Tesis de Doctorado en Física

Aprendizaje y Generalización en Redes Neuronales.

Autor: Leonardo Franco

Director: Dr. Sergio A. Cannas

Facultad de Matemática, Astronomía y Física
Córdoba, Agosto de 2000

A mis padres

Agradecimientos

a Sergio por acercarme al tema de las Redes Neuronales y por su constante apoyo, confianza, tranquilidad y eficiencia a lo largo del desarrollo de esta tesis.

a Pancho por su permanente interés en que todo vaya mejor y su buena onda.

a todo el Gtmc^2 por su predisposicón a colaborar con la solución de los variados problemas que uno debe superar.

a Sara Solla por sus interesantes y motivadoras discusiones sobre redes.

a todas las instituciones y personas que apoyan la ciencia en Argentina y que me apoyaron en el desarrollo de esta tesis. En especial a Conicor y SeCyT-UNC quienes a través de sus becas hicieron posible realizar este trabajo.

a Paula por estar a mi lado en estos años brindándome tanto amor y compañía.

a mis padres, de quienes estoy muy orgulloso y a quienes dedico esta tesis.

a mis amigos de Córdoba y Tucumán por cada momento compartido.

Resumen

Se diseñan arquitecturas de redes neuronales para los problemas de suma de n números, multiplicación de dos números, problema de corrimiento de bits y problema de paridad obteniéndose exactamente el conjunto completo de pesos sinápticos que hacen que la red compute estas funciones para números de precisión arbitraria. La construcción de arquitecturas resulta de interés tanto para estudios de tipo teórico, interesados en las propiedades de aprendizaje y generalización y en conocer el tamaño mínimo necesario de una arquitectura para poder implementar una dada función, como para posibles aplicaciones prácticas en circuitos masivamente paralelos.

Las arquitecturas implementadas para los tres primeros problemas son todas óptimas en cuanto al número de capas que poseen y el número de neuronas en cada capa está acotado polinomialmente, comparándose favorablemente con anteriores implementaciones.

En el caso del problema de paridad se construye una familia de arquitecturas modulares caracterizadas por el número máximo de conexiones por neurona permitido (fan-in max), el cual regula el grado de modularidad de las arquitecturas.

Se estudian las propiedades de aprendizaje y generalización de las redes neuronales construídas, a través del cálculo del mínimo número de ejemplos necesario para obtener generalización total (MNEFG), así como usando simulaciones numéricas. El MNEFG es obtenido a partir de un análisis detallado de las ecuaciones de los ejemplos para redes pequeñas para luego generalizar los resultados a redes de tamaño arbitrario. Para el caso del aprendizaje de la función de paridad en una red de máxima modularidad se presentan evidencias de una transición de fase de memorización a generalización perfecta cuando el número de ejemplos en el conjunto de entrenamiento es del orden del MNEFG.

A partir del estudio de los ejemplos que conforman el MNEFG se deriva un criterio general para la selección de ejemplos independiente de la arquitectura. Se construye un algoritmo de selección de ejemplos basado en el criterio anterior, el cual es probado con éxito con funciones aleatorias implementadas en arquitecturas con campos receptivos locales. A partir de los resultados obtenidos para el problema de paridad implementado en diferentes arquitecturas monolíticas y modulares se observa que la habilidad de generalización es sustancialmente mejorada con la modularidad.

Se propone el MNEFG como parámetro para medir la complejidad de funciones y se discuten diversas cuestiones relacionadas con la capacidad de generalización de redes.

Abstract

We design new feed forward multilayered neural networks which perform different arithmetic operations, such as addition of n numbers, multiplication of two numbers, bit-shifting and parity, obtaining exactly the whole set of synaptic couplings that allow the networks to compute the mentioned functions for arbitrary precision numbers. The design of architectures is a complex task involving both theoretical and practical issues. The resulting architectures could be used as platforms for the study of learning properties and for testing learning algorithms. Also they could be implemented in massively parallel microcircuits, being the size of the networks one of the most important concerning from a practical and theoretical point of view.

The first three architectures are all optimal in depth (number of layers) and the number of neurons in every layer is polinomially bounded, comparing favorably with previous designs.

For the parity function we construct a family of modular architectures characterized by the maximum number of connections per neuron allowed (fan-in max). Through this parameter the modularity of the architectures could be controlled.

We study learning and generalization properties of the architectures constructed through the analytic calculus of the minimum number of examples needed for full generalization(MNEFG) and also through numerical simulations. The MNEFG is obtained from small network equations written for the examples, to later generalize the results to arbitrary size networks. In the parity network with extreme modularity (fan-in max=2) we found evidence of the existence of a phase transition from memorization to perfect learning when the number of examples in the training set approach the MNEFG.

Through the study of the set of examples constituting the MNEFG we derive general criteria for selecting examples in a general architecture and also we construct an algorithm based in the previous criteria which is tested using random functions in local receptive fields networks. From the results obtained in different architectures implementing the parity function it is observed a great improvement in the generalization capacity when using modular networks.

We propose the MNEFG as a parameter to measure the complexity of functions and we discuss many questions related to the generalization abilities of neural networks.

Índice

Lista de Figuras

Capítulo 1

Introducción

1.1 Origen y Desarrollo de los Modelos de Redes Neuronales

El interés por las redes neuronales proviene históricamente de intentos en: 1) modelar matemáticamente el funcionamiento del sistema nervioso de organismos vivos, como por ejemplo el cerebro humano, y 2) construir máquinas capaces de resolver tareas complejas, como una alternativa a las computadoras convencionales tipo máquina de von Neumann, esto es, máquinas digitales, en las cuales la información es procesada secuencialmente mediante un intercambio entre una CPU y una memoria RAM.

Podemos citar como antecedente fundamental en la historia del modelado de sistemas neuronales el trabajo de [McCulloch and Pitts, 1943] en el cual la neurona es modelada como una unidad de cómputo que puede tomar valores bien definidos 0 ó 1, en función de los valores de entrada que recibe provenientes de otras neuronas. Otro hecho de gran significación dentro de las redes neuronales lo constituye el trabajo de Donald Hebb en el cual se propone que *el aprendizaje tiene lugar mediante modificaciones sinápticas.* Este mecanismo de aprendizaje se conoce hoy como *plasticidad sináptica* y representa un concepto fundamental tanto en la neurofisiología como en el desarrollo de las redes neuronales. El mismo establece que el cambio en el valor de las conductancias efectivas de las sinapsis es proporcional a las correlaciones entre las actividades pre y post sinápticas [Hebb, 1949]. En la jerga de las redes neuronales este mecanismo se conoce como *regla de Hebb.*

Frank Rosenblatt extendiendo las ideas de McCullogh y Pitts inventa el *perceptron* en

1957 y con la ayuda de Charles Wightman entre otros, construyen la primer computadora neuronal capaz de procesar una función útil: el *Mark I Perceptron* que funcionaba como un reconocedor de caracteres [Rosenblatt, 1958], [Rosenblatt, 1962]. Las limitaciones computacionales del perceptron simple (dado que solo puede computar las funciones llamadas linealmente separables) son usadas por Minsky and Papert en su libro *Perceptrons* como prueba de que las redes neuronales eran incapaces de computar ciertas funciones simples. Esta conclusión errónea, ya que el problema de cómputo puede ser solucionado agregando una capa intermedia en la arquitectura del perceptron, provoca que el tema de redes neuronales sea fuertemente frenado en el período 1967-1982. [Minsky and Papert, 1969], [Hecht-Nielsen, 1989].

John Hopfield en 1982 aplica la regla de aprendizaje hebbiana en las redes neuronales en un modelo de *memoria asociativa* conocido como *modelo de Hopfield* renovando el interés en las redes neuronales, especialmente dentro del ámbito de la física [Hopfield, 1982], [Hopfield, 1984], [Amit et al., 1989].

A mediados de los años 80 las aplicaciones de las redes neuronales a problemas prácticos como reconocimiento de patrones, reconocimiento de voz, clasificación de datos, predicción de series temporales, compresión de imágenes, etc., comienzan a hacerse posible debido al gran avance en la construcción de computadores cada vez más veloces [Hertz et al., 1991], [Hecht-Nielsen, 1989], [Sejnowski and Rosenberg, 1987], [Pomerleau, 1989]. A la par se desarrola el estudio fundamentalmente teórico y con base matemática en la mecánica estadística de las propiedades de las redes neuronales [Derrida et al., 1987], [Amit et al., 1985], [Watkin et al., 1993], [Solla, 1991].

Un avance de significativa importancia se produce con la introducción del algoritmo de aprendizaje conocido como "back-propagation" o método de retro-propagación de errores que permite realizar el aprendizaje en redes multicapas con gran éxito, convirtiéndose en el algoritmo de aprendizaje más utilizado [Rumelhart et al., 1986a].

Hoy en día el tema de redes neuronales se encuentra bastante desarrollado, aunque persisten cuestiones fundamentales sin aclarar [Toulouse, 1989]. El estudio de las redes encuentra su ámbito de estudio y desarrollo en dos áreas claramente diferenciadas: un área principalmente interdisciplinaria llamada *neurociencia computacional* con interés en modelar funciones análogas a las del cerebro de organismos vivos y un área de lo que llamaríamos *redes neuronales artificiales* mayormente dedicada a las aplicaciones.

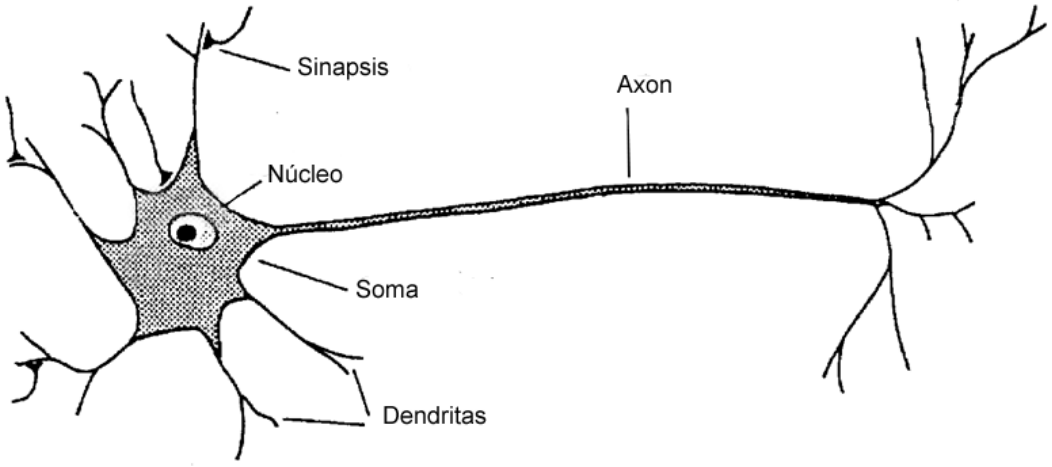


Figura 1.1: Esquema simplificado de una neurona biológica.

1.2 Redes Neuronales

En un sentido amplio, las redes neuronales son sistemas constituídos por un conjunto de unidades a las cuales denominaremos **neuronas**, interconectadas a través de conexiones que llamaremos **sinapsis** o pesos sinápticos.

El funcionamiento de neuronas biológicas es bastante complejo, involucrando complicados procesos químicos en la transmisión de los impulsos y en la computación de los mismos. Un esquema simple de neurona biológica se muestra en la figura 1.1. Los impulsos nerviosos procedentes de las neuronas aferentes generan potenciales eléctricos en las uniones sinápticas que son transmitidos a través de las dendritas hacia el soma o cuerpo celular. En caso que el potencial total en el soma exceda un determinado umbral propio de cada neurona, se genera un potencial de acción que se transmite por el axón hacia otras neuronas, y en este caso decimos que la neurona ha disparado. Luego de este proceso la neurona debe permanecer un tiempo inactiva (período refractario); si en cambio el potencial de membrana no alcanza el valor umbral, éste decae hasta el valor de potencial de reposo. La neurona actúa como un dispositivo integrador, sumando los impulsos provenientes de otras neuronas y emitiendo o no un potencial de acción dependiendo de si la suma de impulsos supera o no un valor umbral.

El proceso químico que se produce entre las sinapsis de distintas neuronas es altamente complejo y esta fuera del alcance de este trabajo. Referiremos al lector interesado a [Kuffler and Nicholls, 1984], [Mac Gregor, R.J. (1987)], [Levine, D.S. (1991)].

No obstante, en lo que respecta al modelado de neuronas, solo se consideran ciertos

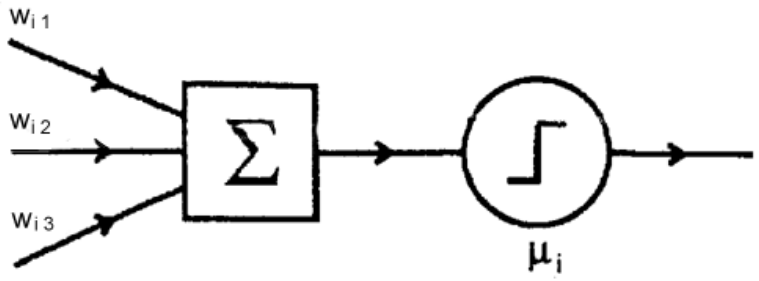


Figura 1.2: Esquema de una neurona de McCullogh y Pitts.

aspectos funcionales esenciales: la neurona como dispositivo de dos estados (en reposo y activada), la cual integra los impulsos provenientes de otras neuronas. Estos modelos, si bien ultra simplificados, contienen elementos suficientes para dar cuenta de diversas funciones cerebrales, tales como memoria asociativa, aprendizaje y generalización.

En 1943 McCullogh y Pitts propusieron un modelo simple de neurona, en el cual el estado de la neurona puede tomar dos valores: (1) estado activado, la neurona *dispara*, (0) estado desactivado, la neurona se encuentra en reposo. El estado S_i en que se encuentra una neurona dependerá del valor que alcance la suma de los impulsos que recibe a través de los pesos sinápticos desde otras neuronas. Si esta suma supera un cierto valor umbral μ_i la neurona se activa, i.e. $S_i = 1$, y en cambio si esta suma es menor que el valor umbral la neurona estará inactiva, i.e., $S_i = 0$:

$$S_i = \begin{cases} 1 & \text{si} \quad h_i = \sum_j w_{ij} S_j \geq \mu_i \\ 0 & \text{si} \quad h_i = \sum_j w_{ij} S_j < \mu_i \end{cases} \tag{1.1}$$

h_i simula un potencial de membrana efectivo, w_{ij} serán los valores de las conexiones sinápticas entre neuronas y μ_i representa el valor umbral de la neurona.

En la figura 1.2 se muestra un esquema del modelo de neurona de dos estados de McCullogh y Pitts.

A las neuronas cuyo estado toma valores discretos las denominaremos neuronas discre-

tas o *neuronas booleanas* y son las que en mayor medida usaremos en este trabajo. Otra clase muy usual de neurona también simple es la llamada neurona *sigmoidal*, cuyo estado de activación dependerá a través de una función de tipo sigmoidea de las entradas que recibe, pudiendo entonces tomar valores continuos, normalmente restringidos al intervalo [0,1].

Según la manera en que acomodemos las neuronas y de las características de las conexiones sinápticas obtendremos distintos tipos de redes neuronales. Podemos decir que existen principalmente 2 clases de modelos de redes neuronales: redes recurrentes o con atractores y redes de tipo feedforward o sin retroalimentación.

Las redes con atractores han surgido como un modelo para el problema de memoria asociativa, el cual consiste en almacenar una serie de patrones de forma tal que cuando a la red neuronal se le presente un patrón cualquiera la red recupere a través de su dinámica el patrón almacenado que más se le asemeje [Muller and Reinhardt, 1989]. Un modelo de fundamental importancia dentro del desarrollo de las redes neuronales y que pertenece a esta clase de arquitecturas lo constituye el modelo de Hopfield basado en la regla de aprendizaje de Hebb. Esta regla determina el valor de los pesos sinápticos (w_{ij}) entre neuronas en función de los patrones que deseamos almacenar:

$$w_{ij} = \frac{1}{N} \sum_{\mu=1}^{p} \xi_i^\mu \xi_j^\mu, \tag{1.2}$$

donde N es el número de neuronas, p es el número de patrones que deseamos almacenar y ξ_i^μ es la componente i del patrón μ, el cual tendrá tantas componentes como número de neuronas, i.e., N.

La aplicación de la regla de Hebb permitirá que los patrones almacenados sean recuperados si un patrón relativamente cercano es presentado a la red. En una idea esquemática podemos ver que esto corresponde a la creación de una superficie de energía cuyos mínimos corresponderán a los patrones almacenados y la propiedad de recuperación de patrones cercanos la visualizamos como las cuencas de atracción que rodean a los mínimos [Hopfield, 1982a], [Amit, 1984], [Hertz et al. 1991], [Montemurro, 1999] (ver figura 1.3).

Las redes de tipo feedforward se diferencian de las redes de atractores en que el flujo de información posee una dirección determinada. En esta clase de modelos se distribuyen las neuronas en *capas*, las cuales según su ubicación podrán ser: *capa de entrada*, encargada de recibir la información a procesar; *capas intermedias*, encargadas de procesar la información

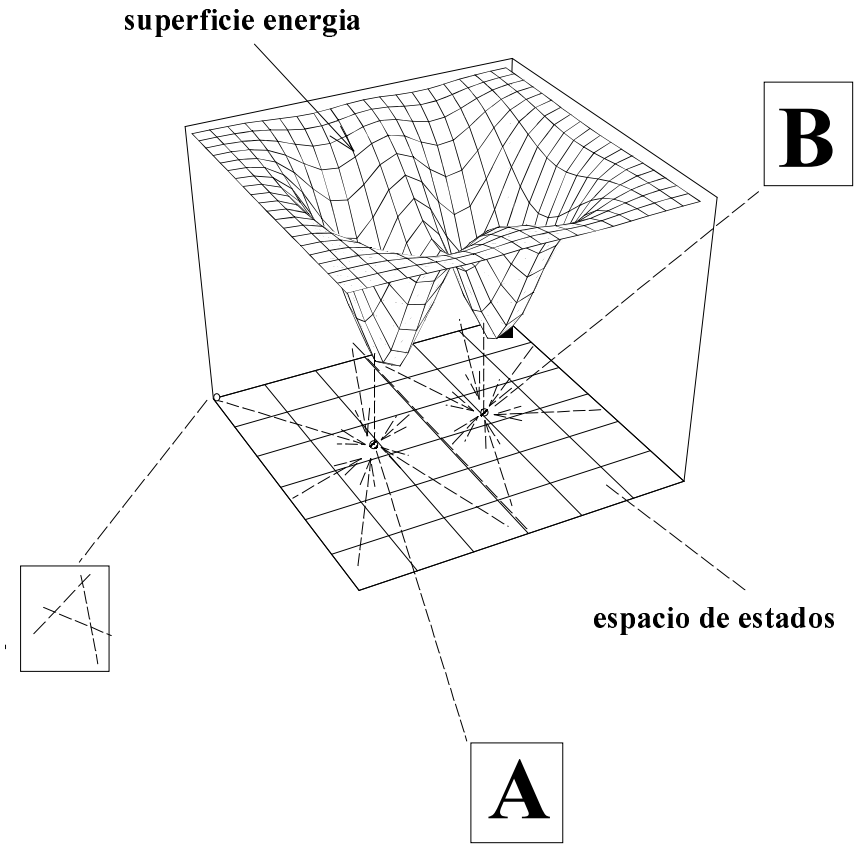


Figura 1.3: Representación esquemática de la superficie de energía asociada al almacenamiento de dos memorias (A, B)

y *capa de salida*, de donde leeremos el resultado del cómputo.

Las redes ordenadas en capas con conexiones de tipo feedforward (sin retroalimentación) son las llamadas **perceptrones** y son las que usaremos a lo largo de esta tesis. En estas, el estado de activación de una neurona en una capa dada depende unicamente del estado de de neuronas en capas anteriores. Es decir, no existen conexiones sinápticas, ni hacia atrás, ni dentro de una misma capa. El número de neuronas en cada capa es absolutamente arbitrario. De esta manera, los perceptrones permiten llevar a cabo asociaciones entre patrones de información de diferente naturaleza (heteroasociación), en tanto que las redes con atractores solo permiten asociaciones entre patrones del mismo tipo (autoasociación). Las redes con atractores son especialmente eficientes para el reconocimiento asociativo de patrones, especialmente visuales. En el caso de los perceptrones, la heteroasociación permite la implementación de tareas complejas, tales como reconocimiento de interlocutores a partir de patrones de voz [Bennani, 1994], predicción de series temporales [Petridis and Kehagias, 1998], reconocimiento de estructuras en datos estadísticos [Haykin, 1995], entre muchas otras. Una de las propiedades más importantes de los perceptrones es que permiten una implementación relativamente sencilla de *algoritmos de aprendizaje mediante ejemplos*, tal como se detalla en las secciones siguientes.

1.2.1 Perceptrones

La estructura más simple dentro de las redes tipo perceptron lo constituye el *perceptron simple* cuya estructura consta de un cierto número N de neuronas en la capa de entrada y una única neurona de salida. Esta arquitectura no posee capas intermedias y la encargada de procesar la información será la neurona de salida.

Las limitaciones computacionales del perceptron simple fueron advertidas por [Minsky and Papert, 1969], ya que esta clase de arquitecturas no pueden computar funciones llamadas *linealmente no separables*, siendo un ejemplo clásico de ellas la función booleana XOR [Hertz et al., 1991], [Muller and Reinhardt, 1987]. A pesar de sus grandes limitaciones el perceptron simple ha sido objeto de numerosos estuidos que han permitido conocer propiedades y características interesantes de las redes neuronales en general [Watkin et al., 1989].

Los problemas de computabilidad del perceptron simple pueden superarse facilmente agregando capas ocultas y más aún, puede demostrarse (ver [Muller and Reinhardt, 1987],

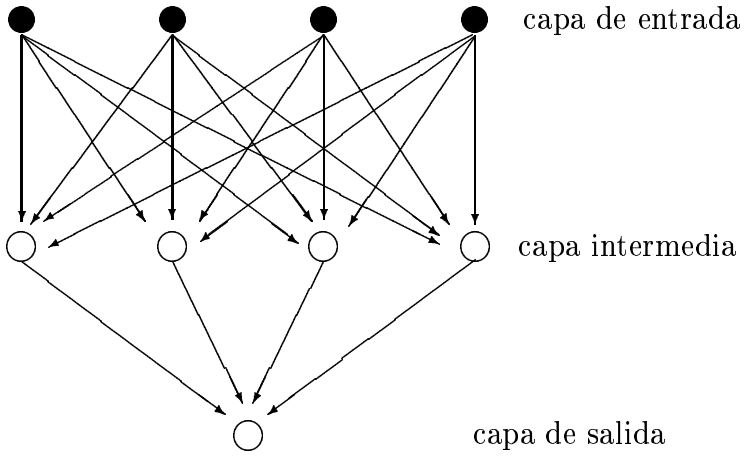


Figura 1.4: Arquitectura de una red neuronal tipo feedforward (perceptron) con una sola capa oculta. Las conexiones representadas por vectores indican el sentido del flujo de la información

[Denker et al., 1987]) que cualquier función booleana puede ser computada con un perceptron de una única capa oculta siempre y cuando exista un número suficiente de neuronas en esta capa, en general exponencial con el número de entradas. Si bien este resultado reviste gran importancia desde el punto de vista teórico, no resulta demasiado práctico a la hora de las aplicaciones debido a la dependencia exponencial en el número de neuronas. Normalmente se recurre a agregar más capas ocultas y de esta manera reducir el número de neuronas aunque incrementando la *profundidad* de la red. La profundidad de una red está definida como el número de capas que posee una red sin contabilizar la capa de entrada cuyas neuronas no realizan ninguna función de cómputo, sino que solamente proveen a la red de los valores a considerar. En la figura 1.4 se muestra una arquitectura de una red neuronal correspondiente a un perceptron con una capa oculta (profundidad 2).

Dependiendo de como organicemos las conexiones entre las neuronas de distintas capas obtendremos distintas clases de perceptrones. La estructura obtenida se denomina

arquitectura de la red.

El aprendizaje en una red neuronal se realiza a través del ajuste de sus parámetros libres (intensidades sinápticas) mediante algoritmos de aprendizaje, basados en procesos de modificación sináptica.

Las diversas clases de algoritmos de aprendizaje guardan una íntima relación con la arquitectura de red que utilicemos y en este contexto podemos dividir los algoritmos de aprendizaje en dos clases principales: *supervisados* y *no supervisados*. Los primeros van modificando los valores de las sinapsis según la respuesta obtenida por la red para un ejemplo dado o un conjunto de ejemplos. La respuesta obtenida por la red es comparada con la respuesta correcta (conocida) y en función a esta comparación se modificará el valor de las sinapsis.

Los algoritmos de aprendizaje supervisado involucran en general la *optimización* de una función costo o función error que podrá definirse de acuerdo al problema y dentro de ciertos criterios. Estos algoritmos son en general del tipo descenso por el gradiente, donde la función a optimizar será alguna función que mida la discrepancia (distancia) existente entre la salida o respuesta obtenida con la respuesta deseada; una opción usual para la función error es utilizar una función cuadrática. No obstante, como todo algoritmo basado en el descenso por el gradiente, el mismo presenta la desventaja quedar atrapado en mínimos locales.

Dentro de los algoritmos de aprendizaje supervisado podemos citar el algoritmo de Back-propagation (Retro-propagación de errores) [Rumelhart et al., 1986a] el cual ha demostrado una gran utilidad, siendo el algoritmo de mayor uso.

El algoritmo de "simulated annealing" (recocido simulado) introducido por [Kirckpatrick et al., 1983], y que debe su nombre a la analogía con el proceso de enfriado lento de un material con el fin de que el mismo llegue a una configuración de equilibrio, ha demostrado una gran practicidad. Este algoritmo realiza una búsqueda aleatoria en el espacio de configuraciones adoptando aquellas en las cuales el error va disminuyendo. Sin embargo, dependiendo de una temperatura auxiliar que irá descendiendo con el paso de las iteraciones, configuraciones con aumento del error son accesibles, permitiendo que el sistema escape de posibles mínimos locales y pueda llegar al mínimo global de la función error que coincidirá con el objetivo de la tarea propuesta [Franco, 1995].

Por otro lado, los algoritmos de aprendizaje no supervisado involucran la definición

a priori de una regla de aprendizaje, que determinará el futuro valor de las sinapsis en función de la distribución de patrones de entrada exclusivamente. Dentro de esta clase encontramos por ejemplo la regla de Hebb [Hopfield, 1982] que ha proporcionado muy buenos resultados, en el problema de memoria asociativa. A través de estos algoritmos es posible dotar a la red de una superficie de error con mínimos locales y cuencas de atracción alrededor de ciertos patrones que deseamos almacenar, de forma tal que cuando le entregamos a la red un patrón de entrada similar a alguno almacenado la propia dinámica de la red nos lleva a obtener el patrón anteriormente almacenado o memorizado [Haykin, 1994], [Hertz, 1988]. Algoritmos de aprendizaje no supervisado también son usados para problemas de clasificación o "clustering", donde se pretende clasificar de acuerdo a algunas características (a veces no conocidas previamente) una cierta cantidad de datos con el fin de agruparlos y tipificarlos.

En esta tesis nos restringiremos a algoritmos de aprendizaje supervisados en los cuales ciertos ejemplos, que formarán parte del conjunto de aprendizaje, son enseñados a la red. Un ejemplo es una relación entrada-salida, un par $\{\vec{x}, d\}$ definido por un conjunto de valores o vector de entrada $\vec{x}$ y una respuesta correcta y conocida d, asociada al valor de entrada. Por ejemplo, en las redes que usaremos en el capítulo 3 para sumar números el vector de entrada estará definido por los números a sumar y la respuesta correspondiente será el resultado correcto de la operación de suma.

El error de aprendizaje se define como la suma de los errores cuadráticos cometidos por la red para los ejemplos que constituyen el conjunto de aprendizaje. Una cantidad relacionada y muy útil para evaluar la perfomance de la red es el error de generalización que se obtiene al sumar los errores cometidos por la red pero en todos los ejemplos posibles (número de ejemplos finitos como en el caso de entradas binarias) o en un gran número de ejemplos mayor que los pertenecientes al conjunto de aprendizaje. El error de generalización suele evaluarse una vez que la red ha aprendido con éxito el conjunto de ejemplos de entrenamiento, es decir una vez que el error de aprendizaje es cero o menor a un cierto valor pequeño predeterminado. Los ejemplos que componen el conjunto de entrenamiento determinarán, conjuntamente con la función usada para computar el error entre la respuesta de la red y la respuesta correcta, la *superficie de error*. Esta superficie, de dimensión igual al número de sinapsis que posee la red, será de fundamental importancia para el éxito de los algoritmos de aprendizaje, pues puede

contener mínimos locales, mesetas, etc., que dificultan la convergencia hacia el o los mínimos globales determinados por los valores que hacen que la red compute con éxito los ejemplos del conjunto de aprendizaje.

La capacidad de generalización es la propiedad de que una red neuronal compute correctamente ejemplos que no le han sido enseñados previamente, siendo una de las propiedades más interesantes de las redes neuronales. Hasta el momento no existe una teoría completa para entender este fenómeno, el cual es influenciado por tres factores principales: tamaño y propiedades del conjunto de entrenamiento, arquitectura de la red y complejidad del problema a tratar [Haykin, 1994].

1.3 Diseño de Arquitecturas

El problema de diseño de arquitecturas es un problema abierto y de gran interés actual, siendo un proceso que involucra la consideración de diversos factores como complejidad de la función a implementar, cantidad de capas ocultas, número de neuronas en cada capa, conexiado, etc. Existen algunos algoritmos para la construcción de redes, entre los que podemos destacar de acuerdo a su naturaleza los algoritmos de tipo "pruning" o de eliminación de sinapsis y algoritmos de crecimiento o "network growing" [Haykin, 1994], [Hertz et al., 1991]. Los algoritmos de tipo pruning se basan en monitorear el valor de los pesos sinápticos e ir eliminando o reemplazando aquellos cuyos valores son muy pequeños o que no se modifican a lo largo del aprendizaje [Sejnowski and Rosenberg, 1987], [Hassibi et al., 1993]. En otra palabras, dada una función a aprender, se parte de una arquitectura redundante que asegure la computabilidad de la misma y se eliminan sistemáticamente conexiones y neuronas hasta alcanzar un tamaño mínimo que resuelva el problema.

Los algoritmos de crecimiento partiendo de una arquitectura mínima van agregando neuronas en diferentes capas hasta alcanzar una arquitectura capaz de computar la función deseada [Marchand et al, 1990]. El algoritmo de tiling [Mézard and Nadal, 1989] consiste en agregar capas a la arquitectura que cada vez contendrán un número menor de neuronas y de esta manera el algoritmo convergerá a una única neurona de salida. El éxito del algoritmo se basa en obtener en cada capa una *representación fiel* de los patrones de entrada, esto es si dos patrones de entrada diferentes producen salidas diferentes, sus representaciones internas (estados de las neuronas intermedias) deberán ser diferentes en

cada capa. La idea del algoritmo consiste en comenzar cada capa con una unidad *maestra* que computará de la mejor manera posible la función deseada y luego agregar unidades *auxiliares* hasta lograr una representación fiel.

Si bien los algoritmos de diseño de arquitecturas han demostrado cierta confiabilidad, la heurística y el conocimiento a priori de las propiedades y simetrías de la función a implementar continúan siendo elementos muy importantes a la hora de la construcción de una arquitectura.

Un modelo de arquitectura usual muy utilizado en los primeros desarrollos de redes neuronales y aún en el presente, consiste en conectar totalmente las neuronas entre capas adyacentes. Este esquema de tipo monolítico tiene como principales desventajas poseer un gran número de conexiones lo que involucra para su implementación una gran capacidad de memoria y largos tiempos de cómputo para el proceso de aprendizaje. También es frecuente observar, con esta clase de conexionado, una pobre capacidad de generalización y un problema de sobreespecialización en algunos ejemplos conocido como "overfitting" [Boers et al., 1993], [Haykin, 1994]. Esto puede entenderse si pensamos el aprendizaje como un proceso de ajuste no lineal de funciones, donde los parámetros de ajuste son las sinapsis. De esta manera, un exceso de parámetros tendrá como consecuencia un mejor aprendizaje de los ejemplos de entrenamiento (puntos del ajuste) a costa de un empobrecimiento en la capacidad de generalización (extrapolación e interpolación), tal como se muestra en la figura 1.3 donde se observan los ejemplos (conjunto e entrenamiento), la función implementada por la red y la función que se desea implementar. [Haykin, 1995]. La manera más evidente de disminuir este efecto indeseado sin perder capacidad de cómputo consiste en disminuir el número de sinapsis de cada neurona, haciendo que las mismas se conecten solo a un subconjunto reducido de neuronas de la capa siguiente. Estos subconjuntos se denominan *campos receptivos locales.* Un tipo particular de arquitectura con campos receptivos locales son las arquitecturas modulares. En éstas la información se procesa en paralelo en módulos neuronales sin conexiones laterales entre sí cuyas salidas se combinan en alguna capa o módulo posterior. Los diferentes módulos pueden entrenarse independiente o simultáneamente, dependiendo de la aplicación particular. Las arquitecturas modulares en comparación con las arquitecturas totalmente conexas han demostrado tener una mejor capacidad de generalización, mayor velocidad en el aprendizaje y una mayor adaptabilidad a modificaciones de la tarea original [Haykin,

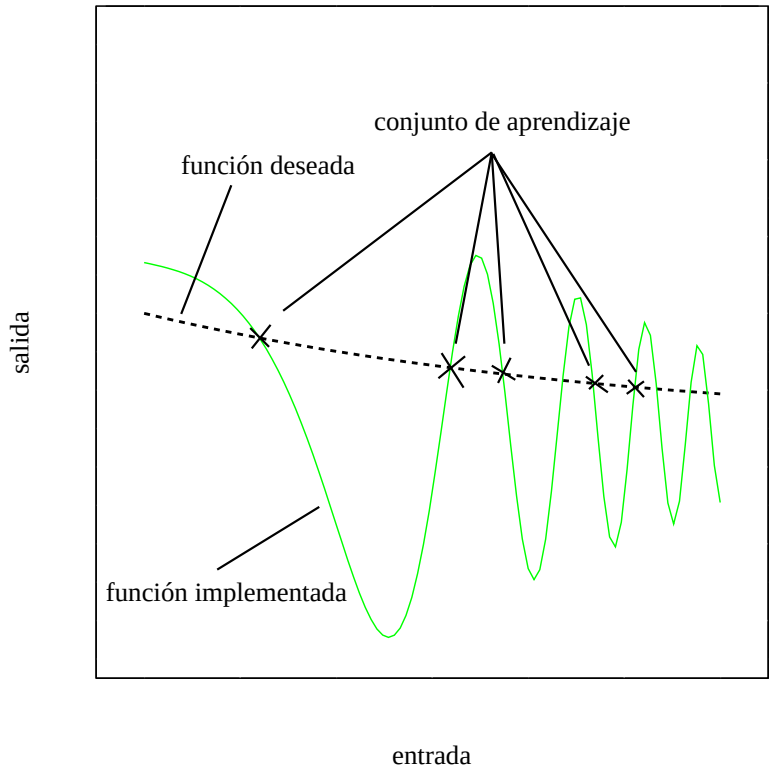


Figura 1.5: Representación esquemática de un proceso de aprendizaje con sobreespecialización ("overfitting") en algunos ejemplos obteniéndose una mala generalización de la función que se desea implementar

1994], [Bennani, 1994], [Franco and Cannas, 2000b]. Por otra parte estructuras modulares resultan de gran interés en el modelado de sistemas neurológicos como por ejemplo en áreas de la corteza visual [Houk, 1992], [Van Essen et al., 1992] y han mostrado eficiencia en diversas aplicaciones tales como reconocimiento de voz e imágenes [Bennani, 1994], [Haykin, 1994], predicción de series temporales [Petridis and Kehagias, 1998], entre muchas otras.

1.4 Selección de Ejemplos y Generalización

La cuestión de elegir y conocer el tamaño adecuado del conjunto de entrenamiento es un problema de un gran interés práctico y teórico [Kinzel and Ruján, 1987], [Jung and Opper, 1996]. Uno de los resultados más generales e interesantes es el obtenido por [Baum and Haussler, 1989] con ejemplos seleccionados al azar en una red tipo perceptron con una capa oculta. El resultado obtenido permite conocer el número N necesario de ejemplos para obtener generalización en función de una tolerancia permitida para el error, ϵ:

$$N \geq \frac{32W}{\epsilon} ln(\frac{32M}{\epsilon}) \tag{1.3}$$

donde W denota el número de pesos sinápticos, M es el número de neuronas en la capa oculta y ln(x) es la función logaritmo natural. En la práctica, en primera aproximación suele considerarse que del orden de

$$N > \frac{W}{\epsilon} \tag{1.4}$$

ejemplos son necesarios para obtener generalización. Así, si permitimos un error de generalización del 10 %, necesitaremos un número de ejemplos de diez veces el número de sinapsis existentes en la red para obtenerlo. El uso de ejemplos al azar como conjunto entrenamiento a menudo resulta poco eficiente, ya que muchos de los ejemplos presentarán información redundante o insuficiente acerca de la función a aprender. Esto implica en general la necesidad de un número elevado de ejemplos para obtener un grado adecuado de generalización, así como un elevado tiempo de cómputo para el entrenamiento. Sin embargo, en muchos casos ocurre que un subconjunto reducido de ejemplos contiene toda la información acerca de la función objetivo. Una selección adecuada del conjunto de entrenamiento debería por lo tanto resultar en un proceso de aprendizaje más rápido y eficiente. Así en los últimos años se desarrollaron diversos algoritmos de selección de

ejemplos, recibiendo diversos nombres en la literatura tales como aprendizaje activo ("active learning") [Cohn et al., 1994], [Plutowski and White, 1993], [Jung and Opper, 1996] ó aprendizaje por preguntas ("query learning") [Baum, 1991], [Sollich, 1994], [Hancock et al., 1994], [Kinzel and Ruján, 1990]. No obstante, la teoría general de selección de ejemplos y su relación con la capacidad de generalización continúa siendo un problema abierto y será una de las cuestiones centrales en esta tesis. Es claro que la selección de un conjunto reducido de ejemplos que garantice un grado adecuado de generalización depende tanto de la complejidad de la función a aprender como de la arquitectura de la red. Resulta así importante definir cantidades que permitan relacionar la capacidad de generalización de una red con el tipo y número de ejemplos utilizados en el entrenamiento. Un parámetro importante en este sentido es la llamada *dimensión VC*, en honor a sus creadores Vapnik y Chervonenkis [Vapnik and Chervonenkis, 1971]. Este parámetro ha demostrado tener una relación fundamental en la determinación de la habilidad una arquitectura de generalizar y está relacionado con la capacidad de la red de implementar diferentes funciones binarias. Consideremos el caso de una clasificación binaria de patrones, es decir la posible respuesta d a la clasificación de un patrón de entrada es bivaluada: $d \in \{0, 1\}$. Definiremos una dicotomía como una regla de decisión binaria. Si $\mathcal{F}$ denota la familia de dicotomías implementadas por una arquitectura, es decir:

$$\mathcal{F} = \{F(x, w) : w \in \mathcal{W}, F : R^p \to \{0, 1\}\} \tag{1.5}$$

donde x es un vector de entrada de dimensión p y w representa los pesos sinápticos pertenecientes al espacio de pesos $\mathcal{W}$. Si definimos $\mathcal{L}$ como un conjunto de N puntos que pertenecen al espacio p-dimensional X de vectores de entrada, entonces una dicotomía implementada por la arquitectura separará el conjunto $\mathcal{L}$ espacio X en dos subespacios $\mathcal{L}_0$ y $\mathcal{L}_1$ de manera que:

$$F(x, w) = \begin{cases} 0 & si \quad x \in \mathcal{L}_0 \\ 1 & si \quad x \in \mathcal{L}_1 \end{cases} \tag{1.6}$$

para un conjunto de valores w fijo. Diremos que $\mathcal{L}$ es *cubierto* por $\mathcal{F}$ si todas las dicotomías de $\mathcal{L}$ pueden ser inducidas por funciones en $\mathcal{F}$.

La dimensión VC de una familia de dicotomías $\mathcal{F}$ es la máxima cardinalidad de cualquier conjunto de puntos $\mathcal{L}$ pertenecientes al espacio de entrada X que es cubierto por $\mathcal{F}$.

Consideremos el siguiente ejemplo para tratar de comprender el concepto de dimensión VC: Tomemos una arquitectura tipo perceptron con N neuronas de entrada, una capa

intermedia con N neuronas y una neurona de salida. Una función binaria o *dicotomía* consistirá en definir un valor binario (0) o (1) de respuesta de la red correspondiente a un conjunto de valores de entrada x (este vector define un punto en el espacio N dimensional). Si definimos solamente un cierto número de puntos, p, podemos ver que exisitirán a lo sumo 2^p distintas funciones binarias. La dimensión VC será entonces el mayor valor de p para el cual la arquitectura es capaz de computar todas las 2^p funciones posibles cualquiera sea el conjunto de p ejemplos [Hertz et al., 1991], [Haykin, 1994]. La dimensión VC es un parámetro dependiente exclusivamente de la arquitectura. De su definición vemos que, dada una arquitectura, es imposible garantizar cualquier grado de generalización si el tamaño del conjunto de entrenamiento es inferior a la dimensión VC, para cualquier función computable por la red. No obstante su utilidad, el cálculo de la dimensión VC resulta sumamente engorroso en los casos prácticos. Por otra parte, dada una función específica, el tener un número de ejemplos superior a la dimensión VC es condición necesaria pero no suficiente para obtener generalización.

1.5 Estructura de la tesis

En el capítulo 2 se muestra la construcción de arquitecturas óptimas en profundidad para implementar distintas operaciones aritméticas: suma de N números, producto de dos números y corrimiento de bits, todas operaciones usuales en microprocesadores, obteniéndose expresiones explícitas para los valores de pesos sinápticos que permiten computar exactamente la función deseada para números de precisión arbitraria. También se construyó una familia de arquitecturas modulares para implementar la función de paridad basada en la arquitectura usual completamente conexa para computar la paridad de N bits. Se analizan las propiedades de estas arquitecturas en términos de número de capas ocultas, número de conexiones, etc. En el capítulo 3 se analizan las propiedades de generalización de las redes construídas en el capitulo 2 y de otras arquitecturas más generales. Para ello introducimos el concepto de mínimo número de ejemplos necesarios para generalización total (MNEFG, "Minimum Number of Examples for Full Generalization"), el cual es calculado en cada caso para tamaños arbitrarios de redes mediante un método de cálculo desarrollado al efecto. Este parámetro se propone como un estimador de la complejidad de las funciones. A partir del método usado para calcular el MNEFG

derivamos un algoritmo general para seleccionar ejemplos independientemente de la arquitectura y de la función elegida. El algoritmo es probado con éxito en funciones aleatorias implementadas en una arquitectura con campos receptivos locales. Las capacidades de generalización de la familia de redes modulares que implementan la función de paridad es analizada, observándose una verdadera mejora en las habilidades de generalización a medida que la modularidad de la red es incrementada. Se analiza también la capacidad de cómputo de las arquitecturas modulares antes mencionadas, mediante el cálculo numérico de la entropía de información de estas redes. Finalmente en el capítulo 4 se discuten los resultados obtenidos y se presentan las conclusiones.

Capítulo 2

Diseño de Arquitecturas

2.1 Introducción

Un problema central en el tema de Redes Neuronales es el diseño de arquitecturas para resolver un problema específico. La implementación óptima de una red tipo "feed-forward" requiere el análisis de diversas cuestiones: complejidad del problema, profundidad de la red (número de capas ocultas), número de neuronas en cada capa, tipo de neuronas (binarias, sigmoidales, etc), tipo de sinapsis (continuas, discretas), convergencia de los algoritmos de aprendizaje, capacidad de generalización, etc.

En este capítulo presentaremos un conjunto de arquitecturas diseñadas para la implementación de redes neuronales para computar los problemas de **Suma de N números, Multiplicación de dos números, Corrimiento de bits (Bit-Shifting) y Paridad.** Consideraremos redes compuestas por neuronas binarias, las cuales pueden tomar dos estados: activada (uno), desactivada (cero). La actividad de una neurona σ_i, vendrá determinada por una función de activación lineal con umbral de la forma:

$$\sigma_i = \Theta\left[\sum_j J_{ij} S_j - T_i\right], \tag{2.1}$$

donde $\Theta(x)$ es la función de activación, que será 1 si $x \geq 0$ y 0 en otro caso, J_{ij} son los pesos sinápticos, $S_j = \{0, 1\}$ son las entradas que la neurona recibe, ya sea del exterior (si están en la capa de entrada) o de otras neuronas, y T_i es el umbral de activación de la neurona σ_i. Es decir, cada neurona computa una función booleana de los valores que recibe dependiente de los pesos sinápticos y de su umbral de activación.

La red más simple que uno podría proponer para resolver un determinado problema sería un perceptron simple, consistente en una capa de entrada con N neuronas todas conectadas a una única neurona de salida. El primer problema que uno enfrenta a la hora de resolver problemas con estos simples circuitos es el de la computabilidad, es decir nos preguntamos si con esta clase de arquitecturas podemos computar la función deseada.

Existen una gran clase de problemas, llamados linealmente inseparables que no pueden ser resueltos por perceptrones simples, siendo la función booleana XOR uno de los más simples y famosos [Rumelhart,1986, Hertz et al., 1991, y Haykin, 1994] y en general cualquier problema más o menos no trivial cae dentro de esta categoría. La solución usual a esta falta de computabilidad de los perceptrones simples consiste en agregar capas intermedias, también llamadas capas ocultas. Puede demostrarse que cualquier función booleana puede ser computada usando una red con una sola capa oculta conteniendo un número de neuronas que crece exponencialmente con el número de neuronas de la entrada [Muller et al., 1991]. La dependencia exponencial de este resultado lo torna totalmente impráctico desde el punto de vista de las aplicaciones y además una red con un número exponencial de neuronas tendrá una muy mala capacidad de generalización (Ver capítulo 3, [Boers et al., 1997], [Haykin, 1994]). De esta manera, una cuestión importante a la hora de implementar una red neuronal es el de conocer el mínimo número de capas ocultas necesarias para poder computar un dado problema con la restricción de que el número de neuronas en cada capa esté **acotado polinomialmente** con el número de neuronas en la capa de entrada, esto es el número de neuronas en cada capa intermedia será una función polinomial del número de neuronas de entrada. Definiremos la profundidad de una red como el número de capas que posee sin tener en cuenta la capa de entrada, la cual no realiza función alguna excepto la de proveer los valores de entrada.

Existen ciertos resultados matemáticos sobre la profundidad mínima que deben tener las redes tipo "feed-forward" para computar algunas funciones aritméticas. Por ejemplo para el caso de la suma de N números A. Hajnal et al. [Hajnal et al., 1987], probaron que esta operación puede ser computada con redes de profundidad 2, mientras que la multiplicación requiere al menos profundidad 3. Estos dos problemas están muy relacionados, pues es casi trivial pasar del primero al segundo. Aparte de la profundidad de la red, otro factor muy importante a tener en cuenta es el máximo "fan-in" del circuito, definido como el máximo número de entradas que una neurona puede recibir. Ambas

características (profundidad y "fan-in") son de gran importancia a la hora de su posible implementación en microcircuitos. La profundidad total del circuito estará relacionada con el tiempo total de procesamiento paralelo y por otro lado la tecnología pone una cota sobre el número de conexiones permitidas.

En cuanto a la cuestión relacionada con el aprendizaje de la función deseada dada la arquitectura, cabe señalar que no existe un teorema general para ningún algoritmo que **asegure** la convergencia del aprendizaje; pese a esto podemos señalar que algoritmos como el de "back-propagation" o del tipo "simulated annealing" han demostrado ser muy exitosos en el aprendizaje de funciones en redes de tipo feed-forward. De cualquier manera en este capítulo nos centraremos en demostrar la **computabilidad** de ciertas funciones y no de su aprendizaje, tema que será tratado en los siguientes capítulos.

Aún en casos en los cuales los algoritmos de aprendizaje son exitosos la teoría general de aprendizaje y generalización está lejos de ser totalmente comprendida, existiendo una serie de problemas abiertos, como por ejemplo: ¿Cuál es el número de capas ocultas necesarias para resolver un dado problema?, ¿Cuántas neuronas poner en cada capa?, ¿Cuántos ejemplos son necesarios para el aprendizaje?, ¿Cómo depende la generalización de la arquitectura elegida?, etc.

El hecho de poseer soluciones exactas para problemas no triviales como los considerados aquí, ayuda considerablemente en el análisis de las cuestiones planteadas anteriormente, sirviendo como plataformas con las cuales experimentar distintos algoritmos de aprendizaje. Por otra parte estas soluciones podrían servir para revelar nuevas características de las redes neuronales tal como ocurre con soluciones exactas obtenidas en el campo de la mecánica estadística.

En este capítulo obtendremos modelos de redes que resuelven el problema de adición de N números, multiplicación de dos números, corrimiento de bits y paridad. Las arquitecturas construídas para los tres primeros problemas son todas óptimas (mínimas) en profundidad, mientras que para el problema de paridad construímos una familia de soluciones modulares. Analizamos en todos los casos la estructura de la red en términos de tamaño, número de conexiones, etc.

Las funciones de suma, producto y corrimiento de bits son funciones básicas en el funcionamiento de procesadores de computadores y ahí radica gran parte del interés en la construcción de ellas, sobre todo a medida que la tecnología incrementa la posibilidad

de construir circuitos paralelos. El problema de corrimiento de bits posee un interés adicional, pues esta clase de circuitos ha sido propuestos para explicar el flujo de información entre grupos de neuronas pertenecientes a diferentes niveles de la estructura visual de organismos vivos [Anderson et al., 1987]. Respecto de la función de paridad podemos decir que es una de las funciones booleanas más usadas para testar algoritmos de aprendizaje, dada su simple definición y su gran complejidad.

2.2 Red de profundidad dos para la adición de N números

En esta sección mostraremos la construcción de una red neuronal de tipo "feed-forward" de profundidad 2, compuesta por neuronas de tipo lineal con umbral con un número polinomial tanto de neuronas como de conexiones, para computar el problema de sumar N números de p bits de longitud [Franco and Cannas, 1998]. Fue probado por Hajnal et al. [Hajnal et al., 1987] que la suma de N números puede ser computada por una red neuronal de tamaño polinomial de profundidad 2. La prueba es no constructiva y hasta el momento una red con estas característica no ha sido construída. Siu & Roychowdhury mostraron que es posible construir una red de profundidad 2 [Siu et al., 1994].

Este problema está muy relacionado con el de multiplicación de dos números, potenciación, etc. La mayor dificultad a la hora de la construcción de una red para sumar viene dado por la dificultad de computar el acarreo que resulta de la suma de los bits con menor valor relativo. En casi todos los diseños previos el procedimiento consiste en transformar la suma de N números en la suma de dos números usando técnicas como el modelo de Dortmunder [Hofmeister et al., 1991] y Suma con reducción de acarreo ("Carry Save Addition") [Patterson et al., 1990, y Lauwereins et al., 1991]. Estos procedimientos introducen al menos dos capas de neuronas intermedias sobrepasando la cota óptima conocida de profundidad 2. Presentaremos a continuación la construcción de una red en la cual todos los números son sumados en una sola operación.

La arquitectura de la red consta de una capa de entrada con Np neuronas correspondientes a los N números de p bits a sumar; la capa de salida contiene $M = \log_2(N(2^p - 1) + 1)$ neuronas suficientes para expresar el resultado que a lo sumo puede ser igual a $N(2^p - 1)$(Asumimos que la operación log entrega un valor **entero** mayor o igual que el

resultado exacto de la operación logaritmo). Cada número a sumar S_j $(j = 1, \ldots, N)$ estará representado por un conjunto de p neuronas binarias: $S_j = S_j^{p-1}, S_j^{p-2}, \ldots, S_j^0$. La capa de entrada consiste en p grupos de N bits cada uno, acomodados según su valor relativo, es decir, las neuronas están acomodadas en el siguiente orden:
$S_N^{p-1}, S_{N-1}^{p-1}, \ldots, S_1^{p-1}, S_N^{p-2}, S_{N-1}^{p-2}, \ldots, S_1^{p-2}, \ldots, S_N^0, S_{N-1}^0, \ldots, S_1^0$. El valor relativo de un bit S_j^i es 2^i y puede tomar los valores uno y cero, i.e., $S_j^i = \{0, 1\}$. Las neuronas de salida las denotaremos por $S_{M-1}^o, S_{M-2}^o, \ldots, S_0^o$. De esta manera el número total de neuronas quedará totalmente determinado por el número de neuronas que ubiquemos de la capa oculta, ya que tanto la capa de entrada como la de salida están claramente definidas por el problema en cuestión. Las neuronas de la capa intermedia las acomodaremos en M grupos. El grupo k tendrá N_k neuronas, cuya salida será conectada solamente a la neurona de la capa de salida S_k^o.

Analizaremos cada bit de la capa de salida independientemente, comenzando por el bit de menor valor relativo S_0^o (correspondiente al valor 2^0). El procedimiento para este caso merece particular atención pues el resto de la estructura, correspondientes a los otros bits, será similar. El bit con menor valor relativo, es decir el bit de salida S_0^o, debe ser **uno** si el resultado de la suma H de los bits de entrada (S_i^0),

$$H = \sum_{i=1}^{N} S_i^0, \tag{2.2}$$

es **par** (incluyendo el cero), y **cero** si H es **impar**:

$$S_0^o = \begin{cases} 1 & \text{si H es impar} \\ \\ 0 & \text{si H es par} \end{cases} \tag{2.3}$$

En otras palabras, este bit computa la paridad de H. Podemos computar esta función en la siguiente manera: ponemos en la capa intermedia $N_0 = N$ neuronas, $\sigma_i^0 = 0, 1$ $(i = 0, ..., N_0 - 1)$, que pertenecerán al grupo $k = 0$. Las conexiones sinápticas entre estas neuronas y el grupo $k = 0$ de la entrada serán elegidas de tal manera que más de la mitad de las neuronas intermedias estén prendidas $(\sigma_i^0 = 1)$ cuando H sea impar mientras que menos de la mitad de las neuronas estarán activadas $(\sigma_i^0 = 0)$ cuando H sea par. ¿Cómo seleccionamos los valores de las sinapsis? Dado que en el problema de paridad todos los bits tienen el mismo peso, es razonable poner todas las sinapsis que conectan las neuronas

H=Resultado Suma	N_o	Neuronas ON	Neuronas OFF	S_0^o
impar	par	$(N_o/2)+1$	$(N_o/2)-1$	ON
impar	impar	$(N_o+1)/2$	$(N_o-1)/2$	ON
par	par	$N_o/2$	$N_o/2$	OFF
par	impar	$(N_o-1)/2$	$(N_o+1)/2$	OFF

Tabla 2.1: Número de neuronas prendidas(ON) y apagadas(OFF) en el grupo intermedio ($k=0:\ \sigma_i^o,\ i=0,\ldots,N_0-1$), para diferentes combinaciones de N_0 y de la suma de bits de entrada H (Ec. 2). La última columna indica el valor requerido para computar la paridad de H (Ec. 3).

de entrada con las intermedias iguales a un único valor $w_i^{0,0}$. Por lo tanto tenemos que,

$$\sigma_i^0 = \Theta\left(w_i^{0,0}\sum_{j=1}^{N} S_j^0 - T_i\right) = \Theta\left(w_i^{0,0}H - T_i\right). \tag{2.4}$$

Seleccionamos entonces los valores de los parámetros $\{w_i^{0,0}\}$ y $\{T_i\}$ de la siguiente manera:

$$\{T_i\} = \left\{\pm 1, \pm 3, \pm 5, ..., \begin{array}{ll} \pm(N_0-2), N_0 & \text{si } N_0 \text{ es impar} \\ \pm(N_0-1) & \text{si } N_0 \text{ es par} \end{array}\right\} \tag{2.5}$$

y

$$w_i^{0,0} = \left\{\begin{array}{rl} 1 & \text{si } T_i > 0 \\ -1 & \text{si } T_i < 0 \end{array}\right. \tag{2.6}$$

Notemos que para valores pares de H, las neuronas intermedias cuyos umbrales tienen el mismo valor absoluto $|T_i|$ están una prendida y una apagada. En el caso en que H es impar esto también se cumple para todos los pares de neuronas con valores absolutos iguales excepto para aquellas para las cuales $|T_i| = H$, las que estarán ambas prendidas (recordemos que con la definición de $\Theta(x)$ utilizada $\Theta(0)=1$). En el caso en que N es impar la neurona con umbral $T_i = N$ está apagada para todos los valores pares de H. De esta forma, tenemos que

$$\sum_{i=0}^{N_0-1}\sigma_i^0 = \left\{\begin{array}{ll} > N_0/2 & \text{if } H \text{ es impar} \\ \leq N_0/2 & \text{si } H \text{ es par} \end{array}\right. \tag{2.7}$$

para todos los valores de N_0, tal cual era lo deseado (ver tabla 2.1).

Finalmente, podemos ver de la tabla 2.1 que ajustando los valores de las sinapsis del grupo intermedio ($k=0$) hacia la neurona de salida S_0^o todas iguales a uno, i.e.,

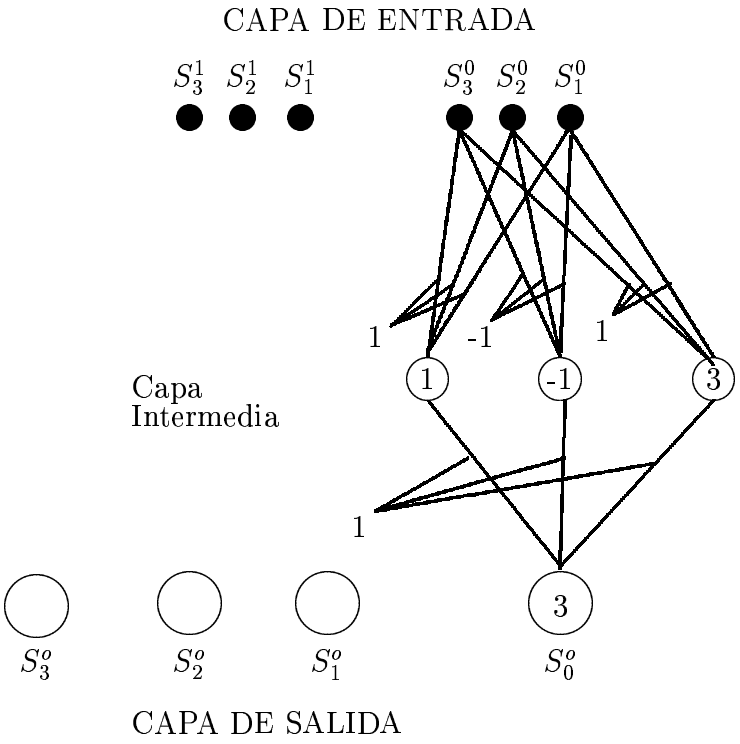


Figura 2.1: Estructura de una red para computar la suma del bit menos significativo, con valor 2^0, para el caso de tres números S_j $(j = 1, 2, 3)$. Los valores dentro de los círculos indican el valor del umbral de la respectiva neurona

$S_0^o = \Theta\left(\sum_{i=0}^{N_0-1} \sigma_i^0 - T_0\right)$, y tomando

$$N_0/2 < T_0 \leq (N_0 + 1)/2, \tag{2.8}$$

S_0^o computa el problema de paridad para todos los valores de N. En la figura 2.1 se muestra un ejemplo de la estructura para el bit de salida con menor valor relativo.

La solución encontrada es una solución particular y simple; una solución más general puede obtenerse a partir de la anterior permitiendo que los umbrales $\{T_i\}$ tomen un rango de valores en lugar de un valor definido. De esta forma los umbrales que en la solución simple tomaban los valores de m $(m = \pm 1, \pm 3, \ldots)$, ahora pueden tomar cualquier valor arbitrario en el intervalo $(m-1, m]$. Por ejemplo, el umbral que anteriormente tenía el valor $T_i = 1$ ahora puede tomar cualquier valor en el intervalo$(0, 1]$ y el anterior umbral $T_i = -1$ ahora puede tomar cualquier valor perteneciente al intervalo $(-2, -1]$. El resto de las sinapsis y umbrales mantienen sus antiguos valores, aunque también pueden

encontrarse rangos de valores para ellos, aún cuando esto involucra una relación un poco más complicada entre diferentes sinapsis y umbrales. De cualquier manera lo interesante de destacar es que los valores de umbrales y sinapsis tienen en general un rango de valores dentro del cual pueden computar la función deseada y por lo tanto no requieren una precisión extrema para su ajuste. Este tipo de consideraciones resultan de importancia a la hora de hallar estas soluciones mediante el algoritmo de aprendizaje. Soluciones al problema con valores puntuales de las intensidades sinápticas estarán representadas por un conjunto de medida nula en el espacio de parámetros.

Consideremos ahora el resto de los bits de salida. Para computar el segundo bit con menor valor relativo S_1^o, tenemos que computar la suma de N bits de entrada pertenecientes al grupo $k = 1$ (con valor relativo 2^1) más el valor del acarreo producto de la suma de los N bits de entrada del grupo $k = 0$ (valor relativo 2^0). Procederemos de la misma manera que en el caso anterior pero considerando que son necesarios dos bits con valor relativo 2^0 para formar un bit con valor relativo 2^1 y de esta forma estaremos contemplando el acarreo. Consecuentemente, los bits con valor relativo 2^0 estarán conectados a neuronas intermedias por sinapsis que tendrán un valor igual a la mitad del valor que tienen las sinapsis que conectan la capa intermedia con neuronas de valor relativo 2^1. Procedemos como si estuvieramos sumando $N_1 = N + int(N/2)$ bits, por lo que pondremos ese número de neuronas intermedias en el grupo $k = 1$, y repetimos el procedimiento usado en el primer caso (La operación *int* toma la parte entera del argumento).

Tendremos entonces, que

$$\sigma_i^1 = \Theta\left[w_i^{1,1}\sum_{j=1}^{N} S_j^1 + w_i^{1,0}\sum_{j=1}^{N} S_j^0 - T_i\right] \tag{2.9}$$

para $i = 0, \ldots, N_1 - 1$, con

$$\{T_i\} = \left\{\pm 1, \pm 3, \pm 5, ..., \begin{array}{ll} \pm(N_1-2), N_1 & \text{si } N_1 \text{ es impar} \\ \pm(N_1-1) & \text{si } N_1 \text{ es par} \end{array}\right\} \tag{2.10}$$

y

$$w_i^{1,j} = \left\{ \begin{array}{ll} \frac{1}{2^{1-j}} & \text{si } T_i > 0 \\ -\frac{1}{2^{1-j}} & \text{si } T_i < 0 \end{array} \right. \quad (j = 0, 1) \tag{2.11}$$

Este procedimiento puede ser generalizado para cualquier bit de salida. Cuando $k < p$, el bit de salida S_k^o proviene de la suma de N bits de entrada del grupo k, más el acarreo

de los $k-1$ grupos de entrada. Esta operación es equivalente a la adición de N_k números, con

$$\begin{aligned} N_k &= Int\left(N + N/2 + \cdots + N/2^k\right) = \\ &= 2N - Int\left[\tfrac{N}{2^k}\right] \end{aligned}$$

para $k < p$. Cuando $p \leq k \leq M-1$, el bit de salida S_k^o dependerá sólo del acarreo de los p grupos de entrada dado que no hay grupos de entrada con $k' \geq p$. De esta forma, el número de neuronas intermedias en el grupo k será

$$N_k = \begin{cases} 2N - Int\left[\frac{N}{2^k}\right] & \text{si } k < p \\ \frac{N_p}{2^{k-p+1}} = Int\left[\frac{N}{2^{k-p}}(1 - \frac{1}{2^{p+1}})\right] & \text{si } k \geq p \end{cases} \tag{2.12}$$

y

$$\sigma_i^k = \Theta\left[\sum_{l=0}^{k} w_i^{k,l} \sum_{j=1}^{N} S_j^l - T_i\right] \tag{2.13}$$

para $i = 0, \ldots, N_k - 1$, con

$$\{T_i\} = \left\{ \pm 1, \pm 3, \pm 5, ..., \begin{array}{ll} \pm(N_k - 2), N_k & \text{si } N_k \text{ es impar} \\ \pm(N_k - 1) & \text{si } N_k \text{ es par} \end{array} \right\} \tag{2.14}$$

y

$$w_i^{k,l} = \begin{cases} \frac{1}{2^{k-l}} & \text{si } T_i > 0 \\ -\frac{1}{2^{k-l}} & \text{si } T_i < 0 \end{cases} \quad (l = 0, \ldots, k) \tag{2.15}$$

El número total de neuronas en la capa intermedia será:

$$N_I = \sum_{k=0}^{p-1} \left\{ 2N - Int\left[\frac{N}{2^k}\right] \right\} + \sum_{k=p}^{M-1} Int\left[\frac{N}{2^{k-p}}(1 - \frac{1}{2^{p+1}})\right] \simeq$$

$$\simeq 2Np - 2 + Int\left[\frac{N}{2^p} - \frac{1}{2^p - 1}\right] \tag{2.16}$$

Finalmente en

$$S_k^o = \Theta(\sum_{i=0}^{N_k - 1} \sigma_i^k - T_0^k) \tag{2.17}$$

ponemos

$$N_k/2 < T_0^k \leq (N_k + 1)/2 \tag{2.18}$$

para $k = 0, \ldots, M-1$.

La estructura completa de una red de suma de N números de p bits cada uno tendrá $\mathcal{O}(3Np)$ neuronas y un número de sinapsis $\mathcal{O}(N^2p^2)$. Como se mostró, todo el procedimiento de suma puede ser computado simplemente conociendo la manera de computar

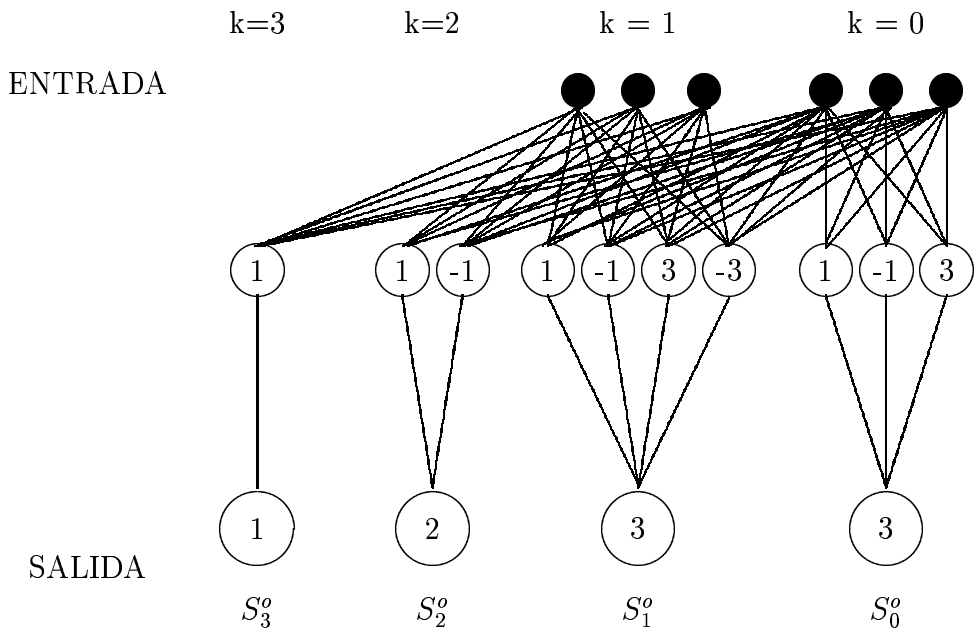


Figura 2.2: Estructura de una red para computar la suma de tres números de dos bits cada uno. Los valores dentro de los círculos indican los valores del umbral de la respectiva neurona (Ver el texto por detalles)

Tamaño (N, p)	Neuronas	Sinapsis	Profundidad	Fan-in Max
(4 , 4)	50	352	2	15
(8 , 8)	195	5016	2	63
(16 , 16)	772	73936	2	255
(32 , 32)	3077	1121664	2	1023
(N , p)	$\mathcal{O}(3Np)$	$\mathcal{O}(N^2p^2)$	2	$Np-1$

Tabla 2.2: Algunos parámetros de una red neuronal para computar la suma de N números.

el problema de **paridad** y ajustando las sinapsis según el valor relativo de los bits de entrada y poder así computar el acarreo.

La estructura completa de una red para el caso de $N = 3$ $p = 2$ se muestra en la figura 2.2 y algunos parámetros de las arquitectura obtenida para diferentes valores de N y p son presentados en la tabla 2.2.

2.3 Multiplicador Neuronal de profundidad 3

El multiplicador de dos números de n bits consiste en una red neuronal de 3 capas intermedias [Franco and Cannas, 1998]. La operación de multiplicación se realiza en dos etapas: la primera consiste en transformar el producto del multiplicador y multiplicando de la capa de entrada en una suma de n números de $2n-1$ bits; en la segunda etapa se sumarán los n números obtenidos. La primera etapa es simplemente una multiplicación bit a bit, de la misma forma que uno multiplica dos números a mano (en este caso números binarios); ver ejemplo en la figura 2.3. Para realizar esta operación no necesitamos capas intermedias dado que el producto de dos bits es equivalente a efectuar la operación booleana AND, la cual es linealmente separable [Hertz et al., 1991]. De esta forma, la primera etapa puede ser efectuada por un perceptron simple, obteniendo n números de $2n-1$ bits para sumar, que acomodaremos en la primer capa intermedia. Notemos que de los $2n-1$ bits de cada número solamente n de ellos son diferentes de cero, lo que permitirá reducir el número de neuronas en esta capa de $2(n^2-n)$ a n^2. Las neuronas de esta primera capa intermedia están conectadas por medio de dos sinapsis a la capa de entrada, una conectada a un bit del multiplicando y otra a un bit del multiplicador. Los valores de las sinapsis y umbrales pueden ser elegidos en una manera muy simple,

Tamaño	Neuronas	Sinapsis	Profundidad	Fan-in Max
4x4	60	341	3	16
8x8	216	4743	3	64
16x16	816	70539	3	256
32x32	3168	1087763	3	1024
nxn	$3(n^2+n)$	$\mathcal{O}(n^4)$	3	n^2

Tabla 2.3: Algunos parámetros de la red construída para multiplicar dos números para ciertos tamaños clásicos.

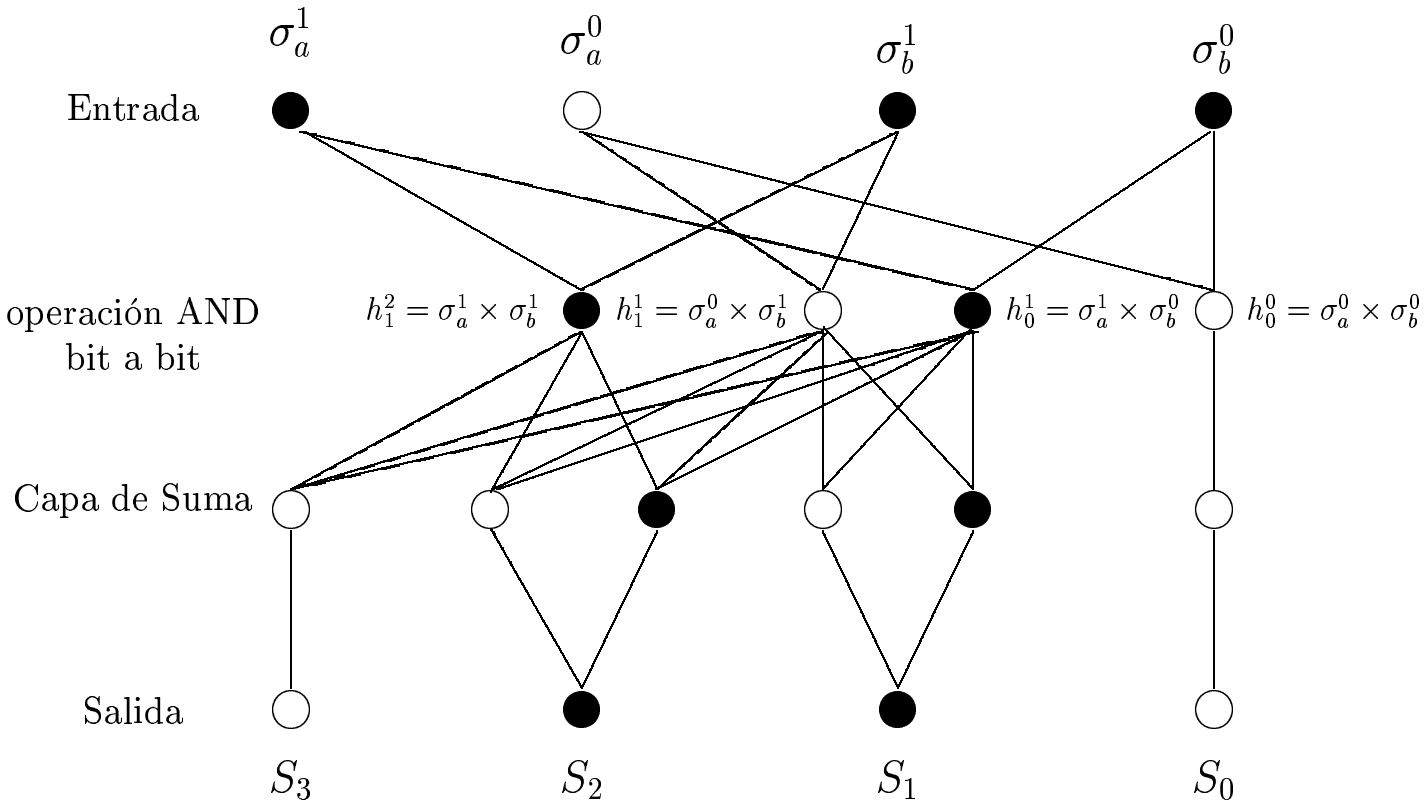
ponemos todas las sinapsis iguales a +1 y todos los umbrales iguales a +2. Esta solución simple puede extenderse, siguiendo la misma filosofía de la sección anterior, a una forma más general consistente en poner cada umbral(T) a un valor positivo y las dos sinapsis correspondientes (W_1, W_2) tales que:

$$W_1, W_2 < T \leq W_1 + W_2. \tag{2.19}$$

Entonces esta primera etapa involucra $2n$ neuronas en la capa de entrada, n^2 neuronas en la primera capa oculta y $2n^2$ sinapsis conectando ambas capas. La segunda y última operación consiste en sumar los n números de $2n-1$ bits obtenidos en la primera etapa. En la sección anterior resolvimos este problema en una forma más general en la cual construímos una red para sumar N números de p bits. En este caso, en el cual los números provienen de considerar un producto de dos números podemos realizar una reducción en el número de neuronas, debido a que $n-1$ bits de cada número son iguales a cero. De esta forma, la segunda etapa necesitará $2n^2-n$ neuronas en la segunda capa oculta que junto con las $2n$ neuronas de la capa de salida totalizan $3(n^2+n)$ neuronas y el número total de conexiones será n^4. La estructura de una red para multiplicación de dos números de 2 bits se muestra en la figura 2.3 y en la tabla 2.3 se muestran ciertos parámetros (número de neuronas, número de conexiones, etc.) para el caso de redes de tamaños clásicos, así también como para el caso general.

2.4 Red para corrimiento de bits ("bit-shifting")

Dado un número de entrada de M bits $S = S_{M-1}S_{M-2}....S_1S_0$, con $S_i = \{0,1\}$, un corrimiento de 1 bit a la izquierda consiste en obtener un nuevo número de M bits en el



$$
\begin{array}{cccccl}
 & & 1 & 0 & = & \sigma_a \\
 & \times & & & & \\
 & & 1 & 1 & = & \sigma_b \\
\hline
 & & 1 & 0 & = & \sigma_a \times \sigma_b^0 = h_0 \\
 & 1 & 0 & & = & \sigma_a \times \sigma_b^1 = h_1 \\
\hline
0 & 1 & 1 & 0 & = & S
\end{array}
$$

Figura 2.3: Estructura esquemática de una red para computar el producto de dos números de 2 bits: $\sigma_a = \sigma_a^1 \sigma_a^0$ y $\sigma_b = \sigma_b^1 \sigma_b^0$ ($\sigma_{a,b}^i = 0, 1$), donde $\sigma^0 = \sigma_a \times \sigma_b$ es la salida. Cada neurona en la primer capa oculta realiza el producto binario de un bit del multiplicando σ_a y un bit del multiplicador σ_b. Las capas restantes están diseñadas, como se describió en la sección 2 para computar la adición de los número resultantes. Círculos llenos y vacíos indican neuronas activas (ON) e inactivas (OFF) respectivamente. En este ejemplo particular los números de entrada $\sigma_a = 10$ y $\sigma_b = 11$ son transformados en los números $h_o = 10$ y $h_1 = 100$, los cuales son sumados para obtener la respuesta igual a $S^0 = 0110$.

cual todos los bits del número original han sido corridos un lugar a la izquierda y un cero ocupa el lugar dejado vacío, mientras que el bit más hacia la izquierda es simplemente descartado (Ver figura 2.4). La generalización de este procedimiento a un corrimiento de c bits es directa, corremos todos los bits de entrada c lugares a la izquierda y ponemos ceros en los c lugares vacíos a la derecha, mientras que los c bits $S_{M-1}, \ldots, S_{M-c}$ son descartados. Construímos una red de profundidad 2 que puede computar esta operación, la cual es óptima en el número de capas dado que el problema es linealmente inseparable. La estructura de la red es la siguiente: La capa de entrada contiene los M bits del número inicial S más $K = log_2(M+1)$ neuronas que indicarán el número c de lugares a correr $c \leq M$), *i.e.*, $c = S^c_{K-1} \ldots S^c_1 S^c_0$. La capa de salida contendrá M neuronas correspondientes al resultado $S^o = S^o_{M-1} \ldots S^o_1 S^o_0$. En nuestro modelo la capa intermedia actúa como un conjunto de compuertas que deja pasar el correspondiente valor de entrada, dependiendo del valor que posean los bits de corrimiento. Acomodaremos las neuronas de la capa intermedia en M grupos cada uno correspondiente a una neurona de salida . Esta red es fácilmente generalizable al caso de corrimiento de bits en ambas direcciones, permitiendo corrimientos a izquierda y derecha en una única y simple estructura. Para entender el funcionamiento de la capa intermedia comenzaremos con el caso más fácil del corrimiento de un único bit hacia la izquierda. Esta estructura luego la generalizaremos para el caso del corrimiento de c bits [Franco and Cannas, 1998].

2.4.1 Corrimiento de un bit

Para el caso del corrimiento de un único bit es suficiente con poner una sola neurona indicadora de corrimiento, $S^c = S^c_0$. Cuando $S^c_0 = 0$ la salida deberá ser igual a la entrada, i.e., $S^o_i = S_i \ \forall i$ (no hay corrimiento), mientras que para $S^c_0 = 1$ habrá que correr todos los bits de entrada un lugar, i.e., $S^o_i = S_{i-1}$ para $i = 1, \ldots, M-1$ y $S^o_0 = 0$. Vemos que el estado de la neurona de salida S^o_i depende solamente de los valores S_i, S_{i-1} y S^c_0; por lo tanto, ponemos solamente dos neuronas intermedias $\sigma_{i,0}$ y $\sigma_{i,1}$ en cada grupo i (ver figura 2.4) con $i = 1, \ldots, M-1$ (el caso $i = 0$ lo consideraremos luego). La neurona ubicada más hacia la izquierda, $\sigma_{i,0}$ transportará el valor de la neurona de entrada S_i hacia la neurona de salida S^o_i, dejando pasar su valor según el estado de la neurona indicadora S^c_0. De esta forma, la neurona $\sigma_{i,0}$ estará conectada solamente con S_i (a través de la sinapsis $u_{i,i}$) y con S^c_0 (a través de $v^0_{i,0}$). Por otro lado, la neurona $\sigma_{i,1}$ llevará información

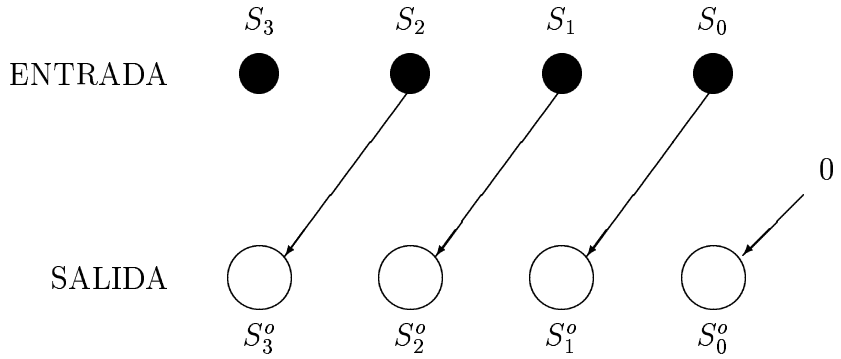


Figura 2.4: Representación esquemática de una transformación para el corrimiento de 1 bit. En el ejemplo el número de entrada $S = S_3S_2S_1S_0$ ($S_i = 0, 1$) es transformado en el número
$S^o = S_3^oS_2^oS_1^oS_0^o$ con $S_i^o = S_{i-1}$ para $i = 1, 2, 3$ y $S_0^o = 0$.

acerca del estado de la neurona S_{i-1} hacia S_i^o , y entonces estará conectada solamente con S_{i-1} (a través de $u_{i,i-1}$) y con S_0^c (a través de la sinapsis $v_{i,1}^0$). Denotando por $T_{i,j}$ los umbrales de las neuronas $\sigma_{i,j}$ ($j = 0, 1$) tenemos que:

$$\sigma_{i,j} = \Theta\left[u_{i,i-j}S_{i-j} + v_{i,j}^0 S_0^c - T_{i,j}\right] \quad (i = 1, \ldots, M-1\,;\; j = 0, 1) \tag{2.20}$$

El grupo intermedio $i = 0$ debe ser considerado por separado. Dado que hay un solo bit a considerar necesitamos una única neurona intermedia $\sigma_{0,0} = \Theta\left[u_{0,0}S_0 + v_{0,0}^0 S_0^c - T_{0,0}\right]$. Finalmente, ambas neuronas $\sigma_{i,0}$ y $\sigma_{i,1}$ estarán conectadas con S_i^o a través de sinapsis $w_{i,0}$ y $w_{i,1}$:

$$S_i^o = \Theta\left[w_{i,0}\sigma_{i,0} + w_{i,1}\sigma_{i,1} - T_i\right] \tag{2.21}$$

En la figura 2.5 se muestra una figura esquemática de un grupo i. La estructura funciona de la siguiente manera: cuando $S_0^c = 0$ tenemos que $\sigma_{i,0} = S_i$ y $\sigma_{i,1} = 0\ \forall i$. Cuando $S_0^c = 1$ tenemos que $\sigma_{i,0} = 0$ y $\sigma_{i,1} = S_{i-1}$ para $i = 1, \ldots, M-1$, es decir, las dos neuronas intermedias nunca están activas al mismo tiempo, en cualquier caso al menos una de ellas es cero y la otra se encarga de transmitir el valor correcto desde la entrada hacia la salida. Estos resultados se muestran en la tabla 2.4 y si los reemplazamos en la Eq.(2.20) encontramos que

$$u_{i,i} \;\geq\; T_{i,0} > 0 \tag{2.22}$$

S_0^c	$\sigma_{i,0}$	$\sigma_{i,1}$	S_i^0
0	S_i	0	S_i
1	0	S_{i-1}	S_{i-1}

Tabla 2.4: Estados de activación de las neuronas intermedias asociadas con el bit de salida S_i^0 para un corrimiento de 1 bit.

$$v_{i,0}^0 < T_{i,0} - u_{i,i} \tag{2.23}$$

para $i = 0, \ldots, M-1$ y

$$0 < u_{i,i-1} < T_{i,1} \tag{2.24}$$

$$0 < v_{i,1}^0 < T_{i,1} \tag{2.25}$$

$$u_{i,i-1} + v_{i,1}^0 \geq T_{i,1} \tag{2.26}$$

para $i = 1, \ldots, M-1$. De la tabla 2.4 vemos que S_i^o debe satisfacer $S_i^o = \sigma_{i,0}$ si $S_0^c = 0$ y $S_0^c = \sigma_{i,1}$ si $S_0^c = 1$. Dado que $\sigma_{i,0}$ y $\sigma_{i,1}$ no pueden ser nunca ambas iguales a uno (al mismo tiempo), la condición anterior es satisfecha si $S_i^o = (\sigma_{i,0} \,\mathrm{OR}\, \sigma_{i,1})$ (operación lógica OR). Esta última operación puede ser realizada tomando $T_i \geq 0$ y

$$w_{i,j} \geq T_i \qquad (j = 0, 1) \tag{2.27}$$

para todo i.

De esta manera completamos la construcción de la red para el caso del corrimiento de un bit, obteniendo para el caso de M bits de entrada una estructura compuesta por $4M-1$ neuronas, $6M-3$ pesos sinápticos y un número máximo de dos conexiones por neurona.

2.4.2 Corrimiento de c bits

Consideremos ahora el caso más general de corrimiento de c bits, que permite correr entre 0 y M bits, siendo M el número total de bits del número de entrada. Será necesario poner $K = log_2(M+1)$ bits indicadores de corrimiento lo que totalizará $N = M + K$ neuronas en la capa de entrada. En este caso el grupo intermedio i tiene que poder llevar información de cualquiera de las $i+1$ neuronas de entrada $\{S_i, S_{i-1}, \ldots, S_1, S_0\}$ hacia

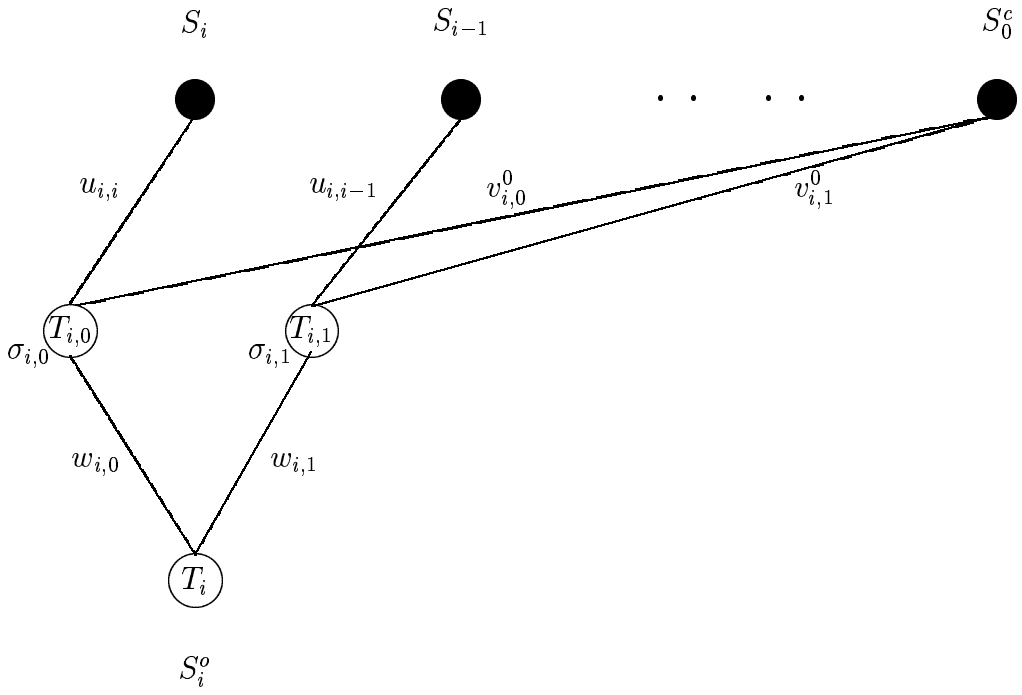


Figura 2.5: Esquema de conexiones asociadas con la i-ésima neurona de salida S_i^o para la operación de corrimiento a izquierda de 1 bit.

la neurona de salida S_i^o, dejando pasar solamente el valor S_{i-c}. Ponemos, entonces en este grupo $i+1$ neuronas $\sigma_{i,j}$ con $j = 0, \ldots, i$, lo que da un número total de neuronas intermedias $M(M+1)/2$; $\sigma_{i,j}$ estará conectada con S_{i-j} a través de la sinapsis $u_{i,i-j}$ y con las K neuronas indicadoras S_l^c a través de $v_{i,j}^l$ $(l = 0, \ldots, K-1)$:

$$\sigma_{i,j} = \Theta\left[u_{i,i-j}S_{i-j} + \sum_{l=0}^{K-1} v_{i,j}^l S_l^c - T_{i,j}\right] \quad (i = 1, \ldots, M-1\,;\, j = 0, \ldots, i). \tag{2.28}$$

Las sinapsis $\{u_{i,i-j}, v_{i,j}^l\}$ tendrán valores tales que, en un corrimiento de c bits, en cada grupo i, $\sigma_{i,j} = 0$ para $j \neq c$ y $\sigma_{i,c} = S_{i-c}$.

Comencemos con el análisis de la primer neurona intermedia de cada grupo $\sigma_{i,0}$. Tenemos que

$$\sigma_{i,0} = \begin{cases} S_i & \text{si } S_l^c = 0\ \forall l \\ 0 & \text{en otro caso.} \end{cases} \tag{2.29}$$

Reemplazando en Eq.(2.28) encontramos que

$$u_{i,i} \geq T_{i,0} > 0 \tag{2.30}$$

$$v_{i,0}^l < T_{i,0} - u_{i,i} \quad \forall l \tag{2.31}$$

para $i = 0, \ldots, M-1$. Notemos que todas las sinapsis son inhibitorias excepto $u_{i,i}$.

Consideremos ahora el resto de las neuronas intermedias. Dado que $\sigma_{i,j} = 0\ \forall j > 0$ si $S_l^c = 0\ \forall l$, encontramos usando Eq.(2.28) que $T_{i,j} > 0$ para todos los valores de i y j, y que:

$$u_{i,i-j} < T_{i,j} \qquad \forall j > 0 \tag{2.32}$$

Para la segunda neurona de cada grupo $\sigma_{i,1}$ tenemos que

$$\sigma_{i,1} = \begin{cases} S_{i-1} & \text{si } S_0^c = 1\ y\ S_l^c = 0\ \forall l \neq 0 \\ 0 & \text{en otro caso} \end{cases} \tag{2.33}$$

para $i = 1, \ldots, M-1$. Reemplazando en la Ec.(2.28) encontramos que

$$v_{i,1}^0 < T_{i,1} \tag{2.34}$$

$$u_{i,i-1} + v^0_{i,1} \geq T_{i,1} \tag{2.35}$$

$$T_{i,1} - (u_{i,i-1} + v^0_{i,1}) > v^l_{i,1} \quad \forall l \neq 0. \tag{2.36}$$

De las Ecs.(2.32), (2.34) y (2.35) vemos que $u_{i,i-1} > 0$ y $v^0_{i,1} > 0$, *i.e.*, ambas sinapsis son excitatorias mientras el resto son inhibitorias. Notemos que obtenemos un conjunto similar de desigualdades para las sinapsis asociadas con cualquier neurona intermedia que pueda estar prendida solamente cuando el bit indicador S^c_l esté encendido, *i.e.*, cuando $c = 2^m$ con $m = 0, 1, \ldots, K-1$. En otras palabras, para cualquier grupo i con $i \geq 2^m$ tenemos que:

$$\sigma_{i,2^m} = \begin{cases} S_{i-2^m} & \text{si } S^c_m = 1 \ y \ S^c_l = 0 \ \forall l \neq m \\ 0 & \text{en otro caso} \end{cases} \tag{2.37}$$

y por lo tanto

$$0 < v^m_{i,2^m} < T_{i,2^m} \tag{2.38}$$

$$u_{i,i-2^m} + v^m_{i,2^m} \geq T_{i,2^m} \tag{2.39}$$

$$T_{i,2^m} - (u_{i,i-2^m} + v^m_{i,2^m}) > v^l_{i,m} \quad \forall l \neq m. \tag{2.40}$$

En todos los casos solamente las sinapsis $u_{i,i-2^m}$ y $v^m_{i,2^m}$ serán excitatorias y menores que sus respectivos umbrales, mientras que el resto de las sinapsis serán inhibitorias de acuerdo con (2.39) y (2.40).

El procedimiento anterior puede ser aplicado a cualquiera de las neuronas intermedias. Entonces, dado $\sigma_{i,j}$, suponemos que hay p bits $S^c_l = 1$ cuando $c = j$. Sea $\{l_1, \ldots, l_p\}$ el conjunto de índices para los cuales $S^c_l = 1$. Luego

$$0 < v^l_{i,j} < T_{i,j} \quad \forall l \in \{l_1, \ldots, l_p\} \tag{2.41}$$

$$u_{i,j} + v^l_{i,j} \geq T_{i,j} \quad \forall l \in \{l_1, \ldots, l_p\} \tag{2.42}$$

Tamaño	Neuronas	Sinapsis	Fan-in Max
4	21	40	4
8	56	144	8
16	177	544	16
32	597	2112	32
$M \gg 1$	$\mathcal{O}(M^2)$	$2(M^2+M)$	M

Tabla 2.5: Algunos parámetros para una red de corrimiento de bits de profundidad 2 para tamaños clásicos.

$$T_{i,j} - (u_{i,j} + \sum_{\forall l \in \{l_1,\ldots,l_p\}} v_{i,j}^{l}) > v_{i,m}^{l'} \quad \forall l' \notin \{l_1,\ldots,l_p\}. \tag{2.43}$$

Finalmente, vemos que para cualquier valor de c sólo la neurona $\sigma_{i,c} = S_{i-c}$ puede estar activa (si $S_{i-c} = 1$) para cada grupo $i \geq c$, mientras que el resto de las neuronas intermedias $\sigma_{i,j} = 0 \; \forall j \neq c$. Podemos entonces calcular el estado de la neurona de salida S_i^o como $S_i^o = (\sigma_{i,0}$ OR $\sigma_{i,1}$ OR$\ldots$ $\sigma_{i,i-1}$ OR $\sigma_{i,i})$. Esta operación puede ser computada si ponemos $T_i > 0 \; \forall i$ y $w_{i,j} \geq T_i$ para $j = 0,\ldots,i$.

De esta forma el procedimiento conduce para el caso de una red que computa el corrimiento de M bits a una estructura con 2 capas ocultas y un total de neuronas del orden $\mathcal{O}(M^2)$, $2(M^2+M)$ sinapsis y un número máximo de conexiones por neurona (fan-in max) igual a M.

Una red para el caso de M=3 se muestra en la figura 2.6, y algunos parámetros para distintos tamaños son mostrados en la tabla 2.5.

2.4.3 Corrimiento de bits en ambas direcciones

Las redes previamente diseñadas pueden ser generalizadas para el caso de un corrimiento bidireccional. El funcionamiento será esencialmente el mismo pero habrá que adicionar neuronas en la capa intermedia para posibilitar el nuevo corrimiento hacia la derecha. Una red diseñada exclusivamente para correr bits hacia la **derecha** tendrá la misma estructura descripta en la subsección anterior, con la diferencia que las neuronas intermedias estarán ordenadas al revés: las neuronas intermedias correspondientes al primer bit tendrán que ser intercambiadas con las correspondientes al último bit y así sucesivamente. Finalmente,

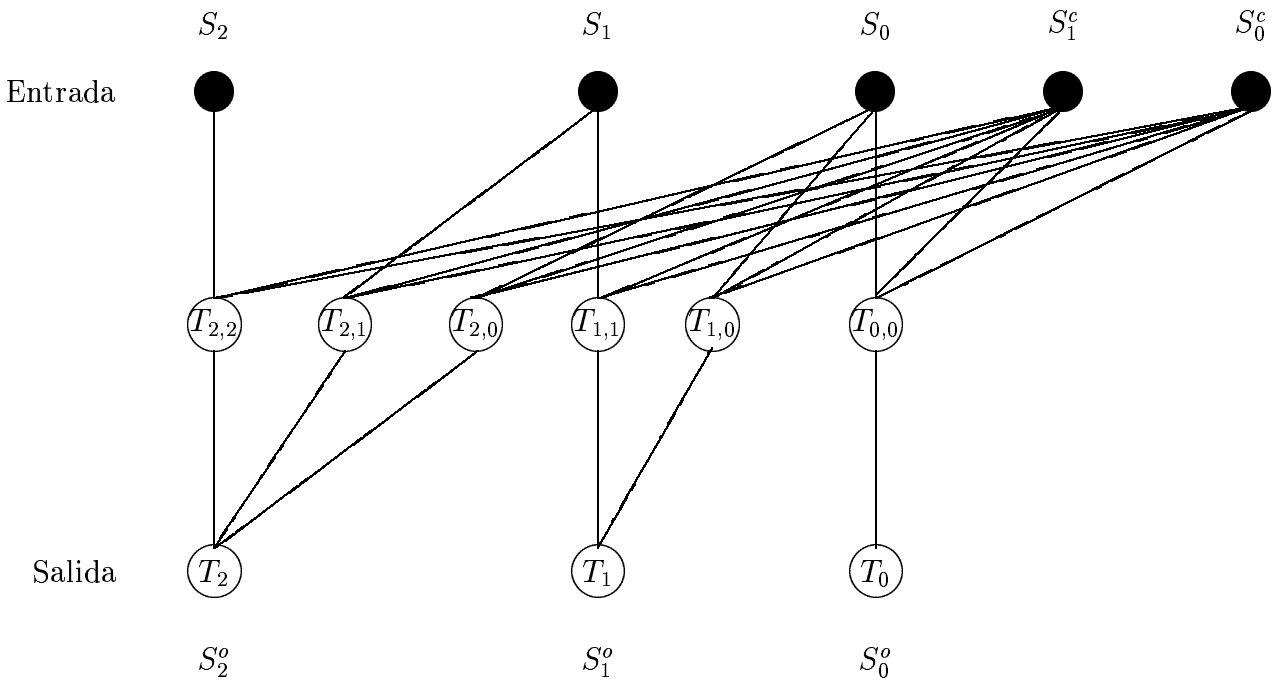


Figura 2.6: Arquitectura de una red neuronal para computar el corrimiento de bits para un número de entrada de 3 bits $S = S_2S_1S_0$. El número de bits a correr es indicado por las neuronas $S^c = S_1^cS_0^c$.(ver el texto por más detalles)

ambas estructuras pueden ser combinadas para obtener un corrimiento bidireccional, con el agregado de una nueva neurona indicadora S_{sh} la cual indicará si el corrimiento es hacia la derecha $S_{sh} = 0$ o a la izquierda $S_{sh} = 1$. De esta forma, cada grupo de la capa intermedia contendrá M neuronas que estarán conectadas a una neurona de entrada y a todas las neuronas indicadoras de corrimiento S_l^c. La neurona indicadora de la dirección del corrimiento S_{sh} tendrá conexiones inhibitorias con todas las neuronas intermedias encargadas de dejar pasar valores desde su izquierda, ya que que cuando S_{sh} es 1 los valores que pasarán provendrán de la derecha. Las neuronas encargadas de transmitir valores de neuronas de entrada hacia su derecha tendrán conexiones excitatorias provenientes de la neurona S_{sh} y también su umbral será incrementado respecto del valor calculado en la subsección anterior de manera de evitar su funcionamiento cuando $S_{sh} = 0$.

Las neuronas intermedias $\sigma_{i,0}$ serán las únicas no conectadas con S_{sh} dado que ellas solamente llevan información hacia la neurona de salida S_i^o cuando ningún corrimiento es realizado. De esta forma la red poseerá M^2 neuronas intermedias y un total de $M^3 + M^2(\log_2(K) + 2)$ sinapsis. En la figura 2.7 se muestra un ejemplo de una red para el corrimiento bidireccional de bits para el caso de $M = 5$ neuronas de entrada (las conexiones entre la entrada y la capa intermedia no están representadas por una cuestión de claridad). En el ejemplo de la figura 2.7, el número en la entrada es 01010 y como las neuronas indicadoras marcan un corrimiento de un bit a derecha la salida correspondiente es 00101.

2.5 Una familia de arquitecturas para el problema de paridad

Distintas medidas de complejidad han sido propuestas para clasificar las funciones booleanas, tales como orden de predicado, criterio de entropía, tamaño de la red, etc.,[ver Denker et al., 1988, Carnevali y Patarnello, 1987, Parberry, 1994). La función de paridad es clasificada como una función difícil usando cualquiera de aquellas medidas, siendo la clave de esta dificultad el hecho de que el cambio de cualquier bit de la entrada produce un cambio en la salida (al cambiar la paridad del número de neuronas prendidas). Debido a este hecho, a su facilidad de definición y a algunas otras razones históricas la función de paridad es una de las funciones más estudiadas y más usadas para testear algoritmos

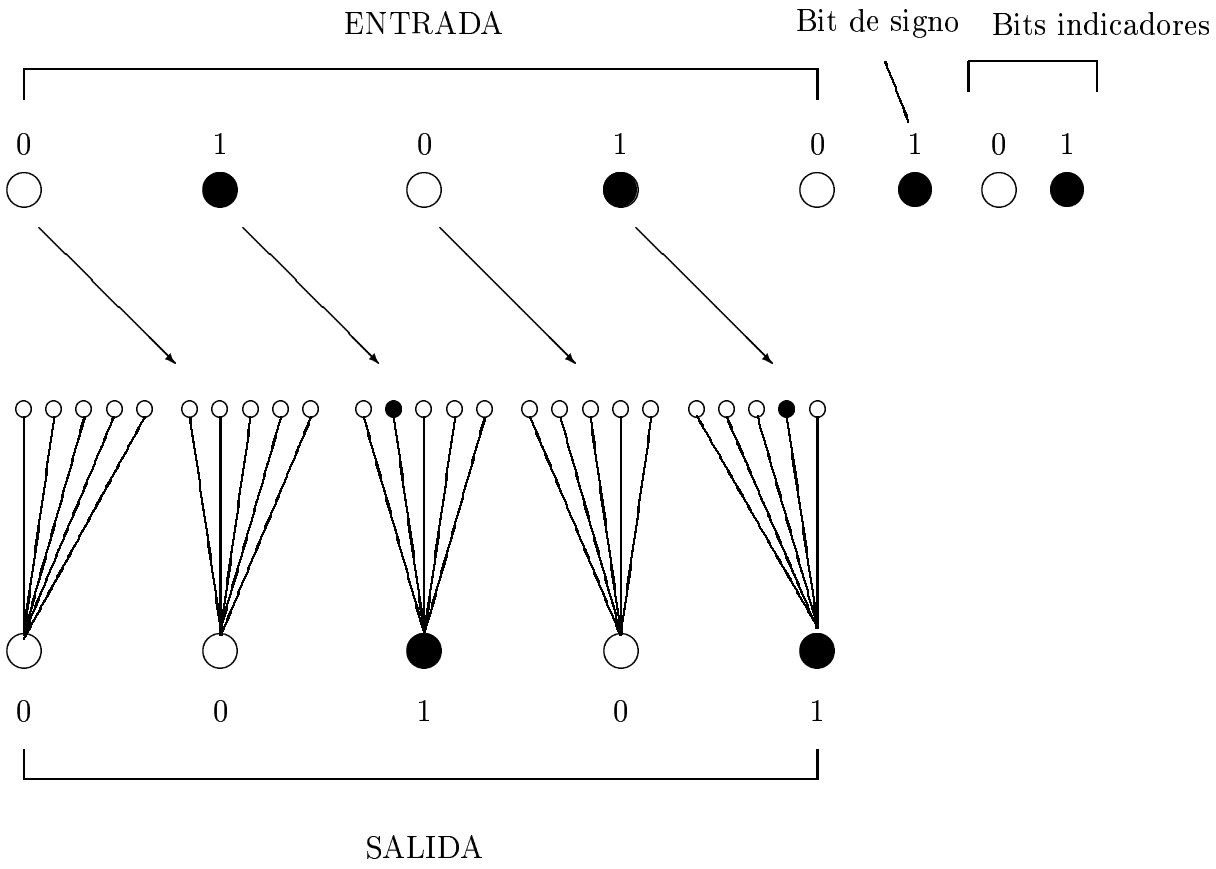


Figura 2.7: Ejemplo del corrimiento de 1 bit a derecha con una red para corrimiento bidireccional. Círculos llenos y vacíos indican neuronas activas (ON) e inactivas (OFF) respectivamente. En el ejemplo el número de entrada es 01010, $S_{sh} = 1$, y $S^c = 01$, indicando el corrimiento de 1 bit a derecha.

de aprendizaje. Dentro de los trabajos donde la función de paridad es estudiada y analizada podemos citar [Minsky y Papert, 1969, Rumelhart y McClelland, 1990, Hertz et al., 1992, Haykin, 1994, Denker et al., 1994, Van de Broeck y Kawai, 1990]. La cuestión acerca de cual es el tamaño mínimo necesario para computarla usando neuronas lineales con umbral, ha sido analizada dentro de lo que llamaríamos el área de "Complejidad de Circuitos" [Parberry, 1994]. Dentro de esta línea [Impagliazzo et al., 1990] encontraron que la función de paridad con N bits de entrada necesita una capa oculta con al menos $N^{\frac{1}{2}}$ neuronas, mientras que hasta el momento la red de menor tamaño construída tiene del orden $\mathcal{O}(N)$ neuronas.

La solución conocida más simple para la función de paridad de N bits (ver [Rumelhart y McClelland, 1986]) usando neuronas lineales con umbral consiste en una red de tres capas, esto es una capa de entrada con N neuronas, una capa oculta conteniendo N neuronas, las cuales están totalmente conectadas tanto con la entrada como con la única neurona de salida. Esta arquitectura funciona eligiendo los valores de las sinapsis y umbrales en una forma muy simple: todos los pesos conectando la entrada con las neuronas de la capa oculta valen 1, los umbrales de las neuronas intermedias son los números semi-enteros comprendidos en el rango $\frac{1}{2}$ y $N - \frac{1}{2}$; las conexiones desde las neuronas intermedias hacia la única neurona de salida son valuadas alternadamente a 1 y -1 y el valor del umbral de la neurona de salida es $\frac{1}{2}$. De aquí en más nos referiremos a esta arquitectura como la "arquitectura básica o standard". En la figura 2.8 se muestra un ejemplo de esta arquitectura para el caso de 4 neuronas de entrada, $N = 4$.

Veamos como funciona la red con esta elección de pesos y umbrales: cuando i de las N neuronas de entrada están prendidas, i de las neuronas intermedias (las que tienen umbrales menores que i) estarán prendidas. Como los pesos sinápticos conectando neuronas intermedias con la neurona de salida tienen valores alternados 1 y -1, sólo cuando un número impar de neuronas intermedias estén prendidas la suma de sus respectivos pesos que las conectan a la neurona de salida, excederá el valor $\frac{1}{2}$ del umbral de esta neurona de salida y consecuentemente estará activá. Un número par de neuronas de entrada prendidas resultará en un número par de neuronas intermedias activadas y por lo tanto la suma de sus pesos se cancelará resultando igual a cero y la neurona de salida quedará apagada.

Usando como base la solución anterior introduciremos una familia de arquitecturas, que resolverán la función de paridad [Franco and Cannas, 2000b]. Las arquitecturas están

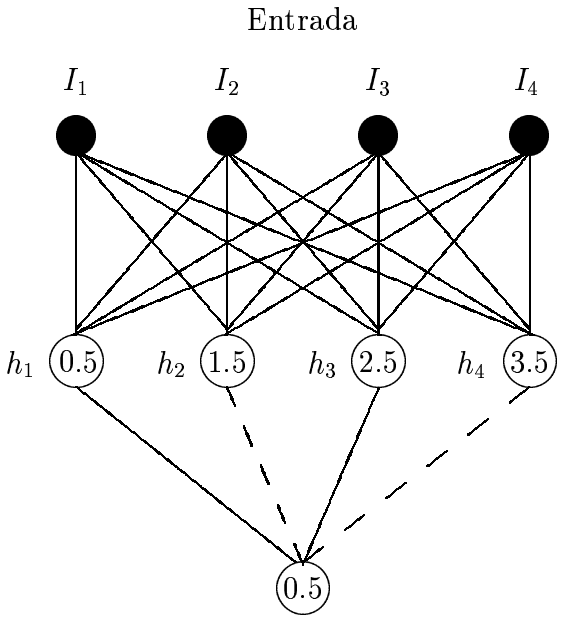


Figura 2.8: Estructura de una red neuronal para computar la función de paridad de 4 bits de entrada usando la arquitectura básica con una única capa oculta. Los valores de los umbrales están indicados dentro de las neuronas mientras que los valores de las sinapsis están indicados con líneas sólidas para sinapsis con valor 1 y con líneas cortadas para los valores -1.

construídas teniendo en cuenta la propiedad de que la suma de dos números pares o dos impares da como resultado un número par, mientras que la suma de un número par con uno impar es impar. Entonces, la paridad de un número arbitrario de bits puede ser computada dividiendo las neuronas en grupos y computando la paridad de estos grupos en forma independiente y repitiendo este mismo procedimiento con los resultados obtenidos hasta obtener un único bit de salida. La función de paridad puede ser computada en cada etapa con la arquitectura básica presentada anteriormente existiendo diversas maneras de agrupar los bits en cada etapa, cada una asociada con una arquitectura particular. Una de estas maneras es construir los grupos, mientras sea posible, con un mismo número de neuronas m. Esta elección genera una estructura modular donde la arquitectura básica con m bits es repetida recursivamente. Ilustremos la idea con un ejemplo simple para el caso $m = 2$. Considereremos por simplicidad que N, el número de neuronas de entrada, es del tipo 2^k ($k > 0$) y dividamos las neuronas en $\frac{N}{m} = \frac{N}{2}$ grupos de $m = 2$ neuronas. Computando la paridad de estos pares obtenemos $\frac{N}{2}$ salidas binarias, que volvemos a agrupar en $\frac{N}{m^2} = \frac{N}{4}$ pares y así hasta obtener una única salida. Esta arquitectura computará la paridad de N-bits en $k = \log_2 N$ etapas involucrando un número total de $2k - 1$ capas ocultas. Un ejemplo para el caso $N = 4$ se muestra en la figura 2.9. La generalización al caso de tener $N \neq 2^k$ es directo: cada vez que necesitemos computar la paridad de un número impar de neuronas, dejamos una neurona suelta, computamos la paridad de las restantes de la manera descripta y en la etapa siguiente incluímos la neurona sin computar. Escribiendo $N = 2^k - l$ (con $l < 2^{k-1}$), el problema de paridad correspondiente puede ser resuelto por una arquitectura con $2k-1$ capas ocultas. La estructura contendrá neuronas conectadas a través de sinapsis que saltean capas, las cuales pueden ser eliminadas agregando neuronas en la capa salteada cuya única función será la de propagar el valor de la neurona que no intervino en el cómputo y las cuales no tendrán sinapsis salteando capas. Esta estructura tendrá un número máximo de sinapsis que entran en una neurona (fan-in max) de $m = 2$.

Podemos generalizar fácilmente este procedimiento al caso $m > 2$ usando la estructura básica con m bits. Analizaremos los casos donde $N = m^k$, ya que cuando $N \neq m^k$ los podemos tratar con el mismo análisis anteriormente mencionado. En el caso $N = m^k$ obtenemos una estructura modular que resulve la función de paridad en $k = \log_m N$

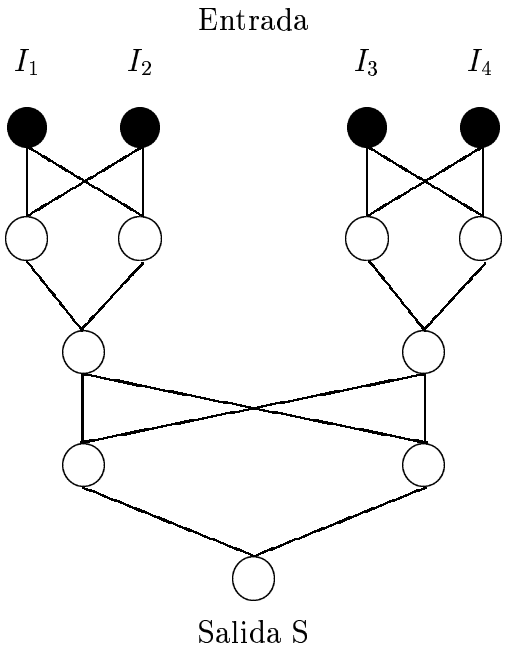


Figura 2.9: Estructura de una red neuronal para computar la función de paridad de 4 bits de entrada con $f_{max} = 2$. Primero se computa la paridad de pares de bits de entrada y luego la paridad de los resultados usando el mismo procedimiento.

etapas con $2k-1$ capas ocultas, un número total de neuronas igual a

$$N_n = 1 + 2m\frac{N-1}{m-1}, \tag{2.44}$$

y un número total de pesos sinápticos

$$N_s = \frac{m(m+1)(N-1)}{m-1}. \tag{2.45}$$

El número máximo de conexiones entrantes, f_{max}, será igual a m, sirviendo este parámetro para medir o ajustar el grado de modularidad de las redes: el caso $m = N$ corresponderá la arquitectura básica de tipo monolítica, mientras que para $m = 2$ obtendremos la arquitectura con la mayor modularidad posible. Notemos que tanto N_n y N_s crecen linealmente con el número de entradas N. Para valores grandes de m, N_n se torna casi independiente de m mientras que N_s se incrementa linealmente con m: en redes con un gran número de entradas el número de sinapsis puede ser reducido usando pequeños m que aseguran solamente un pequeño incremento en el número total de neuronas, siendo estas consideraciones importantes a los efectos de su posible implementación en hardware. En la figura 2.10 se muestra una estructura de red que permite computar la paridad de $N = 9$ entradas con $m = 3$.

2.6 Conclusiones del capítulo

Presentamos en este capítulo la construcción de 4 arquitecturas neuronales que permiten implementar los problemas de suma de N números, multiplicación de dos números, corrimiento de bits y problema de paridad en forma modular. El diseño de las tres primeras arquitecturas puede compararse favorablemente, en relación al número de capas y número de neuronas, con los diseños previos existentes para estas funciones [Lauwereins y Bruck, 1991], [Hofmeister et al., 1991], [Anderson y Van Hessen, 1987], con la destacable ventaja de reducir el número de capas a la profundidad **óptima** de 3,4 y 3 capas respectivamente.

La red de suma presenta conexiones hacia atrás, i.e., neuronas de un grupo i están conectadas con neuronas intermedias de grupos j con $j \leq i$. Los correspondiente pesos sinápticos tienen valores que decaen exponencialmente con el valor de $i-j$ (ver ec. 2.15). Esta estructura proviene del efecto de acarreo de la suma de bits y ha sido encontrada anteriormente en un problema relacionado [Cannas, 1995].

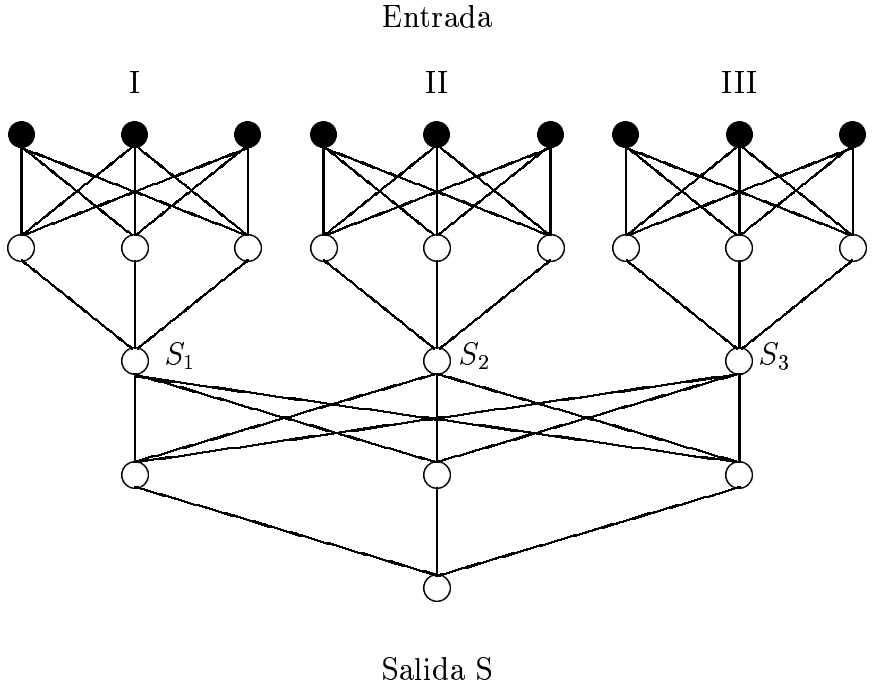


Figura 2.10: Estructura de una red neuronal para computar la función de paridad de 9 bits de entrada con un $f_{max} = 3$. Ver texto por detalles.

Método	Neuronas	Sinapsis	Fan-in Max	Profundidad
Block Save Addition (BSA)	$\mathcal{O}(N^3)$	$\mathcal{O}(N^4 logN)$	$\mathcal{O}(N^2)$	3
BSA optimizado y Dortmunder	$\mathcal{O}(N^2)$	$\mathcal{O}(N^3 logN)$	$\mathcal{O}(N logN)$	3
Nuestro	$\mathcal{O}(N^2)$	$\mathcal{O}(N^4)$	$\mathcal{O}(N^2)$	2

Tabla 2.6: Comparación entre las principales características correspondientes a los diferentes métodos utilizados para construir una red neuronal que computa la suma de N números de N bits.

Diferentes características de estas redes para sumar N números de p bits por distintos métodos se muestran en la tabla 2.6.

Se mostró también la construcción de una estructura para permitir el corrimiento bidireccional de bits con profundidad 2 independientemente del valor del corrimiento, siendo esta arquitectura, dentro de nuestro conocimiento, la primer solución a este problema con esta característica, ya que la red propuesta por [Anderson y Van Hessen, 1987] tiene una profundidad dependiente del número de neuronas en la capa de entrada.

Las misma ideas expuestas en la construcción de la arquitectura para el corrimiento de bits pueden extenderse para el caso de dos dimensiones, lo cual resulta de interés tanto para el procesamiento artificial de imágenes como para la comprensión de mecanismos de corrección en circuitos visuales. Vale la pena notar que siendo el problema de corrimiento de bits linealmente no separable, la construcción de una arquitectura con una sola capa oculta asegura que estamos en el óptimo en cuanto a la profundidad.

Las tres funciones anteriormente citadas, son operaciones usuales en microprocesadores y tienen entonces un interés adicional desde el punto de vista de la construción de procesadores neuronales o paralelos.

Respecto de la familia de arquitecturas modulares para computar la función de paridad, estas serán usadas para estudiar las propiedades de generalización y selección de ejemplos en redes modulares que trataremos en el capítulo siguiente. Estas arquitecturas están basadas en la arquitectura básica para computar la función de paridad, pero nos permitirán analizar los efectos de la modularidad en el aprendizaje. La modularidad aparece como un principio muy importante en la arquitectura de los sistemas nerviosos de vertebrados, mientras que redes totalmente conectadas son rara vez encontradas en la naturaleza [Haykin, 1994].

Las arquitecturas diseñadas sirven como plataforma de entrenamiento para estudiar propiedades de aprendizaje y generalización de redes neuronales, las cuales serán analizadas en el capítulo siguiente y también son útiles para comparar la eficiencia de distintos algoritmos de aprendizaje.

Capítulo 3

Generalización y Selección de Ejemplos

3.1 Introducción

En este capítulo estudiaremos como la selección de ejemplos influye en el proceso de aprendizaje en una red neuronal booleana (compuesta por neuronas binarias) analizando su relación con la complejidad de la función estudiada y su arquitectura.

Estudiaremos la capacidad de generalización en el aprendizaje de diferentes funciones a través del cálculo analítico del mínimo número de ejemplos necesarios para obtener *generalización total*,(i.e., error de generalización cero), en diferentes arquitecturas. El análisis de los conjuntos de entrenamiento asociados con ese parámetro nos permitirá proponer un criterio general para seleccionar ejemplos independiente de la arquitectura. Este criterio es chequeado por medio de simulaciones numéricas en distintas funciones con arquitecturas particulares diseñadas al efecto (ver capítulo anterior), así como también con funciones aleatorias en perceptrones con campo receptivo sin solapamiento (NORFP). En todos los casos los resultados son satisfactorios, obteniéndose una mejora en la capacidad de generalización de la red comparada con la obtenida usando ejemplos seleccionados al azar.

También mostraremos que para el caso de la función de paridad (implementada en la arquitectura básica), uno de los problemas más usados para testar algoritmos de aprendizaje, es necesario el uso de la *totalidad de los ejemplos* en el conjunto de aprendizaje para asegurar generalización total. Veremos que esta dificultad puede ser superada usando

una arquitectura modular para computarla.

Hoy en día no existe todavía una teoría general que explique el proceso completo de aprendizaje y generalización en redes neuronales, existiendo pocos resultados generales especialmente en el tema de la generalización, siendo la mayoría de lo resultados existentes para problemas y arquitecturas particulares. Un problema interesante es el de encontrar o seleccionar un conjunto limitado de ejemplos que pueda mejorarnos la habilidad de generalización; cuando seleccionamos u obtenemos ejemplos al azar, la mayoría de las veces la información es redundante; es entonces importante tener algún criterio con el cual poder seleccionar ejemplos. Desde un punto de vista más amplio el problema es también de interés para las ciencias cognitivas, ver por ejemplo [Anderson, 1998], donde entre otras muchas cuestiones es interesante conocer que clase de arquitecturas pueden utilizarse para aprender las funciones aritméticas y también cuales ejemplos son más útiles para el aprendizaje.

La cuestión de selección de ejemplos ha sido previamente estudiada dentro del contexto de las redes neuronales donde es conocida bajo el nombre de aprendizaje activo ("active learning"), o aprendizaje basado en preguntas ("query-based learning"). En un sentido general estas definiciones se refieren a cualquier forma de aprendizaje en las cuales los algoritmos de aprendizaje tienen algún control sobre los ejemplos usados para el entrenamiento.

Diferentes enfoques han sido desarrolladas para abordar la cuestión de obtener criterios para seleccionar ejemplos en distintos tipos de redes, desde perceptrones simples [Kinzel and Ruján, 1990] y clasificadores lineales [Jung and Opper, 1996] hasta redes multicapas con salidas continuas. Entre otros, podemos mencionar los trabajos de [Plutowski and White, 1993], [Cohn et al., 1994], [Cohn, 1996] en los cuales redes parcialmente entrenadas son usadas para determinar *regiones de incerteza* alrededor de las cuales los ejemplos serán seleccionados. Otra manera de tratar el problema es el introducido por [Baum, 1991] quien propuso un algoritmo basado en las respuestas de los ejemplos ("query-based algorithm") para buscar iterativamente las regiones de clasificación en el conjunto de entrada.

Una pregunta importante, relacionada con el problema de la selección de ejemplos es: ¿ Cuál es el *número mínimo de ejemplos* que *asegure generalización total* en el proceso de aprendizaje? Por generalización total, entendemos error de generalización cero en el caso de redes booleanas (como es en este trabajo) o un error menor que alguna constante

positiva para el caso de redes con neuronas de salida continua. Este conjunto mínimo de ejemplos contiene *toda la información* acerca de la función a aprender y nos dará una cota mínima para el número de ejemplos necesarios para obtener generalización total en *cualquier proceso de selección de ejemplos* (dada la función y usando una arquitectura fija). Usualmente se puede el calcular el mínimo número *promedio* de ejemplos necesarios para obtener generalización con ejemplos seleccionados al azar [Baum and Haussler, 1989]; en este caso la dimensión "VC" (Vapnik-Chervonenkis), (ver por ejemplo [Haykin, 1994], capítulo 1) juega un papel importante para determinar la capacidad de generalización de una cierta arquitectura y para cualquier función en general.

A pesar de que el mínimo número de ejemplos necesarios para obtener generalización total (**MNEFG**, Minimum Number of Examples for Full Generalization) está relacionado con la complejidad intrínseca de la función a computar, también depende de la arquitectura de la red (redes incorrectamente diseñadas no podrán usar toda la información contenida en el conjunto de aprendizaje necesitando un mayor número de ejemplos). En este sentido el MNEFG resulta un parámetro más específico que la dimensión VC, ya que está relacionado con la complejidad combinada de la arquitectura y la función , mientras que la dimensión VC está relacionada exclusivamente con la arquitectura. De hecho el MNEFG es una cota superior para la dimensión VC.

Primero analizaremos tres diferentes funciones booleanas implementadas en diferentes arquitecturas usando neuronas lineales con umbral y obtendremos cotas mínimas para el MNEFG. La esencia del método es la siguiente: supongamos una red booleana con N neuronas de entrada, una única neurona de salida y una función a implementar. Como la red es booleana, el número de ejemplos posibles es finito y será 2^N. Cada uno de estos 2^N ejemplos puede representarse como un conjunto de restricciones (inecuaciones) que deben satisfacer los pesos sinápticos. En general los 2^N conjuntos de restricciones no serán independientes, sino solamente un subconjunto de ellos. Dado que cada ejemplo está relacionado con ciertas restricciones, el subconjunto independiente nos permitirá conocer el mínimo número de ejemplos que asegurará el aprendizaje de todos los ejemplos.

Los problemas que estudiaremos son: suma de dos números, corrimiento de bits y función de paridad, implementadas en arquitecturas en las cuales su computabilidad está asegurada. Mas aún, para cada uno de estos problemas existe un conjunto exacto de soluciones (ver [Cannas, 1995] para el primer caso, [Franco and Cannas, 1998] para el

segundo y [Franco and Cannas, 2000a] para el último caso). Todas las arquitecturas tienen un solo bit de salida para poder comparar los resultados, por lo que en los dos primeros casos solo el bit con mayor valor relativo será considerado. Los tres problemas son linealmente no separables, por ello las arquitectura mínimas (respecto del número de capas) para computar los problemas serán arquitecturas con una sola capa oculta. En nuestro análisis obtendremos que en estas arquitecturas **óptimas en profundidad** el MNEFG escala polinomialmente con el número de bits de entrada N, para el caso de los problemas de suma y paridad y exponencialmente en el caso de la función de paridad. Desde el punto de vista de las arquitecturas, la diferencia entre la arquitectura usada para implementar la función de paridad respecto de las otras, es que esta contiene N neuronas en la capa intermedia *totalmente conectadas*, mientras que en los otros dos casos hay un número menor de neuronas ocultas que están parcialmente conectadas con las neuronas de entrada. La topología de las redes parcialmente conectadas utilizadas en estos casos responde a un conocimiento a priori de las simetrías de los problemas correspondientes [Cannas, 1995], [Franco and Cannas, 1998]. Analizamos también la computabilidad (en el sentido del número de funciones que pueden implementarse) de las redes utilizadas.

La presencia de campos receptivos locales y de un menor número de neuronas ocultas puede pensarse como una solución particular de una red totalmente conectada con un conjunto de sinapsis iguales a cero. De esta forma el MNEFG obtenido puede considerarse como una cota mínima a este valor cuando la función sea implementada en redes totalmente conectadas.

Dado que los tres problemas analizados tienen en principio diferente complejidad, el MNEFG aparece como un parámetro interesante para **clasificar la complejidad de funciones** y estudiar su **interrelación con la arquitectura** en la cual la función es implementada.

A partir del análisis de los subconjuntos de ejemplos que determinan el MNEFG desarrollamos un criterio para la selección de ejemplos. Este primer criterio se probó usando una función particular en una arquitectura de tipo NORFP (Non-Overlapping Receptive Field Perceptron, ver [Hancock, 1994]) y a partir de este ejemplo pudimos refinar las ideas iniciales para proponer un criterio general para la selección de ejemplos en redes booleanas. Este criterio es chequeado con funciones **aleatorias** a través de simulaciones numéricas, obteniéndose una considerable mejora en la capacidad de generalización com-

parada con la obtenida usando ejemplos al azar, especialmente en lo que concierne a la probabilidad de obtener generalización total.

Mostramos también algunas simulaciones numéricas para el caso de suma y corrimiento de bits implementadas en redes pequeñas.

El caso de la función de paridad es analizado en detalle tanto para sinapsis binarias ± 1 como continuas. Se estudia primero la función de paridad en la arquitectura standard *totalmente conectada*, obteniéndose el resultado de que un número exponencial de ejemplos es necesario para obtener generalización total en el caso de sinapsis continuas, hecho que no es observado en el caso de sinapsis binarias. Continuamos el estudio analizando la misma función pero implementada en una familia de *arquitecturas modulares*, diseñadas al efecto (ver capítulo 2), mostrando como la capacidad de generalización es notoriamente incrementada con el uso de estas arquitecturas modulares, tanto para sinapsis continuas como binarias.

3.2 Selección de Ejemplos en el problema de Suma

Estudiaremos las propiedades de generalización de una red construída para computar la operación de suma de dos números de N bits cada uno. La arquitectura elegida [Cannas,1995] posee una única capa oculta y dado que el problema de suma es linealmente no separable estamos en presencia de una arquitectura óptima en cuanto al número de capas (profundidad). La red está compuesta por $2N$ neuronas binarias en la capa de entrada, correspondientes a los dos número de N bits a sumar, N neuronas intermedias y N neuronas de salida. Con el fin de simplificar el análisis y permitir la comparación con cualquier función que tendrá al menos una neurona de salida, estudiaremos solamente un bit de salida, en este caso aquel con el mayor valor relativo. La generalización al caso de N bits de salida es directa dado que el bit con mayor valor relativo fijará las sinapsis compartidas en la estructura completa a los valores correctos. Así el resto de las sinapsis podrán ser fijadas por un conjunto independiente de ejemplos seleccionados por el mismo procedimiento.

La red, para computar el bit con mayor valor relativo, poseerá unicamente dos neuronas intermedias, una totalmente conectada a las $2N$ neuronas de entrada y la otra conectada a $2N-2$ bits de entrada (todos menos los dos bits con mayor valor relativo); ver figura 3.1a.

Existirá también conexión directa desde los dos bits de entrada con mayor valor relativo hacia la neurona de salida. Un ejemplo de una red para el caso $N = 3$ se muestra en la figura 3.1a, donde se muestra la arquitectura para el bit con mayor valor relativo. Podemos ver que la arquitectura es casi totalmente conexa, no existiendo conexión únicamente entre dos bits de entrada y una de las dos neuronas intermedias. Esta simplificación esta basada en el conocimiento previo de la función de suma, en la cual existe una asimetría producida por el acarreo de bits, que siempre ocurre desde los bits de menor valor relativo hacia los de mayor valor(Cannas, 1995). Esperamos, dado que la red es casi totalmente conexa, que la arquitectura sea lo bastante general como para computar un gran número de funciones, por lo menos aquellas que comparten la propiedad de asimetría.

A los fines del análisis, la red puede simplificarse aún más usando propiedades de la función de suma; sabemos que los bits de igual valor relativo tendrán un impacto similar en el resultado y por lo tanto es razonable buscar soluciones en las cuales las sinapsis que surgen de estos pares de bits con igual valor relativo tengan un mismo valor. Esta simplificación puede visualizarse como otra red con N neuronas de entrada, en lugar de las $2N$ previas, las cuales podrán tomar los valores {0,1,2} en vez de ser binarias. De esta manera , por ejemplo, el par de entradas [11-10-00:0], [11-01-00:0] se transforman en un único ejemplo [2-1-0:0] en la arquitectura simetrizada.

Comenzaremos nuestro estudio con el caso de suma de dos números de 3 bits para luego generalizar el resultado al caso de números de N bits. En la figura 3.1b se muestra la arquitectura correspondiente al caso con sinapsis simetrizadas.

La neurona de salida, S, recibe conexiones directas de las 2 neuronas intermedias $D, E = 0, 1$ a través de las sinapsis d_1, e_1 y desde la neurona A, a través de la sinapsis a_0, y computa la siguiente función:

$$S = \theta\{a_0 A + d_1\theta[a_1 A + b_1 B + c_1 C - T_d] + e_1\theta[b_2 B + c_2 C - T_e] - T_s\} \qquad (3.1)$$

donde $A, B, C = 0, 1, 2$; $\theta(x)$ es la función escalón de Heaviside y T_α ($\alpha = d, e, s$) son los valores de los umbrales.

Los valores de la sinapsis a_0 y de los umbrales T_d y T_e han sido restringidos a la condición de ser mayores que cero con el fin de reducir las posibles representaciones internas a una única (ver capítulo 1).

El requerimiento de que la red compute el conjunto completo de 27 ejemplos puede

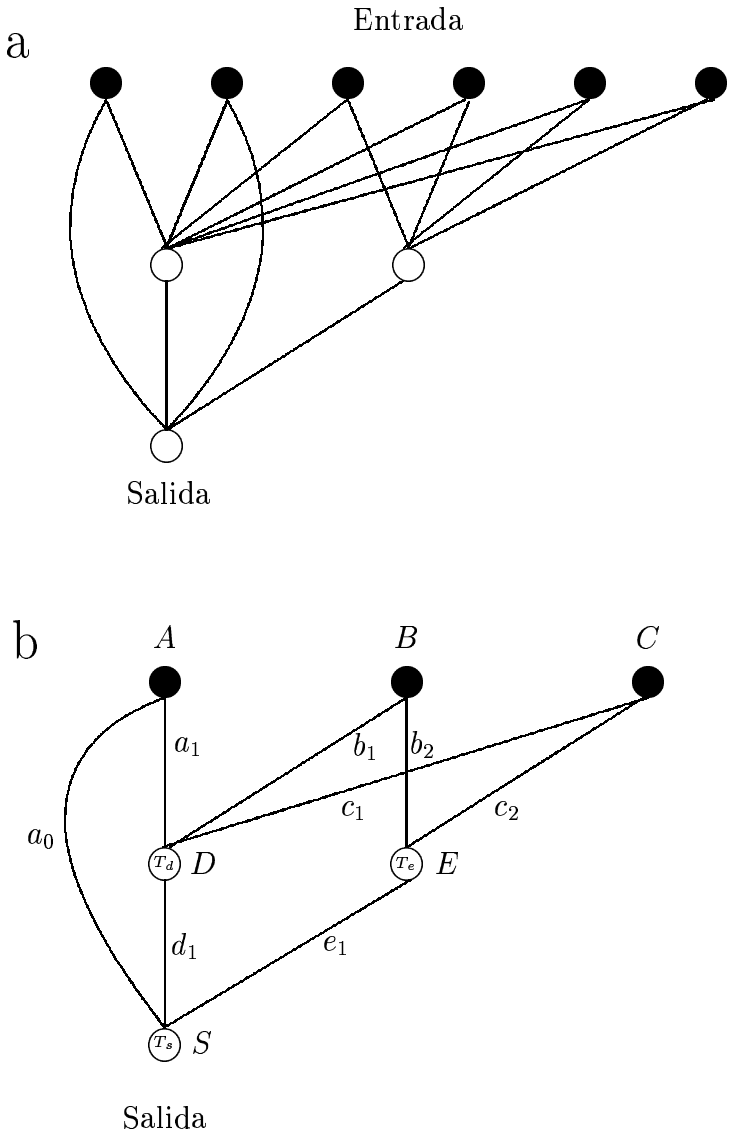


Figura 3.1: Estructura de una red neuronal para computar el bit de mayor valor relativo para la función de suma de dos números de tres bits. a) Estructura general para sinapsis arbitrarias b) En este caso las sinapsis correspondientes a bits de entrada con igual valor relativo han sido simetrizadas de modo que los 6 bits de entrada pueden ser reemplazados por tres neuronas de entrada (denotadas por A, B, C), que podrán tomar los valores {0,1,2}

expresarse en las siguientes condiciones necesarias y suficientes:

$$T_d > \quad a_1 \quad \geq \frac{T_d}{2} \tag{3.2}$$

$$a_1 + 2b_1 \geq \quad T_d \quad > 2b_1 + 2c_1 \tag{3.3}$$

$$a_1 + b_1 + 2c_1 \geq \quad T_d \quad > a_1 + b_1 + c_1 \tag{3.4}$$

$$2b_2 \geq \quad T_e \quad > 2c_2 \tag{3.5}$$

$$b_2 + 2c_2 \geq \quad T_e \quad > b_2 + c_2 \tag{3.6}$$

$$e_1 \geq \quad T_s \quad > 0 \tag{3.7}$$

$$a_0 \geq \quad T_s \quad > 2a_0 + d_1 \tag{3.8}$$

$$2a_0 + d_1 + e_1 \geq \quad T_s \quad > a_0 + d_1 + e_1 \tag{3.9}$$

Mostraremos que si la red es capaz de computar un conjunto seleccionado de 12 ejemplos las condiciones anteriores serán satisfechas y la red será capaz de computar eficientemente todos los ejemplos y por lo tanto la red adquirirá **generalización total** con menos de la mitad de todos los ejemplos (64). Denotaremos cada ejemplo escribiendo entre corchetes los tres valores correspondientes a cada uno de los bits de entrada y a continuación separado por un punto y coma el valor de salida. De este modo:

- Del ejemplo $[000:0]$ obtenemos la parte derecha de la Ec. (3.7).
- De los ejemplos $[100:1]$ y $[200:0]$ obtenemos:

  $$d_1 < -a_0 \tag{3.10}$$

  y por lo tanto se verifican las Ecs. (3.2) y (3.8)
- Del ejemplo $[020:1]$ obtenemos:

  $$d_1\theta[2b_1 - T_d] + e_1\theta[2b_2 - T_e] > T_s$$

  que conjuntamente con $T_s > 0$ y la Ec.(3.10) verifican la parte izquierda de la Ec.(3.7) y la parte izquierdade la Ec.(3.5).
- Del ejemplo $[120:0]$ obtenemos la parte izquierda de la Ec.(3.3) junto con la parte derecha de la Ec.(3.9).

- De los ejemplos [111 : 1] y [112 : 0] obtenemos la Ec.(3.4) y también que $c_1 > 0$.
- De los ejemplos [011 : 0], [012 : 1] y [002 : 0] obtenemos la Ec.(3.6) y la parte derecha de la Ec.(3.5).
- Del ejemplo [222 : 1] obtenemos la parte izquierda de la Ec.(3.9).
- Finalmente, del ejemplo [022 : 1] obtenemos la parte derecha de la Ec.(3.3) y consecuentemente la verificación del conjunto completo de Ecs.(3.2-3.9).

Para el caso más general con N bits de entrada, podemos ver que por cada bit que agregamos en la entrada es suficiente agregar cuatro ejemplos para obtener generalización total. Por ejemplo, para el caso $N = 4$ tomamos los doce ejemplos correspondientes al caso de 3 bits, convertidos para el caso de 4 bits agregando un cero en el bit ubicado más a la derecha; veamos esta conversión con un ejemplo: el ejemplo[112:0] correspondiente al caso de 3 bits estará representado en el caso de 4 bits por el ejemplo [1120:0]. También, como mencionamos, es necesario agregar 4 nuevos ejemplos, 2 pares de ejemplos con respuestas opuestas relacionados con el nuevo bit. Para el ejemplo considerado, $N = 4$ estos dos pares de ejemplos son {[0111:0] [0112:1]} y {[1111:1] [1112:0]}. El mismo procedimiento se repite a medida que aumentamos el número de bits de entrada, obteniendo el resultado general de que con $4N$ ejemplos puede obtenerse generalización total para este problema. Si consideraramos el caso de N bits de salida, el número total de ejemplos requerido se eleva a $2N(N+1)$. Vemos que el MNEFG crece polinomialmente con N, en tanto que el número total de ejemplos crece siempre exponencialmente. En la tabla 3.1 se muestran estos resultados conjuntamente con los obtenidos para otras funciones que trataremos a continuación.

3.3 Selección de Ejemplos para la función de Corrimiento de Bits

La función de corrimiento de bits ("bit-shifting") es una operación básica en circuitos de computadoras siendo también usada para modelar circuitos biológicos de visión. (Ver [Franco and Cannas, 1998] y referencias allí citadas). La estructura para computar esta función ya fue descripta en detalle en el capítulo 2 y esencialmente consta de tres capas: una capa de entrada con $N + log_2(N+1)$ bits, N neuronas en la capa intermedia y N

	Suma	Corrimiento de bits	Paridad(pesos continuos)
Número de sinapsis	$2N+4$	$N(2+log_2(N+1))$	N^2+N
Número total de ejemplos	3^N	$2^N(N+1)$	2^N
MNEFG	$4N$	$(N+1)^2$	$\mathcal{O}(2^N)$

Tabla 3.1: Algunas características de las redes usadas para computar los problemas de: suma de dos números, corrimiento de bits y paridad.

neuronas de salida. Para simplificar el estudio y poder comparar con otra redes analizaremos un único bit de salida. Comenzaremos nuestro análisis con el caso de una red con $N = 3$, es decir 5 bits en la capa de entrada, 3 bits correspondientes al número a correr o número de entrada, más dos bits indicadores de corrimiento. Estos resultados particulares serán luego generalizados para el caso de un número de entrada de cualquier tamaño. La estructura para el caso de 3 bits de entrada, 2 bits indicadores y una única neurona de salida se muestra en la figura 3.2, donde la neurona de salida S computa la función:

$$S = \theta[J_a\theta(a_1I_1 + a_2S_1 + a_3S_0 - T_a) + J_b\theta(b_1I_2 + b_2S_1 + b_3S_0 - T_b) + \\ + J_c\theta(c_1I_3 + c_2S_1 + c_3S_0 - T_c) - T] \tag{3.11}$$

$I_1, I_2, I_3 = 0, 1$ son los bits de entrada a correr; $S_0, S_1 = 0, 1$ son los bits indicadores , J_a, J_b, J_c son las sinapsis conectando las neuronas intermedias A, B, C y la neurona de salida S, y a_i, b_i, c_i son las sinapsis entre las respectivas neuronas intermedias (A, B, C) y los bits de entrada e indicadores.

Del mismo modo que en la sección anterior, impondremos condiciones sobre los umbrales de las neuronas intermedias T_a, T_b, T_c, restringiendo sus valores a ser mayores que cero con el fin de reducir las posibles representaciones internas a una única representación. También para simplificar la generalización al caso de tener N bits en la entrada las sinapsis J_a, J_b y J_c tendrán que ser mayores que el umbral de la neurona de salida T. Estas restricciones no producen grandes cambios en los resultados finales siendo su función la de simplificar la obtención de los resultados.

Las Ecs. de los 32 ejemplos posibles pueden expresarse en las siguientes condiciones necesarias y suficientes que deben cumplir los pesos y umbrales:

$$T > \quad 0 \tag{3.12}$$

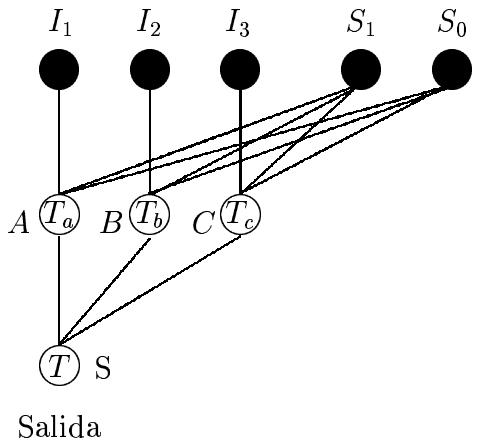


Figura 3.2: Estructura de una red neuronal para computar el bit de salida ubicado más a la izquierda para una operación de corrimiento de bits, donde I_1, I_2, I_3 son los bits de entrada y S_1, S_0 son los bits indicadores de corrimiento.

$$a_1 \geq T_a > a_1 + a_3 \tag{3.13}$$

$$a_1 + a_2 < T_a \tag{3.14}$$

$$b_1 < T_b \tag{3.15}$$

$$b_1 + b_3 \geq T_b > b_3 \tag{3.16}$$

$$b_1 + b_2 + b_3 < T_b \tag{3.17}$$

$$c_1 < T_c \tag{3.18}$$

$$c_1 + c_2 \geq T_c > c_2 \tag{3.19}$$

$$c_1 + c_2 + c_3 < T_c \tag{3.20}$$

Denotaremos los ejemplos escribiendo entre corchetes los tres bits de entrada más los dos bits indicadores y el bit de salida correspondiente separado por dos puntos.

Al igual que se mostró en la sección anterior para la red de suma, se puede verificar que los diez ejemplos siguientes aseguran el cumplimiento de las inecuaciones anteriores por lo que su aprendizaje producirá generalización total: $\{[000-00:0], [100-00:1], [100-01:0], [100-10:0], [010-01:1], [010-11:0], [010-00:0], [001-00:1], [001-10:1], [001-11:0]\}$. Por ejemplo, del ejemplo $[000-00:0]$ derivamos $T > 0$ Ec.(3.12); del ejemplo $[100-00:1]$ obtenemos el lado izquierdo de Ec.(3.13). Siguiendo con el resto de los

ejemplos mencionados se verifican el resto de las inecuaciones. Notemos que este conjunto de ejemplos involucra todos aquellos ejemplos con un bit de entrada prendido (i.e., igual a 1) y todas las combinaciones posibles para el estado de los bits indicadores. Desde el punto de vista del aprendizaje podemos ver que a partir del aprendizaje del ejemplo $[000 - 00 : 0]$ que fijará el valor del umbral T a algún valor positivo, cada ejemplo con un bit prendido y alguna combinación de bits prendidos y apagados en los bits indicadores van fijando ciertas sinapsis a sus valores correctos. Por ejemplo el aprendizaje del ejemplo $[100 - 00 : 1]$ fijará la sinapsis conectando el bit de entrada I_1 con la neurona intermedia A a un valor mayor al valor del umbral T_a y en el mismo modo el resto de los ejemplos irán fijando el resto de las sinapsis.

La extensión de este resultado al caso de un número de entrada con N bits involucra todos los ejemplos que poseen un solo bit de entrada prendido junto con todas las combinaciones posibles de bits indicadores, más los ejemplos con todos los bits de entrada apagados (i.e., iguales a 0) con todas las combinaciones de los bits indicadores. Este procedimiento conduce al resultado de que para el caso de N bits de entrada es suficiente la enseñanza de $(N+1)^2$ ejemplos a fin de obtener generalización total. En la tabla 3.1 se muestran ciertas características de esta red y el número de ejemplos necesarios para obtener generalización conjuntamente con los resultados para otras arquitecturas.

3.4 Selección de ejemplos y modularidad en el problema de Paridad

La función de paridad es una de las funciones más usadas para testear algoritmos aprendizaje debido a su simple definición y su gran complejidad dado que el cambio de un bit en la entrada produce un cambio de salida (Rumelhart and McClelland, 1986; Tessauro and Janssens, 1988). Analizaremos a continuación la influencia de la selección de ejemplos en el aprendizaje de la función de paridad en diferentes arquitecturas, siguiendo los procedimientos de las secciones anteriores.

3.4.1 Selección de Ejemplos en la Arquitectura Standard

La función de paridad posee una única neurona de salida que tendrá que activarse cuando el número de bits de entrada prendidas sea impar mientras que deberá estar apagada

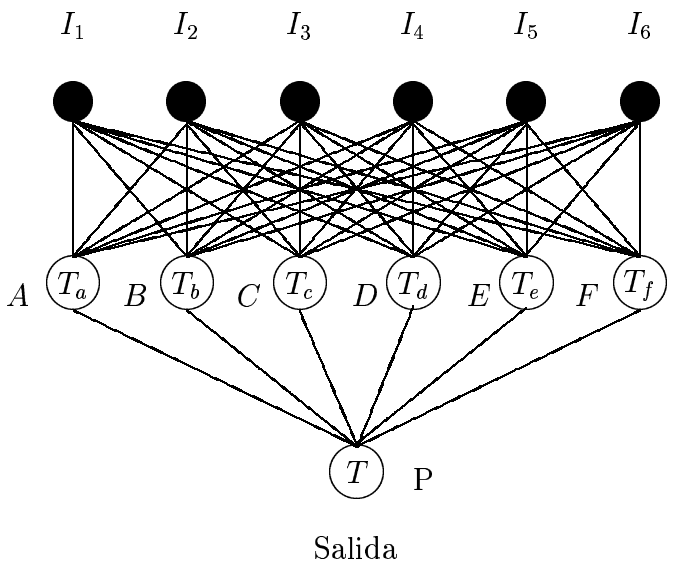


Figura 3.3: Estructura de una red neuronal totalmente conexa compuesta por una única capa oculta para computar la función de Paridad de 6 bits de entrada.

cuando el número de bits prendidos en la entrada sea par. Analizaremos primero la selección de ejemplos en la arquitectura que denominamos "arquitectura básica" y que fue detalladamente analizada en el capítulo 2.

Al igual que hicimos en las secciones anteriores primero analizaremos un caso particular con 6 neuronas de entrada para luego generalizar al caso de N bits de entrada. En la figura 3.3 se muestra una arquitectura standard para computar esta función.

El funcionamiento standard de esta red (ver [Minsky and Papert, 1969], [Rumelhart and McClelland, 1986], y [Hertz et al., 1991]) se basa en el siguiente comportamiento de las seis neuronas de la capa intermedia: el número de neuronas en esta capa que tienen que estar prendidas es igual al número de bits prendidos en la entrada. Como vimos en la sección 2.6, las neuronas intermedias están conectadas a la neurona de salida P, a través de sinapsis con valores alternados $+1$ y -1, de forma tal que cuando el número de neuronas prendidas en la capa de entrada (y por lo tanto en la intermedia) es par,

sus contribuciones sobre la capa de salida son canceladas debido a la alternancia de los valores sinápticos; pero cuando el número es impar todas menos una contribución (la cual es positiva) se anulan, permitiendo activar la neurona de salida.

El funcionamiento correcto de las neuronas intermedias puede obtenerse considerando las siguientes condiciones:

1. La neurona intermedia A, tiene que estar prendida cuando una o más bits de entrada estén prendidos, en otro caso A tiene que estar apagada.

2. La neurona intermedia B, tiene que estar prendida cuando dos o más bits de entrada estén prendidos, en otro caso B tiene que estar apagada.

3. La neurona intermedia C, tiene que estar prendida cuando tres o más bits de entrada estén prendidos , en otro caso C tiene que estar apagada.

4. La neurona intermedia D, tiene que estar prendida cuando cuatro o más bits de entrada estén prendidos, en otro caso D tiene que estar apagada.

5. La neurona intermedia E, tiene que estar prendida cuando cinco o más bits de entrada estén prendidos, en otro caso E tiene que estar apagada.

6. La neurona intermedia F, tiene que estar prendida cuando seis bits de entrada estén prendidos, en otro caso F tiene que estar apagada.

Claramente permutando el orden de las neuronas intermedias obtenemos otros modos de funcionamiento de la red, pero que son totalmente equivalentes, en cuanto solo involucran cambios en el ordenamiento de la red y no en la estructura de funcionamiento.

Denotaremos con $P = 0, 1$ el valor de la neurona de salida y tendremos entonces que P computa la siguiente función:

$$P = \theta \left\{ a\theta \left[\sum_{i=1}^{6} (a_i I_i - T_a) \right] + b\theta \left[\sum_{i=1}^{6} (b_i I_i - T_b) \right] + c\theta \left[\sum_{i=1}^{6} (c_i I_i - T_c) \right] + \right.$$

$$\left. + d\theta \left[\sum_{i=1}^{6} (d_i I_i - T_d) \right] + e\theta \left[\sum_{i=1}^{6} (e_i I_i - T_e) \right] + f\theta \left[\sum_{i=1}^{6} (f_i I_i - T_f) \right] - T \right\} \qquad (3.21)$$

donde $I_i = 0, 1 (i = 1, \ldots, 6)$ son los valores de las neuronas de entrada.

De las condiciones 1-6 obtenemos el siguiente conjunto de inecuaciones para los valores de las sinapsis que aseguran el correcto funcionamiento de la red.

$$a_i \geq T_a \ \ \forall i \tag{3.22}$$

$$b_i + b_j < T_b \ \ \forall\, i, j \tag{3.23}$$

$$c_i + c_j + c_k \geq T_c \ \ \forall\, i, j, k \tag{3.24}$$

$$d_i + d_j + d_k + d_l < T_d \ \ \forall\, i, j, k, l \tag{3.25}$$

$$e_i + e_j + e_k + e_l + e_m \geq T_e \ \ \forall\, i, j, k, l, m \tag{3.26}$$

$$f_i + f_j + f_k + f_l + f_m + f_n < T_f \ \ \forall\, i, j, k, l, m, n \tag{3.27}$$

$$\tag{3.28}$$

Si fijamos los valores de los umbrales $T_a, T_b, T_c, T_d, T_e, T_f$, obtenemos un conjunto de $2^6 - 1$ inecuaciones *independientes* para los valores de las sinapsis $\{a_i\}, \{b_i\}, \ldots, \{f_i\}$. Por otro lado, el número total de ejemplos es 2^6 y, dado que cada ejemplo asegura el cumplimiento de una sola inecuación, será necesario para obtener generalización total el aprendizaje de todos los ejemplos excepto el ejemplo más simple [000000 : 0], el cual determina el signo del umbral de la neurona de salida. De esta manera no es posible obtener generalización usando un subconjunto cualquiera de ejemplos.

Finalmente, la generalización de este resultado para el caso de la función de paridad con N entradas es directa, porque en tal caso tendremos 2^N ejemplos y $2^N - 1$ desigualdades.

El análisis previo estuvo basado en una representación interna particular, ya que la red presentada admite otras representaciones internas para computar la función de paridad, como por ejemplo la representación usada en [Franco and Cannas, 1998] para la construcción de una red de suma, resultado mostrado en el capítulo 2. En estos casos siguiendo el mismo análisis que el realizado previamente puede mostrarse que si bien no es necesario el aprendizaje de todos los ejemplos, sí son necesarios del orden $\mathcal{O}(2^N)$. Este resultado sugiere que para el caso de usar la arquitectura básica para computar la función de paridad el mínimo número de ejemplos para obtener generalización total es siempre de orden exponencial en el número de entradas, y se corrobora en simulaciones numéricas en redes pequeñas.

3.4.2 Selección de Ejemplos en Arquitecturas Modulares

En la subsección anterior vimos que es casi imposible obtener generalización total en el problema de paridad implementado en la arquitectura standard (arquitectura totalmente conexa) dado que es necesario el uso de casi todos los ejemplos para que la red generalice.

Calcularemos ahora una cota superior M, para el mínimo número de ejemplos para obtener generalización total en las arquitecturas modulares construídas en el capítulo 2 para computar la función de paridad [Franco and Cannas, 2000b]. Este número depende tanto de la función a implementar (en este caso la función de paridad) como de la arquitectura elegida y podemos calcularlo analíticamente analizando directamente las ecuaciones obtenidas a partir de los ejemplos, del mismo modo que hicimos en las secciones anteriores.

Para empezar el cálculo, restringiremos los valores de los pesos sinápticos a valores **discretos** y finitos, en particular tomaremos $J_{ij} = \{\pm 1\}$ y posteriormente extenderemos los resultados al caso de pesos continuos. Sabemos de las soluciones obtenidas en el capítulo 2 que estos valores de pesos alcanzan para computar la función de paridad.

Analicemos primero un caso similar al de la subsección anterior, arquitectura básica $N = m$, y tomemos un caso particular con $N = m = 4$, tal como el que se muestra en la figura 2.8. Sean $I_i = \{0, 1\}$ $(i = 1, 2, 3, 4)$ los 4 bits de entrada,

$$h_i = \Theta \left[\sum_{j=1}^{4} J_{ij} I_j - T_i \right] \qquad i = 1, 2, 3, 4 \tag{3.29}$$

es la actividad de las neuronas intermedias, T_i son los umbrales de estas neuronas, los cuales no están restringidos, y $J_{ij} = \pm 1$ son las sinapsis conectando las neuronas de entrada con las neuronas intermedias. La salida de la red S vendrá dada por:

$$S = \Theta \left[\sum_{k=1}^{4} w_k h_k - T \right] \tag{3.30}$$

donse T es real y sin restricciones en su rango de valores, y $w_k = \pm 1$ son las conexiones entre las neuronas de la capa intermedia y la neurona de salida.

Diferentes elecciones de los umbrales $\{T_i, T\}$ permiten obtener diferentes soluciones al problema de paridad, cada una de ellas asociada con una diferente representación interna, *i.e.*, diferentes estados de activación de las neuronas intermedias $\{h_i\}$ para cada ejemplo de entrada (ver [Rumelhart and McClelland, (1986)], [Franco and Cannas, (1999)]).

Elegiremos los umbrales de la misma manera que hicimos con pesos no restringidos (continuos) en la subsección anterior: $T = 0.5$ y $T_i = 0.5, 1.5, \ldots, N - 0.5$, para $i = 1, \ldots, N$. Puede verse que con esta elección de umbrales existe una sola representación interna posible para el cómputo de la función de paridad, excepto por permutaciones triviales en el orden de las neuronas intermedias.

Denotaremos los ejemplos, escribiendo entre corchetes los valores de las neuronas de entrada y separado por dos puntos la salida correcta correspondiente a ese ejemplo. Con nuestra elección de umbrales el ejemplo [0000:0] es automáticamente computado por la red. Consideremos ahora el ejemplo [1000:1]. Dado que todos los umbrales son positivos y en particular $T_2, T_3, T_4 > 1$, tendremos que $h_2 = h_3 = h_4 = 0$ y la Ec.(3.30) con $S = 1$ requirirá $w_1 = h_1 = 1$. De la Ec.(3.29)obtenemos que $J_{11} = 1$. Del mismo modo el aprendizaje de todos los ejemplos con un solo bit prendido: [1000:1], [0100:1], [0010:1] y [0001:1] implicarán las condiciones: $w_1 = J_{1j} = 1$, $j = 1, 2, 3, 4$.

Si ahora consideramos los ejemplos con dos bits prendidos, por ejemplo el [1100:0], vemos que $h_1 = 1$ y de la Ec.(3.30) obtenemos $h_2\, w_2 + 0.5 < 0$, lo cual implica $w_2 = -1$ y $h_2 = 1$. De la Ec.(3.29) obtenemos $J_{21} = J_{22} = 1$. Repitiendo este procedimiento con el resto de los ejemplos con dos bits prendidos obtenemos $J_{2j} = 1$, $j = 1, 2, 3, 4$, pero notemos que solo 2 de los 6 ejemplos posibles que cumplen esta condición son necesarios para hacer cumplir la condición $J_{2j} = 1$, siempre que los bits encendidos en los dos ejemplos sean todos distintos, (por ejemplo podemos tomar: [1100:0] y [0011:0]).

Usemos también los ejemplos con tres bits de entrada prendidos, viendo que dos de los cuatro ejemplos posibles son suficientes y necesarios para determinar que $w_3 = J_{2j} = 1$, $j = 1, 2, 3, 4$. Finalmente tomando el ejemplo con todos los bits prendidos, $[1111 : 0]$, obtenemos $w_4 = -1$ y $J_{4j} = 1$, para $j = 1, 2, 3, 4$.

De esta forma, vemos que es posible para el caso $N = 4$ y pesos discretos (± 1), tomar 9 ejemplos seleccionados entre los 16 ejemplos posibles, para obtener generalización total. Recordemos que para el caso de pesos continuos necesitamos casi todos los ejemplos para obtener generalización total.

La generalización de este resultado al caso de N bits de entrada es directa: acomodamos los 2^N ejemplos posibles en i grupos que contienen los ejemplos que tienen i bits prendidos y de cada grupo tomamos el mínimo número de ejemplos que asegura que cada bit de entrada aparece al menos una vez encendido en los ejemplos seleccionados. Esto nos lleva

a que el número de ejemplos necesarios para obtener generalización total sea:

$$M = \sum_{i=1}^{N} Int_{+}\left(\frac{N}{i}\right), \tag{3.31}$$

donde $Int_{+}(x)$ es igual a x si x es entero e igual al entero **mayor** más próximo a x si x no es entero. Para valores grandes de N, este número escala como $M \propto Nlog(N)$, bastante menor que el número total de ejemplos 2^N y que el número total de sinapsis $N^2 + N$.

Analicemos ahora una red modular con fmax $= m$, y $m < N$ y tomemos un caso particular con $N = 9$, $m = 3$ correspondiente a la arquitectura mostrada en la figura 2.10 . Esta red tiene una estructura modular derivada de la arquitectura básica con $N = m = 3$.

Como se explico anteriormente, el funcionamiento de las redes modulares para el cómputo de la función de paridad (ver Capítulo 2) es el siguiente: los tres módulos de entrada (indicados con I, II y III en la figura 2.10 computan independientemente la paridad de los correspondientes bits de entrada, mientras que el módulo de salida computa la paridad de los resultados obtenidos por los módulos de entrada S_1, S_2 y S_3 . Una elección natural para la selección de ejemplos es comenzar con todos los sub-ejemplos necesarios para aprender la función de paridad en cada uno de los módulos; esto es, seleccionamos los ejemplos usados en el caso de la arquitectura standard para cada uno de los módulos, por lo que en esta arquitectura esos ejemplos consistirán en bits de entrada que tienen el sub-ejemplo a usar en la entrada correspondiente al módulo y cero en los bits de los otros módulos (por ejemplo, si queremos enseñarle al módulo I el ejemplo [xxx:y], en esta arquitectura ese ejemplo será el [xxx 000 000:y]). En este caso las neuronas intermedias S_1, S_2 y S_3, respuestas de los respectivos módulos, estarán prendidas a lo sumo una a la vez. Necesitamos entonces $3\sum_{i=1}^{3} Int_{+}\left(\frac{3}{i}\right) = 18$ ejemplos para fijar todos los pesos sinápticos de los módulos de entrada I, II y III y las sinapsis asociadas con la primer neurona oculta del módulo de salida (neurona con umbral $\frac{1}{2}$).

Una vez realizado el aprendizaje de estos ejemplos, cada módulo actuará como una unidad, computando eficientemente la función de paridad de los bits de entrada correspondientes a cada módulo. Para fijar los restantes pesos sinápticos del módulo de salida, consideraremos este módulo como una red del tipo standard pero en la cual las entradas serán los valores de las neuronas (S_1, S_2, S_3), que son las neuronas de salida correspondiente a los módulos I, II y III. De esta forma utilizaremos un nuevo conjunto de ejemplos

similar al utilizado en cada módulo de entrada, pero notando que los ejemplos correspondientes a un solo bit prendido, en este caso correspondiente a las neuronas (S_1, S_2, S_3), ya han sido considerados. Entonces serán necesarios 2 ejemplos con dos bits prendidos en las neuronas S_i, como por ejemplo los ejemplos [111 111 000:0] con valores para las neuronas S_i: $(S_1 = 1, S_2 = 1, S_3 = 0)$ y el ejemplo [111 000 111:0] con neuronas S_i: $(S_1 = 1, S_2 = 0, S_3 = 1)$. Todavía nos falta un ejemplo correspondiente al caso $(S_1 = S_2 = S_3 = 1)$ que puede ser el ejemplo [100 010 001:1]. Vemos entonces que con $M = 21$ ejemplos seleccionados de un total de $2^9 = 512$ es posible asegurar el aprendizaje de la función de paridad en esta arquitectura con pesos ± 1. Recordemos que para la arquitectura básica con pesos discretos hacen falta $M = 29$ ejemplos, valor que obtenemos de la Ec.(3.31) con $N = m = 9$.

La generalización de este resultado al caso general $N = m^k$ es directo: vamos tomando conjuntos de ejemplos que nos aseguran el aprendizaje de los pesos sinápticos del primer grupo de m^{k-1}; luego le enseñamos a la red los ejemplos para aprender los pesos correspondientes al segundo grupo de m^{k-2} módulos y así sucesivamente hasta considerar el último módulo de salida.

El número de ejemplos necesarios para el aprendizaje de la función de paridad en los m^{k-1} módulos de entrada es:

$$m^{k-1} \sum_{i=1}^{m} Int_+ \left(\frac{m}{i}\right) = N + m^{k-1} \sum_{i=2}^{m} Int_+ \left(\frac{m}{i}\right).$$

Una vez que el aprendizaje de estos ejemplos se ha realizado, necesitamos ejemplos para el segundo conjunto de módulos pero con la salvedad que ejemplos con un bit prendido para cada módulo ya están considerados en el aprendizaje del grupo de módulos anterior, por lo que necesitamos:

$$m^{k-2} \sum_{i=2}^{m} Int_+ \left(\frac{m}{i}\right)$$

ejemplos seleccionados para fijar el resto de los pesos sinápticos correspondientes a estos módulos. Repitiendo este procedimiento para los restantes conjuntos de módulos obtenemos el número total de ejemplos necesarios:

$$M(N) \;=\; N + \left[\sum_{j=0}^{k-1} m^j\right] \left[\sum_{i=2}^{m} Int_+ \left(\frac{m}{i}\right)\right]$$

$$= \quad N + \frac{N-1}{m-1} \sum_{i=2}^{m} Int_{+} \left(\frac{m}{i} \right) \tag{3.32}$$

para $N = m^k$. Notemos que para valores grandes de N y una gran modularidad $m \ll N$, $M(N)$ escala linealmente con N en lugar del comportamiento $M(N) \sim Nlog(N)$ cuando $m \sim N$.

Extender estos resultados para los casos en que $N = m^k - l$, con $l < m^{k-1}$, es un poco más complicado, pero puede ser resuelto con la construcción de una red modular con m^k bits de entrada y entrenar esta red con ejemplos donde los l bits extras son tomados iguales a cero. Una cota máxima al número de ejemplos necesarios para entrenar esta red es entonces $M(m^k - l) \leq M(m^k)$, donde $M(m^k)$ está dada por la Ec.(3.32).

Consideremos ahora el caso de redes modulares pero implementadas con pesos J_{ij} *continuos*. Nuevamente utilizaremos la estructura modular de las redes para realizar nuestro análisis recursivamente a partir de las propiedades de los módulos.

Para pesos continuos las restricciones impuestas por el aprendizaje de K ejemplos aparecen en la forma de K inecuaciones simultáneas que deben satisfacer los valores de los pesos sinápticos. En una sección previa vimos que para la arquitectura básica con m bits de entrada y con valores de umbrales elegidos del mismo modo que ahora, la generalización total implica el cumplimiento de $2^m - 1$ inecuaciones *independientes*: solamente el aprendizaje de todos los ejemplos asegura la generalización total con esta arquitectura.

Seguiremos los mismos pasos que en el procedimiento usado en el caso de pesos discretos para $N = m^k$ y $m < N$. En una primera etapa le enseñamos a la red los N ejemplos con un solo bit prendido. Esto fijará las sinapsis del primer grupo que conectan la entrada con neuronas que tienen valor de umbral $1/2$. Para fijar el resto de las sinapsis, debemos enseñarle a la red todos los $m^{k-1} \sum_{i=2}^{m} \left(\begin{smallmatrix} m \\ i \end{smallmatrix} \right)$ ejemplos conteniendo más de un bit prendido en cada uno de los módulos de entrada y cero en el resto ($\left(\begin{smallmatrix} m \\ i \end{smallmatrix} \right)$ denota el coeficiente binomial $\frac{m!}{i!(m-i)!}$). Para las capas sucesivas, notemos que los ejemplos con un bit prendido ya han sido enseñados, por lo que para fijar los restantes pesos sinápticos del j-ésimo conjunto de módulos necesitaremos $m^{k-j} \sum_{i=2}^{m} \left(\begin{smallmatrix} m \\ i \end{smallmatrix} \right)$ ejemplos. De esta forma el número total de ejemplos será

$$M(N) \quad = \quad N + \frac{N-1}{m-1} \sum_{i=2}^{m} \left(\begin{smallmatrix} m \\ i \end{smallmatrix} \right) =$$

Número de neuronas	Número de sinapsis N_s	Profundidad	Ejemplos necesarios para generalización M	
			Pesos restringidos ± 1	Pesos continuos (no restringidos)
$1 + 2m\frac{N-1}{m-1}$	$\frac{m(m+1)(N-1)}{m-1}$	$2\log_m N$	$N + \frac{(N-1)}{m-1}\sum_{i=2}^{m} Int_+\frac{m}{i}$ $\sim \mathcal{O}(N)$ para $m \ll N$ $\sim \mathcal{O}(NlogN)$ para $m \sim N$	$N + \frac{N-1}{m-1}(2^m - m - 1)$ $\sim \mathcal{O}(N)$ para $m \ll N$ $\sim \mathcal{O}(2^N)$ para $m \sim N$

Tabla 3.2: Algunas características de las redes modulares usadas para computar la función de paridad de N bits con un fan-in max igual a m. Referirse al texto por la definición de la función $Int_+(x)$

$$= \quad N + \frac{N-1}{m-1}(2^m - m - 1) \tag{3.33}$$

para una red modular con pesos continuos y $N = m^k$. El resultado es similar al obtenido para el caso de pesos discretos (Ec. 3.32), pero debemos reemplazar el término $Int_+\left(\frac{m}{i}\right)$ por $\binom{m}{i}$. Para $m = N$ recuperamos el resultado previamente obtenido de $M = 2^m - 1$. Como antes, $M(N)$ en Ec. 3.33 nos da una cota superior para el caso $N = m^k - l$. Vemos que para $m \ll N$ y $N \gg 1$, $M(N)$ escala linealmente con N para pesos continuos. Los resultados obtenidos para las arquitecturas modulares para computar el problema de paridad, en cuanto a sus características y número de ejemplos necesarios paa obtener generalización total son mostrados en la tabla 3.2.

3.4.3 Simulaciones Numéricas en el problema de Paridad

Realizamos simulaciones numéricas en redes de tamaño $N = 8$ con pesos discretos, ± 1, para implementar la función de paridad y usando simulated annealing como algoritmo de aprendizaje. Los parámetros del algoritmo de simulated annealing (*i.e.*, temperatura inicial, descenso de la temperatura, etc.) fueron mantenidos constantes en todas las simulaciones. Los valores de los umbrales fueron fijados a los valores citados anteriormente. También realizamos pruebas con valores de umbrales libres obteniendo similares resultados pero observando una mayor lentitud en la convergencia de los algoritmos de aprendizaje [Franco and Cannas, 2000b].

Con el fin de estudiar el efecto de la modularidad comparamos tres redes diferentes

con $f_{max} = 2$ ($m = 2$, modularidad máxima), $f_{max} = 8$ ($m = N = 8$, arquitectura básica monolítica) y el caso intermedio $f_{max} = 4$. Este último caso es una arquitectura mixta en el sentido que consta de dos módulos de entrada con $m = 4$ y un módulo de salida con $m = 2$. Calculamos para estas tres redes las curvas de aprendizaje: el error de generalización promedio ϵ_g *versus* la fracción de ejemplos $\rho_e = N_e/2^N$, siendo N_e el número de ejemplos seleccionados aleatoriamente en el conjunto de aprendizaje. Los resultados fueron promediados sobre diferentes conjuntos de N_e ejemplos y diferentes configuraciones iniciales aleatorias de pesos sinápticos $\{J_{ij} = \pm 1\}$. Tamaños típicos de muestras van entre 50 y 100 realizaciones. El entrenamiento de la red para cada condición inicial y cada conjunto de ejemplos es realizado hasta que el error de aprendizaje llega a cero y entonces el error de generalización es calculado sobre todo el conjunto de 2^N ejemplos.

En la figura 3.4 comparamos las curvas de aprendizaje para las tres redes. Para el caso $f_{max} = 8$, observamos que el error de generalización, ϵ_g, decae muy lentamente aproximándose a $\epsilon_g = 0$ asintóticamente a medida que ρ_e se aproxima a 1, mostrando una falta de generalización que es a menudo encontrada en redes neuronales sobredimensionadas en el número de conexiones [Boers et al., 1993]. Una mejora sistemática se observa a medida que incrementamos la modularidad, ocurriendo un dramático cambio de comportamiento cuando $f_{max} = 2$. En este último caso observamos que ϵ_g se mantiene casi constante $\epsilon_g \sim 0.5$ mientras que $N_e < M$, pero decae abruptamente a cero con unos cuantos ejemplos más, sugiriendo una transición de fase de un estado de *memorización* a uno de *aprendizaje perfecto o generalización.* Este efecto ya había sido indicado por [Patarnello and Carnevali, 1987] para redes neuronales compuestas por unidades booleanas con un $f_{max} = 2$. Pareciera que la transición es un efecto de la extrema modularidad ya que para $f_{max} = 4$, ϵ_g decae suavemente a cero, anulándose para $\rho \sim 0.5$.

A fin de obtener una idea sobre la relación entre la transición anteriormente mencionada y la modularidad analizamos como el proceso de aprendizaje afecta a los diferentes módulos constituyentes de la red a medida que incrementamos ρ_e para $f_{max} = 2$. Para $N = 8$ la red resultante posee tres conjuntos de módulos, que denotaremos como: de entrada, intermedio y de salida. Calculamos la probabilidad de aprender los valores correctos de los pesos sinápticos en un dado módulo (recordemos que dado que los umbrales son fijos, existe un único conjunto correcto de pesos). Las simulaciones numéricas fueron

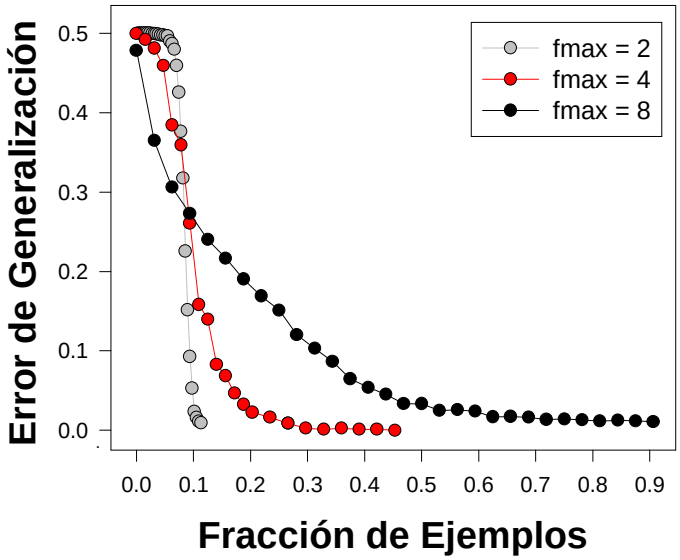


Figura 3.4: Error de generalización vs. Fracción de ejemplos aleatorios para tres redes que implementan la función de paridad con 8 bits de entrada usando diferentes arquitecturas. En negro los resultados correspondientes al caso de una arquitectura con una sola capa oculta con $f_{max} = 8$, en gris oscuro resultados correspondientes a una arquitectura con tres capas y un $f_{max} = 4$ y en gris claro los resultados para una red con 5 capas y un $f_{max} = 2$.

llevadas a cabo sobre muestras de tamaño 1000, verificando primero que módulos en una misma capa tuvieran la misma probabilidad tal como se espera por simetría. En la figura 3.5 comparamos las probabilidades obtenidas como función de la fracción de ejemplos ρ_e para módulos en diferentes capas. Observemos que el aprendizaje ocurre desde las capas de salida hacia las de entrada: a medida que incrementamos ρ_e las probabilidades de aprendizaje para los módulos de entrada e intermedios se mantienen en valores muy bajos ($\sim 10\%$) mientras que la probabilidad para el módulo de salida se incrementa en forma constante. Cuando ρ_e alcanza un valor de $\rho_e \sim 0.06 \sim M/2^N$, siendo la probabilidad del módulo de salida alrededor del 40%, la probabilidad de los módulos de entrada e intermedios experimenta un brusco crecimiento, volviéndose sus valores aproximadamente iguales y convergiendo conjuntamente con la probabilidad del módulo de salida rápidamente a uno. Podemos interpretar estos resultados de la siguiente manera: dada la estructura jerárquica de estas redes modulares, el aprendizaje correcto de un dado módulo tiene mayor influencia en el proceso global de aprendizaje, cuanto más cerca de la salida se ubique el mismo. En otras palabras, no es posible el aprendizaje de módulos cercanos a la entrada en tanto las sinapsis de los módulos en capas subsiguientes no estén fijados a sus valores correctos. El uso de un número de ejemplos del orden del MNEFG asegura el aprendizaje del módulo de salida (el más importante) con alta probabilidad ($\sim 50\%$. Una vez que esto ocurre el proceso se repite recursivamente en los módulos de las capas anteriores. No obstante, para las subredes anteriores el número de ejemplos necesarios para su aprendizaje será menor que el requerido si fueran entrenadas independientemente, ya que cierto número de sinapsis en dichas subredes ya habían sido fijadas a los valores correctos durante el aprendizaje del módulo de salida. De esta manera, se genera un efecto en cascada en el cual, una vez aprendido el módulo de salida, bastan unos pocos ejemplos más para producir generalización perfecta. Esperaríamos entonces que este efecto sea más marcado cuanto mayor sea N, generando una transición de fase, posiblemente discontinua.

Con el fin de establecer la ventaja práctica de las redes modulares, medimos el tiempo promedio de cómputo necesario para aprender un número N_e de ejemplos suficiente para que $\epsilon_g \leq 0.05$. Para ser más precisos, hicimos simulaciones numéricas con N_e fijo hasta un determinado número máximo de iteraciones, repitiendo este procedimiento con diferentes condiciones iniciales y diferentes conjuntos de N_e ejemplos. En algunas corridas el aprendizaje no es exitoso, es decir el algoritmo no converge a error de aprendizaje nulo.

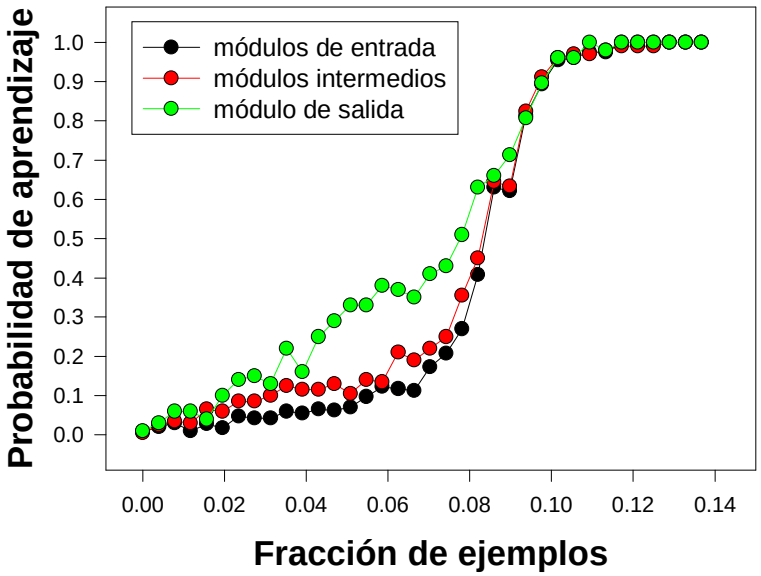


Figura 3.5: Probabilidad de generalización vs. fracción de ejemplos calculada para los distintos módulos (de entrada, intermedios y de salida) de una red modular con $N = 8$ y $f_{max} = 2$.

Computamos, entonces, el tiempo total de cómputo (en ms) necesario para obtener K corridas de aprendizaje exitosas (incluyendo los casos en que la red no aprende) y luego dividimos por K ($K = 50$ fue suficiente para estabilizar el promedio).

En la figura 3.6 mostramos una comparación para los tiempos de cómputo para las tres redes modulares aquí consideradas con $N = 8$, para conjuntos de ejemplos seleccionados y al azar. Mientras que para el primer caso la mejora es considerable y crece monótonamente a medida que f_{max} disminuye, en el caso de ejemplos elegidos al azar una gran habilidad de generalización con $f_{max} = 2$ es obtenida al costo de una muy baja eficiencia (notar la escala logarítmica en el eje de las ordenadas en la figura 3.6). Sin embargo el tiempo de cómputo en este último caso no se incrementa monótonamente con f_{max}, siendo mínimo $f_{max} = 4$.

También analizamos la influencia de la selección de ejemplos en la capacidad de aprendizaje de redes modulares. En la figura 3.7 comparamos los valores de ϵ_g *vs* $N_e/2^N$

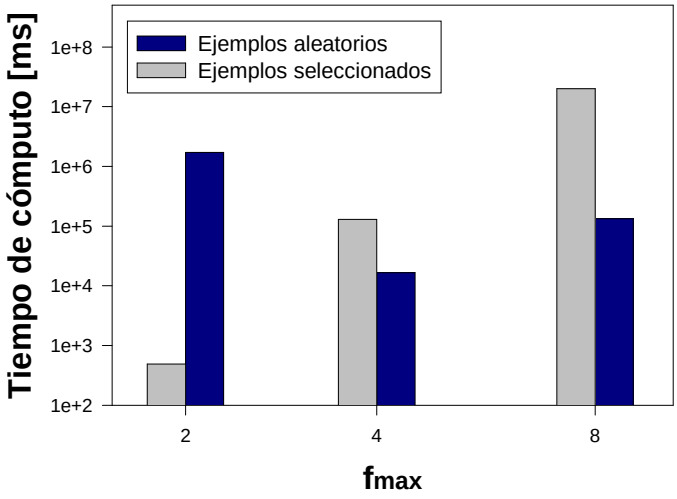


Figura 3.6: Tiempo de cómputo utilizado en el aprendizaje de la función de paridad implementada en tres diferentes arquitecturas con $f_{max} = 2, 4$ y 8 usando conjuntos de ejemplos seleccionados y aleatorios.

($N = 8$) para ejemplos aleatorios en el conjunto de aprendizaje y para ejemplos seleccionados dentro del conjunto de ejemplos que aseguran generalización total. Resultados para $f_{max} = 2, 4$ y 8 se muestran en las figuras 3.7 a,b y c respectivamente. Vemos que para $f_{max} = 2$ la influencia de la modularidad es tan fuerte que poco se obtiene con la selección de ejemplos. En cambio para el caso no modular ($f_{max} = N = 8$) observamos que la selección de ejemplos conduce a una gran mejora en la habilidad de generalización pero al costo de una baja eficiencia (ver figura 3.6).

Finalmente, analizamos el la relación entre el mínimo número de ejemplos necesarios para obtener generalización total M, con el número de sinapsis N_s. De la tabla 3.2 puede verse que para redes modulares ($m \ll N$) M escala linealmente con N_s, $M \sim \alpha(m)N_s$, tanto para pesos discretos como continuos. En el primer caso $\alpha(m)$ disminuye lentamente con m. Este comportamiento lineal ha sido observado en el número de ejemplos aleatorios necesario para obtener generalización en redes totalmente conectadas [Haykin, 1994]. Para redes no modulares ($m \sim N$) observamos un comportamiento totalmente diferente: mientras que para pesos discretos $\frac{M}{N_s} \sim ln\frac{N_s}{2N_s^{\frac{1}{2}}}$, para pesos continuos obtenemos que $M \sim 2^{N_s^{\frac{1}{2}}}$.

3.5 Algoritmo para la selección de ejemplos

En las secciones anteriores analizamos tres problemas problemas: suma de 2 números, corrimiento de bits y paridad. Si bien no existe una definición concreta o completamente aceptada de como medir la complejidad de un dado problema, es bastante claro que la función de paridad es uno de los problemas más difíciles de aprender, al menos con redes de profundidad 2. Los resultados, en estas clase de arquitecturas se muestran en la tabla 3.1.

El mínimo número de ejemplos para obtener generalización total (MNEFG) obtenido en los casos considerados puede ser tomado como una cota mínima en el número de ejemplos para el problema correspondiente implementado en una arquitectura fija, como podría ser una red de dos capas totalmente conectada, con el mismo número de neuronas intermedias; de esta forma los resultados pueden ser comparados, resultando el MNEFG un parámetro posible para clasificar la complejidad de las funciones bajo estudio.

A partir de un detallado análisis de los ejemplos seleccionados para el caso de los

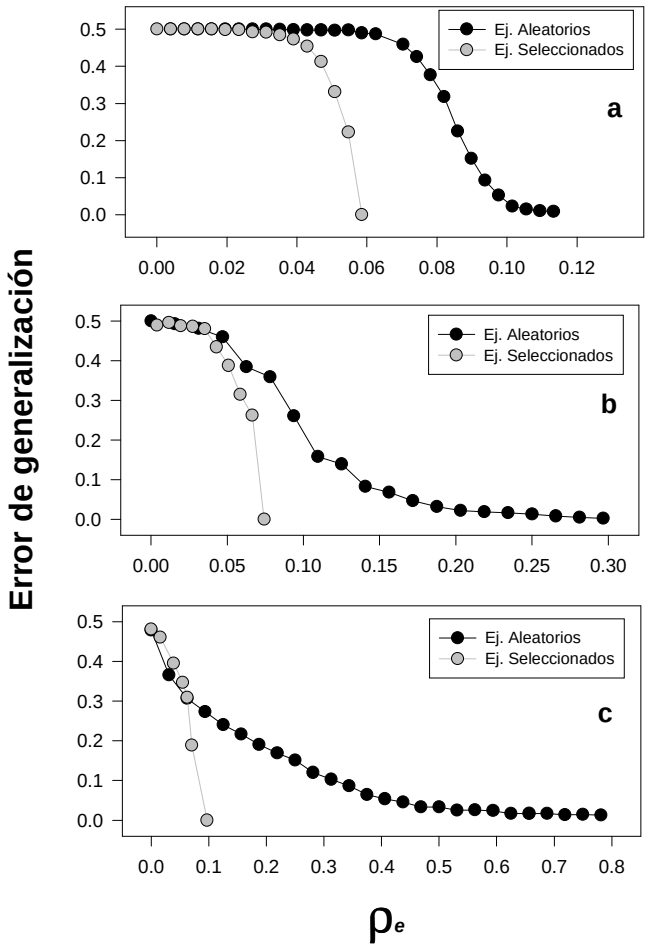


Figura 3.7: Error de generalización vs. fracción de ejemplos seleccionados y aleatorios para tres redes que implementan la función de paridad con 8 bits de entrada usando diferentes arquitecturas: a)$f_{max} = 2$, b) $f_{max} = 4$ y c) $f_{max} = 8$.

problemas de suma y de corrimiento de bits, pudimos ver que la mayoría de ellos pueden ser agrupados en pares consistentes en ejemplos muy similares difirendo en un bit de entrada y con salida distinta. Es decir, seleccionamos los pares más cercanos (considerando la distancia de Hamming entre sus entradas) ubicados a ambos lados de un límite de clasificación, esto es un hiperplano separando ejemplos con distinta respuesta. Así esto aparece como un criterio simple y general para la selección de ejemplos.

En la figura 3.8 se muestran resultados de las simulaciones numéricas realizadas calculando el error de generalización en función del número de ejemplos seleccionados con el criterio descripto anteriormente. Estos resultados son comparados con el error de generalización obtenido a través de ejemplos seleccionados aleatoriamente. En la figura 3.8a la comparación es realizada con una red para el problema de corrimiento de bits con 3 bits de entrada (arquitectura mostrada en la figura 3.2) y en la figura 3.8b se muestran los resultados correspondientes a una red que implementa la función de suma de dos números de 4 bits (arquitectura del tipo de la que se muestra en la figura 3.1). Todas las simulaciones fueron realizadas usando "recocido simulado" (simulated annealing) como algoritmo de aprendizaje y los resultados fueron promediados sobre diferentes conjuntos de ejemplos y diferentes configuraciones iniciales para los pesos sinápticos. Tamaños típicos de muestras van entre 25 y 100.

Con el fin de chequear el funcionamiento del criterio obtenido para la selección de ejemplos en una arquitectura general consideraremos a continuación una red neuronal conocida como perceptron con campos receptivos sin solapamiento (Non Overlapping Receptive Field Perceptron: NORFP) cuya estructura consta de varios perceptrones simples conectados a una única neurona de salida. En términos de complejidad de arquitectura podemos ubicarlos a mitad de camino entre los perceptrones simples y las redes con una capa oculta totalmente conectada y en este sentido han sido elegidas en el pasado como las candidatas naturales para continuar con el estudio comenzado en perceptrones simples.(Ver Copelli M. et al., 1995). Un ejemplo de estas arquitecturas con dos neuronas en la capa intermedia y $N = 6$ neuronas de entrada se muestra en la figura 3.9. Algunas propiedades de estas arquitecturas pueden encontrarse en [Hancock T. et al., 1994 y referencias citadas] y en [Priel A. et al., 1994].

De todas las funciones que están arquitecturas pueden computar [Priel, A. et al., 1994], seleccionamos una función simple de analizar y cuya computabilidad está asegurada:

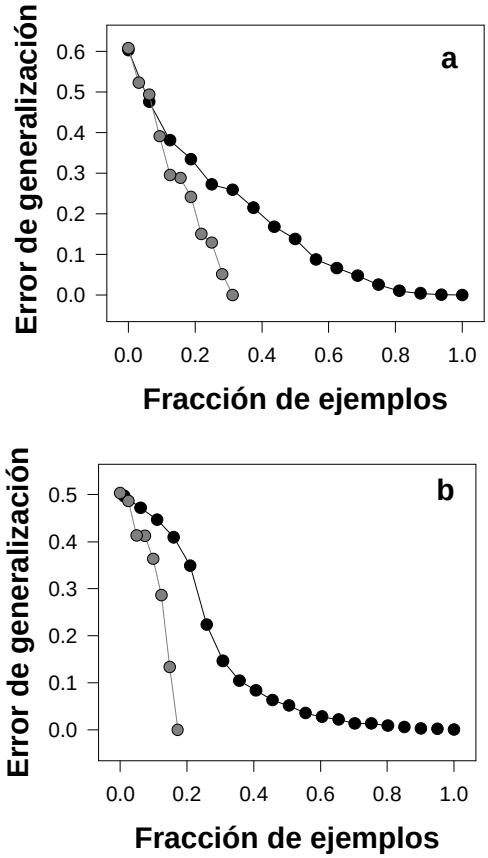


Figura 3.8: Comparación entre el error de generalización obtenido con selección de ejemplos y con ejemplos aleatorios en una red neuronal para computar la función de corrimiento de 3 bits (parte a) y para una red que computa la suma de dos números de 4 bits (parte b). El error de generalización para el caso de usar ejemplos aleatorios fue calculado asegurando la no repetición de ejemplos.

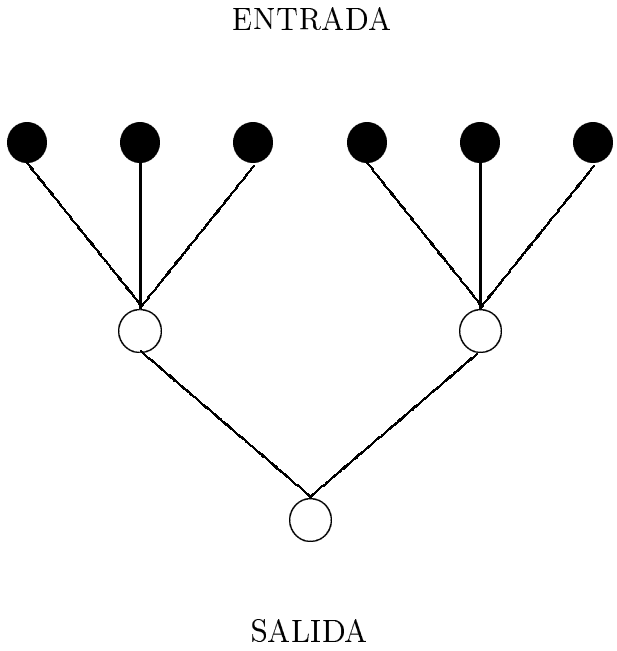


Figura 3.9: Perceptron con campo receptivo sin solapamiento (NORFP) con $N = 6$ neuronas de entrada y dos neuronas en la capa intermedia.

consideremos el caso de N par y dividamos los bits de entrada en dos grupos de $N/2$ bits; la salida de la red debe ser 1, sí y solo sí hay dos o más bits activados en cada grupo al mismo tiempo. Llamaremos a esta función F_2. Es fácil ver que está función será implementable por esta arquitectura tipo NORFP con dos neuronas intermedias, como la mostrada en la figura 3.9: el problema de determinar si dos o más bits de entrada están prendidos entre un conjunto de $N/2$ es linealmente separable y por eso puede ser implementado con una de las neuronas intermedias y por lo tanto el problema total puede resolverse con dos de estas neuronas acopladas a una neurona en la capa de salida que computa la función booleana AND, que también es linealmente separable.

De la definición de la función puede verse que el número de ejemplos que difieren en un bit de entrada y que poseen diferentes salidas escala como $\mathcal{O}(N^2\, 2^{N/2})$ para N grande, por lo que no esperamos que un conjunto de ejemplos seleccionados con este criterio mejore la capacidad de la red comparado con un conjunto aleatorio. Verificamos esta hipótesis a través de simulaciones numéricas. Sin embargo, no todos ejemplos son necesarios para asegurar generalización total. Siguiendo con el método utilizado en las secciones anteriores podemos encontrar un conjunto de ejemplos reducido con el cual

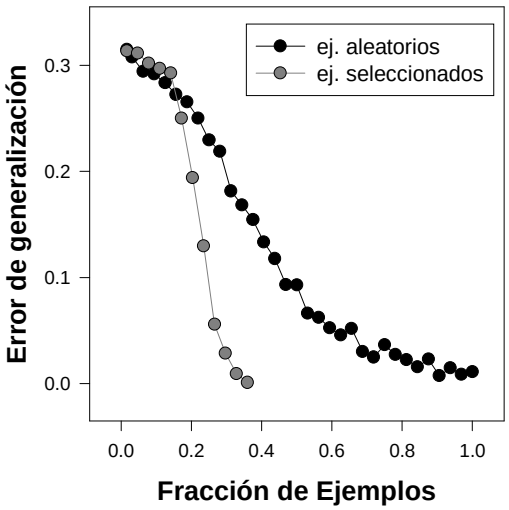


Figura 3.10: Comparación entre el error de generalización obtenido con selección de ejemplos y con ejemplos aleatorios en una red neuronal tipo NORFP con 6 neuronas de entrada (ver figura 3.9) construída para realizar el aprendizaje de la función F_2 (ver el texto para su definición).

obtener el MNEFG, el cual consistirá en los siguientes pares de ejemplos: un ejemplo con dos bits prendidos en un grupo y solo uno en el otro grupo y el ejemplo teniendo los mismo ejemplos prendidos más otro en el segundo grupo. Por ejemplo, para el caso de $N = 6$ que se muestra en la figura 3.9, un posible par es el formado por los ejemplos [110-100:0] y [110-110:1]. Un simple conteo muestra que el MNEFG escala en este caso como $\mathcal{O}(N^2)$ para N grande. En la figura 3.10 se muestra un gráfico del error de generalización obtenido a través de simulaciones en función del número de ejemplos en el conjunto de aprendizaje seleccionados con el criterio mencionado y en el mismo gráfico se compara la perfomance obtenida con ejemplos seleccionados en forma completamente aleatoria.

Este último resultado nos lleva a proponer el siguiente *criterio general* para la selección de ejemplos en redes booleanas: seleccionar al azar pares de ejemplos que difieren solamente en un bit de entrada y que tengan respuesta diferente, pero seleccionando con

mayor probabilidad aquellos pares con una menor cantidad de bits de entrada prendidos. Criterios relacionados han sido implementados conjuntamente con otros métodos de aprendizaje activo [Baum and Haussler, 1989],[Kinzel and Ruján, 1990].

Verificamos este último criterio en la red tipo NORFP de la figura 3.9 para *funciones aleatorias*. Con el fin de asegurar que las funciones analizadas sean computables, las construímos sorteando aleatoriamente los pesos sinápticos y observando que funciones implementa la red. Los resultados fueron promediados usando diferentes funciones y diferentes condiciones iniciales para las sinapsis. Tamaños de muestras van entre 100 y 1000. Los ejemplos fueron seleccionados con una probabilidad proporcional a $\propto 1/n_a^\beta$, donde n_a es el número de bits prendidos en la entrada y $\beta > 0$ es algún exponente arbitrario. Se usaron valores de β alrededor de 3, no observándose una variación importante en los resultados al variar β. Los resultados obtenidos se muestran en la figura 3.11 donde son comparados con los resultados correspondientes con ejemplos seleccionados aleatoriamente. Cabe notar que en redes booleanas generalización total siempre puede ser obtenida con un número suficiente de ejemplos. Observemos que obtuvimos una mejora en el error de generalización (figura 3.11a), pero más impresionante es el comportamiento de la probabilidad de obtener generalización total (figura 3.11b), para la cual obtenemos una probabilidad finita (del orden del 10%) con un 10% de los ejemplos, alcanzando un valor de más del 80% para la mitad de los ejemplos. Una comprobación similar fue realizada para las arquitectura usadas para los problemas de suma y corrimiento de bits mostradas en las figuras 3.1 y 3.2. Los resultados son cuantitativamente similares a los obtenidos con NORFP, especialmente en lo que concierne a la probabilidad de obtener generalización total. Observemos que el comportamiento de la generalización con respecto a la selección de ejemplos según el criterio propuesto está bastante relacionada con lo obtenido para el MNEFG. En el caso de la función de paridad con la arquitectura standard mostramos que el MNEFG escala exponencialmente con el número total de ejemplos, lo que es coincidente con que el criterio de selección de ejemplos no funcione mejor que la selección aleatoria en este caso. Este resultado nos muestra como la capacidad de generalización depende tanto de la función considerada como de la arquitectura utilizada. La estructura standard para implementar la función de paridad es la arquitectura más simple que puede implementarla y tiene una capacidad de generalización muy pobre. En cambio cuando modularizamos las arquitecturas la capacidad de generalización aumenta considerablemente.

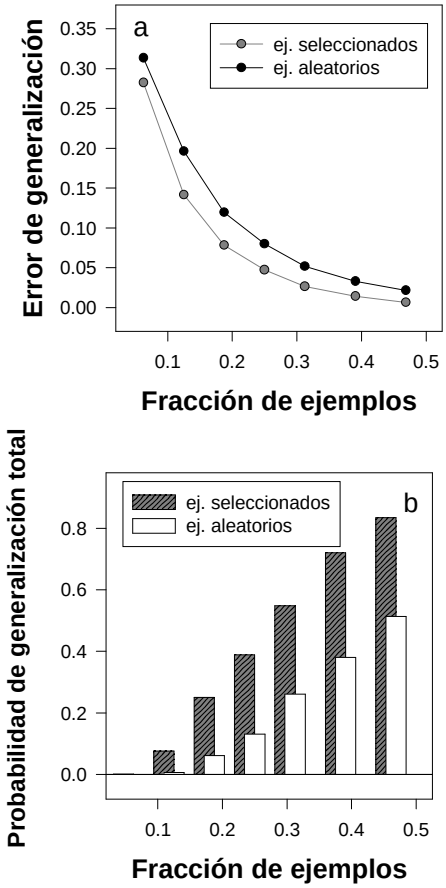


Figura 3.11: Comparación en el aprendizaje de funciones aleatorias usando selección de ejemplos versus ejemplos aleatorios en una red tipo NORFP con $N = 6$ (ver figura 3.9). a) Error de generalización promedio *vs.* fracción del número total de ejemplos en el conjunto de aprendizaje b) Probabilidad de obtener generalización total (error de generalización cero).

	Corrimiento de bits			F_2			Paridad		
	Selecc.	Aleatorios		Selecc.	Aleatorios		Selecc.	Aleatorios	
Fracción de Ejemplos	0.28	0.28	0.81	0.4	0.4	1	0.43	0.43	0.68
Error de Generalización	0.0	0.26	0.02	0.0	0.15	0.02	0.0	0.4	0.0
Tiempo de CPU [sec.]	7.0	0.2	1.1	7.7	9.0	93.3	160	8.8	332

Tabla 3.3: Tiempo promedio de cómputo (tiempo de CPU) utilizado por los diferentes métodos de selección en el aprendizaje de las funciones de corrimiento de bits, F_2 y función de paridad con diferentes fracciones de ejemplos en el conjunto de aprendizaje. El error promedio de generalización se muestra para su comparación.

También calculamos el tiempo de cómputo (tiempo de CPU en una computadora personal Pentium II a 300 Mhz) necesario para realizar el proceso de aprendizaje para tres funciones distintas: *i*) corrimiento de bits con $N = 5$ usando la arquitectura de la figura 3.2; *ii*) función F_2 function ($N = 6$) en una arquitectura tipo NORFP (ver figura 3.9) y *iii*) función de paridad ($N = 4$) en la arquitectura de la figura 3.3.Los resultados obtenidos se muestran en la tabla 3.3. En todos los casos calculamos el tiempo de cómputo necesario para obtener error de generalización cero con ejemplos seleccionados. A fin de poder comparar la perfomance con el aprendizaje usando ejemplos aleatorios calculamos dos tiempos de cómputo en este caso: primero calculamos el tiempo de aprendizaje usando la misma fracción de ejemplos que en el caso seleccionado (lo que da un error de generalización distinto de cero) y segundo calculamos el tiempo necesario para obtener un error de generalización cercano a cero (lo que involucra un conjunto de ejemplos bastante mayor que en el caso seleccionado).

Observamos que en el peor caso (problema de corrimiento de bits) el tiempo promedio necesario para obtener error de generalización cero con ejemplos seleccionados es cerca de siete veces mayor que el correspondiente usando ejemplos aleatorios. Sin embargo, vale la pena notar que *generalización total* es siempre alcanzada (i.e., con probabilidad uno) usando solamente una fracción de los ejemplos, mientras que en el segundo caso aún usando un gran número de ejemplos la probabilidad de aprendizaje es menor que uno. De esta forma, obtenemos una mayor capacidad de generalización aunque a un mayor costo computacional. En los otros dos casos el tiempo de cómputo es menor para el caso de ejemplos seleccionados que con ejemplos al azar.

3.6 Computabilidad de redes modulares y complejidad de funciones

El estudio de la capacidad de cómputo de redes neuronales se basa en conocer la cantidad de funciones que una arquitectura puede computar tratando de conocer cuáles o qué clase de funciones pueden ser implementadas.

Una arquitectura de una red neuronal es capaz de computar diferentes funciones en función de los valores que posea de pesos sinápticos y umbrales. Este conjunto de funciones es en principio restringido sobre todo para redes con un número polinomial de neuronas (tanto para el caso de valores de sinapsis restringidas a valores discretos como continuos) de manera que existen una gran cantidad de funciones que no pueden implementarse con una dada arquitectura. Para el caso de redes booleanas el número total de funciones posibles para una red con N entradas binarias es de 2^{2^N}. 2^N es el número de ejemplos posibles para una arquitectura con N entradas booleanas y la especificación del valor de salida (respuesta de la red) para cada uno de los ejemplos determina una dada función. El número total de funciones posibles 2^{2^N} lo obtenemos de considerar las posibles maneras en las que se pueden otorgar los valores 0 y 1 a los 2^N diferentes ejemplos. Una medida útil relacionada con la computabilidad de las arquitecturas es la *capacidad* o dimensión VC de una arquitectura (ver capítulo 1 para su definición) y si bien no nos da una medida precisa del número de funciones que una arquitectura puede computar, muestra una relación de esta propiedad entre diferentes arquitecturas. Otra cantidad utilizada, más directamente relacionada con la computabilidad es la denominada entropía de información de una arquitectura [Shannon, 1948], [Denker et al., 1987], [Haykin, 1995], estando relacionada con la variabilidad de funciones que una arquitectura puede implementar y el volumen en el espacio de configuraciones que cada función ocupa. Está definida como :

$$S = -\sum_i p_i \log_2 p_i, \tag{3.34}$$

donde p_i es el volumen ocupado por una dada función i en el espacio de configuraciones, es decir, es la fracción del espacio de pesos sinápticos que implementan esa función. Para el caso de pesos discretos p_i es la fracción de configuraciones que implementan la función deseada en relación al número total de configuraciones de sinapsis (Ω), que es lo mismo que la probabilidad de que una cierta configuración aleatoria de sinapsis de la red implemente la función i. Puede verse que esta entropía es mayor cuanto mayor es el número de

funciones implementadas, estando su valor acotado entre 0 y $\log_2 \Omega$. El valor $\log_2 \Omega$ se obtiene en el caso de que cada configuración implemente una función distinta, mientras que el valor igual a 0 es obtenido si todas las configuraciones implementan una única función. Para el caso de pesos continuos también puede calcularse la entropía y el volumen que ocupa cada función [Sprinkhuizen-kuyper and Boers, 1986] aunque puede resultar más fácil reemplazar los pesos continuos por pesos discretos sin alterar la computabilidad de las redes y aplicar el análisis anterior. Puede demostrarse que la computabilidad de redes conteniendo neuronas que implementan una función lineal con umbral, como es en los casos tratados en este trabajo, no se modifica si reemplazamos los valores continuos por un conjunto finito de valores discretos [Parberry, 1996], [Schmitt, 1994]. Desgraciadamente el cómputo de entropías requiere de una gran cantidad de cálculos dado que es necesario conocer la probabilidad de obtener cada función y sólo es posible calcularla exactamente en muy pocos casos [Sprinkhuizen-Kuyper and Boers, 1996]. Por otra parte, dado que el número de funciones booleanas crece como 2^{2^N}, sólo es posible obtener resultados numéricos en redes pequeñas. A continuación analizamos la capacidad de cómputo de algunas redes similares a las utilizadas a lo largo de este trabajo intentando comparar la capacidad de las redes modulares con campos receptivos locales con la de redes monolíticas totalmente conectadas. Comparamos la capacidad de cómputo de tres arquitecturas diferentes con $N = 4$ neuronas de entrada y una única neurona de salida. Las arquitecturas que consideraremos serán:

- Una red totalmente conectada (monolítica) con una sola capa oculta con 4 neuronas y un total de $N^2 + N = 20$ pesos sinápticos (ver figura 2.8).
- Una red del tipo modular como la usada para implementar la función de paridad en el capítulo 3. La red posee tres capas ocultas con 4, 2 y 2 neuronas respectivamente, y un total de 18 sinapsis (ver figura 2.9).
- Una red tipo NORFP con campos receptivos locales con una sola capa intermedia conteniendo 4 neuronas y un total de 6 pesos sinápticos (similar a la figura 3.9).

Se usaron pesos sinápticos con valores discretos restringidos a 1, -1, mientras que en el caso de los umbrales, los correspondientes a las neuronas de capas internas fueron fijados a valores iguales a los usados en las diferentes implementaciones (secciones 3.2, 3.3 y 3.4)

Arquitectura	# de config.	# de funciones	entropía	neuronas
Perceptron monolítico	2^{21}	5280	0.339	5
Arquitectura modular	2^{19}	520	0.1534	11
Perceptron tipo NORFP	2^{7}	68	0.657	3

Tabla 3.4: Número de funciones computables, entropía y algunas características de 4 diferentes arquitecturas.

pero se permitió al umbral de la neurona de salida tomar solamente los valores $\{0.5$ y $-0.5\}$. Para estas tres arquitecturas calculamos por enumeración exhaustiva la cantidad de funciones que cada arquitectura puede computar y la entropía relativa de cada arquitectura obtenida al dividir el valor de la entropía *absoluta* definida por la ec. 3.34 por su valor máximo $\log_2 \Omega$. Los resultados obtenidos se muestran el la tabla 3.4. Vemos que la entropía relativa del NORFP es considerablemente mayor a la de la red monolítica, en tanto que la de la red modular es menor. Esto indica un mayor aprovechamiento en la implementación de diferentes funciones de las diferentes configuraciones sinápticas disponibles. Respecto del número total de funciones computadas, vemos que el NORFP computa muchas menos funciones que las otras redes. Esto es resultado del número pequeño de sinápsis disponibles. En este sentido la más ineficiente resulta ser la arquitectura modular. No obstante, estos resultados son extremeadamente dependientes del tamaño pequeño de las redes consideradas y debemos tener cuidado en extrapolarlos a redes grandes. Por ejemplo, para $N = 4$ vemos que la arquitectura monolítica dispone de un número de configuraciones sinápticas superior al número de posibles funciones booleanas 2^{16} (no obstante lo cual tampoco es capaz de computarlas a todas). Esto no ocurre para N grande, ya que el número de sinapsis crece polinomialmente con N, y por lo tanto se espera que la computabilidad sea considerablemente menor. Por otra parte, la entropía nos da una medida exclusivamente acerca del número de funciones computables, sin ninguna información acerca del tipo de funciones que computa la red. De hecho una gran cantidad de funciones booleanas son altamente triviales y por lo tanto sin interés práctico. Pensemos, por ejemplo, en todas aquellas funciones que asignan el valor 1 a una única entrada y cero a las $2^N - 1$ restantes. Siguiendo con esta línea de razonamiento podemos considerar que una función será más compleja (y por lo tanto de mayor interés)

cuanto mayor sea el porcentaje de *pares de entradas* a las cuales asigne valores diferentes. Este criterio de complejidad de funciones es consistente con el utilizado en el algoritmo de selección de ejemplos introducido en la sección anterior. En este sentido las funciones mas complejas serán aquellas que asignan el valor 1 a la mitad de sus valores de entrada (2^{N-1}) y cero a la otra mitad (la función de paridad se encuentra en esta última categoría). A fin de analizar esta cuestión calculamos el número de funciones computadas con i bits de salida prendidos y el resto cero. Dada la simetría de la función de activación (ec. 2.1) con la elección de pesos y umbrales utilizados este número será igual al número de funciones computables con i bits apagados (0) y $2^{N-1} - i$ bits prendidos (1). En la figura 3.12 mostramos la fracción entre esta cantidad y el número total de funciones computables para cada una de las tres arquitecturas analizadas. A los fines de analizar estos resultados es importante tomar en cuenta que el número de funciones booleanas *existentes* con i bits prendidos tiene una fuerte dependencia con i, siendo igual $C(2^N, i) = \frac{(2^N)!}{(2^N-i)!\,i!}$. Así en la figura 3.13 mostramos la fracción entre el número de funciones *computadas* y el número de funciones *posibles* con i bits prendidos. Para el caso del NORFP vemos que, si bien este computa una cantidad relativamente alta de funciones, la mayoría son funciones relativamente simples. En los otros dos casos vemos que la distribución de funciones es semejante para ambas arquitecturas, computando una cantidad relativamente alta de funciones complejas (notar la escala logarítmica en los gráficos).

3.7 Análisis y Conclusiones del capítulo

Analizamos en este capítulo la capacidad de generalización en el aprendizaje de tres funciones booleanas: suma de dos números, corrimiento de bits y paridad, implementadas en redes neuronales de tipo feed-forward. Basamos nuestro análisis en el cálculo analítico del mínimo número de ejemplos para obtener generalización total (MNEFG) y de su comportamiento (propiedades de escala) en relación con el número de bits de entrada N y con el número de sinapsis presentes en la red. Obtuvimos los valores del MNEFG a través de un análisis detallado, para redes pequeñas, de las inecuaciones en términos de los pesos sinápticos de la red que pueden escribirse para los posibles ejemplos y seleccionando entre ellas un conjunto mínimo, a partir del cual pudieran ser derivadas el resto de las inecuaciones. El comportamiento de MNEFG para valores grandes de N se obtuvo de

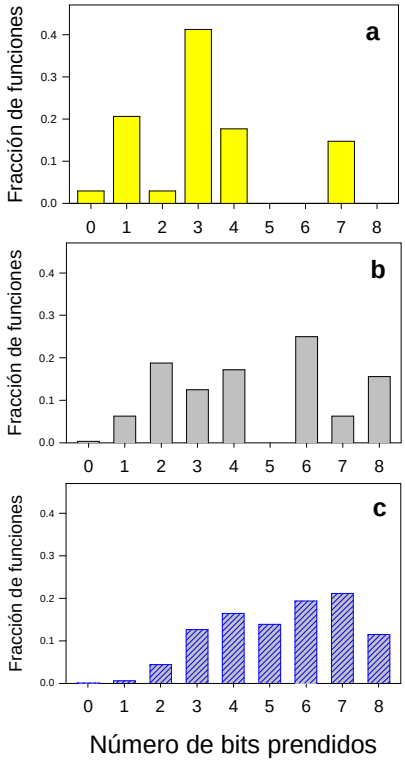


Figura 3.12: Fracción de funciones implementadas en relación al número total de funciones *implementadas* por cada arquitectura en función del número de bits prendidos ó bits apagados en tres arquitecturas diferentes con 4 neuronas de entrada a) NORFP, b) Modular, c) Totalmente conexa.

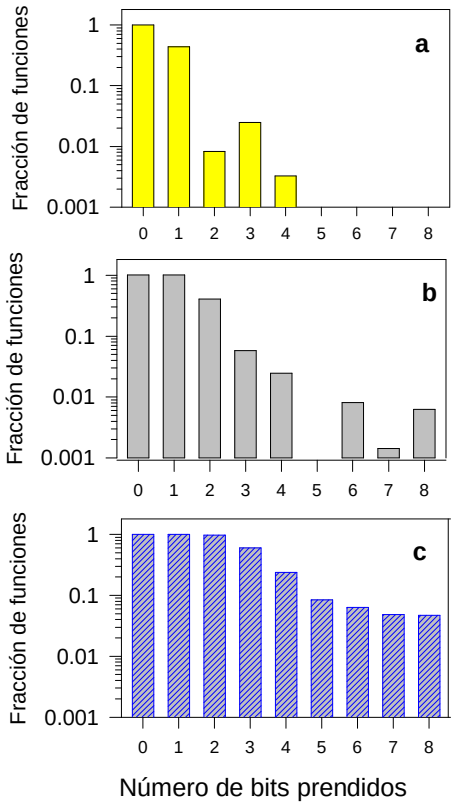


Figura 3.13: Fracción de funciones implementadas en relación al número total de *funciones booleanas existentes* con i bits prendidos (o apagados) en función de i para tres arquitecturas diferentes con 4 neuronas de entrada a) NORFP, b) Modular, c) Totalmente conexa.

la generalización de los resultados para redes de tamaño arbitrario. El MNEFG aparece como un parámetro de complejidad interesante, consistente con la noción intuitiva de que a mayor complejidad de problema, mayor la dificultad de su aprendizaje. Nuestros resultados muestran como la interrelación **función-arquitectura** puede ser estudiada con este parámetro. En el caso de redes con profundidad dos, mínimas en cuanto al número de capas posibles, para funciones relativamente simples como la suma de dos números y corrimiento de bits el MNEFG escala polinomialmente con el número de bits de entrada, mientras que para el caso de la función de paridad escala exponencialmente, mostrando la complejidad de esta función. Este último resultado es muy interesante si consideramos que la función de paridad con la arquitectura considerada es una de las redes más usada para probar algoritmos de aprendizaje. También mostramos como este comportamiento no deseado (falta de generalización) puede ser superado si consideramos estructuras con un mayor número de capas, con campos receptivos locales. En efecto, mostramos como en una familia de arquitecturas modulares la capacidad de generalización es mejorada sustancialmente a medida que el grado de modularidad de la red es incrementado, tanto para el caso de un conjunto de entrenamiento con ejemplos seleccionados como con ejemplos aleatorios. Más aún encontramos evidencia de la existencia de una transición de fase de memorización a generalización total para el caso de modularidad extrema. Con respecto a la computabilidad de estas redes modulares mostramos, que si bien la misma se reduce considerablemente comparativamente con la de las redes monolíticas, la distribución de funciones computables de acuerdo a su complejidad es semejante en ambas redes. Una observación minuciosa del conjunto de ejemplos que determina el MNEFG en los diferentes casos nos llevó a proponer un criterio muy simple para poder seleccionar ejemplos que conlleven a una mejora en las propiedades de generalización de la red: seleccionar ejemplos con entradas muy similares (una distancia de Hamming lo menor posible entre sus bits de entrada) pero que posean una respuesta correcta con valores opuestos. Los ejemplos son elegidos en esta forma y privilegiando a los ejemplos con menor número de bits de entrada prendidos. Este criterio, independiente de la arquitectura, no siempre asegura obtener generalización total, como ocurre con la función de paridad implementada en una red de profundidad dos con pesos continuos. Aquí es donde entonces comienza a jugar el rol de la arquitectura y su relación con la función elegida. Nuestros resultados sugieren que el criterio funciona bien cuando MNEFG escala polinomialmente con N, y esto

parece ocurrir solamente (en el caso de pesos continuos) cuando tenemos en la red campos receptivos locales (no conectividad total). A partir del criterio obtenido se construyó un algoritmo para seleccionar ejemplos, el cual fue probado con éxito en funciones aleatorias implementadas en arquitecturas con campo receptivos locales tipo NORFP (Ver sección 3.5).

Capítulo 4

Conclusiones

En la primera parte de esta tesis (capítulo 2) hemos construído arquitecturas de redes neuronales que implementan diferentes funciones. El problema de construcción de arquitecturas es un problema central dentro del marco de las redes neuronales.

En primer lugar, obtuvimos soluciones *exactas* para arquitecturas que implementan diferentes operaciones aritméticas con números enteros de tamaño arbitrario (la extensión de los resultados obtenidos a números reales es directa usando la representación de punto flotante), a saber: suma de N números de p bits, multiplicación de dos números y problema de corrimiento de bits ("bit-shifting"). Estas funciones son usuales en los microprocesadores de computadoras secuenciales, por lo que las arquitecturas diseñadas pueden ser de gran utilidad a la hora de implementar un circuito análogo pero masivamente paralelo, que en estos momentos encuentra limitaciones tecnólogicas principalmente respecto a la cantidad de conexiones por neurona que involucran (ver tablas 2.2, 2.3 y 2.5). Sin embargo, en aplicaciones concretas que necesitan una sobreespecialización en ciertas funciones, como por ejemplo en tareas de compresión de imágenes o cálculos de transformación de coordenadas que involucran productos y sumas matriciales, las arquitecturas diseñadas pueden implementarse en circuitos programables. En particular la función de bit-shifting ha sido utilizada para modelar circuitos de sistemas visuales de organismos vivos. [Anderson and Van Essen, 1987]

Por otra parte, las arquitecturas diseñadas sirven como plataforma de prueba y análisis para los algoritmos de aprendizaje y para estudiar cuestiones relacionadas con sus propiedades de aprendizaje y generalización tal como se hizo en esta tesis en el capítulo 3.

Notemos que las arquitecturas construídas para estos tres problemas son óptimas en

cuanto al número de capas, dado que las funciones de suma y de corrimiento de bit son linealmente no separables y por lo tanto arquitecturas con una sola capa intermedia constituyen arquitecturas mínimas en este sentido. Para el caso de la red de producto es posible demostrar, si bien el análisis es complejo, que una arquitectura conteniendo dos capas ocultas es la estructura mínima necesaria con un número polinomial de neuronas que puede computar esta función [Siu and Roychowdhury, 1994].

La profundidad mínima de una red para computar una dada función se relaciona con la complejidad de esta última. De hecho la profundidad mínima puede ser usada como una medida de complejidad de una función [Hofmeister et al., 1991], [Siu et al., 1993], [Paterson et al., 1990].

Finalmente, en relación al número de neuronas y sinapsis, las redes aquí presentadas se comparan favorablemente en relación a diseños pre-existentes para las mismas funciones.

También en el capítulo 2 se construyó una familia de arquitecturas para computar la función de paridad, la cual constiutye una de las funciones más estudiadas por razones de complejidad e históricas; recordemos que en el caso de dos bits de entrada la función de paridad es equivalente a la función booleana XOR. Las arquitecturas obtenidas en la sección 2.6, a partir de la arquitectura clásica completamente conexa para computar este problema, tienen como característica principal la de ser modulares y el hecho de que esta modularidad es controlada a través del número de conexiones por neurona (f_{max}, "fan-in max"). Notemos que en estas arquitecturas tanto el número de neuronas como el número de sinapsis crece linealmente con el número de neuronas en la capa de entrada para una modularidad fija (f_{max}=cte); para el caso en que el número de entradas está fijo el número de sinapsis crece linealmente con el f_{max}, o sea que a mayor modularidad (menor f_{max}) el número de conexiones es menor, en tanto que el número de neuronas en la red es casi independiente del f_{max}.

Las redes modulares han demostrado ser sumamente útiles y eficientes en diversas aplicaciones como reconocimiento de voz, predicción de series temporales, aproximación de funciones, etc. El gran éxito de las redes modulares parece basarse en su capacidad para captar tanto detalles *globales* como *locales*, pudiendo los diferentes módulos dedicarse a distintas tareas haciendo más fácil su implementación y más veloz su aprendizaje [Jacobs et al., 1991]. La interpretación de los datos representados por una red es una tarea muy difícil y en ese sentido las redes modulares aparecen como una solución al permitir

descomponer una tarea compleja en múltiples tareas más simples [Rueckl et al., 1989].

Desde el punto de vista biológico la modularidad aparece como un concepto de fundamental importancia tanto por cuestiones de tipo estructural, cantidad de espacio disponible, como por cuestiones relacionadas con el aprendizaje de diferentes tareas, permitiendo una mayor optimización de algunos módulos y también una mejor codificación. La existencia de estructuras modulares parece evidente en ciertas áreas de la corteza visual de organismos vivos [Haykin, 1994], [Van Essen, 1985].

En el capítulo 3 analizamos las propiedades de generalización de algunas de las arquitecturas mencionadas anteriormente y de algunas otras como por ejemplo las de una red para sumar 2 números [Cannas, 1995], calculando el número mínimo de ejemplos necesarios para obtener generalización total (MNEFG). Este parámetro introducido en [Franco and Cannas, 2000a] es obtenido a partir de un estudio analítico de las ecuaciones que impone el conjunto de ejemplos de redes pequeñas. A través de un método inductivo obtenemos el comportamiento del MNEFG para redes de *tamaño arbitrario*, constituyendo uno de los pocos resultados analíticos que existen en este sentido.

En el caso de redes con campos receptivos locales (no totalmente conectadas) el MNEFG escala polinomialmente con el número de entradas N de la red, mientras que en el caso de una red totalmente conexa, implementando la función de paridad en una arquitectura con una única capa oculta y sinapsis continuas, el MNEFG escala en forma exponencial (ver tabla 3.2). Este último resultado demuestra una mala capacidad de generalización en esta red atribuíble a lo inadecuado de una arquitectura de este tipo para implementar una función tan compleja como la función de paridad. El uso de sinapsis discretas modifica este comportamiento, haciendo que el MNEFG escale polinomialmente con N. No obstante, simulaciones numéricas en redes pequeñas con ejemplos tomados al azar muestran que la red continúa siendo incapaz de obtener generalización perfecta. Este efecto desaparece con el uso de redes modulares, donde el MNEFG escala como $N \log N$, lo cual refuerza las expectativas existentes sobre las redes modulares. Mas aún las simulaciones numéricas con ejemplos tomados al azar muestran que cualquier grado de modularización permite obtener generalización perfecta con sólo una fracción del total de los ejemplos. Este efecto se vuelve más marcado al aumentar la modularidad, llegando a manifestarse en una transición de fase de memorización a generalización total en el caso de modularidad extrema ($f_{max} = 2$). Esta transición, posiblemente discontinua, ocurre

cuando el tamaño del conjunto de entrenamiento es aproximadamente igual al MNEFG. En contraposición, mostramos que los tiempos de cómputo se vuelven elevados en el caso de modularidad extrema. Nuestros resultados sugieren que los tiempos de cómputo encuentran un valor óptimo para valores intermedios de modularización. En el caso de ejemplos seleccionados dentro de un subconjunto de máxima información (esto es, entre aquellos de los cuales se derivó el MNEFG) los tiempos de cómputo son mínimos en el caso de modularidad extrema.

En función de los resultados obtenidos surge la pregunta de conocer la cantidad de funciones que las arquitecturas modulares pueden computar; este tema fue analizado en la subsección 3.5.1 mostrándose que si bien hay una gran reducción debido al incremento de la modularidad esto está relacionado con el hecho de que cuanto más modulares las arquitecturas menor es la cantidad de pesos sinápticos que poseen y obviamente su capacidad de cómputo se ve reducida (ver tabla 3.4). En compensación, la distribución de funciones computables por las redes modulares de acuerdo con su complejidad es semejante a la de las redes monolíticas. En otras palabras, si bien las redes modulares computan menos funciones que las no-modulares, el porcentaje de funciones complejas es semejante. Por otra parte, cabe mencionar que la estructura de los módulos aquí utilizada está determinada por simetrías globales de la función paridad. El mismo tipo de criterio puede aplicarse en el diseño de los módulos para computar funciones con otras simetrías, donde podemos esperar que este tipo de estructuras presenten una alta computabilidad dentro de subconjuntos de funciones que comparten las misma simetrías. A partir del análisis de los conjuntos de ejemplos que constituyen el MNEFG pudo elaborarse un criterio para seleccionar ejemplos en problemas generales. El criterio se basa en tomar para el conjunto de entrenamiento pares de ejemplos similares, es decir que difieren solamente en pocos bits de salida (distancia de Hamming pequeña), pero con respuestas diferentes y privilegiando a los ejemplos que contienen una menor cantidad de bits prendidos. Con esta idea desarrollamos un algoritmo de selección de ejemplos que fue probado con éxito utilizando funciones aleatorias en diferentes arquitecturas con campos receptivos locales, obteniéndose una mejora sustancial respecto de las propiedades de aprendizaje observadas utilizando conjunto de ejemplos seleccionados aleatoriamente. Uno de los puntos centrales de esta tesis ha sido la relación entre las propiedades de generalización de perceptrones y la complejidad de las funciones a ser aprendidas. La complejidad de una función es un

concepto difuso, de difícil definición y que va más allá del área de las redes neuronales. Diferentes medidas de complejidad han sido propuestas en diferentes áreas [Parberry, 1996], [Denker et al., 1987], [Parisi, 1992]. Es probable que no sea posible encontrar una única cantidad que cuantifique de manera consistente la complejidad de cualquier función, siendo necesario combinar diferentes cantidades de acuerdo a cada contexto específico. Creemos que los resultados presentados en este trabajo constituyen un aporte importante al desarrollo de medidas de complejidad en el contexto de aprendizaje en perceptrones, como discutiremos a continuación. Una primera medida de complejidad de una función puede ser la arquitectura mínima de red necesaria para computarla. Tomemos por ejemplo un perceptron de N entradas con una única capa oculta conteniendo h neuronas completamente conectadas a la capa de entrada. Para $h = 1$ esta arquitectura sólo puede computar funciones linealmente separables (consideradas simples). A medida que aumentamos h esta arquitectura computará un número creciente de funciones hasta computar las 2^{2^N} posibles cuando h sea de orden exponencial en N. De esta manera el h mínimo necesario para implementar una función nos dará una primera idea de su complejidad. Ahora bien, dada una arquitectura fija y un conjunto de funciones podemos comparar su complejidad considerando la capacidad de generalización de la red en el aprendizaje de cada una de ellas. Funciones más complejas requieren de un número mayor de ejemplos para ser aprendidas. Hemos mostrado en este trabajo la existencia de conjuntos mínimos de ejemplos que contienen toda la información acerca de una función. Estos ejemplos pueden agruparse en pares de ejemplos con respuestas diferentes cuyas entradas presentan una distancia de Hamming mínima. La cardinalidad de estos conjuntos es el MNEFG, el cual surge entonces como una medida posible de complejidad, a través de su dependencia asintótica con el número de entradas de la red N. El aprendizaje de los ejemplos contenidos en uno de estos conjuntos nos garantiza la generalización total por medio de *cualquier algoritmo de aprendizaje* que se utilice y de esta manera nos está dando una idea de la cantidad de información total que hay que suministrarle a una dada arquitectura para aprender la función. Una medida usual de la capacidad de aprendizaje de una arquitectura es la dimensión VC, la cual es independiente de la función a implementar. El MNEFG depende tanto de la arquitectura como de la función. Dado que la generalización sólo es posible cuando el número de ejemplos supera la dimensión VC el MNEFG calculado para cualquier función computable por la red constituye una cota

superior para la primera. Vemos así que en un proceso de aprendizaje, al ir aumentando el tamaño del conjunto de entrenamiento la generalización comienza cuando se alcanza la dimensión VC y termina cuando se alcanza el MNEFG. Por otra parte, la dimensión VC es en general sumamente díficil de calcular en cualquier arquitectura de tamaño y complejidad mediana. El cálculo de las propiedades asintóticas del MNEFG es más sencillo que el de la dimensión VC [Mitchinson and Durbin, 1989], [Kowalczyk, 1997] a través del método aquí presentado, si bien requiere de información detallada acerca de la función utilizada. Esto lo convierte potencialmente en un instrumento teórico valioso para el estudio de propiedades de generalización. Una extensión interesante del presente trabajo es la siguiente: es sabido que el uso de campos receptivos locales es fundamental para obtener una buena capacidad de generalización. En particular hemos mostrado como las redes modulares pueden presentar una alta capacidad de generalización (bajo MNEFG) en el aprendizaje de funciones complejas. El problema que se presenta en situaciones prácticas, es que en general no se dispone de información suficientemente detallada acerca de la función a aprender que permita el diseño de una red modular eficiente. Una posibilidad interesante es el desarrollo de algoritmos de construcción de arquitecturas (tanto de crecimiento como de "pruning") en los cuales las unidades básicas añadidas o eliminadas durante el aprendizaje sean distintos tipos de módulos, en lugar de neuronas o sinapsis aisladas. En estos algoritmos los conjuntos de entrenamiento podrían escogerse de acuerdo con el criterio de selección aquí introducido, lo cual tendería a optimizar la capacidad de generalización para cada tipo de módulo, a la vez de disminuir los tiempos de cómputo.

Capítulo 5

Referencias

- Alon, N., and Bruck, J. (1991). Explicit constructions of depth-2 majority circuits for comparison and addition, *SIAM J. Discrete Math.* **7**, pp. 1-8.
- Amit, D.J., Gutfreund, H., and Sompolinsky, H. (1985). *Phys. Rev. A* **32**, pp. 1007.
- Amit, D.J.(1989). *Modelling Brain Function: The world of attractor neural networks*, Cambridge University Press, New York.
- Anderson, C.H. and Van Essen, D.C. (1987). Shifter circuits: A computational strategy for dynamic aspects of visual processing, *Proc. Natl. Acad. Sci. USA* **84**, pp. 6297-6301.
- Anderson, J.A. (1995). *An introduction to Neural Networks*, MIT Press, Cambridge, MA.
- Baum, E. B. and Haussler, D. (1989). What Net Size Gives Valid Generalization?, *Neural Computation* **1**, pp. 151-160.
- Baum, E. B. (1991). Neural Net Algorithms that learn in Polynomial Time from Examples and Queries, *IEEE Transactions on Neural Networks* **2**, 1, 5.
- Bennani, Y. (1994). Multi-expert and hybrid connectionist approach for pattern recognition: speaker identification task, *Intl. Journal of Neural Systems* **5**, 3, pp. 207-216.

- Binder, K., and Heerman, D.W. (1988) *Monte Carlo Simulation in Statistical Mechanics.* Springer-Verlag, Berlin.
- Boers E. J. W., Kuiper H., Happel B. L. M. and Sprinkhuizen-Kuyper I. G. (1993). Designing Modular Artificial Neural Networks in Wijshoff H. A. (ed.), Proceedings of Computing Science in the Netherlands (CSN'93), Stichting Matematisch Centrum, Amsterdam, pp. 87.
- Budinich, M., and Milotti, E. (1992). Properties of feedforward neural networks, *J. Phys. A: Math. and Gen.* **25**, pp. 1903-1914.
- Cannas, S.A. (1995). Arithmetic Perceptrons, *Neural Computation*, **7**, 1, pp. 173.
- Cannas, S.A. (2000). Redes Neuronales, Física, Biología o Computación ?. Notas de Computación, FaMAF-UNC. (en prensa).
- Carnevali, P. and Patarnello, S. (1987) Exhaustive Thermodynamical Analysis of Boolean Learning Networks, *Europhysics Letters* **4**, 10, pp. 1199-1204.
- Cohn, D., Atlas, L. and Ladner, R. (1994). Improving Generalization with Active Learning, *Machine Learning* **15**, 2, pp. 201.
- Cohn, D. (1996). Neural Network Exploration using Optimal Experiment Design, *Neural Networks*, **9**, 6, pp. 1071.
- Copelli, M. and Caticha, N. (1995). On-line learning in the committee machine, *J. Phys. A: Math. and Gen.* **28**, pp. 1615.
- Cover, T.M. (1965). Geometrical and Statistical Properties of Systems of Linear Inequalities with Applications to Pattern Recognition. *IEEE Trans. Elect. Comp.* **14**, 326-334.
- Denker, J., Schwartz, D., Wittner, B., Solla, S., Howard,R., Jackel, L., and Hopfield, J. (1987). Large Automatic Learning, Rule Extraction, and Generalization, *Complex Systems* **1**, pp. 877-922.
- Derrida, B., Gardner, E., and Zippelius, A. (1987). An exactly solvable asymmetric neural network model. *Europhysics letters* **4**, 2, pp. 167-173.

- Franco, L. (1995). Trabajo Especial de la Licenciatura en física. FaMAF-UNC.
- Franco, L., and Cannas, S.A.(1997) Energy Landscape and learning on a neural network for the sum problem. Proceedings of the Second International Workshop on (semi)numerical techniques in polynomial equation solving Tera'97. Trabajos de Matemática, serie B. FaMAF-UNC.
- Franco, L., and Cannas, S.A. (1998). Solving arithmetic problems using feed-forward neural networks, *Neurocomputing* **18**, pp. 61-79.
- Franco, L., and Cannas, S.A. (2000a). Generalization and Selection of Examples in Feed Forward Neural Networks, *Neural Computation.* **12**, 10.
- Franco, L., and Cannas, S.A. (2000b). Generalization properties of modular networks implementing the Parity function. (sometido para publicación).
- Hajnal, A., Maass, W., Pudlak, P., Szegedy M., and Turan, G.(1987). Threshold Circuits of bounded depth, *IEEE Symp. Found. Comp. Sci.* **28**, pp. 99-110.
- Hancock, T.R., Golea, M., and Marchand, M. (1994). Learning Nonoverlapping Perceptron Networks From Examples and Membership Queries, *Machine Learning* **16**, pp. 161.
- Haykin, S. (1994). *Neural Networks: A comprehensive foundation.* McMillan.
- Hebb, D.O. (1949). *The Organization of Behaviour: A Neurophysiological Theory*, Wiley, New York.
- Hecht-Nielsen, Robert. (1989) *Neurocomputing*, Addison Wesley, MA.
- Hertz, J., Krogh, A., and R. Palmer. (1991). *Introduction to the Theory of Neural Computation*, Addison Wesley, Santa fe Institute.
- Hofmeister, T., Hohberg,W., and Kohling, S. (1991). Some notes on Threshold Circuits and multiplication in depth 4, *Inform. Proccesing Lett.* **39**, pp. 99-110.
- Hopfield, J.J. (1982). Neural Networks and physical systems with emergent collective computational abilities, *Proc. Natl. Acad. Sci. USA* **79**, pp. 2554-2558.

- Hopfield, J.J. (1984). Neunons with graded response have collective computational properties like those of two-styate neurons, *Proc. Natl. Acad. Sci. USA* **79**, pp. 3088-3092.

- Impagliazzo, R., Paturi R., and Saks, M.E. (1997). Size-Depth Trade-offs for Threshold Circuits. *SIAM Journal on Computing* **26**, 3, pp. 693.

- Jacobs, R.A., Jordan, M.I. and Barto, A.G. (1991). Task decomposition through competition in a modular connectionist architecture: The what and where vision tasks. *Cognitive Science* **15**, pp. 219-250.

- Jung, G. and Opper, M. (1996). Selection of Examples for a Linear Classifier, *J. Phys. A: Math. and Gen.* **29**, 7, pp. 1367.

- Kinzel, W., and Ruján, P. (1990). Improving a Network Generalization Ability by Selecting Examples. *Europhysics Letters* **13**, 5, pp. 473.

- Kirckpatrick, S. , Gelatt, C.D., and Vecchi, M.P. (1983). Optimization by Simulated Annealing. *Science*, **220**, pp. 671.

- Klingman, E. (1977).*Microprocessor systems design*, Prentice Hall, New Jersey.

- Kuffler, S.W., and Nicholls, J.G. (1984) De la neurona al cerebro, Reverté, S.A., Madrid.

- Lauwereins, R., and Bruck, J. (1991). Efficient Implementation of a Neural Multiplier, IBM Research, Tech. Report RJ 8138.

- Levine, D.S. (1991) *Introduction to Neural and Cognitive Modelling*, Lawrence Erlbaum Ass., NY.

- Mac Gregor, R.J. (1987) *Neural and Brain Modelling*, Academic Press, NY.

- Mc Culloch, W.S., and Pitts, W. (1943). A Logical Calculus of Ideas Immanent in Nervous Activity. *Bulletin of Mathematical Biophysics* **5**, pp. 115-133.

- Mézard M., and Nadal J.P.(1989). Learning in feedforward layered networks: the tiling algorithm, *J. Phys. A: Math. and Gen.* **22**, pp. 2191-2203.

- Minsky, M.L., and Papert, S.A. (1969). *Perceptrons*, MIT Press, Cambridge, MA.
- Mitchinson G.J. and Durbin R.M. (1989) Bounds on the leaarning capacity of some multi-layers networks. *Bio. Cyber.* **60**, pp. 345-356.
- Montemurro, M. Trabajo especial de la licenciatura en física. FaMAF-UNC.
- Muller, B., and Reinhardt, J. (1991). *Neural Networks: An introduction.*, Springer Verlag, Berlin.
- Parberry, I. (1996). Circuit Complexity and Feedforward Neural Networks, in *Mathematical Perspectives on Neural Networks*, P. Smolensky, M. Mozer, D. Rumelhart,(Eds.), Lawrence Erlbaum Associates, pp. 85-111.
- Parisi, G. (1992). On the classification of learning machines. *Network* **3**, pp. 259-265.
- Patarnello, S., and Carnevali, P. (1987). Learning Networks of Neurons with Boolean Logic, *Europhysics Letters* **4**, 4, pp. 503-508.
- Paterson, M., Pippenger, N., and U. Zwick. (1990). Optimal Carry Save Networks, Research Report RR166, Dept. of Computer Science, Univ. of Warwick, Coventry, UK.
- Petridis, V., and Kehagias, A. (1998) *Predictive Modular Neural Networks: Applications to Time Series*, Kluwer Academic Publishers.
- Plutowski, M. and White, H. (1993). Selecting Concise Training Sets from Clean Data, *IEEE Transactions on Neural Networks* **4**, 2, pp. 305.
- Pomerleau, D.A. (1989). ALVINN: An autonomous land vehicle in a neural network, Advances in Neural Information Processing Systems I, Touretzky, D.(Ed.). San Mateo: Morgan Kaufmann, pp. 305-313.
- Prechelt, L. (1996). A Quantitative Study of Experimental Evaluations of Neural Network Learning Algorithms: Current Research Practice, *Neural Networks* **9**, 3, pp. 457-462.

- Priel, A., Blatt, M., Grossman, T., Domany, E. and Kanter, I. (1994). Computational Capabilities of Restricted two layers Perceptrons, *Physical Review E*, **50**, pp. 577.

- Rosenblatt, F.(1958). The perceptron: A probabilistic model for information storage and organization in the brain, *Psychol. Rev* **65**, pp. 386-408.

- Rosenblatt, F.(1962). *Principles of Neurodynamics*, Spartan, New York.

- Rueckl, J.G., Cave, K.R., and Kosslyn, S.M. (1989). Why are 'what' and 'where' processed by separate cortical visual systems? A computational investigation. *Journal of Cognitive Neuroscience* **1**, pp. 171-186.

- Rumelhart, D.E., Hinton, G.E., and Williams, R.J.(1986a). Learning representations by back-propagating errors, it Nature **323**, pp. 533-536.

- Rumelhart, D.E., McClelland, J.L. (1986b). *Parallel Distributed Processing*, vol. 1, MIT Press, Cambridge, MA.

- Sejnowski, T.J., and Rosenberg, C.R. (1987). Parallel networks that learn to pronounce english text, *Complex Systems* **1**, pp. 145-168.

- Shannon, C.E. (1948). A mathematical theory of communication. *Bell System Technical journal*, **27**, pp. 379-423, pp. 623-656.

- Siu, K., Bruck, J., and Kailath, T. (1993). Depth Efficient Neural Networks for Division and Related problems, *IEEE Transactions on information Theory* **39**, 3.

- Siu, K., and Roychowdhury, V. (1994). On optimal depth threshold circuits for multiplication and related problems. *SIAM J. Discrete Math.* **7**, 2.

- Solla, S.A. (1991). Supervised Learning: a Theoretical Framework. Proceeding of the 1990 Workshop on non linear Modelling and Forecasting, Casdaghi, M. and Eubank, S.(eds.), Addison-Wesley.

- Sollich, P. (1994). Query construction, entropy, and generalization in neural-network models. *Physical Review E* **49**, 5, pp. 4637-4651.

- Sprinkhuizen-Kuyper, I.G., and Boers, E.J.W. (1996). Probabilities and entropy of some small neural networks for boolean functions. Technical Report 96-30, Dept. of Computer Science, Leiden University, Holland.

- Tessauro, G. and Janssens, R. (1988). Scaling Relationships in Back-propagation Learning: Dependence on Predicate Order. Technical Report CCSR-88-1. Center for Complex System Research. Urbana-Champaign.

- Toulouse, G. (1989). Perspectives on neural network models and their relevance to neurobiology, *J. Phys. A: Math. and Gen.* **22**, pp. 1959-1968.

- Van den Broeck, C., and Kawai, R. (1990). Learning in feedforward Boolean Networks, *Physical Review A* **42**, 10, pp. 6210-6218.

- Van Essen, D.C. (1985). Functional organization of the primate visual cortex, In Cerebral Cortex, Peters and Jones, Eds. Plenum, New York. pp. 259-329.

- Vapnik V.N., and Chervonenkis, A.Y. (1971) On the Uniform Convergence of Relative Frequencies of Events to their Probabilities. *Th. Prob. and its applications* **17**, 2, pp. 264-280.

- Watkin, T.L.H, Rau, A. and Biehl, M. (1993). The statistical mechanics of learning a rule, *Rev. Mod. Phys.* **65**, 2, pp. 499-556.

- Winder, O.R. (1966). Partitions of N-space by hyperplanes, *J. SIAM Appl. Math.* **14**, 4, pp. 811-818.