APLICACIÓN DE LAS REDES DE NEURONAS DE COMPRESIÓN A LA EXTRACCIÓN DE CARACTERÍSTICAS PARA EL RECONOCIMIENTO A PARTIR DE IMÁGENES DE LA OREJA

AUTOR: MIGUEL Á. CARREIRA PERPIÑÁN

TUTOR: ÁNGEL SÁNCHEZ CALLE

FACULTAD DE INFORMÁTICA

UNIVERSIDAD POLITÉCNICA DE MADRID

SEPTIEMBRE DE 1995

Resumen

El problema de la identificación personal por ordenador goza actualmente de gran popularidad, especialmente cuando dicha identificación tiene lugar a partir de imágenes faciales. Sin embargo, la tarea —realizada de manera espontánea por las personas— presenta dificultades considerables para su automatización.

Nosotros introducimos en este trabajo la identificación a través de imágenes de la *oreja*, que presenta algunas ventajas sobre la que se basa en imágenes de la cara, entre las que destacan su menor tamaño y variabilidad y sobre todo su mayor capacidad discriminante.

De los múltiples problemas asociados a un sistema de identificación, nosotros abordamos dos: la *extracción de características* a partir de la imagen, para su posterior empleo en una etapa de clasificación o identificación; y el *reconocimiento*, es decir, responder sí o no a la pregunta "¿Representa la imagen una oreja?" La extracción de características se realiza por medio de un tipo especial de redes de neuronas artificiales, las llamadas *redes autoasociativas de compresión*, capaces de llevar a cabo durante su entrenamiento un proceso similar al del análisis de componentes principales. Las características obtenidas son precisamente las proyecciones de la imagen sobre estas componentes principales. El reconocimiento se implementa mediante una simple regla de rechazo sobre el error producido por la red al reconstruir la imagen: ésta es reconocida si y solamente si dicho error es menor que un cierto valor umbral prefijado de antemano.

Se dedica gran parte del texto a justificar teóricamente el comportamiento de las redes de compresión, así como a describir en detalle los experimentos realizados, los cuales se repitieron para dos tamaños distintos de imagen (20×32 y 30×48) y diferente número de características extraídas.

Finalmente, se resumen los resultados obtenidos y se describen varias posibilidades de continuación del trabajo. Para facilitar la misma, se cita abundante bibliografía y se detallan en un apéndice aspectos de implementación.

Palabras clave: procesamiento facial, identificación personal, reconocimiento de patrones, redes de neuronas artificiales, compresión de imágenes, segmentación de imágenes, memorias autoasociativas, transformada de Karhunen-Loève, análisis de componentes principales.

Abstract

Presently, the problem of computer personal identification enjoys considerable popularity, particularly when the identification takes place by means of face images. However, the automatisation of this task —which is easily accomplished by human beings— poses considerable technical challenges.

In this work, we introduce the use of *outer ear* images for identification. The ear presents some advantages over the face for identification matters, such as its smaller area and variability and, more interestingly, its greater discriminant capacity.

From the many problems associated to an identification system, we approach two: *extraction of features* from the image —to be used later in a classification or identification stage—; and *recognition*, i.e. to be able to answer 'yes' or 'no' to the question "Does the image represent an ear?" The feature extraction is attained by means of a special type of artificial neural networks, called *compression autoassociative networks*, which are able to carry out a process similar to principal components analysis during training. The features obtained are precisely the image projections on the principal components. The recognition phase is implemented with a simple rejection rule on the error produced by the network when reconstructing an image: the image is recognised if and only if its error is less than a certain fixed threshold value.

A large part of the text is dedicated to theoretically justify the behaviour of the compression networks, as well as to describe in detail the experiments performed, which were repeated for each of the image sizes (20×32 y 30×48) and number of features extracted.

Finally, our results are summarised and several possible continuations of the work are described. To ease future work, a number of bibliographic references are listed and implementation details are given in an appendix.

Keywords: face processing, personal identification, pattern recognition, artificial neural networks, image compression, image segmentation, autoassociative memories, Karhunen-Loève transform, principal component analysis.

Índice General

Índice de Figuras

Índice de Tablas

Capítulo 1

Introducción

1.1 Introducción al problema de la identificación personal

1.1.1 Actualidad del problema

El reconocimiento e identificación personal, principalmente a través de imágenes faciales, es un tema que goza en la actualidad de un éxito considerable. Buena prueba de ello son el elevado número de artículos publicados en revistas y congresos en los últimos años o el volumen de comunicaciones que aparecen en grupos de Usenet relacionados con el tema, como `comp.ai.neural-nets`, `sci.image.processing` y otros. Existen ya congresos especializados en el tema (por ejemplo, el *International Workshop on Automatic Face- and Gesture-Recognition*, celebrado los días 26 al 28 de junio de 1995 en Zurich) y desde 1992 se han publicado 3 *surveys* sobre el reconocimiento facial [48, 56, 10], en revistas como *Pattern Recognition* y *Proc. of the IEEE*. Son muchos los grupos en centros de investigación y universidades de todo el mundo que están abordando el tema desde perspectivas distintas: psicología, telecomunicaciones, procesamiento de señal, redes de neuronas, informática, biología, etc.

Aparte de los primeros intentos más o menos aislados —por ejemplo, los de Harmon *et al.* [22], Kaufman y Breeding [27] o Kohonen y sus colaboradores [29], entre otros—, el interés generalizado arranca de principios de los 90, aproximadamente; a partir de ese momento, el volumen de publicaciones ha crecido cada año. Si bien ya se han obtenido algunos resultados prometedores, es mucho mayor el camino que queda por recorrer.

1.1.2 Reconocimiento e identificación

En este trabajo se definirá **reconocimiento** como *el proceso por el cual se asigna un objeto percibido a una clase determinada*; por ejemplo, se dice que "el objeto es una oreja." El reconocimiento es el objetivo principal de un sistema visual. Y consideraremos como **identificación** al *proceso que tiene lugar tras haber reconocido un objeto como perteneciente a una clase y por el cual se identifica dicho objeto con una instancia particular de dicha clase*; por ejemplo, "el objeto es la oreja de Pantito." La figura 1.1 aclara la situación.

Existe cierta confusión en la literatura en cuanto a los conceptos de reconocimiento e identificación; nosotros emplearemos las definiciones anteriores, tal como también hacen Samaria [49] y Samal e Iyengar [48]. No ocurre lo mismo con, por ejemplo, O'Toole *et al.* [39].

1.1.3 Aproximaciones al problema del reconocimiento e identificación personal. El procesamiento facial

El método más fiable conocido para la identificación personal es el del *dibujo papilar*, responsable de la aparición de las huellas dactilares, como es sabido. Sin embargo, hay ciertas aplicaciones en las que tal método no es apropiado; por ejemplo, para acceder a una base de datos utilizando como clave una imagen del rostro de una persona, o para identificar a un sospechoso del que se tiene una fotografía pero no sus huellas. En este caso resulta necesario un sistema que realice las funciones pedidas a partir de imágenes faciales.

El reconocimiento e identificación facial son tareas que las personas realizamos con facilidad pasmosa —incluso en condiciones en las que la cara está alterada por una expresión, por maquillaje, barba, peinado

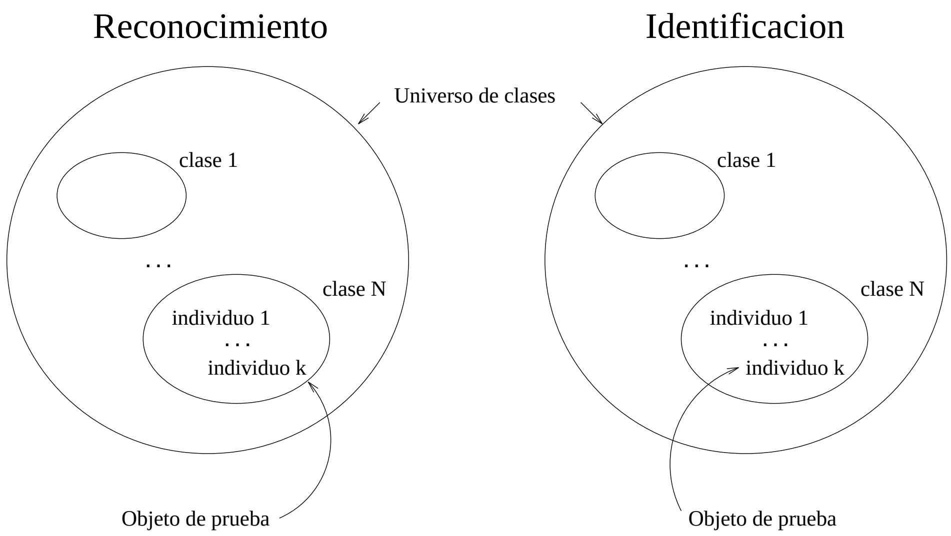

Figura 1.1: Reconocimiento frente a identificación.

o gafas, o se ve desde distintos puntos de vista (frente, perfil, tres cuartos, etc.) y bajo distinta iluminación—, pero que plantean grandes problemas para ser reproducidas por un ordenador.

La teoría de reconocimiento de patrones proporciona el marco en el que encuadrar la tarea del reconocimiento y posterior identificación facial: en una primera fase[1], llamada de *segmentación*, se localiza y extrae de la imagen global aquella parte que nos interesa —la que contiene exclusivamente la cara—; en una segunda, se obtiene a partir de la imagen segmentada un vector de características numéricas, que idealmente representa unívocamente la cara que aparece en la imagen; en una tercera fase se utiliza alguna técnica para decidir si la imagen es o no una cara (reconocimiento) y de quién es (identificación).

Los primeros intentos de extracción de características a partir de imágenes faciales [48] generalmente empleaban una codificación geométrica obtenida a través de medidas de distintas relaciones entre los rasgos faciales (por ejemplo, la distancia interocular, la longitud del segmento que va desde la barbilla hasta el extremo de la frente, el ángulo formado por dicho segmento y el contorno de la frente, etc.; es decir, distancias entre ciertos puntos clave —llamados puntos *somatométricos*— y ángulos entre segmentos que los unen). En esta labor, a menudo resulta útil hacer uso de imágenes de perfil, en las que es relativamente fácil extraer de manera automática la silueta del perfil [27] y ajustarla a una plantilla [23] (hecha a base de *splines*, por ejemplo, o, de manera más simple, una quebrada), a partir de la cual pueden situarse en la escala vertical diversos puntos y rasgos relevantes que se buscan en la imagen frontal [1] (p. ej. la altura a la que están los ojos, el punto extremo de la barbilla, etc.). Sin embargo, este procedimiento descarta información importante sobre la textura y la forma detallada de la cara, así como variaciones sutiles en la configuración de los rasgos faciales. Además, la extracción automática de características es dificultosa y propensa a errores, incluso contando con fotos de buena calidad y en las que son claramente visibles los puntos clave[2]. Finalmente, si bien es cierto que las personas extraen una parte de la información sobre la cara basándose en rasgos geométricos —así, podemos decir que "tal persona tiene la nariz muy estrecha"—, también lo es que esto no ocurre de manera muy localizada ni mucho menos numérica, y que no se descarta el resto de información sobre la imagen (textura, etc.).

Más tarde se empezaron a considerar representaciones más simples de las caras basadas en el vector de la imagen (obtenido concatenando las filas de píxeles de la misma); dicha representación puede usarse para caras completas, parciales o componentes suyos (ojos, boca), pero el enfoque es mucho más general y sirve para cualquier tipo de imagen, evidentemente. Esta representación permite la reconstrucción de la imagen original y, por tanto, lleva codificada de manera implícita la representación geométrica antes mencionada (ya que puede extraerse a partir de la imagen reconstruida), mientras que al revés no es cierto. Pero además preserva información detallada sobre la textura y la forma. Como veremos, estos códigos basados en vectores imagen van asociados generalmente a un modelo que representa, implícita o explícitamente, los vectores (las caras) como una combinación lineal de autovectores de la matriz de covarianzas de los mismos. Desde este punto de vista, los coeficientes asociados a los autovectores (*eigenvectors*) en dicha

[1] Tras, posiblemente, un preprocesamiento previo de la imagen (realce, restauración, etc.) que facilite la segmentación.

[2] El enfoque geométrico es útil para construir modelos tridimensionales de la cabeza a partir de los puntos somatométricos, obtenidos de una imagen de frente y otra de perfil (véase Akimoto [1]), pero este problema no está ya directamente relacionado con el que nos ocupa.

combinación lineal pueden considerarse como "macrorrasgos" (*macrofeatures*) extraídos de manera no supervisada. En la literatura en inglés se les dan diversos nombres a estos autovectores, relacionados con el hecho de provenir de imágenes faciales y de ser capaces de generar los macrorrasgos: *eigenpictures*, *eigenfeatures*, *eigenfaces*, *holons*, etc. También veremos que este enfoque puede abordarse desde diversos puntos de vista: estadístico (análisis de componentes principales), de teoría de señales o procesamiento de imágenes (compresión, transformada de Karhunen-Loève), conexionista[3] [56] y otros.

Indiquemos que, tal como señalan Dony y Haikin [13], el procesamiento de la información visual en las personas (en los mamíferos, en general) lo realizan redes de unidades de proceso conectadas de manera masivamente paralela, desde la retina hasta las estructuras de orden superior de la corteza visual; es decir, la información visual es procesada de manera paralela, contrariamente al procesamiento del lenguaje, que tiene lugar de manera secuencial, conforme las palabras van percibiéndose de una en una. Los sistemas artificiales de procesamiento de información presentan también ambas modalidades: la secuencial, apropiada para la implementación de fórmulas y algoritmos (que suelen venir expresados de manera lingüística), y la paralela, uno de cuyos casos particulares son las redes de neuronas artificiales[4] (RNAs). Existe una analogía entre las RNAs y sus homólogas en sistemas neurobiológicos: las primeras nacieron inspiradas por los segundos, pero ahora constituyen un paradigma con personalidad propia. Actualmente se da una influencia mutua entre las redes artificiales y las biológicas. Las características de las RNAs, como su estructura masivamente paralela, su alto grado de interconexión, su capacidad para almacenar experiencia y para autoorganizarse, encuentran características similares en nuestro propio sistema visual. Se puede decir, pues, que las RNAs son, por su propia naturaleza, análoga a la de los mecanismos visuales biológicos, especialmente apropiadas para el proceso de imágenes. Esto se ha comprobado de manera práctica: muchos enfoques conexionistas al proceso de imágenes obtienen rendimientos parecidos o mejores que los de los enfoques tradicionales [13].

1.1.4 Problemas básicos del procesamiento de imágenes faciales

Samal e Iyengar [48] reducen los problemas básicos del procesamiento de imágenes faciales a los siguientes:

- **Representación** de las caras: en un sistema de procesamiento de caras, éstas deben almacenarse y transmitirse en algún formato, que puede ser una codificación por rasgos (vector de características) o la propia imagen de píxeles. En este caso el fondo forma parte de la imagen y puede dar problemas si no está normalizado (p. ej., uniformemente blanco para todas las imágenes). En cualquier caso, la representación debe ser compacta pero preservar información suficiente para permitir el reconocimiento e identificación.

- **Detección** de la cara en una imagen: salvo para imágenes controladas (como las fotos del pasaporte), la posición de la cara en la imagen es desconocida. Se plantea el difícil problema, pues, de determinar si hay alguna cara en la imagen y, en caso afirmativo, hallar su posición[5].

- **Identificación**: consiste en asociar la cara detectada en la imagen a una cara almacenada en una base de datos.

- **Análisis de expresiones faciales**: se trata de determinar, a partir de la representación de una cara, si la expresión que presenta es de alegría, miedo, repulsión, etc., cosa que una persona realiza sin esfuerzo aparente, pero muy difícil de automatizar. Este problema aún no ha sido abordado de manera seria.

- **Clasificación**: en cuanto al sexo (masculino, femenino), edad, raza, etc. De nuevo, un problema complejo fácilmente resuelto por las personas.

Ninguno de estos problemas ha sido resuelto hasta ahora de manera satisfactoria en el caso general para imágenes fijas, y mucho menos para secuencias animadas. Dada la complejidad de la tarea, casi todos los intentos realizados se basan en imágenes frontales de la cara tomadas bajo condiciones muy controladas

[3] Por modelos *conexionistas* se entienden aquéllos que usan algoritmos que pueden implementarse en paralelo y que usan mecanismos distribuidos o no localizados de almacenamiento. El ejemplo más conspicuo lo constituyen las redes de neuronas artificiales.

[4] Existe cierta confusión en la literatura entre el término *redes de neuronas*, de connotaciones biológicas, y el término *redes de neuronas artificiales* (RNAs), que son las tratadas en este trabajo, y muchos autores tienden a referirse con el primero a lo segundo; lo mismo ocurre con los términos *neurona* y *unidad*, y *sinapsis* y *conexión*. Nosotros trataremos de evitar esta confusión en lo posible, usando los términos apropiados en cada caso.

[5] Las imágenes usadas en este trabajo pueden considerarse normalizadas, ya que el proceso seguido en su captación permite un control riguroso sobre las mismas (iluminación, posición, tamaño, etc.). Ello simplifica el problema de detección, convirtiéndolo en un simple reconocimiento (la imagen es o no es una cara).

de fondo, iluminación, posición y calidad que simplifiquen la segmentación u otros tratamientos que se empleen.

Se puede considerar que un sistema de procesamiento de caras se acercará al humano cuando sea capaz de reconocer e identificar, en tiempo real, caras en una secuencia de vídeo tomada en un aeropuerto u otro escenario complejo y abigarrado.

De los problemas enumerados, en este trabajo se abordan sólo el de reconocimiento y el de extracción de características.

1.1.5 Aplicaciones

Samal e Iyengar [48] y Chellappa *et al.* [10] detallan varias aplicaciones del procesamiento de imágenes faciales. La mayor parte de ellas están relacionadas con el campo legal: los sistemas de seguridad —que aparecen en el momento en el que una instalación dada permite el acceso a la misma sólo a un grupo de individuos[6], desde un ordenador hasta un cajero automático, pasando por zonas de acceso restringido, etc.—, la identificación de delincuentes, la vigilancia u observación de grupos de personas a través de cámara de video (en una tienda, por ejemplo) y otros. Pero también hay otras aplicaciones de interés comercial, como el de las interfaces de usuario, la videotelefonía, el acceso a bases de datos usando como clave la cara, etc.

Asimismo señalan el hecho de que en Estados Unidos el problema que más preocupa a los ciudadanos es el de la delincuencia —superando al paro, la sanidad y la economía—, por lo que es previsible un aumento de los fondos dedicados a proyectos que tengan aplicación en el campo legal, como es el caso del procesamiento facial.

1.1.6 Empleo de imágenes de la oreja para la identificación

Si bien, como ha quedado dicho, la identificación personal basada en el uso de imágenes faciales goza de gran aceptación, esto no significa que la cara sea la parte del cuerpo humano que contiene la mayor cantidad de información identificativa; existen otras que la superan en ese aspecto. La mayor riqueza identificativa la poseen los dibujos papilares, seguidos por, curiosamente, **la oreja**.

Es cierto que las personas poseemos una capacidad extraordinariamente desarrollada para el reconocimiento e identificación de caras; sin embargo, actualmente [10] se discute la posibilidad de que las personas llevemos "cableada" en el cerebro dicha capacidad; es decir, esta capacidad podría ser aprendida, ya que su eficiencia es notablemente menor al tratar de recordar y distinguir caras de otra raza (japonés, por ejemplo); a este efecto se le conoce como "efecto de la otra raza" (*other-race effect*), y se debe a que no estamos acostumbrados a ver caras de esa otra raza. Análogamente, una persona común no está acostumbrada a fijarse en las orejas para identificar a otra persona (a menos que sean realmente inusuales), pero un buen fisonomista (como algunos funcionarios policiales) sí puede hacerlo. Con esto queremos decir que los rasgos faciales parecen no tener nada de especial a priori frente a los de la oreja, aparte del hecho de que los conocemos muy bien. A un ordenador, este hecho le es indiferente, como es obvio.

El hecho es que la riqueza identificativa de la oreja es enorme. Las múltiples variaciones de su complicada estructura entre persona y persona, así como de su tamaño y forma, hace verdadero el dicho: *no hay dos orejas iguales*. Tan sólo en los gemelos univitelinos pueden dejar de diferenciarse de manera destacada [44]. Citamos textualmente del libro *Estudios de Policía Científca: Identificación* [19, pág. 49], de la División de Formación y Perfeccionamiento de la Dirección General de la Policía (1992):

[6] Normalmente se emplean métodos tales como tarjetas, palabras clave, códigos, etc., pero desde hace cierto tiempo ya están en funcionamiento sistemas de identificación *biométrica* [51], que emplean partes del cuerpo humano para tal fin, como por ejemplo:

- La *mano*: existen sistemas comerciales que obtienen, a partir de una imagen digitalizada de la mano del individuo que desea obtener el acceso a la instalación (para lo cual debe apoyar la mano sobre un dispositivo digitalizador situado a su entrada), una representación de tan sólo 9 bytes de ciertos rasgos geométricos de la mano. El sistema emplea 1.2 segundos en el proceso y tiene una tasa de aceptación o rechazo falsos del 0.2%, algo pobre pero adecuada para aplicaciones específicas.
- El *ojo*: otros sistemas, igualmente ya en la calle, exploran el patrón de la retina con rayos infrarrojos y obtienen un código de entre 48 y 256 bytes con el que alcanzan tasas mucho menores (0.00076% de acuerdo con los fabricantes) en un tiempo de 4 a 7 segundos.

Todos estos sistemas presentan, frente a uno basado en la cara, el inconveniente de ser molestos; por ejemplo, en el caso del reconocimiento a través del patrón de la retina, el sujeto debe acercar el ojo a una distancia de 8 centímetros del dispositivo; en el de la mano, los usuarios acusan al procedimiento de "poco higiénico." Frente a los dibujos papilares, la cara puede presentar las ventajas de una representación más compacta, más barata y sobre todo el hecho de evitar el "estigma" criminal de la toma de las huellas, algo que mucha gente detesta.

La oreja constituye la facción más precisa para afirmar o negar una identidad, particularmente en las confrontas de fotografías, dada la diversidad de sus formas y detalles, que aportan más datos a la confronta que el resto de los rasgos fisonómicos juntos. Por ello ya algunos tratadistas, como Frigerio, la tomaron como referencia para su sistema identificativo, que éste denominó "otométrico", ideando incluso un aparato, el "otómetro", para medir los distintos rasgos.

Dicho libro dedica cuatro páginas más a la descripción física de la oreja, haciendo énfasis en aquellos rasgos de mayor interés para la identificación, así como en sus variaciones en cuanto a tamaño, forma, disposición, etc. Nosotros nos contentaremos con señalar los más relevantes en la figura 1.2, que son los cinco *relieves* (*hélix*, *antihélix*, *lóbulo*, *trago* y *antitrago*) y las cuatro *depresiones* (*fosa navicular*, *foseta digital*, *concha* y *canal intertraguiano*). Para mayores detalles remitimos al lector a [19].

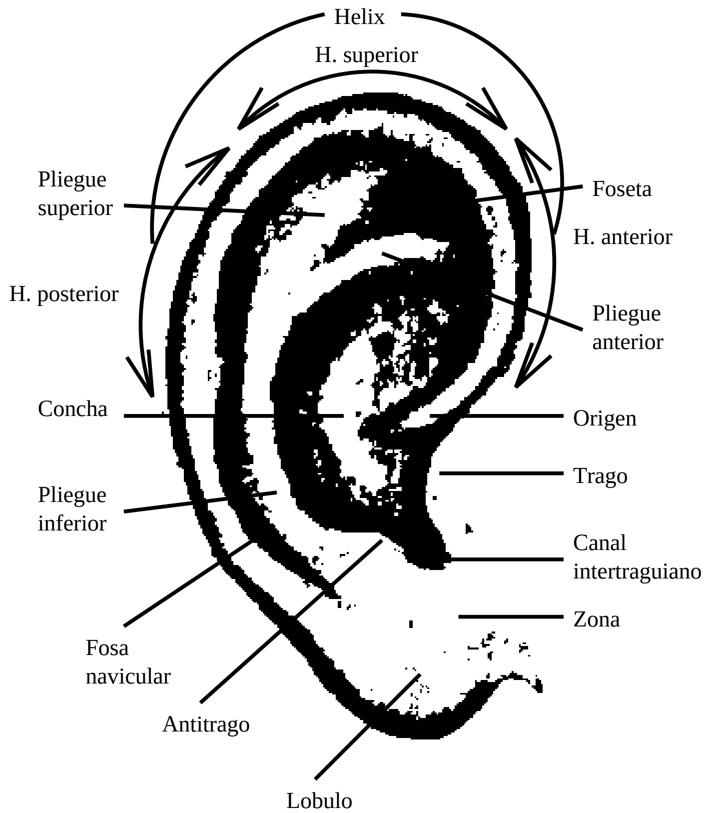

Figura 1.2: Representación esquemática de la oreja, mostrando sus rasgos más sobresalientes. A la derecha, una foto real en tonos de gris de una oreja.

La oreja cuenta con ventajas adicionales sobre la cara:

- La oreja no cambia de manera significativa a partir del estado adulto del individuo; la cara, por el contrario, sigue cambiando —lentamente— durante toda la vida del individuo (cuántas veces hemos dicho "¡Cuánto has cambiado!" ante una persona a quien no veíamos en dos o tres años).
- La cara cambia notablemente de aspecto con la expresión (tristeza, alegría, miedo, sorpresa, etc.), lo cual es una seria dificultad para su procesamiento por el ordenador. Por el contrario, la forma y aspecto de la oreja son fijos (salvo mutilaciones u otros casos extremos[7]).
- La distribución de color es más uniforme en la oreja que en la cara (piénsese en los ojos, por ejemplo), por lo que se pierde menos información al trabajar con imágenes de tonos de gris o blanco y negro.
- La superficie ocupada por la oreja es más pequeña (aproximadamente 1/20 ó 1/25 de la de la cara), lo que permite utilizar imágenes de menor resolución y hacer más eficiente y económico el proceso.

[7] La integración de su tejido, extraordinariamente rico en glándulas sebáceas, hace que cicatrices e injertos en su pabellón sean muy difíciles de disimular, lo que los convierte en "rasgos" añadidos de la misma.

Sin embargo, las orejas pueden aparecer total o parcialmente tapadas por el pelo o por pendientes; aunque lo mismo le ocurre a la cara —si bien en menor medida— con el maquillaje, barba, gafas, peinado, etc.

Como en cualquier representación plana de un objeto espacial, hay información que se pierde: en el caso de la cara, la longitud de la nariz, etc. (aunque con ayuda de la imagen de perfil pueden recuperarse en gran medida); en el caso de la oreja, la separación del cráneo, la profundidad de los diversos pliegues, etc. En ambos casos hay que tener cuidado con las sombras creadas por la iluminación lateral.

En la parte experimental del presente trabajo consideraremos imágenes de la oreja para su reconocimiento (previa extracción de características). Sin embargo, todo el tratamiento matemático presentado y las redes de neuronas artificiales creadas son igualmente válidas para imágenes faciales o de cualquier otro tipo. Por esta razón, en las explicaciones a menudo nos referiremos globalmente a ambas (caras y orejas).

1.2 Planteamiento del trabajo. Objetivos

De lo explicado en las secciones anteriores se deduce que el problema del reconocimiento, identificación, etc. personales a partir de imágenes faciales o de la oreja, no sólo está lejos de obtener una solución global ni parcial para sus subproblemas asociados, sino que cada uno de ellos conlleva además una gran cantidad de trabajo.

Por ello, nosotros nos restringiremos a una parte: la **extracción de características de manera no supervisada** a partir de imágenes de la oreja. Para ello emplearemos un tipo particular de RNAs, las redes de compresión. Con fines comparativos, repetiremos los experimentos para varias resoluciones de imagen y para distintos parámetros de la red (número de unidades ocultas). Como base sobre la que comparar se empleará una técnica bien conocida en estadística, el análisis de componentes principales.

Como aplicación inmediata de las redes al reconocimiento, se enunciará una regla de rechazo basada en el error de la red al reproducir un patrón, que permitirá responder con sí o no a la pregunta "¿Representa la imagen una oreja?" (aunque a veces la respuesta sea errónea a juicio de una persona).

No se abordará el problema de la identificación, que alargaría excesivamente este trabajo; no obstante, se indica cuál sería un posible proceso a seguir a partir del vector de características generado por la red de compresión. La resolución práctica de dicho problema podría ser motivo para otro proyecto de fin de carrera. Con objeto de facilitar la continuación del trabajo, el autor ha dado todas las explicaciones que ha podido, tanto en lo que se refiere a los fundamentos teóricos de las redes empleadas (con abundantes referencias a la literatura especializada), como a las herramientas utilizadas.

1.3 Panorámica del resto del libro

Una vez introducidos en el entorno de la identificación personal, lo primero es dar la nomenclatura y notación empleada, así como una serie de resultados teóricos sobre matrices reales simétricas semidefinidas positivas y su descomposición en formas más simples. Esto permite comprender la función que realizan las redes de compresión y relacionarlo con técnicas esencialmente equivalentes y que son conocidas en teoría de señales y estadística desde mucho tiempo antes de que aparecieran las redes de neuronas artificiales. La sección 1.4 se encarga de esto.

El capítulo 2 describe las redes de neuronas artificiales de interés para nosotros, relacionándolas con el problema de optimización no lineal no restringido. A continuación se describen los conjuntos de entrenamiento empleados para enseñar a la red, cuya importancia es tan grande como la de la propia arquitectura elegida.

Sobre la base de los dos capítulos anteriores, es ya sencillo introducir el análisis de componentes principales y la transformada de Karhunen-Loève, así como las arquitecturas de RNA para la extracción de componentes principales. Esto se hace en el capítulo 3.

El capítulo 4 detalla los experimentos realizados con cada una de las redes creadas en cuanto a extracción de características y componentes principales. Se dan gráficos, etc. que justifican la validez del enfoque conexionista para la extracción de características. Para comparar, se repiten los cálculos pero usando técnicas tradicionales de análisis numérico sobre la matriz de covarianzas de los patrones de entrada. Así obtenemos constancia experimental de la similitud subyacente a ambos enfoques.

En el siguiente capítulo se define la regla de rechazo, que permite el reconocimiento de un patrón dado. Su validez se comprueba experimentalmente sobre las redes del capítulo 4.

Finalmente, el capítulo 6 resume todos los resultados obtenidos desde una perspectiva más global y los compara, lo que permite ver las condiciones en las que un enfoque es más ventajoso que otro. Se dedica una sección a posibles ampliaciones del trabajo realizado.

Los apéndices recogen información sobre el proceso seguido para la captación de las imágenes, las herramientas empleadas (incluyendo software externo, como *SNNS*, *Mathematica* o *xv*, y los *shellscripts* creados por el autor) y varias observaciones y demostraciones adicionales. También se dedica un apéndice a bases de datos existentes con imágenes faciales.

1.4 Nomenclatura y notación empleadas[8]

Una letra minúscula cursiva representará un número entero o real; así, n denotará generalmente la dimensión de un vector y p el número de vectores de un conjunto de vectores.

Para abreviar la escritura, emplearemos la siguiente notación:

$$\langle x \rangle = \sum_{i=1}^{p} x_i,$$

es decir, $\langle x \rangle$ es la suma de los x_i (en su caso, serán vectores, matrices, etc.) extendida a todo el conjunto de los x_i, $i = 1, \ldots, p$. Será particularmente adecuada para escribir de manera concisa sumatorios extendidos a una nube de vectores o a un conjunto de patrones de entrada de una RNA. Asimismo, usaremos la notación de Kronecker:

$$\delta_{ij} = \begin{cases} 0, & i \neq j \\ 1, & i = j \end{cases}$$

especialmente útil para escribir productos escalares mutuos de vectores ortonormales.

Una letra minúscula en negrita representará un vector columna. Además, $\mathbf{x}$ irá generalmente asociado a vectores centrados[9], cuya media $\overline{\mathbf{x}}$ es 0, es decir,

$$\overline{\mathbf{x}} = \frac{1}{p} \sum_{i=1}^{p} \mathbf{x}_i = \frac{1}{p} \langle \mathbf{x} \rangle = \mathbf{0}$$

mientras que $\mathbf{y}$ irá asociado a vectores generales (no centrados). $\mathbf{u}$ denotará un autovector de la matriz de covarianzas, asociado al autovalor λ.

Una letra mayúscula en negrita representará una matriz (no necesariamente cuadrada; su orden se especificará debidamente). A menos que se indique lo contrario, todas las matrices que aparezcan serán reales. En particular, $\mathbf{I}$ será la matriz identidad del orden que le corresponda e $\mathbf{Y}$ será la matriz de orden $n \times p$ cuyas columnas son los p vectores columna $\mathbf{y}_1, \ldots, \mathbf{y}_p$. La matriz de covarianzas de dichos vectores $\mathbf{y}$, $\mathbf{\Sigma} = (\sigma_{ij})$, será entonces:

$$\mathbf{\Sigma} = \frac{1}{p}\mathbf{Y}\mathbf{Y}^T - \overline{\mathbf{y}}\overline{\mathbf{y}}^T = \frac{1}{p}\mathbf{X}\mathbf{X}^T$$

donde $\mathbf{x} = \mathbf{y} - \overline{\mathbf{y}}$ son los vectores centrados. Puede comprobarse que

$$\sigma_{ij} = \frac{1}{p} \big\langle (\mathbf{y}_i - \overline{\mathbf{y}}_i)(\mathbf{y}_j - \overline{\mathbf{y}}_j) \big\rangle = \frac{1}{p} \langle \mathbf{x}_i \mathbf{x}_j \rangle$$

En general, será más útil emplear directamente la matriz $\mathbf{X}\mathbf{X}^T = p\mathbf{\Sigma}$ (también llamada matriz de correlación). En este caso conviene observar que:

$$\mathbf{\Sigma}\mathbf{u} = \lambda\mathbf{u} \Rightarrow \mathbf{X}\mathbf{X}^T\mathbf{u} = p\lambda\mathbf{u}$$

es decir, los autovectores de $\mathbf{X}\mathbf{X}^T$son los de $\mathbf{\Sigma}$ multiplicados por el número de vectores p. También es interesante ver cómo se transforma $\mathbf{\Sigma}$ ante traslaciones, homotecias y rotaciones de los $\mathbf{y}$. Fácilmente se demuestran los resultados resumidos en la tabla 1.1, si $\mathbf{t} \in \mathbb{R}^n$, $a \in \mathbb{R}$ y $\mathbf{R}$ es ortogonal, y λ y $\mathbf{u}$ son los autovalores y autovectores, respectivamente, de $\mathbf{\Sigma}$.

La operación de trasposición se indicará con el superíndice ^{T}. Así, $\mathbf{x}^T\mathbf{Q}\mathbf{x}$ denotará una forma cuadrática en $\mathbf{x}$ de matriz $\mathbf{Q}$, $\mathbf{x}^T\mathbf{y} = (\mathbf{x}, \mathbf{y})$ el producto escalar de los vectores $\mathbf{x}$ e $\mathbf{y}$, y $\mathbf{x}\mathbf{y}^T$ su producto externo.

Dada una forma cuadrática de matriz $\mathbf{Q}$, su signatura será $\operatorname{sig}\mathbf{Q} = (i_p, i_n)$ donde i_p es el número de autovalores positivos de $\mathbf{Q}$ (su índice positivo) e i_n el de autovalores negativos (su índice negativo).

---

[8] El programa de *Mathematica* listado en el apéndice D.4 da definiciones para todas estas matrices y operaciones, entre otras.

[9] Es decir, referidos al sistema de centro de masas, cuyo origen está en la media de los datos y cuyos ejes son paralelos a los del sistema original.

Tabla 1.1: Autovectores y autovalores de la matriz de covarianzas ante diversas transformaciones.

Traslación	Homotecia	Rotación
$\mathbf{y}' = \mathbf{y} + \mathbf{t}$	$\mathbf{y}' = a\mathbf{y}$	$\mathbf{y}' = \mathbf{R}\mathbf{y}$
$\overline{\mathbf{y}}' = \overline{\mathbf{y}} + \mathbf{t}$	$\overline{\mathbf{y}}' = a\overline{\mathbf{y}}$	$\overline{\mathbf{y}}' = \mathbf{R}\overline{\mathbf{y}}$
$\mathbf{\Sigma}' = \mathbf{\Sigma}$	$\mathbf{\Sigma}' = a^2\mathbf{\Sigma}$	$\mathbf{\Sigma}' = \mathbf{R}\mathbf{\Sigma}\mathbf{R}^T$
$\lambda' = \lambda$	$\lambda' = a^2\lambda$	$\lambda' = \lambda$
$\mathbf{u}' = \mathbf{u}$	$\mathbf{u}' = \mathbf{u}$	$\mathbf{u}' = \mathbf{R}\mathbf{u}$

$p_{\mathbf{A}}(\lambda) = |\mathbf{A} - \lambda\mathbf{I}|$ representará el polinomio característico de $\mathbf{A}$.

Denotaremos por $L(\mathbf{A})$ al subespacio generado por los vectores columna de la matriz $\mathbf{A}$ (que coincide con su subespacio imagen $\operatorname{im}\mathbf{A}$) y por $\ker\mathbf{A}$ al núcleo de $\mathbf{A}$ (vectores que se transforman en el cero). El superíndice $^\perp$ denotará el subespacio ortogonal del referido.

Tanto para vectores como para matrices se emplearán las normas euclídeas:

$$||\mathbf{x}||_2 = \sqrt{\mathbf{x}^T\mathbf{x}} = \sqrt{(\mathbf{x},\mathbf{x})} = \sqrt{\sum_{i=1}^{n} x_i^2}$$

$$||\mathbf{A}||_2 = \sqrt{\operatorname{tr}\mathbf{A}^T\mathbf{A}} = \sqrt{\sum_{i=1}^{m}\sum_{j=1}^{n} a_{ij}^2}$$

donde $\mathbf{A}$ es de orden $n \times m$ y $\operatorname{tr}\mathbf{M} = \sum_{i=1}^{n} m_{ii}$ es la traza de la matriz cuadrada $\mathbf{M}$, de orden $n \times n$. La norma matricial dada $||\cdot||_2$ es compatible con la definición de norma matricial:

$$||\mathbf{A}|| = \sup_{\mathbf{y}\neq 0} \frac{||\mathbf{A}\mathbf{y}||}{||\mathbf{y}||} \tag{1.1}$$

Las propiedades siguientes de la traza serán de utilidad más adelante:

$$\operatorname{tr}(\mathbf{A}+\mathbf{B}) = \operatorname{tr}\mathbf{A} + \operatorname{tr}\mathbf{B}$$

$$\operatorname{tr}\mathbf{ABC} = \operatorname{tr}\mathbf{BCA} = \operatorname{tr}\mathbf{CAB} \tag{1.2}$$

Representaremos por $\operatorname{vec}\mathbf{A} = (a_{11}\ldots a_{1n}\ldots a_{m1}\ldots a_{mn})^T$ al vector concatenación de $\mathbf{A}_{m\times n}$ por filas.

A continuación se introducen de manera más rigurosa algunas definiciones y propiedades de utilidad posterior. Comenzamos relacionando las propiedades espectrales de una matriz $\mathbf{X}$ con las de sus productos cruzados y externos, $\mathbf{X}^T\mathbf{X}$ y $\mathbf{X}\mathbf{X}^T$, respectivamente[10]:

Proposición 1.4.1. *Sea $\mathbf{X}_{n\times p}$ con $n \geq p$. Entonces:*

1. *$\mathbf{X}\mathbf{X}^T$ y $\mathbf{X}^T\mathbf{X}$ son semidefinidas positivas.*
2. *Los autovalores de $\mathbf{X}\mathbf{X}^T$ y $\mathbf{X}^T\mathbf{X}$ son reales no negativos.*
3. *Los autovalores positivos de $\mathbf{X}\mathbf{X}^T$ lo son de $\mathbf{X}^T\mathbf{X}$ y al revés.*
4. *$n \geq p \geq \operatorname{rg}\mathbf{X}\mathbf{X}^T = \operatorname{rg}\mathbf{X}^T\mathbf{X} = \operatorname{rg}\mathbf{X} = \operatorname{rg}\mathbf{X}^T$*

Demostración. Las demostraciones se dan sólo para $\mathbf{X}^T\mathbf{X}$. Para $\mathbf{X}\mathbf{X}^T$ son análogas, sustituyendo $\mathbf{X}$ por $\mathbf{X}^T$ y al revés.

1. $\mathbf{v}^T\mathbf{X}^T\mathbf{X}\mathbf{v} = (\mathbf{X}\mathbf{v})^2 \geq 0 \quad \forall \mathbf{v} \in \mathbb{R}^p$
2. Por ser $\mathbf{X}^T\mathbf{X}$ simétrica sus autovalores son reales. Además, sea $\mathbf{u} \neq \mathbf{0}$ autovector de $\mathbf{X}^T\mathbf{X}$ con autovalor λ:
$$\mathbf{X}^T\mathbf{X}\mathbf{u} = \lambda\mathbf{u} \Rightarrow \mathbf{u}^T\mathbf{X}^T\mathbf{X}\mathbf{u} = \mathbf{u}^T\lambda\mathbf{u} = \lambda||\mathbf{u}||^2 \Rightarrow \lambda = \frac{||\mathbf{X}\mathbf{u}||^2}{||\mathbf{u}||^2} \geq 0$$

[10] Recordemos que una matriz cuadrada $\mathbf{X}$ es semidefinida positiva si, para cualquier vector $\mathbf{y}$, $\mathbf{y}^T\mathbf{X}\mathbf{y} \geq 0$. Si la desigualdad es estricta, la matriz es definida positiva. Invirtiendo las desigualdades se obtienen las definiciones de matriz semidefinida negativa y definida negativa.

3. Supongamos que existe $\lambda > 0$ autovalor de $\mathbf{X}^T\mathbf{X}$ y sea $\mathbf{u} \neq \mathbf{0}$ un autovector suyo:

$$\mathbf{X}^T\mathbf{X}\mathbf{u} = \lambda\mathbf{u} \neq \mathbf{0} \Rightarrow \left\{ \begin{array}{rcl} \mathbf{X}\mathbf{X}^T\mathbf{X}\mathbf{u} & = & \lambda\mathbf{X}\mathbf{u} \\ \mathbf{X}\mathbf{u} & \neq & \mathbf{0} \end{array} \right\}$$

Luego $\mathbf{X}\mathbf{u}$ es autovector de $\mathbf{X}\mathbf{X}^T$ con autovalor λ.

4. Veamos que $\ker\mathbf{X} = \ker\mathbf{X}^T\mathbf{X}$. Sea $\mathbf{v} \neq \mathbf{0} \in \mathbb{R}^p$:

$$\mathbf{X}\mathbf{v} = \mathbf{0} \Rightarrow \mathbf{X}^T\mathbf{X}\mathbf{v} = \mathbf{0} \Rightarrow \ker\mathbf{X} \subset \ker\mathbf{X}^T\mathbf{X}$$

$$\mathbf{X}^T\mathbf{X}\mathbf{v} = \mathbf{0} \Rightarrow 0 = \mathbf{v}^T\mathbf{X}^T\mathbf{X}\mathbf{v} = \|\mathbf{X}\mathbf{v}\|^2 \Rightarrow \mathbf{X}\mathbf{v} = \mathbf{0} \Rightarrow \ker\mathbf{X}^T\mathbf{X} \subset \ker\mathbf{X}$$

Luego $\dim(\operatorname{im}\mathbf{X}) = \operatorname{rg}\mathbf{X} = \operatorname{rg}\mathbf{X}^T\mathbf{X}$, ya que $\dim(\ker\mathbf{X}) + \dim(\operatorname{im}\mathbf{X}) = p$ siempre. Análogamente se prueba que $\ker\mathbf{X}^T = \ker\mathbf{X}\mathbf{X}^T$, sustituyendo $\mathbf{X}$ por $\mathbf{X}^T$. Aplicando $\operatorname{rg}\mathbf{X} = \operatorname{rg}\mathbf{X}^T$ se obtiene el enunciado. □

La propiedad anterior permite, si $p < n$, pasar de un problema de autovalores de tamaño $\mathcal{O}(n^2)$ a uno $\mathcal{O}(p^2)$, es decir, obtener los autovalores de $\mathbf{X}\mathbf{X}^T$ a partir de los de $\mathbf{X}^T\mathbf{X}$ y los autovectores de $\mathbf{X}\mathbf{X}^T$ asociados a autovalores positivos a partir de los de $\mathbf{X}^T\mathbf{X}$.

Observación 1.4.1. Si $\mathbf{X}\mathbf{u} = \mathbf{0}$, entonces $\mathbf{X}^T\mathbf{X}\mathbf{u} = \mathbf{0}$, luego $\mathbf{u}$ es autovector de $\mathbf{X}^T\mathbf{X}$ asociado a $\lambda = 0$, pero no nos produce un autovector no trivial (no nulo) $\mathbf{X}\mathbf{u}$ de $\mathbf{X}\mathbf{X}^T$ asociado a $\lambda = 0$. Si $\mathbf{X}_{n\times p} = (\mathbf{x}_1, \dots, \mathbf{x}_p)$ y $\langle\mathbf{x}_i\rangle = \mathbf{0}$ (es decir, la media de las columnas de $\mathbf{X}$ es $\mathbf{0}$), entonces $\mathbf{X}\mathbf{1} = \mathbf{0}$ para $\mathbf{1} = (1, 1, \dots, 1)^T$. En general, si $\operatorname{rg}\mathbf{X}^T\mathbf{X} = p - 1$, este método no nos permite obtener ningún autovector asociado a $\lambda = 0$ (pero sí a los autovalores positivos).

Ahora se introducen los conceptos de matriz de proyección y de matriz pseudoinversa, a partir de la descomposición en valores singulares de una matriz.

Definición 1.4.1 (Matriz de proyección). Una matriz $\Pi_{n\times n}$ se dice de proyección si es simétrica ($\Pi^T = \Pi$) e idempotente ($\Pi^2 = \Pi$). Como caso particular importante, la proyección según la dirección del vector $\mathbf{v}$, $\Pi_{\mathbf{v}}$, debe cumplir:

$$(\mathbf{x} - \|\Pi_{\mathbf{v}}\mathbf{x}\|\mathbf{v}) \perp \mathbf{v}) \Leftrightarrow \mathbf{v}^T(\mathbf{x} - \|\Pi_{\mathbf{v}}\mathbf{x}\|\mathbf{v}) = 0 \Leftrightarrow \Pi_{\mathbf{v}}\mathbf{x} = \frac{\mathbf{v}^T\mathbf{x}}{\mathbf{v}^T\mathbf{v}}\mathbf{v} = \frac{\mathbf{v}\mathbf{v}^T}{\mathbf{v}^T\mathbf{v}}\mathbf{x}$$

Luego $\Pi_{\mathbf{v}} = \frac{\mathbf{v}\mathbf{v}^T}{\mathbf{v}^T\mathbf{v}}$. Si $\mathbf{v}_1 = \frac{\mathbf{v}}{\|\mathbf{v}\|}$ es el vector unitario en la dirección de $\mathbf{v}$: $\Pi_{\mathbf{v}} = \Pi_{\mathbf{v}_1} = \mathbf{v}_1\mathbf{v}_1^T$

Teorema 1.4.1 (Teorema espectral). *Sea* $\mathbf{A}$ *una matriz real simétrica.* $\mathbf{A}$ *admite una diagonalización ortogonal de la forma* $\mathbf{A} = \mathbf{U}\boldsymbol{\Lambda}\mathbf{U}^T$*, donde* $\boldsymbol{\Lambda} = \operatorname{diag}(\lambda_1, \dots, \lambda_n)$ *y* $\mathbf{U} = (\mathbf{u}_1, \dots, \mathbf{u}_n)$*.* $\{\mathbf{u}_1, \dots, \mathbf{u}_n\}$ *es base ortonormal de autovectores de* $\mathbf{A}$ *asociados a sus autovalores* $\{\lambda_i\}$*.* $\mathbf{A}$*,* $\mathbf{U}$ *y* $\boldsymbol{\Lambda}$ *son de orden* $n \times n$*.*

Demostración. Por su longitud la omitimos; puede encontrarse en cualquier libro de álgebra lineal (por ejemplo, en [53, pág. 309]). □

El teorema espectral nos permite obtener el siguiente resultado fundamental:

Teorema 1.4.2 (Descomposición en valores singulares (DVS)). *Sea* $\mathbf{A}_{m\times n}$ *tal que* $\operatorname{rg}\mathbf{A} = r$*. Se puede factorizar* $\mathbf{A}$ *como:*

$$\mathbf{A} = \mathbf{U}\mathbf{S}\mathbf{V}^T = \sum_{i=1}^{r} \mathbf{u}_i\, s_i\, \mathbf{v}_i^T$$

donde $\mathbf{U}_{m\times m} = (\mathbf{u}_1, \dots, \mathbf{u}_m)$ *y* $\mathbf{V}_{n\times n} = (\mathbf{v}_1, \dots, \mathbf{v}_n)$ *son matrices ortogonales y* $\mathbf{S}_{m\times n}$ *tiene los valores singulares* $s_i > 0, 1 \leq i \leq r$ *de* $\mathbf{A}$ *en su diagonal y 0 en el resto.*

Además:

- $\{\mathbf{u}_1, \dots, \mathbf{u}_r\}$ *es base de* $L(\mathbf{A})$*,* $\{\mathbf{u}_{r+1}, \dots, \mathbf{u}_m\}$ *de* $\ker\mathbf{A}^T$*,* $\{\mathbf{v}_1, \dots, \mathbf{v}_r\}$ *de* $L(\mathbf{A}^T)$ *y* $\{\mathbf{v}_{r+1}, \dots, \mathbf{v}_m\}$ *de* $\ker\mathbf{A}$*.*
- $\mathbf{u}_i$ *es autovector de* $\mathbf{A}\mathbf{A}^T$ *asociado a su autovalor* s_i^2 *porque* $\mathbf{A}\mathbf{A}^T = \mathbf{U}\mathbf{S}\mathbf{V}^T\mathbf{V}\mathbf{S}^T\mathbf{U}^T = \mathbf{U}\mathbf{S}\mathbf{S}^T\mathbf{U}^T$ *que es la diagonalización (ver el teorema espectral 1.4.1) de* $\mathbf{A}\mathbf{A}^T$*. Análogamente,* $\mathbf{v}_i$ *es autovector de* $\mathbf{A}^T\mathbf{A}$ *asociado a su autovalor* s_i^2*.*

La descomposición es única salvo permutaciones y combinaciones lineales de columnas de $\mathbf{U}$ *y* $\mathbf{V}$ *cuyos valores singulares sean iguales.*

Demostración. Por el teorema espectral 1.4.1 podemos hacer $\mathbf{A}^T\mathbf{A} = \mathbf{V}\mathbf{\Lambda}\mathbf{V}^T$ donde $\mathbf{\Lambda} = \operatorname{diag}(\lambda_i)$ contiene los autovalores de $\mathbf{A}^T\mathbf{A}$ ($\lambda_i > 0$ para $1 \le i \le r$, $\lambda_i = 0$ para $r \le i \le n$) y $\mathbf{V} = (\mathbf{v}_1, \dots, \mathbf{v}_n)$ sus autovectores normalizados asociados. Además $\|\mathbf{A}\mathbf{v}_i\|^2 = \mathbf{v}_i^T\mathbf{A}^T\mathbf{A}\mathbf{v}_i = \mathbf{v}_i^T\lambda_i\mathbf{v}_i = \lambda_i$. Construyamos $s_i = \sqrt{\lambda_i}$ en la diagonal de $\mathbf{S}$ y completemos el resto de $\mathbf{S}$ con ceros. Hagamos $\mathbf{u}_i = \mathbf{A}\mathbf{v}_i/s_i$ para $1 \le i \le r$, con lo que $\{\mathbf{u}_1, \dots, \mathbf{u}_r\}$ será ortonormal:

$$\mathbf{u}_i^T\mathbf{u}_j = \frac{\mathbf{v}_i^T\mathbf{A}^T\mathbf{A}\mathbf{v}_j}{s_i s_j} = \frac{\lambda_i}{s_i s_j}\mathbf{v}_i^T\mathbf{v}_j = \delta_{ij}$$

Ampliando los $\mathbf{u}_i$ anteriores a una base ortonormal de $\mathbb{R}^n$ (por el procedimiento de Gram-Schmidt, por ejemplo) obtengamos $\mathbf{U}$. Entonces, la entrada i, j de la matriz $\mathbf{U}^T\mathbf{A}\mathbf{V}$ será:

$$\mathbf{u}_i^T\mathbf{A}\mathbf{v}_j = \begin{cases} \mathbf{v}_i^T\mathbf{A}^T\mathbf{A}\mathbf{v}_j/s_i = \lambda_i\mathbf{v}_i^T\mathbf{v}_j/s_i = 0 & \text{si } j > r \\ \mathbf{u}_i^T\mathbf{S}\mathbf{u}_j = \begin{cases} 0 & \text{si } i \ne j \le r \\ s_j & \text{si } i = j \le r \end{cases} \end{cases}$$

Luego $\mathbf{U}^T\mathbf{A}\mathbf{V} = \mathbf{S} \Leftrightarrow \mathbf{A} = \mathbf{U}\mathbf{S}\mathbf{V}^T$. □

La DVS de una matriz $\mathbf{A}_{m\times n}$ permite ver que $\mathbf{A}$ transforma una esfera unidad n-dimensional en un elipsoide m-dimensional cuyos ejes principales coinciden con los valores singulares de $\mathbf{A}$.

Definición 1.4.2 (Número de condición). El valor $c(\mathbf{A}) = \max\{s_i\}/\min\{s_i\}, 1 \le i \le r$ se llama número de condición de la matriz $\mathbf{A}$. Si es muy grande, es decir, del orden del inverso de la precisión del ordenador empleado (aproximadamente $3 \cdot 10^{-8}$ en precisión simple y 10^{-15} en doble), se dice que la matriz está *mal condicionada* y se producirán errores de redondeo grandes.

Proposición 1.4.2. *Si λ es autovalor de $\mathbf{A}$, λ^2 lo es de $\mathbf{A}^2$.*

Demostración. $\mathbf{A}\mathbf{u} = \lambda\mathbf{u} \Rightarrow \mathbf{A}^2\mathbf{u} = \mathbf{A}\lambda\mathbf{u} = \lambda^2\mathbf{u}$. □

Proposición 1.4.3. *Sea $\mathbf{A}_{m\times n}$ y $c(\mathbf{A}) = c$ su número de condición. Entonces, $c(\mathbf{A}\mathbf{A}^T) = c(\mathbf{A}^T\mathbf{A}) = c^2$.*

Demostración. $c(\mathbf{A}) = c = \max\{s_i\}/\min\{s_i\}$ con $s_i = \sqrt{\lambda_i}, 1 \le i \le r, r = \operatorname{rg}\mathbf{A}$ y $\lambda_i > 0$ son los autovectores positivos de $\mathbf{A}^T\mathbf{A}$. Por ser $\mathbf{A}^T\mathbf{A}$ simétrica, la proposición 1.4.2 implica que los autovalores de $(\mathbf{A}^T\mathbf{A})^T(\mathbf{A}^T\mathbf{A}) = (\mathbf{A}^T\mathbf{A})^2$ son los de $\mathbf{A}^T\mathbf{A}$ al cuadrado, luego $c(\mathbf{A}^T\mathbf{A}) = c^2$. Para $\mathbf{A}\mathbf{A}^T$ la prueba es análoga. □

Observación 1.4.2. La matriz de covarianzas de un conjunto de p vectores $\{\mathbf{y}\}$, $\mathbf{\Sigma} = \frac{1}{p}\mathbf{Y}\mathbf{Y}^T - \overline{\mathbf{y}}\overline{\mathbf{y}}^T = \frac{1}{p}\mathbf{X}\mathbf{X}^T$, con $\mathbf{x} = \mathbf{y} - \overline{\mathbf{y}}$, cumple $c(\mathbf{\Sigma}) = c(\mathbf{X}\mathbf{X}^T)$.

Observación 1.4.3. Desde el punto de vista del cálculo numérico (cf. [43, págs. 51–63]) conviene tener en cuenta lo siguiente:

- La ortogonalización de Gram-Schmidt es numéricamente nefasta por la acumulación de errores de redondeo. Resulta mejor descomponer la matriz de vectores dada en valores singulares y usar $\mathbf{U}$ como base ortonormal.
- La inversión de $\mathbf{A}$ (caso de ser posible) es también más eficiente y precisa a partir de la DVS de $\mathbf{A}$: $\mathbf{A}^{-1} = \mathbf{V}\mathbf{S}^{-1}\mathbf{U}^T$, pues $\mathbf{U}$ y $\mathbf{V}$ únicamente se trasponen y solamente pueden ser problemáticos los elementos $1/s_i$ de $\mathbf{S}^{-1}$ cuando el número de condición $c(\mathbf{A})$ se aproxima a la precisión del ordenador.

Definición 1.4.3 (Penrose). La matriz pseudoinversa de $\mathbf{A}_{m\times n}$, o inversa generalizada o inversa de Moore-Penrose, que denotamos por $\mathbf{A}^+$, es aquella matriz de orden $n \times m$ que cumple las condiciones:

$$\mathbf{A}\mathbf{A}^+,\ \mathbf{A}^+\mathbf{A} \quad \text{son simétricas} \tag{1.3a}$$

$$\mathbf{A}\mathbf{A}^+\mathbf{A} = \mathbf{A} \tag{1.3b}$$

$$\mathbf{A}^+\mathbf{A}\mathbf{A}^+ = \mathbf{A}^+ \tag{1.3c}$$

Puede demostrarse [7] que $\mathbf{A}^+$ existe para cualquier $\mathbf{A}$ y es única. Una manera eficiente de calcularla, como se comprueba de inmediato por sustitución en las condiciones (1.3), es a partir de la DVS de $\mathbf{A}$: $\mathbf{A} = \mathbf{U}\mathbf{S}\mathbf{V}^T \Rightarrow \mathbf{A}^+ = \mathbf{V}\mathbf{S}^+\mathbf{U}^T$, donde $\mathbf{S}^+$ es igual a $\mathbf{S}^T$ pero con los elementos s_i de $\mathbf{S}$ sustituidos por $1/s_i$. También puede demostrarse que $\mathbf{A}^T$ es el primer término en el desarrollo en serie de $\mathbf{A}$.

Las dos propiedades siguientes, relacionadas con la matriz pseudoinversa, son importantes para la sección 3.3.1:

Proposición 1.4.4. $\mathbf{AA}^+ = \Pi_{L(\mathbf{A})}$, *la matriz de proyección sobre el subespacio generado por las columnas de* $\mathbf{A}$.

Demostración. Probemos primero la simetría e idempotencia de $\mathbf{AA}^+$:

- $\mathbf{AA}^+$ es simétrica por (1.3a).
- $(\mathbf{AA}^+)^2 = \mathbf{AA}^+\mathbf{AA}^+ = \mathbf{AA}^+$ por (1.3b) ó (1.3c).

Luego $\mathbf{AA}^+$ es una matriz de proyección. Para ver que es justamente $\Pi_{L(\mathbf{A})}$, veamos que $\mathbf{x} \in L(\mathbf{A}) \Rightarrow \mathbf{AA}^+\mathbf{x} = \mathbf{x}$ y $\mathbf{x} \in L(\mathbf{A})^\perp \Rightarrow \mathbf{AA}^+\mathbf{x} = \mathbf{0}$:

- $\mathbf{AA}^+\mathbf{A} = \mathbf{A}$ por (1.3b).
- $\mathbf{x} \in L(\mathbf{A})^\perp \Rightarrow \mathbf{A}^T\mathbf{x} = \mathbf{0} \Rightarrow \mathbf{AA}^+\mathbf{x} = (\mathbf{AA}^+)^T\mathbf{x} = (\mathbf{A}^+)^T\mathbf{A}^T\mathbf{x} = \mathbf{0}$. □

Observación 1.4.4. $\Pi_{L(\mathbf{A})} = \mathbf{AA}^+ = \sum_{i=1}^r \mathbf{u}_i\mathbf{u}_i^T$, usando $\mathbf{A} = \mathbf{USV}^T$, con $\{\mathbf{u}_1, \ldots, \mathbf{u}_r\}$ base ortonormal de $L(\mathbf{A})$.

Proposición 1.4.5. *Sea el sistema* $\mathbf{Ax} = \mathbf{b}$. $\mathbf{x} = \mathbf{A}^+\mathbf{b}$ *es su* solución óptima en el sentido de mínimos cuadrados*: minimiza* $||\mathbf{Ax} - \mathbf{b}||^2$ *si el sistema no admite solución y* $||\mathbf{x}||^2$ *si admite infinitas.*

Demostración. El número de soluciones del sistema $\mathbf{Ax} = \mathbf{b}$ depende de los rangos respectivos de $\mathbf{A}$ y de la matriz ampliada $(\mathbf{A}, \mathbf{b})$, como es sabido por el teorema de Rouché-Frobënius. Veamos cada caso por separado:

- No existe solución, es decir, $\mathbf{b} \notin L(\mathbf{A})$. La proyección ortogonal de $\mathbf{b}$ sobre $L(\mathbf{A})$, que según la proposición 1.4.4 es $\mathbf{AA}^+\mathbf{b}$, es el vector de $L(\mathbf{A})$ de distancia (euclídea) mínima a $\mathbf{b}$. Por tanto, $\mathbf{x} = \mathbf{A}^+\mathbf{b}$ es la solución buscada.
- Existen infinitas soluciones, es decir, $\mathbf{b} \in L(\mathbf{A})$. Supongamos $\mathbf{b} \neq \mathbf{0}$ (si $\mathbf{b} = \mathbf{0}$ la solución óptima en todos los sentidos es $\mathbf{x} = \mathbf{0}$, evidentemente):
  - Cualquier solución $\mathbf{x} \in \mathbb{R}^n$ puede describirse como $\mathbf{x} = \mathbf{x}_1 + \mathbf{x}_2$, con $\mathbf{x}_1 \in \ker \mathbf{A}^\perp$ y $\mathbf{x}_2 \in \ker \mathbf{A}$, es decir, $\mathbf{x}_1 \perp \mathbf{x}_2 \Leftrightarrow \mathbf{x}_1^T\mathbf{x}_2 = 0$.
  - El vector $\mathbf{x}_1$ es único, ya que si suponemos que $\mathbf{x}_3 \neq \mathbf{x}_1, \mathbf{x}_3 \in \ker \mathbf{A}^\perp$ tenemos $\mathbf{Ax}_1 = \mathbf{Ax}_3 = \mathbf{b} \Rightarrow \mathbf{A}(\mathbf{x}_1 - \mathbf{x}_3) = \mathbf{0} \Rightarrow \mathbf{x}_1 - \mathbf{x}_3 = \mathbf{0}$.
  - Veamos que $\mathbf{x}_1 = \mathbf{A}^+\mathbf{b}$: $\mathbf{Ax}_1 = \mathbf{AA}^+\mathbf{b} = \mathbf{b} \neq \mathbf{0}$, por la proposición 1.4.4. Luego $\mathbf{x}_1 = \mathbf{A}^+\mathbf{b}$ es solución y pertenece a $\ker \mathbf{A}^\perp$.
  - Por ser $\mathbf{x}_1 \perp \mathbf{x}_2$, la norma de cualquier solución $\mathbf{x}$ es $||\mathbf{x}||^2 = ||\mathbf{x}_1||^2 + ||\mathbf{x}_2||^2$, que es mínima para $\mathbf{x}_2 = \mathbf{0}$.

  Por tanto, $\mathbf{x} = \mathbf{A}^+\mathbf{b}$ es la solución de norma mínima (y además minimiza también la distancia $||\mathbf{Ax} - \mathbf{b}|| = 0$).
- Si $\mathbf{A}$ es invertible, $\mathbf{x} = \mathbf{A}^+\mathbf{b} = \mathbf{A}^{-1}\mathbf{b}$ es la única solución. □

Corolario 1.4.1. $||\mathbf{I} - \mathbf{AA}^+|| = \min_{\mathbf{B}}||\mathbf{I} - \mathbf{AB}||$.

Demostración. Por la proposición 1.4.5, dado $\mathbf{b}$, $\mathbf{B} = \mathbf{A}^+$ minimiza $||(\mathbf{I} - \mathbf{AB})\mathbf{b}||$ y por tanto minimiza también $\frac{||(\mathbf{I}-\mathbf{AB})\mathbf{b}||}{||\mathbf{b}||}$. El enunciado se sigue de la definición 1.1 de norma. □

Observación 1.4.5. Si $\mathbf{A}_{m\times n}$ es de rango completo, es decir, $\operatorname{rg}\mathbf{A} = \min\{m, n\} = \operatorname{rg}\mathbf{A}^T\mathbf{A} = \operatorname{rg}\mathbf{AA}^T$ (por la proposición 1.4.1), los cálculos son más simples:

$$
\begin{array}{lll}
\mathbf{A}^+ = (\mathbf{A}^T\mathbf{A})^{-1}\mathbf{A}^T & \Pi_{L(\mathbf{A})} = \mathbf{AA}^+ = \mathbf{A}(\mathbf{A}^T\mathbf{A})^{-1}\mathbf{A}^T & \text{para} \quad \operatorname{rg}\mathbf{A} = n \leq m \\
\mathbf{A}^+ = \mathbf{A}^T(\mathbf{AA}^T)^{-1} & \Pi_{L(\mathbf{A})} = \mathbf{AA}^+ = \mathbf{AA}^T(\mathbf{AA}^T)^{-1} = \mathbf{I} & \text{para} \quad \operatorname{rg}\mathbf{A} = m \leq n
\end{array}
$$

como puede comprobarse por sustitución en las condiciones de Penrose (1.3). Igualmente puede verse que si $\mathbf{A}$ es cuadrada e invertible, $\mathbf{A}^{-1} = \mathbf{A}^+$.

A continuación se dan unos resultados que relacionan las varianzas direccionales de un conjunto de p vectores $\{\mathbf{y}\}$ con los autovalores de la matriz $\mathbf{XX}^T$. Veamos primero un resultado sobre la maximización de formas cuadráticas.

Proposición 1.4.6 (Maximización de una forma cuadrática sobre la esfera de radio unidad). *Sea* $\mathbf{y}^T\mathbf{Q}\mathbf{y}$ *una forma cuadrática de matriz* $\mathbf{Q}$ *(simétrica). Entonces,* $\max_{\|\mathbf{y}\|=1} \mathbf{y}^T\mathbf{Q}\mathbf{y} = \lambda_{\max}$ *y ocurre para* $\mathbf{y} = \mathbf{u}_1$*, donde* $\lambda_{\max} = \lambda_1$ *es el autovalor dominante de* $\mathbf{Q}$ *y* $\mathbf{u}_1$ *un autovector unitario asociado a él.*

Demostración. Con ayuda del teorema espectral 1.4.1, diagonalizamos la forma cuadrática: $\mathbf{y}^T\mathbf{Q}\mathbf{y} = \mathbf{y}^T\mathbf{U}\mathbf{\Lambda}\mathbf{U}^T\mathbf{y} = \mathbf{v}^T\mathbf{\Lambda}\mathbf{v} = \sum_{i=1}^n \lambda_i v_i^2$ para $\mathbf{v} = \mathbf{U}^T\mathbf{y}$. $\mathbf{\Lambda} = \text{diag}(\lambda_1, \ldots, \lambda_n)$ contiene los autovalores de $\mathbf{Q}$ (que supondremos, por simplicidad, ordenados: $\lambda_1 \geq \cdots \geq \lambda_n$) y $\mathbf{U} = (\mathbf{u}_1, \ldots, \mathbf{u}_n)$ sus autovectores asociados. Entonces, y teniendo en cuenta que $\|\mathbf{y}\| = 1 = \|\mathbf{v}\|$ (ya que $\mathbf{U}$ es ortogonal y preserva la métrica), es evidente que $\sum_{i=1}^n \lambda_i v_i^2$ es máxima si $v_1 = \pm 1$ y $v_2 = \cdots = v_n = 0$. Por tanto, $\mathbf{y}^T\mathbf{Q}\mathbf{y} = \lambda_{\max} = \lambda_1$ y:

$$\mathbf{v} = \mathbf{U}^T\mathbf{y} = (\pm 1, 0, \ldots, 0)^T \Leftrightarrow \left\{ \begin{array}{lcl} \mathbf{u}_1^T\mathbf{y} & = & \pm 1 \\ \mathbf{u}_i^T\mathbf{y} & = & 0, 1 < i \leq n \end{array} \right\} \Leftrightarrow \mathbf{y} = \pm\mathbf{u}_1 = \pm\mathbf{u}_{\max}$$

□

Observación 1.4.6. Si $\lambda_{\max}$ es múltiple (caso degenerado), hay infinitas posibilidades para $\mathbf{y}$ (cualquier vector unitario en el subespacio de los autovectores de $\lambda_{\max}$ es válido).

Definición 1.4.4 (Varianza direccional). La varianza de los p vectores $\{\mathbf{y}\}$ en la dirección del vector $\mathbf{v} \neq \mathbf{0}$ es la varianza de las normas de las proyecciones ortogonales de cada vector $\mathbf{y}$ en la dirección de $\mathbf{v}$, es decir:

$$\text{var}_{\mathbf{v}}(\mathbf{y}) = \frac{1}{p} \left\langle \|\Pi_{\mathbf{v}}\mathbf{y} - \overline{\Pi_{\mathbf{v}}\mathbf{y}}\|^2 \right\rangle$$

.

Proposición 1.4.7. $\text{var}_{\mathbf{v}}(\mathbf{y}) = \frac{1}{p}\mathbf{v}_1^T\mathbf{X}\mathbf{X}^T\mathbf{v}_1 = \frac{1}{p}\|\mathbf{X}^T\mathbf{v}_1\|^2$, *donde* $\mathbf{v}_1 = \frac{\mathbf{v}}{\|\mathbf{v}\|}$ *es un vector unitario en la dirección de* $\mathbf{v}$ *y* $\mathbf{x}$ *son los vectores* $\mathbf{y}$ *centrados.*

Demostración.

$$\begin{aligned} var_{\mathbf{v}}(\mathbf{y}) = \\ & \frac{1}{p} \left\langle \|\Pi_{\mathbf{v}}\mathbf{y} - \overline{\Pi_{\mathbf{v}}\mathbf{y}}\|^2 \right\rangle = \frac{1}{p} \left\langle \|\Pi_{\mathbf{v}}\mathbf{y} - \Pi_{\mathbf{v}}\overline{\mathbf{y}}\|^2 \right\rangle = \frac{1}{p} \left\langle \|\Pi_{\mathbf{v}}(\mathbf{y} - \overline{\mathbf{y}})\|^2 \right\rangle = \frac{1}{p} \left\langle \|\Pi_{\mathbf{v}}\mathbf{x}\|^2 \right\rangle = \\ & \frac{1}{p} \left\langle \|\Pi_{\mathbf{v}_1}\mathbf{x}\|^2 \right\rangle = \frac{1}{p} \left\langle (\mathbf{v}_1\mathbf{x}^T\mathbf{v}_1)^T(\mathbf{v}_1\mathbf{x}^T\mathbf{v}_1) \right\rangle = \frac{1}{p} \left\langle \mathbf{v}_1^T\mathbf{x}\mathbf{x}^T\mathbf{v}_1 \right\rangle = \frac{1}{p}\mathbf{v}_1^T \left\langle \mathbf{x}\mathbf{x}^T \right\rangle \mathbf{v}_1 = \\ & \frac{1}{p}\mathbf{v}_1^T\mathbf{X}\mathbf{X}^T\mathbf{v}_1 = \frac{1}{p}\|\mathbf{X}^T\mathbf{v}_1\|^2 \end{aligned}$$

□

Proposición 1.4.8. *La mayor varianza direccional es igual al valor del autovalor dominante* $\lambda_1 = \lambda_{\max}$ *de* $\mathbf{X}\mathbf{X}^T$ *y su dirección la de un autovector asociado suyo* $\mathbf{u}_1$*.*

Demostración. $\text{var}_{\mathbf{v}}(\mathbf{y}) = \frac{1}{p}\mathbf{v}_1^T\mathbf{X}\mathbf{X}^T\mathbf{v}_1$ por la proposición 1.4.7, que es máxima si $\mathbf{v}_1$ es autovector de $\mathbf{X}\mathbf{X}^T$ asociado a $\lambda_{\max}$, y vale $\text{var}_{\mathbf{v}}(\mathbf{y}) = \frac{1}{p}\lambda_{\max}$ por la proposición 1.4.6. □

Capítulo 2

Las redes de neuronas artificiales

Una definición general de las redes de neuronas artificiales (RNA) nos tomaría demasiado espacio y además no es necesaria para comprender las herramientas usadas en este trabajo. En este capítulo nos limitaremos a describir las redes perceptrón multicapa, haciendo especial énfasis en las redes autoasociativas. Una introducción más general a las RNAs puede hallarse en el artículo de Lippmann [34] o en cualquier libro sobre el tema [25, 24, 17].

2.1 El perceptrón multicapa

Un perceptrón multicapa es una RNA formada por una capa de unidades o neuronas de entrada (círculos negros en la figura 2.1), una de unidades de salida (círculos blancos) y 0 o más capas internas de unidades internas u ocultas (círculos blancos). Las capas consecutivas están conectadas completamente (al menos en principio), es decir, cada unidad de la capa k recibe entradas de cada una de las unidades de la capa $k-1$. Las conexiones van siempre hacia adelante (*feedforward*), en la dirección entrada–salida. Cada

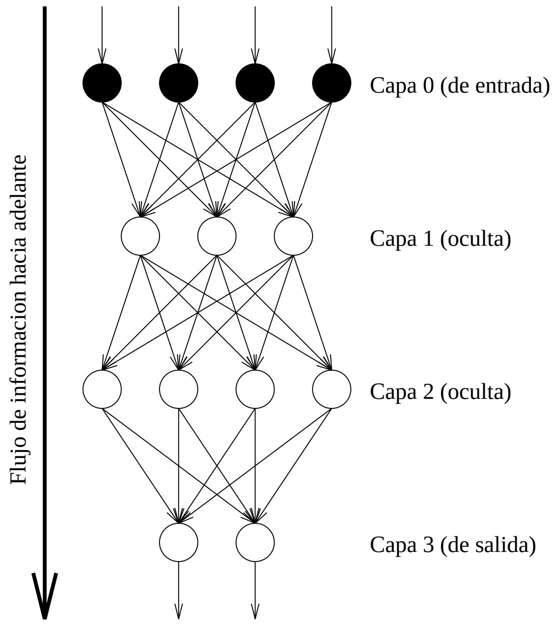

Figura 2.1: Estructura general del perceptrón multicapa (de 3 capas en este caso).

unidad calcula su salida (excepto las de la capa de entrada, que simplemente copian su entrada a la salida) como una suma ponderada por pesos $w_1, \ldots, w_n$ de las n entradas $e_1, \ldots, e_n$ que recibe, a la que se aplica una función real f, llamada *función de activación*:

$$s = f\left(\sum_{i=1}^{n} e_i w_i + w_0\right) = f\left(\sum_{i=0}^{n} e_i w_i\right) \quad \text{con} \quad e_0 = 1$$

El valor w_0 se llama sesgo (*bias*) y puede considerarse como la contribución de una entrada constante de valor 1 conectada a la unidad en cuestión por un peso de valor w_0. El argumento de la función de activación, $\sum_{i=0}^{n} e_i w_i$, se llama *activación* de la unidad.

Los valores de los pesos pueden ser fijos (por haberse calculado a priori) u obtenerse por medio de un proceso, llamado *aprendizaje*, que minimiza cierto criterio[1]. En el caso que a nosotros nos concierne, llamado *aprendizaje supervisado*, las salidas s_i que produce la red en su capa de salida deben ser lo más parecidas posible a unas salidas prefijadas, para unos valores de entrada (llamados *patrones*) dados. Es decir, buscamos que la RNA ajuste lo mejor posible (en el sentido de mínimos cuadrados) una función prefijada $\mathbf{s}_i = F(\mathbf{y}_i)$, con $\mathbf{s}_i \in \mathbb{R}^m$, $\mathbf{y}_i \in \mathbb{R}^n$, $1 \leq i \leq p$ si hay p patrones, n unidades de entrada y m de salida. Por tanto, el aprendizaje de la red debe llevar los pesos a unos valores tales que $E = \left\langle \|\mathbf{s} - F(\mathbf{y})\|^2 \right\rangle$ sea mínimo[2]. Veremos en la sección 2.2 que el llamado algoritmo de *retropropagación* o *regla delta generalizada* [46, 322–328], el algoritmo de aprendizaje más empleado para perceptrones multicapa, minimiza justamente E. Dicho algoritmo requiere pasar iterativamente cada patrón por la red y "retropropagar" unas correcciones dependientes del error producido en la salida hasta que el error es suficientemente pequeño o bien hasta que no puede reducirse más (lo cual no necesariamente indica que se haya alcanzado el mínimo global; puede estarse en un mínimo local o en un punto de silla, tal como se indica en la sección 2.2).

Una propiedad muy deseable del algoritmo de aprendizaje es que sea una regla *local*, es decir, que solamente utilice datos que le están a mano en cada unidad o conexión (salida de unidades adyacentes y valores de pesos que le llegan o que salen de ella). Esto permite que la regla sea implementada en hardware, mediante conexiones y elementos de proceso físicos. Como veremos, hay reglas locales y no locales.

La RNA es, pues, capaz de construir una representación interna (a través de sus pesos) que aproxima la función dada $F : \mathbb{R}^n \longrightarrow \mathbb{R}^m$. Con qué precisión depende del número de capas y del número de unidades de cada capa (aparte de de la propia F). En general, usando el teorema de Kolmogorov, puede demostrarse que bastan 2 capas de pesos en una RNA entrenada con retropropagación para aproximar cualquier función *razonable* (entre las que se encuentran todas las continuas o continuas a trozos) hasta un error cuadrático medio tan pequeño como se desee (Hecht-Nielsen [24, págs. 122–132]), pero no se dice nada del número de unidades necesarias, aunque suele ser enorme. A menudo es más conveniente usar más capas de menos unidades; los números exactos de niveles y de unidades en cada nivel se suelen determinar mediante prueba y error, pues desde el punto de vista teórico resulta muy difícil de analizar.

Cuando las salidas de un perceptrón multicapa coinciden con las entradas, el aprendizaje deja de ser supervisado para llamarse *autosupervisado* (o *no supervisado*, según los autores) y la red se llama *autoasociativa*, pues $F(\mathbf{y}_i) = \mathbf{y}_i \; \forall i = 1, \ldots, n$. Si además la red tiene dos capas de pesos y hacemos que la capa de unidades ocultas tenga un número $h < n$ de unidades, estamos obligando a la red a que construya en dicha capa una representación de dimensión h de los patrones de entrada y a que, a partir de ella, reconstruya lo mejor que pueda (en el sentido de mínimos cuadrados) la entrada. Es decir, el vector $\mathbf{y}$ es *comprimido* al pasar por un canal estrecho (de anchura h). Por esa razón, a estas redes (empleadas, entre otros, por Fleming y Cottrell [15]) se les suele llamar también *redes de compresión* o *redes de codificación* n-h-n. En este trabajo, dado que nos referiremos muchas veces a ellas, las llamaremos redes Ξ para abreviar (donde Ξ alude a la forma de la red, ver la figura 2.2). Los valores de las activaciones de las unidades de la capa oculta pueden tomarse como características que representan al vector de entrada.

En el presente trabajo se emplearán solamente redes *lineales*, es decir, la función de activación será la identidad[3]. La red queda conceptualmente muy simple y es susceptible de un análisis teórico detallado, que se dará en posteriores capítulos.

En forma matricial, la función calculada por la red puede expresarse como $F(\mathbf{y}) = \mathbf{ABy}$ con $\mathbf{B}_{h\times n} = (b_{ij})$ y $\mathbf{A}_{n\times h} = (a_{ij})$. b_{ij} es valor del peso que conecta la unidad j de entrada con la i de la capa oculta, y a_{ki} es el valor del peso que conecta la unidad i de la capa oculta con la k de la de salida. La función de error será $E = \left\langle \|\mathbf{y} - F(\mathbf{y})\|^2 \right\rangle = \left\langle \|(\mathbf{I} - \mathbf{W})\mathbf{y}\|^2 \right\rangle$ con $\mathbf{W} = \mathbf{AB}$. El proceso de aprendizaje puede reformularse como $\min_{\mathbf{A},\mathbf{B}} E$. Obsérvese que, a menos que $h = n$, $\mathbf{AB}$ no puede ser igual a la identidad $\mathbf{I}_{n\times n}$, pues rg $\mathbf{AB} \leq h$ (por la proposición 1.4.1).

[1] Aquí se pone de manifiesto una diferencia fundamental entre las RNAs y los ordenadores convencionales: mientras que a estos hay que proporcionarles, en la forma de un programa, una secuencia de instrucciones perfectamente determinada que —supuestamente— resuelve la tarea encomendada, a aquéllas se las entrena con un conjunto de patrones seleccionados y es la red la que encuentra —durante el aprendizaje— una representación interna que resuelve la tarea.

[2] Aquí, la suma $\langle\cdot\rangle$ va extendida a todos los patrones.

[3] Dos tipos muy usados de función de activación son la sigmoide, $f(x) = \frac{1}{1+e^{-x}} \in [0, 1]$, y la arcotangente hiperbólica, $\operatorname{th} x = \frac{e^x - e^{-x}}{e^x + e^{-x}} \in [-1, 1]$

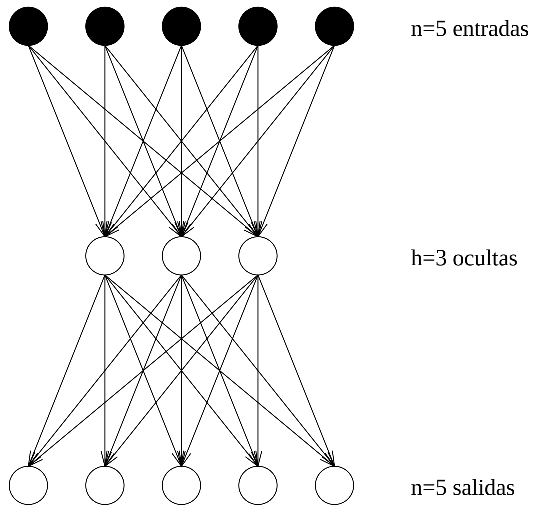

Figura 2.2: Red autoasociativa, de compresión o red Ξ.

2.1.1 Consideraciones sobre las redes lineales y las no lineales

Existe una tendencia a considerar poco interesantes a las redes lineales, comparadas con las no lineales. De acuerdo con Baldi y Hornik [4], los dos argumentos principales que justifican esta actitud son los siguientes:

- La función identidad es un caso particular de función no lineal. Por tanto, lo que una red lineal haga puede hacerlo también una no lineal.
- Una red lineal de H capas es equivalente a una red lineal de una sola capa si multiplicamos las matrices correspondientes a cada capa: $\mathbf{A}_1\mathbf{A}_2\ldots\mathbf{A}_H = \mathbf{A}$.

Los argumentos anteriores son ciertos, pero un tanto sofísticos:

- Las redes lineales son un caso particular de las no lineales, pero muy importante, porque permite en muchos casos analizar la red desde un punto de vista formal y predecir sus resultados, capacidad y limitaciones (tal como se hace en el presente trabajo). Algo parecido sucede en la teoría de ecuaciones diferenciales, en la que se concede gran importancia al caso de las ecuaciones lineales por ser posible su tratamiento matemático y desde ahí poder abordar el caso no lineal general; además, el caso no lineal puede reducirse o simplificarse por linealización (método que hasta ahora no se ha practicado mucho en las RNAs, a pesar del hecho observado de que muchas unidades de redes no lineales operan en su rango lineal la mayoría del tiempo[4], tanto en redes artificiales como en ciertos mecanismos biológicos).
- Por otro lado, las redes lineales presentan una relación muy importante con técnicas fundamentales de estadística y de proceso de señales, como el análisis de componentes principales, la regresión lineal y la transformada de Karhunen-Loève, por citar algunos.
- Además, el paso $\mathbf{A}_1\mathbf{A}_2\ldots\mathbf{A}_H = \mathbf{A}$ no contempla el hecho de que la matriz $\mathbf{A}$ está restringida, es decir, sus elementos no pueden adoptar cualquier valor, porque el rango de $\mathbf{A}$ es menor o igual que el rango más pequeño de las $\mathbf{A}_i$ (que es menor o igual que el número de unidades en la capa i-ésima de la red). Esto impide, por ejemplo, que $\mathbf{A} = \mathbf{I}$ para las redes Ξ si $h < n$.
- Finalmente, si la función F que desea aproximarse con la RNA es lineal, no se gana nada usando unidades no lineales; es decir, el error alcanzado usando unidades no lineales no es menor que el alcanzado usando exclusivamente unidades lineales (Bourlard and Kamp [9]). Esto ha sido comprobado por diversas simulaciones en la práctica (Cottrell *et al.* [12]).

[4] Según Fleming y Cottrell [15], y desde un punto de vista empírico, el empleo de una función de activación no lineal —p. ej. la sigmoide, $f(x) = 1/(1+e^{-x})$— no supone gran diferencia en las redes Ξ, porque observan que, una vez el aprendizaje ha terminado, la activación de las unidades no recorre uniformemente su intervalo de variación sino que tiende a concentrarse en la región lineal de la sigmoide (aproximadamente $[-0.8, 0.8]$). Nosotros realizamos unas pocas pruebas usando la sigmoide y constatamos que frecuentemente el punto de convergencia daba un valor superior para E al de la misma red pero lineal, lo cual sugiere que dicho punto era un mínimo local (o un punto de silla). Además, el aprendizaje era incluso más lento que con la red lineal.

2.2 Redes de neuronas artificiales y optimización

2.2.1 El problema general de optimización no lineal sin restricciones

El problema general de optimización no lineal sin restricciones se puede expresar como:

$$\min f(\mathbf{y}) \qquad \mathbf{y} \in \mathbb{R}^n \tag{2.1}$$

donde la función real $f : \mathbb{R}^n \longrightarrow \mathbb{R}$ es la función objetivo que se desea minimizar sobre el espacio de búsqueda $\mathbb{R}^n$. Una dificultad en la resolución de este problema la plantea la propia función f: la existencia de mínimos locales y de puntos de silla.

2.2.2 El vector gradiente

Para la sección 2.2.5, será necesario hacer uso del concepto de vector *gradiente* de la función f (que supondremos derivable), definido como:

$$\nabla f(\mathbf{y}) = \left(\frac{\partial f(\mathbf{y})}{\partial \mathbf{y}_1}, \ldots, \frac{\partial f(\mathbf{y})}{\partial \mathbf{y}_n} \right)^T$$

El vector gradiente verifica las siguientes propiedades:

- $\nabla f(\mathbf{y}) = \mathbf{0}$ si $\mathbf{y}$ es un punto estacionario de f (máximos o mínimos, globales o locales, y puntos de silla).

- $\nabla f(\mathbf{y})$ apunta en la dirección en la que *localmente* (es decir, en un entorno de $\mathbf{y}$) f crece más rápidamente. En efecto, si desarrollamos f en serie de Taylor en torno a $\mathbf{y}$ hasta el primer orden:

$$f(\mathbf{y} + \Delta\mathbf{y}) = f(\mathbf{y}) + (\nabla f(\mathbf{y}), \Delta\mathbf{y}) + \mathcal{O}(\Delta\mathbf{y}^2) \tag{2.2}$$

  El producto escalar $(\nabla f(\mathbf{y}), \Delta\mathbf{y})$ será máximo, para $\|\Delta\mathbf{y}\|$ constante, cuando $\Delta\mathbf{y}$ apunte en el sentido de $\nabla f(\mathbf{y})$, y mínimo cuando lo haga en la contraria.

- Es perpendicular a las líneas de nivel de f, ya que para que $f(\mathbf{y} + \Delta\mathbf{y}) = f(\mathbf{y})$ en la ecuación (2.2) debe darse $\nabla f(\mathbf{y}) \perp \Delta\mathbf{y}$.

2.2.3 Funciones convexas

Un caso particular de funciones reales son las convexas. f es convexa sobre su dominio $R \subset \mathbb{R}^n$ si para cualesquiera puntos $\mathbf{x}_1, \mathbf{x}_2 \in R$:

$$f(\lambda\mathbf{x}_1 + (1-\lambda)\mathbf{x}_2) \leq \lambda f(\mathbf{x}_1) + (1-\lambda) f(\mathbf{x}_2)$$

donde $\lambda \in [0,1]$. f es estrictamente convexa si, para $\mathbf{x}_1 \neq \mathbf{x}_2$, la desigualdad es estricta. Cambiando las desigualdades ($\leq$ por $\geq$) se tienen definiciones análogas para la concavidad. Una función f diferenciable y convexa tiene las siguientes propiedades:

- $f(\mathbf{x}_2) - f(\mathbf{x}_1) \geq (\nabla f(\mathbf{x}_1))^T (\mathbf{x}_2 - \mathbf{x}_1) \quad \forall \mathbf{x}_1, \mathbf{x}_2 \in R$.

- La matriz hessiana $\nabla^2 f(\mathbf{x}) = \mathbf{H} = (h_{ij})$ de f, definida a partir de las derivadas segundas de f con respecto a $\mathbf{x}$ como:

$$h_{ij}(\mathbf{x}) = \left. \frac{\partial^2 f}{\partial x_i \partial x_j} \right|_{\mathbf{x}} \quad 1 \leq i, j \leq n$$

  es semidefinida positiva para cualquier $\mathbf{x} \in R$ (definida positiva si f es estrictamente convexa).

- f no tiene mínimos ni máximos locales en R, es decir, si f alcanza un mínimo (o máximo) local en $\mathbf{x}$, también lo es global. Si f es estrictamente convexa, el mínimo (o máximo) se alcanza a lo sumo una vez.

2.2.4 Función objetivo asociada al aprendizaje en redes Ξ

El proceso de entrenamiento de una red Ξ con h unidades en su capa oculta resuelve el problema de optimización (2.1) para la función objetivo suma de errores cuadráticos:

$$E(\mathbf{A}, \mathbf{B}) = \left\langle \|\mathbf{y} - \mathbf{A}\mathbf{B}\mathbf{y}\|^2 \right\rangle$$

donde $\mathbf{y}_1, \dots, \mathbf{y}_p$ son los patrones de entrada (vectores de orden $n \times 1$) y $\mathbf{A}$ y $\mathbf{B}$ son matrices de orden $n \times h$ y $h \times n$, respectivamente. El espacio de búsqueda es $\mathbb{R}^{2nh}$. Como se verá en la sección 3.3.1, las propiedades de convexidad de E nos permiten demostrar que $E(\mathbf{A}, \mathbf{B})$ tiene la ventaja de no presentar mínimos locales, aunque sí muchos puntos de silla; esto, unido a su carácter *cuasicuadrático* (ver el apéndice A.1), hace que sea una función relativamente fácil de tratar, tanto desde el punto de vista teórico como numérico.

2.2.5 Métodos del gradiente

De entre los distintos tipos de métodos que existen[5] para resolver el problema (2.1), el grupo de los llamados *métodos del gradiente* utiliza la primera derivada de la función objetivo en los cálculos. Son métodos iterativos, en los que la transición del punto $\mathbf{y}^{(t)}$ al $\mathbf{y}^{(t+1)}$ viene dada por:

$$\mathbf{y}^{(t+1)} = \mathbf{y}^{(t)} + \Delta\mathbf{y}^{(t)} = \mathbf{y}^{(t)} + \eta^{(t)}\mathbf{s}^{(t)} \tag{2.3}$$

Es decir, se desplaza $\mathbf{y}^{(t)}$ en la dirección de $\mathbf{s}^{(t)}$ una distancia $\eta^{(t)}\|\mathbf{s}^{(t)}\|$. La elección de $\eta^{(t)}$ y $\mathbf{s}^{(t)}$ (así como de la norma) en cada paso caracteriza los distintos métodos del gradiente. En particular, el método de *la mayor pendiente* (*steepest descent*) hace $\mathbf{s}^{(t)} = -\nabla f(\mathbf{y}^{(t)})$; $\eta^{(t)} > 0$ puede ser constante (o poder ser ajustado en cada paso), o bien puede obtenerse minimizando $f(\mathbf{y}^{(t)} - \eta^{(t)}\nabla f(\mathbf{y}^{(t)}))$ —es decir, minimizando f a lo largo de la línea descrita por la ecuación (2.3)— mediante un método unidimensional (búsqueda de Fibonacci, razón áurea, etc.). En el primer caso (η fijo o ajustable), η no debe ser ni demasiado pequeño (ya que el método empleará muchos ciclos) ni demasiado grande (pues la sucesión se hace oscilante).

Convergencia de los métodos del gradiente

Si η es suficientemente pequeño, o se obtiene por minimización unidimensional, un método de descenso estricto según la mayor pendiente termina en cualquier punto estacionario (para el cual el gradiente se anula), sea un mínimo o un punto de silla. En este último caso es necesario salir de él mediante otro método distinto.

Son métodos de convergencia muy lenta. Para verlo, supongamos que la función objetivo puede escribirse de la forma (más adelante veremos que $E(\mathbf{A}, \mathbf{B})$ se reduce a ella):

$$f(\mathbf{y}) = f(\zeta_1, \dots, \zeta_n) = \sum_{i=1}^{n} \lambda_i \zeta_i^2 + k, \qquad \lambda_i \geq 0 \tag{2.4}$$

Si el paso de las variables originales y_i a las ζ_i es lineal (en el caso de la ecuación (A.4) lo es), podemos emplear directamente el método del gradiente sobre la ecuación (2.4):

$$\Delta\zeta_i = -\eta(\nabla f)_i = -\eta\frac{\partial f}{\partial \zeta_i} = -2\eta\lambda_i\zeta_i$$

El esquema iterativo queda:

$$\zeta_i^{(t+1)} = \zeta_i^{(t)} + \Delta\zeta_i^{(t)} = (1 - 2\eta\lambda_i)\zeta_i^{(t)}$$

donde vemos que la convergencia es tan sólo lineal (de primer orden): el valor en el nuevo paso es proporcional al del paso anterior. Para que sea convergente debe ser:

$$|1 - 2\eta\lambda_i| < 1 \,\forall i \Rightarrow \eta < \frac{1}{\max\{\lambda_i\}}$$

Por tanto, la dirección de mayor curvatura (mayor λ_i) limita la longitud del salto η. Esto hace que la convergencia en direcciones asociadas a λ_i pequeños (pequeña curvatura) sea muy lenta: la constante por la que se multiplica el error es (cf. [25, págs. 105–107]):

$$1 - \frac{2\lambda_i}{\lambda_{\max}} \approx 1$$

[5] Para una descripción más detallada, tanto de los métodos del gradiente como del resto de tipos existentes, puede consultarse un texto de optimización no lineal; véase, por ejemplo, el libro de Himmelblau [26, págs. 63–73].

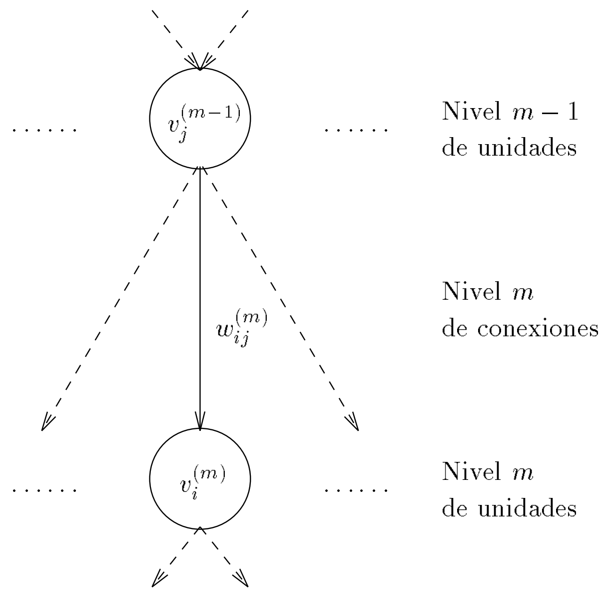

Figura 2.3: Unidades y pesos involucrados en un paso de retropropagación.

Entonces, la velocidad de convergencia depende del cociente $\max\{\lambda_i\}/\min\{\lambda_i\}$. Cuanto mayor sea este cociente, más lento será el proceso. El caso ideal ocurre cuando $\lambda_i = \lambda \; \forall i$ y puede conseguirse escalando adecuadamente las variables ζ_i en la fórmula (2.4). Por desgracia, no es posible incorporar este cambio de variable en la RNA, ni a través de un preproceso de los datos ni modificando el algoritmo de aprendizaje.

Veremos que los $\lambda_1, \ldots, \lambda_n$ coinciden con los autovalores de $\mathbf{X}\mathbf{X}^T$.

Los experimentos realizados (ver el capítulo 4) demostrarán empíricamente cuán lento puede ser el descenso del gradiente. No obstante, el algoritmo *quickprop* para perceptrones multicapa, indicado en la sección 2.2.7, alivia considerablemente esta situación.

2.2.6 El algoritmo de retropropagación

El aprendizaje por retropropagación (*backpropagation*) o regla delta generalizada en una RNA es un método de gradiente, tal y como se demuestra en cualquier libro de texto sobre el tema (por ejemplo, el de Rumelhart [46, págs. 322–328], uno de los coinventores de la retropropagación, o en [25, págs. 115–120] ó [17]).

Suponemos que la red es un perceptrón de M capas de conexiones y, por tanto, $M+1$ niveles de unidades, desde la 0 (entrada) hasta la M (salida). La capa m de pesos conecta las salidas de las unidades del nivel $m-1$ con las entradas del m, para $m = 1, \ldots, M$. El valor del peso de la conexión que va de la unidad j de la capa $m-1$ a la i de la m lo representamos por w_{ij}, siguiendo el convenio usual. Y representaremos por $v_i^{(m)}$ la salida de la unidad i de la capa m. La figura 2.3 muestra la situación. A continuación damos, de manera resumida, el algoritmo de retropropagación:

0. Inicializar todos los pesos de la red a valores aleatorios pequeños.

1. Presentar un vector de entrada $\mathbf{y}$ y su salida deseada $\mathbf{s}$. Aplicar $\mathbf{y}$ a la capa de entrada de la red ($m = 0$). Por tanto, $\mathbf{v}^{(0)} = \mathbf{y}$.

2. Ir calculando las salidas intermedias para cada unidad hacia adelante, de capa en capa ($m = 1, \ldots, M$):
$$v_i^{(m)} = g\left(\sum_j w_{ij}^{(m)} v_j^{(m-1)}\right)$$
donde $g : \mathbb{R} \to \mathbb{R}$ es la función de activación de las unidades (en nuestro caso será siempre la identidad). Para $m = M$ obtenemos en $\mathbf{v}^{(M)} = \mathbf{s}$ las salidas producidas por la red.

3. Calcular los valores δ para cada unidad de la capa de salida a partir del error entre la salida producida por la red y la deseada, según la fórmula:

$$\delta_i^{(M)} = g'\left(v_i^{(M)}\right)\left(s_i - v_i^{(M)}\right)$$

$g'(\cdot)$ es la derivada de $g(\cdot)$.

4. Retropropagar los errores de nivel de unidades en nivel de unidades hacia atrás, $m = M-1,\ldots,1$, para obtener los valores δ en cada capa interna según la fórmula:

$$\delta_i^{(m-1)} = g'\left(v_i^{(m-1)}\right)\sum_j w_{ji}^{(m)}\delta_j^{(m)} \tag{2.5}$$

5. Una vez obtenido el valor δ asociado a cada unidad de la red, obtener la modificación asociada a cada peso como:

$$\Delta w_{ij}^{(m)} = \eta\delta_i^{(m)} v_j^{(m-1)}$$

para un cierto $\eta \in \mathbb{R}^+$ (factor de aprendizaje) fijado o ajustable. Ahora, dependiendo del momento en el que se actualizan los pesos de la red, surgen dos variantes del algoritmo de retropropagación:

- La modificación se suma inmediatamente:

$$w_{ij}^{(m)}(t+1) = w_{ij}^{(m)}(t) + \Delta w_{ij}^{(m)}(t)$$

donde t es el índice de iteraciones, que se incrementa cada vez que se pasa un patrón. Esta variante se conoce como *modo on-line*.

- Las modificaciones debidas a cada patrón del conjunto de entrenamiento se suman de golpe cuando se han pasado todos los patrones por la red. En este caso t se incrementa cuando se pasa el último patrón. Cada pasada de todos los patrones por la red es un *ciclo* (*epoch*). Esta variante se llama *modo batch* o por lotes. A su vez, los patrones pueden pasar por la red siempre en el mismo orden o, por el contrario, en un orden aleatorio en cada ciclo (*shuffled*).

6. Volver al paso 2 si no se cumple cierto criterio (por ejemplo, error total E aceptable). En caso contrario terminar. El error E se define como[6]

$$E(F) = \left\langle \|\mathbf{y} - F(\mathbf{y})\|^2 \right\rangle$$

donde la suma $\langle\cdot\rangle$ va extendida a todos los patrones del conjunto de entrenamiento, $\mathbf{s} = F(\mathbf{y})$ es la salida calculada por la red y $F : \mathbb{R}^n \to \mathbb{R}^q$ es la función que calcula la red ($q = n$ en el caso autoasociativo).

Usando la regla de la cadena, es fácil demostrar que para el modo *batch*:

$$\Delta w_{ij} = -\eta\frac{\partial E}{\partial w_{ij}}$$

con lo cual la retropropagación es un método de gradiente con función objetivo $E(F) = \left\langle \|\mathbf{y} - F(\mathbf{y})\|^2 \right\rangle$ y espacio de búsqueda conformado por los parámetros de la función F, que son los pesos w_{ij} de la red.

Por ser un método de gradiente, el algoritmo de retropropagación comparte todas sus desventajas: convergencia lenta (son métodos de primer orden) y posibilidad de estancarse en puntos de silla y mínimos locales. Sin embargo, y por razones que aún no se comprenden bien, el algoritmo no ha sufrido demasiado

[6] Conviene indicar que la función de error no tiene por qué ser de la forma $E(F) = \left\langle \|\mathbf{y} - F(\mathbf{y})\|^2 \right\rangle$; de hecho, otros autores (p. ej. Sirovich y Kirby [52]) usan el *error cuadrático medio normalizado* (NMSE), definido como $E = \frac{1}{p}\left\langle \frac{\|\mathbf{y}-\hat{\mathbf{y}}\|}{\|\mathbf{y}\|} \right\rangle$, donde $\hat{\mathbf{y}}$ es el vector que aproxima al original $\mathbf{y}$ (calculado por la RNA u otro procedimiento). Mientras que la suma de errores cuadráticos usada por nosotros (y muchas veces cuando se usen RNAs, por ser la función minimizada por el algoritmo de retropropagación) puede considerarse como una medida absoluta del error, el NMSE es una medida relativa. Matemáticamente, es más fácil de manejar la medida absoluta; además la teoría de regresión lineal por mínimos cuadrados está basada en ella y tenemos a mano gran cantidad de teoremas y resultados útiles. Sin embargo, también es cierto que la suma de errores cuadráticos no es una *buena* medida del error para imágenes, como han puesto de manifiesto Hecht-Nielsen [24, págs. 113–114] y Dony y Haikin [13, pág. 300], entre otros, porque, por ejemplo, dicha suma será grande para una imagen y la misma imagen pero más clara (con 30 tonos de gris de diferencia en cada píxel, digamos), mientras que una persona consideraría que representan lo mismo. Dony y Haikin sugieren otras medidas (algunas de ellas subjetivas, basadas en el juicio de varios observadores), pero no son demasiado satisfactorias.

el problema de los mínimos locales y puntos de silla, como han puesto de relieve las muchas simulaciones realizadas por diversos autores.

En el modo *on-line* la retropropagación no equivale ya al descenso de gradiente de manera estricta, pero se ha observado que suele requerir menos iteraciones, sobre todo si los patrones no se presentan siempre en el mismo orden, sino desordenados aleatoriamente (*shuffled*) en cada ciclo. El simulador *SNNS* permite ambos modos, *batch* y *on-line*, y para éste último dispone de una opción *shuffle* que puede activarse o no. En todas nuestras simulaciones se usó el modo *on-line shuffled*.

El aprendizaje por retropropagación presenta una ventaja importante: puede implementarse de una manera local, es decir, cada peso de la red necesita sólo datos locales para ser corregido.

2.2.7 El algoritmo *quickprop*

Una manera de acelerar el aprendizaje de la red es usando información sobre la curvatura de la función de error E. Esto exige calcular las derivadas segundas de la misma. El método *quickprop*, introducido por Scott Fahlman, supone que E es localmente cuadrática —lo cual es cierto hasta el segundo orden para una función general derivable E; en nuestro caso más aún, pues E es cuasicuadrática— e intenta alcanzar de un solo salto el mínimo del paraboloide desde la posición actual.

Quickprop calcula las derivadas en la dirección de cada peso y, tras calcular el primer gradiente por el método de retropropagación a secas, obtiene el salto como:

$$\Delta_{(t+1)} w_{ij} = \frac{S_{(t+1)}}{S_{(t)} - S_{(t+1)}} \Delta_{(t)} w_{ij}$$

donde $\Delta_{(t+1)} w_{ij}$ es el cambio actual, $S_{(t+1)}$ la derivada de E respecto a w_{ij} y $\Delta_{(t)} w_{ij}$ y $S_{(t+1)}$ el cambio y la derivada anteriores, respectivamente.

Como veremos en las simulaciones, *quickprop* es capaz de obtener los mismos resultados que la retropropagación con **muchas menos iteraciones**. Es por tanto el método que conviene usar con las redes Ξ. No obstante, en este trabajo emplearemos ambos algoritmos con fines comparativos.

SNNS implementa, además de la retropropagación, el método *quickprop*[7].

2.3 Relación con las memorias autoasociativas

2.3.1 La regla de aprendizaje de Hebb

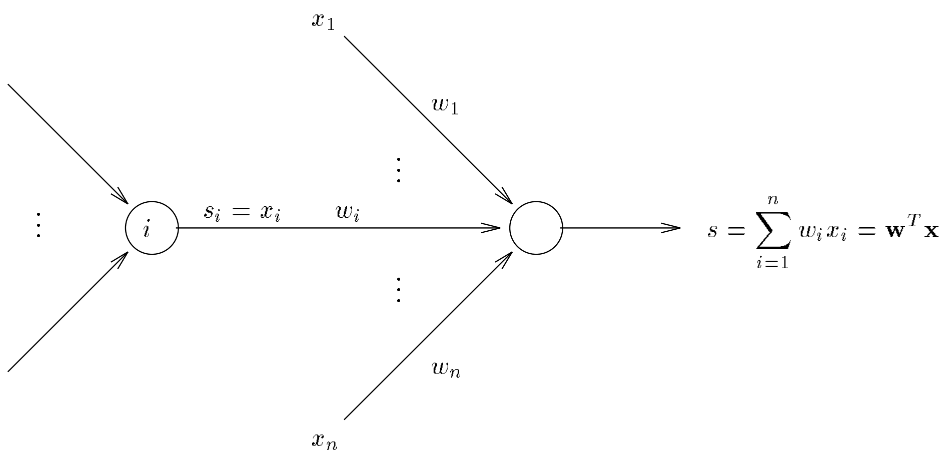

Figura 2.4: Regla de Hebb.

La regla de aprendizaje de Hebb no minimiza ninguna función objetivo, sino que está basada en un mecanismo biológico: se ha observado que cuando dos neuronas tienden a excitarse a la vez, la conexión entre ambas se refuerza. Esto sugiere una formulación matemática de la forma $\Delta w_i = \eta s x_i$, o en forma vectorial $\Delta \mathbf{w} = \eta s \mathbf{x}$, donde el peso w_i conecta una unidad i con la unidad en cuestión, y $s = \mathbf{w}^T \mathbf{x}$ es la salida de dicha unidad ante la entrada $\mathbf{x}$, tal como se muestra en la fig. 2.4. El valor η es el factor

[7] De hecho, la información de este apartado está sacada del manual de *SNNS* [62, págs. 119–120].

de aprendizaje, y, si es positivo, el aprendizaje se llama *hebbiano*, y si es negativo, *antihebbiano*. Ambos tipos de aprendizaje son muy importantes para las RNAs que extraen componentes principales de los patrones, como se verá en la sección 3.4.

Además, la regla de Hebb se emplea en las memorias autoasociativas conexionistas, como veremos a continuación.

2.3.2 La memoria autoasociativa conexionista

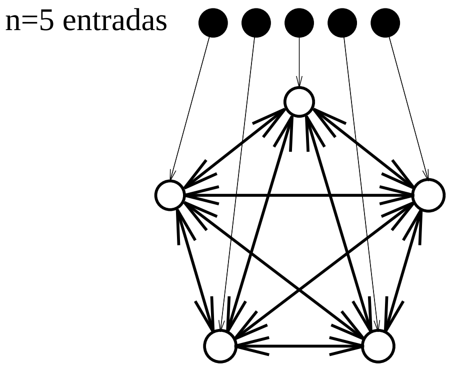

Figura 2.5: Arquitectura de una memoria autoasociativa para vectores de dimensión $n = 5$. Cada unidad está conectada a todas las demás con pesos modificables. Cada patrón de entrada es asociado a sí mismo (idealmente).

La memoria autoasociativa construye, embebida en los pesos, la matriz de correlación $\mathbf{XX}^T = \langle \mathbf{xx}^T \rangle$, puesto que su aprendizaje tiene lugar con la regla de Hebb. La reconstrucción de un vector se consigue presentándolo a la entrada de la memoria, la cual devuelve como estimación $\hat{\mathbf{x}} = \mathbf{XX}^T\mathbf{x}$; su calidad puede comprobarse de la manera usual con la distancia euclídea $\|\mathbf{x} - \hat{\mathbf{x}}\|$ o, como también suele hacerse (p. ej., O'Toole *et al.* [39]), con el ángulo formado por el vector original $\mathbf{x}$: $\cos(\mathbf{x}, \hat{\mathbf{x}}) = \frac{\mathbf{x}^T\hat{\mathbf{x}}}{\|\mathbf{x}\|\|\hat{\mathbf{x}}\|}$, donde un coseno de valor 1 indica reconstrucción exacta. Dado que la matriz de autoasociación $\mathbf{XX}^T$coincide con la matriz sobre la cual se efectúa el pseudoanálisis de componentes principales por la red Ξ, los mismos resultados sobre su descomposición en autovalores, etc. se aplican aquí.

O'Toole *et al.* [39] y originalmente Kohonen [29] usaron memorias autoasociativas (implementadas como RNAs) para almacenar caras y recuperarlas usando un patrón ruidoso o incompleto (p. ej. la mitad de la imagen), ya que la respuesta más fuerte en la red es la del vector que representa dicha imagen.

Una desventaja de la memoria autoasociativa es su limitada capacidad máxima (suponiendo que se desee recuperación exacta)[8]: $0.138n$, donde n es la dimensión de los vectores, y suponiendo que estos son aleatorios y por tanto incorrelados. Dado que esto no es cierto para las caras, la capacidad efectiva es menor, aunque para $n = 151 \times 225 = 33975$ y $p = 159$, resolución de la imagen y número de imágenes usadas por O'Toole *et al.*, respectivamente, esto no es problema, si bien para un caso realista con p grande sí lo sería.

Si los vectores $\mathbf{x}$ son ortogonales, su reconstrucción es exacta (siempre que no se exceda la capacidad de la red), ya que $\mathbf{XX}^T\mathbf{x} = \mathbf{x}$. En un caso real, los vectores no serán ortogonales y su reconstrucción será inexacta: $\hat{\mathbf{x}} = \mathbf{XX}^T\mathbf{x} = \mathbf{x} + \mathbf{x}^* \neq \mathbf{x}$, donde $\mathbf{x}^*$ es el ruido o interferencia debida a los demás patrones de la red. Se puede mejorar la respuesta de la red usando la regla de aprendizaje de Widrow-Hoff o regla delta (que coincide con la de retropropagación para una única capa de unidades), la cual varía iterativamente los pesos (la matriz $\mathbf{W}$) hasta que, en el límite, $\mathbf{W} = \mathbf{UU}^T = \mathbf{UU}^+$, donde $\mathbf{U}$ es la matriz de autovectores de $\mathbf{XX}^T$ (este resultado es análogo al obtenido para las redes Ξ en la sección 3.3.1; allí, $\mathbf{W} = \mathbf{AB} = \mathbf{U}_{\mathcal{I}}\mathbf{U}_{\mathcal{I}}^T$). Nótese que en este momento los autovectores de $\mathbf{XX}^T$ son reconstruidos de manera exacta por la memoria, no así los vectores originales $\mathbf{x}$, pero se minimiza el error cuadrático de su reconstrucción.

Las redes de compresión pueden funcionar también como memorias autoasociativas, como se ve en la sección 5.3.

El simulador de RNAs empleado, *SNNS*, dispone de un modelo de memoria autoasociativa predefinido.

[8]Véase, por ejemplo, [25, págs. 17–20 y 35–41].

2.4 Conjuntos de entrenamiento y validación

Una vez fijada la estructura de la RNA, así como el algoritmo de aprendizaje, es muy importante diseñar correctamente el conjunto de datos usado para el entrenamiento y la validación. Para perceptrones multicapa entrenados con retropropagación, Hecht-Nielsen [24, págs. 115–119] distingue los siguientes conjuntos:

- Conjunto de entrenamiento (*training set*, TS): utilizado para entrenar la red, es decir, para, mediante el aprendizaje, modificar sus pesos, de manera posiblemente iterativa, hasta cumplir cierto criterio.

- Conjunto de prueba del entrenamiento (*training test set*, TTS): utilizado para saber cuándo detener el entrenamiento, para así evitar un ajuste o entrenamiento excesivo (si éste se prolonga demasiado) o bien un entrenamiento insuficiente (ver la fig. 2.6). Se utiliza durante el entrenamiento.

- Conjunto de validación (*validation test set*, VTS): utilizado para validar la red una vez el entrenamiento ha concluido. Si la red no pasa la validación, deben rehacerse los TS y TTS y volver a entrenar la red.

- Conjunto de prueba de aceptación (*acceptance test set*, ATS): lo mantiene oculto el usuario final hasta que el diseñador de la red le entrega ésta. El usuario aceptará la red si y solamente si ésta da buenos resultados sobre el ATS. El ATS, lógicamente, debe ser distinto al VTS.

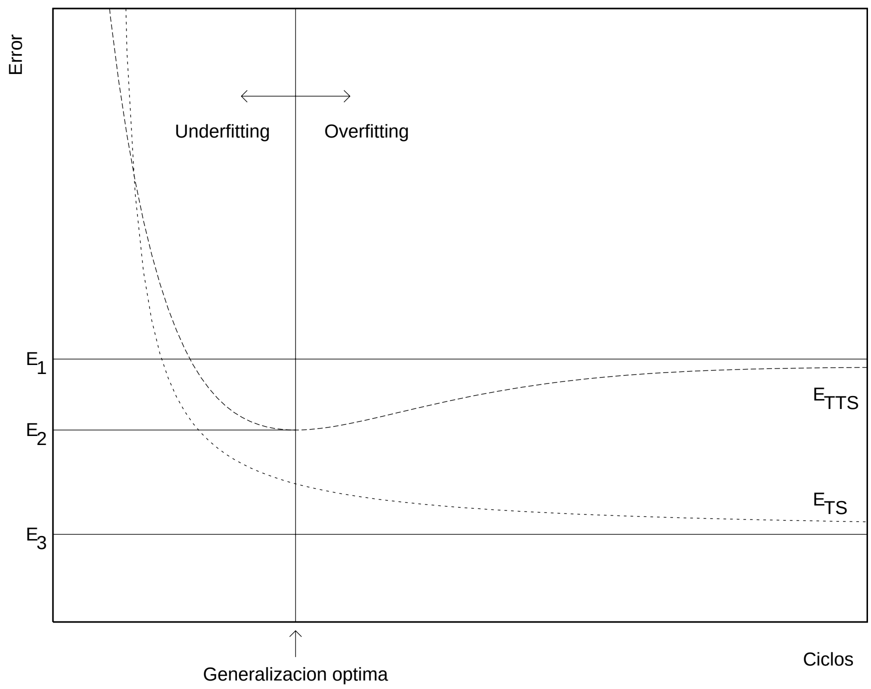

Figura 2.6: Evolución del error E para el TS y el TTS.

Es crucial que los conjuntos VTS y ATS se usen sólo como comprobación una vez el entrenamiento de la red ha finalizado, pero jamás durante el mismo. Si no, la RNA aprenderá estos conjuntos específicos pero no será capaz de generalizar[9].

Las curvas mostradas en la figura 2.6 muestran que —fijados el TS y el TTS— existe un punto (el marcado como "generalización óptima", con un valor asociado de la función de error igual a E_2) en el que el comportamiento de la red es óptimo; si no se alcanza dicho punto, por haberse detenido prematuramente el entrenamiento (zona de entrenamiento insuficiente o *underfitting*), la red dará un resultado pobre pues su ajuste será malo; si se sobrepasa dicho punto, por supuesto que el error en el TS seguirá decreciendo, pero el del TTS comenzará a crecer. Esto se debe a que en la zona de entrenamiento excesivo (*overfitting* u *overtraining*) la red memoriza demasiado el TS y es incapaz de generalizar a nuevos patrones similares. El

[9] En este contexto, *generalizar* equivale a *interpolar*: si el nuevo vector está cerca o entre los patrones del TS, la salida debe estar "cerca" de las salidas para esos patrones; si el nuevo vector está lejos de cualquier patrón conocido, la salida de la RNA no será significativa.

TTS ayuda a evitar esa situación: no participa en el entrenamiento activamente (no causa modificaciones en los pesos), pero indica cuándo detenerlo.

Dicho de otra manera, el entrenamiento excesivo hace que la RNA desarrolle una superficie en el espacio de pesos que ajusta bien los vectores del TS (el error en ellos es lo más pequeño posible), pero pierde su capacidad para interpolar bien entre dichos vectores, porque la superficie se curva demasiado entre esos puntos. Esto ocurre con un número de ciclos de entrenamiento infinito (o muy grande); el error en el TS alcanza su valor límite E_3 (su valor mínimo) y el error en el TTS el valor E_1 (que no es su valor mínimo[10]).

Empíricamente se ha observado que la fig. 2.6 es típica de la evolución de los errores en el TS y el TTS. En la etapa intermedia que rodea al mínimo del error en el TTS, la superficie ajusta aceptablemente los patrones del TS e interpola bien entre ellos, porque aún no se ha curvado en exceso[11].

En cuanto a la elección de los conjuntos TS y TTS, Hecht-Nielsen indica que ambos suelen tomarse de tamaño aproximadamente igual, aunque esto no tiene por qué seguirse a rajatabla, particularmente cuando el número de patrones disponible es escaso (como nos pasa a nosotros); en este caso el TTS debe reducirse e incluso anularse (en este último caso, el entrenamiento debe ser detenido cuando la curva de error en el TS comience a estabilizarse y se tenga un error aceptable).

2.4.1 Elección de los distintos conjuntos de patrones en este trabajo

Se cuenta con un total de 102 imágenes, 6 para cada uno de los 17 individuos disponibles. Estas imágenes aparecen en la figura 2.7. Obsérvese el alto grado de uniformidad que presentan las imágenes, debido al proceso de captación seguido.

Para la extracción de características con la red Ξ, dado que queremos hacer el error E tan pequeño como sea posible, para así poder comparar la base obtenida por la red con la del análisis de componentes principales, no necesitamos el TTS. El entrenamiento se dará por finalizado cuando no se pueda reducir más el error. Sin embargo, usaremos un VTS, porque nos interesará saber cómo reacciona la red, una vez entrenada, ante unos patrones similares a los aprendidos pero distintos. Se decidió tomar $\frac{5}{6}$ del conjunto total de patrones para el TS y el resto, $\frac{1}{6}$, para el VTS; es decir, de cada individuo se reserva una imagen para validación y las otras 5 para entrenamiento. La elección de qué foto va al VTS se tomó al azar. Así pues, el TS consta de 85 imágenes y el VTS de 17. En la figura 2.7, las imágenes correspondientes al VTS son las que, en cada subfila, están a la derecha del todo (ligeramente más separadas de sus 5 compañeras).

Adicionalmente, se incluyen en el VTS una serie de imágenes sobrantes del proceso de captación, así como otras procedentes de fotos de revistas. Para diferenciar ambos conjuntos se les dan los nombres VTS1 (las 17 fotos anteriores) y VTS2 (las sobrantes y de revistas). El conjunto VTS2 consta de otras 17 fotos y se muestra en la figura 2.8. Obsérvese que las fotos externas de este conjunto (las procedentes de revistas, etc.) no presentan unas condiciones en absoluto homogéneas, como sí ocurría con las fotos tomadas con la cámara; algunas orejas llevan pendientes, otras están fuertemente oscurecidas por el pelo, en otras la calidad de la imagen es mala, etc.

También se decidió crear otro conjunto de patrones para comprobar el **comportamiento de la red ante patrones transformados** por traslación, escala, rotación, multiplicación de intensidad, adición de intensidad y adición de ruido. Llamamos a este conjunto "de alteraciones" (AS). Para construirlo, se partió de una imagen determinada, a la que se le aplicaron las transformaciones siguientes (las alteraciones vienen indicadas por los nombres de los ficheros correspondientes y fueron obtenidas con *Mathematica* y *xv*):

1. Rotaciones: `r+15o.pgm`; $+15^\circ = 15$ grados en sentido antihorario.
2. Escalas: `s070%.pgm`; homotecia de razón 0.7.
3. Traslaciones: `t+10%,0%.pgm`; traslación de $-0.1 \times 41 = -4$ píxeles en horizontal y 0 en vertical.
4. Adición/sustracción de intensidad: `a+30%.pgm`; a cada componente se le suman $0.3 \times 255 = 77$ tonos de gris.
5. Multiplicación/división de intensidad: `m060%.pgm`; la intensidad de cada componente se multiplica por 0.6.
6. Adición de ruido: `n+15%.pgm`; se suma a la imagen $0.15 \times \mathcal{N}$, donde $\mathcal{N}$ es una imagen de valores aleatorios entre -255 y 255, contenida en el fichero `Noise.m` (en formato de *Mathematica* 2.0).

[10] Nótese que E_2 *tampoco* es el valor mínimo del TTS.

[11] En el caso de las redes de compresión lineales, éste no es el caso, porque la superficie es un subespacio vectorial; sólo varía su posición, pero no su forma. A pesar de eso, las curvas de la figura 2.6 también son aplicables.

En los casos 4, 5 y 6 se truncan los valores fuera de $[0, 255]$, lo cual equivale a que si eso ocurre se satura la imagen.

Hay 8 ejemplos para cada tipo de alteración 1–6, total 48 patrones. El patrón original se incluye por comodidad y para facilitar comparaciones en el conjunto AS, junto con las 48 alteraciones. Total 49, pues. Se muestran en la figura 2.10.

Para elegir la imagen particular sobre la cual hacer las transformaciones, se buscó que 1) tuviera buena calidad (dentro de lo posible), 2) fuera lo más grande posible (41×65), para soportar bien las rotaciones, etc. y 3) que fuera aceptablemente bien reconocida por la red Ξ (da un error $E = 0.91571$ con la red de $h = 10$ unidades ocultas, que es aproximadamente igual al valor medio de E para los patrones del TS).

Finalmente, y con objeto de evaluar la **capacidad de reconocimiento** de la red, se seleccionó un conjunto de 12 fotos de diversos objetos dispares, llamado conjunto RS (ver figura 2.9). Es de esperar que estos patrones produzcan un error E mucho mayor que el de los patrones que representen una oreja, y eso permitirá reconocer o no al patrón.

La media $\overline{\mathbf{y}}$ de los patrones se obtuvo a partir de los 85 patrones del conjunto TS. A cada imagen de cada conjunto (TS, VTS1 y VTS2) se le restó dicha media, por lo que el único conjunto que aparece centrado es el TS; los demás están en el sistema de referencia centrado en $\overline{\mathbf{y}}$. Estas operaciones se realizaron con *Mathematica*.

Debe tenerse en cuenta que la nueva media del conjunto TS no es exactamente $\mathbf{0}$ como debiera, debido a la truncación de las restas $\mathbf{y} - \overline{\mathbf{y}}$, en la que se pierden decimales. Sin embargo, la media resultante es muy pequeña: para patrones de 20×32 ($n = 640$), $||\overline{\mathbf{x}}|| \approx 7 \cdot 10^{-6}$; para 30×48 ($n = 1440$), $||\overline{\mathbf{x}}|| \approx 1.1 \cdot 10^{-5}$.

En ningún caso se ha empleado el ATS, por razones obvias (ya tenemos un VTS).

Figura 2.7: Patrones obtenidos en el proceso de captación. Se tienen 6 fotos para cada uno de los 17 individuos. A partir de estos patrones se construyen, en cada fase, los conjuntos TS, TTS y VTS1.

Figura 2.8: Las 17 imágenes que conforman el conjunto VTS2. Son imágenes sobrantes del proceso de captación y otras digitalizadas de revistas.

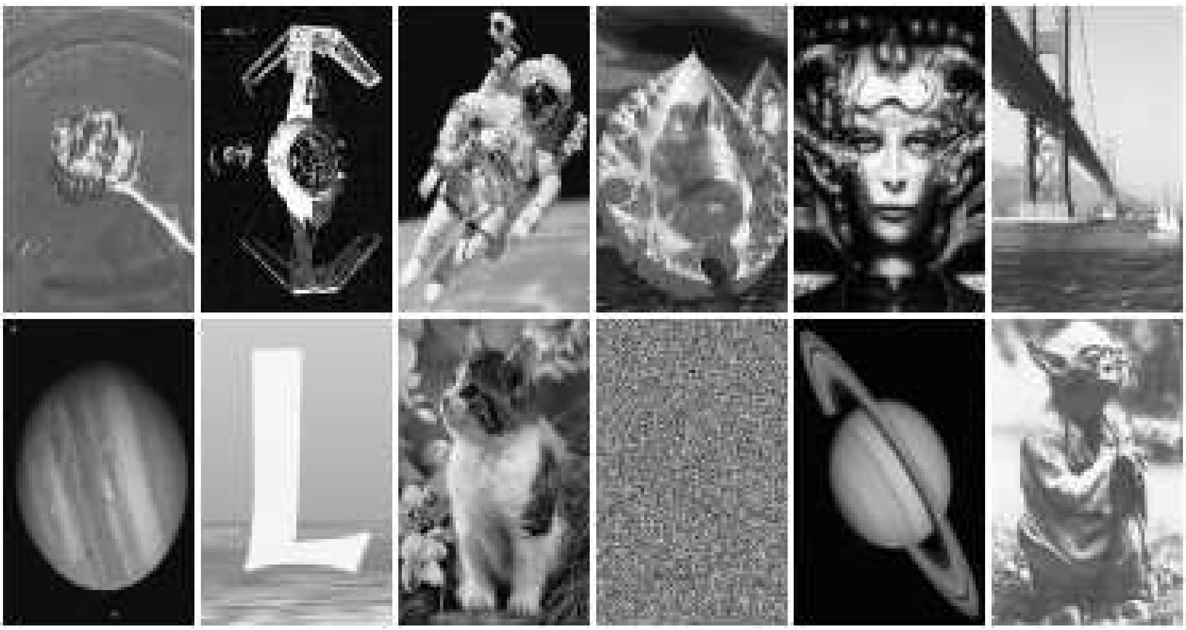

Figura 2.9: Fotos de diversos objetos que la red debería rechazar. Entre ellos se encuentra una imagen de ruido blanco, generada mediante una distribución uniforme $[0, 1]$.

Figura 2.10: Las 49 imágenes que forman el conjunto AS. Son imágenes transformadas por (ordenadas por filas): adición de intensidad, multiplicación de intensidad, adición de ruido, rotación, homotecia y traslación. El patrón original separado a la derecha.

Capítulo 3

El análisis de componentes principales

En este capítulo se describen dos técnicas que están muy estrechamente relacionadas entre sí: el análisis de componentes principales en estadística y la transformada de Karhunen-Loève en teoría de señales. A continuación mostramos cómo las redes Ξ definidas en la sección 2.1 abordan desde otro punto de vista el mismo problema que las dos técnicas citadas. Después de esto, pasamos a ver someramente otras arquitecturas de RNA que, precisamente, son capaces de extraer —con más o menos limitaciones— los primeros componentes principales de un conjunto de patrones dado.

3.1 La técnica del análisis de componentes principales (ACP)

El análisis de componentes principales[1] (*principal components analysis*), ACP, es una técnica estadística que transforma linealmente un conjunto original de variables en un conjunto sustancialmente más pequeño de variables incorreladas que representa la mayor parte de la información contenida en el conjunto original de variables. Su objetivo es reducir la dimensionalidad del conjunto de datos original. Un conjunto pequeño de variables incorreladas es mucho más fácil de entender y usar en tratamientos posteriores que uno grande de variables correladas. La idea fue concebida originalmente por Pearson (1901) y desarrollada independientemente por Hotelling (1933).

El análisis de componentes principales está relacionado con las transformadas de Hotelling (sobre señales discretas) y de Karhunen-Loève (sobre señales continuas). Tiene aplicaciones principalmente en compresión de datos (por ejemplo, en codificación de imágenes se usan subimágenes o bloques de la original como vectores y sobre ellos se hace el ACP) y extracción de características. La compresión de imágenes es posible porque los píxeles vecinos suelen estar fuertemente correlados —salvo en imágenes aleatorias—.

Geométricamente, el primer componente principal (CP) es la línea que ajusta las p observaciones en el espacio n-dimensional. Esta recta minimiza la suma de distancias cuadráticas de las p observaciones a dicha recta *en la dirección perpendicular a la recta*. Los dos primeros CPs definen el plano de ajuste a la nube de puntos en el espacio n-dimensional. Equivalentemente, el segundo CP es la recta de ajuste de los residuos del primer CP. Los tres primeros CPs definen el hiperplano de dimensión 3 de ajuste, etc. Si hay n variables, no puede haber más de n CPs, pero puede haber menos si existen dependencias lineales entre las variables. Si se emplean todos los CPs posibles, se tiene un espacio de la misma dimensión que el original, n, y que por tanto registra toda la variación de las variables. No hay, sin embargo, ventaja alguna en retener *todos* los CPs, ya que no reducimos el número de variables y el problema no se simplifica.

La figura 3.1 muestra el caso de $n = 2$ dimensiones. La nube de puntos se asemeja a un elipsoide más o menos alargado[2] y los CPs se disponen a lo largo de sus ejes principales. En el caso particular de la fig. 3.1, la nube de puntos sigue una distribución normal $N(0, 1)$ en la dirección principal (PCA-1) y una $N(0, 0.3)$ en la secundaria (PCA-2).

La diferencia entre el ACP y la regresión lineal está en la definición de la distancia de ajuste: mientras en el ACP se toma perpendicular a la recta (o hiperplano) de ajuste, en la regresión lineal se toma en la dirección del eje y (ver fig. 3.2).

[1] La mayor parte de esta sección proviene del libro de Dunteman [14].

[2] Bajo ciertas condiciones —por ejemplo, suponiendo normalidad en cada variable—, la nube adoptará la forma de un hiperelipsoide.

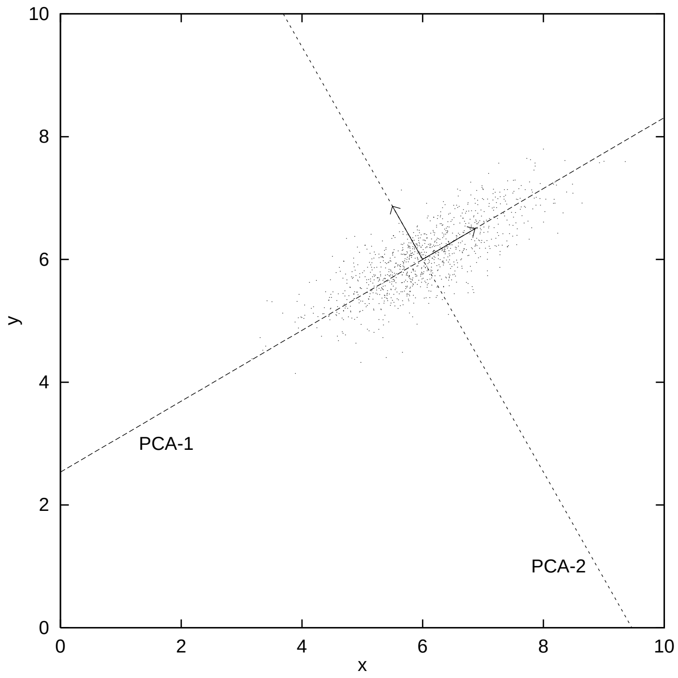

Figura 3.1: Nube de puntos normal en dos dimensiones con sus direcciones principales.

El ACP es un método lineal: si no hay dependencias lineales exactas entre las n variables, habrá n CPs. Las dependencias no lineales (ej. $x_1 = \alpha_1 x_2^2 + \alpha_2 x_2 x_3$) no influyen en el número de CPs.

Una vez seleccionados los CPs, es útil dibujar la nube de puntos original proyectada en pares de CPs (es decir, en planos principales), para identificar puntos que se alejan considerablemente de la nube (*outliers*), agrupamientos y, en general, para comprender la estructura de los datos. Los *outliers* a veces provienen de errores experimentales y puede convenir eliminarlos, pero esto puede —si son influyentes— alterar significativamente los autovalores y CPs obtenidos.

Desde un punto de vista formal[3], supongamos que pasamos del conjunto inicial de p vectores $\{\mathbf{y}\}$ a otro $\{\mathbf{z}\}$ por medio de combinaciones lineales, es decir, cada componente de un vector dado $\mathbf{z}$ se calcula como:

$$z_i = \sum_{j=1}^{n} u_{ij} y_j$$

O, en forma matricial, $\mathbf{z} = \mathbf{U}\mathbf{y}$ para cada par $(\mathbf{y}, \mathbf{z})$. Se trata de un cambio de base: pasamos de la base canónica, en la que están representados los $\mathbf{y}$, a la base $\{\mathbf{u}_1, \ldots, \mathbf{u}_n\}$, en la que están representados los $\mathbf{z}$.

La nueva media $\overline{z}_i$ para cada componente $i = 1, \ldots, n$ será:

$$\overline{z}_i = \frac{1}{p} \left\langle \sum_{j=1}^{n} u_{ij} y_j \right\rangle = \sum_{j=1}^{n} u_{ij} \overline{y}_j$$

Y las nuevas covarianzas λ_{ij}, $i, j = 1, \ldots, n$:

$$\begin{aligned} \lambda_{ij} &= \frac{1}{p} \left\langle (z_i - \overline{z}_i)(z_j - \overline{z}_j) \right\rangle = \\ &\frac{1}{p} \left\langle \left(\sum_{k=1}^{n} u_{ik} (y_k - \overline{y}_k) \right) \left(\sum_{l=1}^{n} u_{jl} (y_l - \overline{y}_l) \right) \right\rangle = \frac{1}{p} \left\langle \sum_{k,l=1}^{n} u_{ik} u_{jl} (y_k - \overline{y}_k)(y_l - \overline{y}_l) \right\rangle = \\ &\frac{1}{p} \sum_{k,l=1}^{n} u_{ik} u_{jl} \left\langle (y_k - \overline{y}_k)(y_l - \overline{y}_l) \right\rangle = \sum_{k,l=1}^{n} u_{ik} u_{jl} \sigma_{kl} \end{aligned} \tag{3.1}$$

[3]En esta sección, los subíndices se refieren a componentes de un vector, nunca a vectores individuales. Las sumas extendidas a todos los vectores del conjunto se notan con el operador $\langle \cdot \rangle$, definido en la sección 1.4.

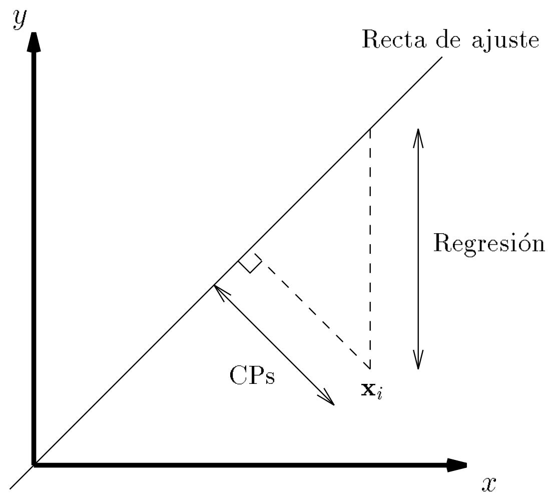

Figura 3.2: Distancias empleadas en el análisis de componentes principales y en la regresión lineal.

donde

$$\sigma_{ij} = \frac{1}{p}\left\langle (y_i - \overline{y}_i)(y_j - \overline{y}_j) \right\rangle$$

son las covarianzas respecto a la base canónica inicial de los vectores $\mathbf{y}$.

Todo lo anterior puede expresarse de manera más cómoda en forma matricial sin más que definir la matriz de covarianzas respecto a $\mathbf{y}$, $\mathbf{\Sigma} = (\sigma_{ij})$, la matriz de covarianzas respecto a $\mathbf{z}$, $\mathbf{\Lambda} = (\lambda_{ij})$ y la matriz de paso $\mathbf{U} = (u_{ij})$, cuyos vectores columna coinciden con los vectores de la nueva base, $\{\mathbf{u}_i\}$. Entonces, es fácil comprobar que la ecuación (3.1) queda:

$$\mathbf{\Lambda} = \mathbf{U}\mathbf{\Sigma}\mathbf{U}^T$$

Si ahora hacemos $\mathbf{U}$ ortogonal ($\mathbf{U}\mathbf{U}^T = \mathbf{U}^T\mathbf{U} = \mathbf{I}$) y $\mathbf{\Lambda}$ diagonal ($\lambda_{ij} = 0$ si $i \neq j$), el conjunto de nuevos vectores $\{\mathbf{u}_i\}$ será incorrelado, y tendremos los componentes principales buscados en $\mathbf{U}$. Salta a la vista que el proceso descrito es simplemente la diagonalización de la matriz $\mathbf{\Sigma}$, que siempre es posible pues $\mathbf{\Sigma}$ es una matriz simétrica semidefinida positiva y por tanto sus autovalores son reales no negativos (ver la proposición 1.4.1). La matriz $\mathbf{\Lambda} = \mathrm{diag}(\lambda_1, \dots, \lambda_n)$ consta de los autovalores de $\mathbf{\Sigma}$, que coinciden con las varianzas respecto de los nuevss vectores $\{\mathbf{u}_i\}$.

Además, el proceso descrito maximiza las varianzas λ_i manteniendo las restricciones de incorrelación y ortonormalidad. Veamos por qué. La proposición 1.4.8 nos garantiza que el primer CP va en la dirección del autovector dominante y que su varianza asociada es máxima y vale $\lambda_1 = \lambda_{\max}$. Si, por inducción, suponemos que los k primeros CPs tienen como varianzas $\lambda_1, \dots, \lambda_k$ y como direcciones $\mathbf{u}_1, \dots, \mathbf{u}_k$ y diagonalizamos la varianza (como en la demostración de la proposición 1.4.6):

$$\mathbf{v}^T\mathbf{\Lambda}\mathbf{v} = \sum_{i=1}^{n} \lambda_i v_i^2$$

vemos que $v_{k+1} = \pm 1$ y $v_i = 0$ si $i \neq k+1$ la maximiza y es ortogonal a $\mathbf{u}_1, \dots, \mathbf{u}_k$, lo que demuestra el caso general.

Nótese cómo las varianzas totales inicial y final coinciden, ya que $\mathrm{tr}\,\mathbf{\Sigma} = \mathrm{tr}\,\mathbf{\Lambda}$ por ser $\mathbf{\Sigma}$ y $\mathbf{\Lambda}$ semejantes.

En otras palabras, el proceso tiene lugar como sigue: el primer CP se toma en la dirección de varianza máxima (que coincide con la del primer autovector); el segundo se toma, dentro del subespacio ortogonal al del primero, en la dirección de varianza máxima (que coincide con la del segundo autovector); el tercero en la dirección de varianza máxima y perpendicular al subespacio de los dos primeros, etc. Como resultado, obtenemos h direcciones (o vectores) ortogonales en el espacio de los datos, $\mathbb{R}^n$, que explican la mayor cantidad de varianza posible siendo mutuamente incorreladas (perpendiculares). De esta manera, la proyección de los $\mathbf{y}$ originales en el subespacio h-dimensional de los h primeros CPs retiene tanta información como es posible y, además, típicamente se tiene que $h \ll n$ mantiene un porcentaje muy grande de la varianza inicial; ahí está la ventaja.

Conviene observar que, dado que la matriz de covarianzas $\mathbf{\Sigma}$ es invariante a traslaciones en los datos (ver la tabla 1.1), el ACP puede realizarse indistintamente sobre los datos centrados o sobre los originales.

Los criterios para decidir cuántos CPs retener son bastante arbitrarios. Además sólo son aplicables *a posteriori*, una vez conocidos los autovalores de $\mathbf{\Sigma}$. Podemos mencionar los siguientes:

- Kaiser, 1960: eliminar $\lambda_i < \overline{\lambda}$, donde $\overline{\lambda}$ es la media de los autovalores de $\mathbf{\Sigma}$. Esto puede descartar autovalores pequeños pero importantes; además, algunas variables pueden no quedar bien representadas por los primeros CPs.
- Jolliffe, 1972: eliminar $\lambda_i < 0.7\overline{\lambda}$.
- Cattell, 1966: sobre un gráfico *scree*, que es aquél que muestra las alturas relativas de los autovalores en orden decreciente (ver fig. 4.1 en la sección 4.2), encontrar una abscisa k tal que la línea que une λ_{k-1} con λ_k tiene una pendiente "mucho más pronunciada" que la que une λ_k con λ_{k+1} y retener k componentes. Equivale a descartar la parte de la curva que empieza a decrecer "despacio."
- Retener tantos CPs como hagan falta para explicar un porcentaje dado de la varianza total (igual a la suma de los autovalores).

Los CPs (al menos los primeros) pueden interpretarse en función del significado de las variables originales. Por ejemplo, si éstas miden el porcentaje de satisfacción de una población con la sanidad (primera variable), la limpieza de las calles (segunda), etc. y el primer CP resulta ser aproximadamente la media de estas variables (caso muy común), este CP podría interpretarse como el "porcentaje de satisfacción general." Por ello, para un estadístico puede resultar útil tomar en cuenta no sólo el porcentaje de varianza total explicada, sino la facilidad de interpretación de los CPs elegidos, así como que cada una de las variables originales quede bien representada por dichos CPs. Muchas veces una rotación de los CPs[4] puede, manteniendo la misma fracción de varianza total explicada, ofrecer unas variables más atractivas desde el punto de vista interpretativo.

Para el caso que nos ocupará más adelante, en el que cada variable es la intensidad de un píxel de la imagen, no hay necesidad de interpretación alguna y los únicos objetivos son obtener una representación con bajo error residual y mantener información de detalle que permita la identificación (esto último puede ser muy difícil). Sin embargo, debe tenerse en cuenta que, una vez fijado el subespacio de los h primeros CPs, cualquier base (ortonormal o no) que lo genere es válida. Como veremos, esto es justamente lo que ocurre con las redes Ξ.

En resumen: el análisis de componentes principales del conjunto de p vectores $\{\mathbf{y}\}$ nos permite encontrar una base ortonormal $\mathbf{U}$ de $\mathbb{R}^n$ respecto a la cual la matriz de covarianzas es diagonal, y por lo tanto los nuevos vectores (o sea, las nuevas variables independientes) de $\mathbf{U}$ son incorrelados. Dicha base lleva los vectores ordenados decrecientemente según su varianza λ_i, lo cual nos permite, reteniendo sólo los primeros h vectores, obtener un subespacio óptimo en el sentido de mínimos cuadrados que explica un porcentaje $\sum_{i=1}^{h} \lambda_i / \sum_{i=1}^{n} \lambda_i$ de la varianza total.

3.1.1 El ACP desde el punto de vista de la teoría de la información

Si suponemos que la distribución de los vectores $\mathbf{x}$ es gaussiana (normal), y que estos vectores pasan por un canal estrecho de transmisión que limita su dimensión a h por medio de una transformación lineal $\mathbf{x}^* = \mathbf{B}\mathbf{x}$, con $\mathbf{B}$ de orden $h \times n$, puede demostrarse[5] que el ACP es óptimo respecto a otro criterio, el de maximización de la información mutua entre los vectores $\mathbf{x}$ que entran en el canal y los $\mathbf{x}^*$ que salen de él. La información mutua entre ambos vectores $I(\mathbf{x}, \mathbf{x}^*)$, definida como

$$I(\mathbf{x}, \mathbf{x}^*) = H(\mathbf{x}) - H(\mathbf{x}|\mathbf{x}^*)$$

donde H es la entropía, es máxima cuando la entropía condicional de $\mathbf{x}$ respecto a $\mathbf{x}^*$, $H(\mathbf{x}|\mathbf{x}^*)$, es mínima, y vale:

$$I(\mathbf{x}, \mathbf{x}^*) = \frac{1}{2} \log((2\pi e)^h \lambda_1 \ldots \lambda_h)$$

donde $\lambda_1, \ldots, \lambda_h$ son los primeros h autovalores de la matriz de covarianzas de los patrones $\mathbf{x}$, $\frac{1}{p}\mathbf{X}\mathbf{X}^T$, que evidentemente es alcanzada por el ACP.

Podemos afirmar, pues, que —en las condiciones de normalidad y linealidad anteriores— el ACP es óptimo también en el sentido de maximización de la información mutua o minimización de redundancia en los vectores transmitidos.

[4] Por ejemplo, usando la técnica *varimax* (Kaiser, 1958), que rota los CPs de modo que las nuevas variables se "parezcan" a las originales.

[5] Para más detalles, véase [5] y [16].

3.2 Codificación por transformadas. La transformada de Karhunen-Loève (KLT)

Un enfoque muy extendido para la compresión de imágenes es el uso de transformadas, normalmente por bloques, sobre la imagen dada, que producen un conjunto ordenado de coeficientes a partir de las cuales es posible la reconstrucción exacta de dicha imagen. Si en lugar de usar todos los coeficientes se usan sólo los primeros h, la reconstrucción presenta un error pero la imagen se comprime.

De entre las transformadas lineales por bloques, la óptima (en el sentido de que minimiza el error cuadrático) es la transformada de Karhunen-Loève[6] (KLT): la imagen original se divide en bloques no solapados de $n = m \times m$ píxeles, que se toman como vectores $\mathbf{y}$ n-dimensionales. La matriz de orden $h \times n$ $\mathbf{U}_h^T$, traspuesta de $\mathbf{U}_h = \{\mathbf{u}_1, \ldots, \mathbf{u}_h\}$, donde $\mathbf{u}_i$ es el autovector de $\mathbf{YY}^T$ asociado al autovalor λ_i y $\lambda_1 \geq \cdots \geq \lambda_n$, da la transformada: $\hat{\mathbf{y}} = \mathbf{U}_h^T \mathbf{y}$, y $\mathbf{U}_h$ la transformada inversa: $\hat{\hat{\mathbf{y}}} = \mathbf{U}_h \hat{\mathbf{y}}$. $\mathbf{W} = \mathbf{U}_h \mathbf{U}_h^T$ minimiza el error cuadrático $\left\langle \|\mathbf{y} - \mathbf{W}\mathbf{y}\|^2 \right\rangle = \left\langle \|\mathbf{y} - \mathbf{U}_h \hat{\mathbf{y}}\|^2 \right\rangle$, como se ve en la sección 3.3.1. Por supuesto, con $h = n$, $\mathbf{W} = \mathbf{U}_h \mathbf{U}_h^T = \mathbf{I}_{n \times n}$ y la reconstrucción es perfecta, pero entonces no hay compresión.

Los vectores $\{\mathbf{u}_i\}$ de la base de la transformada coinciden con los CPs de los datos $\{\mathbf{y}\}$, por lo que puede decirse que el ACP es equivalente a la KLT.

La KLT presenta dificultades prácticas: obtener la matriz $\mathbf{YY}^T$ requiere mucho tiempo de cálculo y mucho espacio de almacenamiento si n es grande, y además el cálculo de los autovectores y autovalores es computacionalmente costoso[7], incluso para matrices reales simétricas semidefinidas positivas, como es el caso de $\mathbf{XX}^T$. Finalmente, el cálculo de la transformada directa y de la inversa es de orden $\mathcal{O}(hn)$ para cada bloque. Por ello, se usan en su lugar transformadas de *base fija*, como la transformada discreta del coseno[8] (DCT), que puede calcularse en $\mathcal{O}(n \log n)$.

Como veremos, las RNAs —tanto algunas de las mencionadas en la sección 3.4, que extraen CPs individuales, como las redes Ξ, que extraen una base del subespacio generado por los primeros CPs, pero no de autovectores— proporcionan un camino distinto, que puede resultar más eficiente que los métodos numéricos tradicionales para el cálculo de los CPs, o dicho de otra manera, de la KLT, si n es grande y $h \ll n$.

La cuantización de vectores, otro mecanismo empleado comúnmente en la compresión de imágenes, puede ser también implementado por RNAs, concretamente por mapas de rasgos autoorganizativos de Kohonen [13].

3.3 Las redes de compresión y el ACP

La sección siguiente contiene el resultado más importante para nosotros: las redes de compresión, o redes Ξ, elaboran durante su aprendizaje una representación interna muy especial, codificada en los valores de sus pesos. Estos pesos, tomados en forma vectorial, nos dan una base del subespacio generado por los h primeros CPs de las patrones usados en el entrenamiento. Veamos en detalle este resultado.

3.3.1 Estudio de la superficie de error $E(\mathbf{A}, \mathbf{B}) = \left\langle \|\mathbf{x} - \mathbf{ABx}\|^2 \right\rangle$

Como se demuestra en el apéndice A.1, la superficie $E(\mathbf{W}) = \left\langle \|\mathbf{x} - \mathbf{Wx}\|^2 \right\rangle$, donde $\mathbf{W}_{n \times n} = (w_{ij})$ es una matriz no restringida (sus elementos w_{ij} son independientes entre sí), resulta ser cuadrática y presenta un mínimo global en $\mathbf{W} = \mathbf{I}$ que da $E = 0$.

Sin embargo, si hacemos $\mathbf{W} = \mathbf{AB}$, con $\mathbf{A}$ y $\mathbf{B}$ matrices rectangulares de órdenes $n \times h$ y $h \times n$, respectivamente, los w_{ij} dejan de ser independientes, $\mathbf{W}$ está restringida y $\operatorname{rg} \mathbf{W} \leq h$. En este caso la superficie $E(\mathbf{A}, \mathbf{B}) = \left\langle \|\mathbf{x} - \mathbf{ABx}\|^2 \right\rangle$ es algo más complicada. Baldi y Hornik [4, 5] dan una descripción completa de ella. Sus resultados más importantes se resumen a continuación (en el apéndice A.2 se detallan algo más); pero previamente demostremos las siguientes proposiciones:

Proposición 3.3.1. $E(\mathbf{W}) = \left\langle \|\mathbf{x} - \mathbf{Wx}\|^2 \right\rangle = \|(\mathbf{I} - \mathbf{W})\mathbf{X}\|^2$.

[6] Para un tratamiento más detallado, relacionado con otros métodos, como la cuantización de vector, véase [13] ó [21].

[7] De orden $\mathcal{O}(n^3)$.

[8] El *Joint Photographics Expert Group* (JPEG) ha adoptado el enfoque de codificación por transformada lineal por bloques para su estándar, usando la DCT como transformada.

Demostración.

$$
\begin{aligned}
E(\mathbf{W}) = \left\langle \|\mathbf{x}-\mathbf{W}\mathbf{x}\|^2 \right\rangle = \\
\left\langle \operatorname{tr}(\mathbf{I}-\mathbf{W})\mathbf{x}((\mathbf{I}-\mathbf{W})\mathbf{x})^T \right\rangle = \left\langle \operatorname{tr}(\mathbf{I}-\mathbf{W})\mathbf{x}\mathbf{x}^T(\mathbf{I}-\mathbf{W})^T \right\rangle = \operatorname{tr}\left\langle (\mathbf{I}-\mathbf{W})\mathbf{x}\mathbf{x}^T(\mathbf{I}-\mathbf{W})^T \right\rangle = \\
\operatorname{tr}(\mathbf{I}-\mathbf{W})\left\langle \mathbf{x}\mathbf{x}^T \right\rangle(\mathbf{I}-\mathbf{W})^T = \operatorname{tr}(\mathbf{I}-\mathbf{W})\mathbf{X}\mathbf{X}^T(\mathbf{I}-\mathbf{W})^T = \operatorname{tr}(\mathbf{I}-\mathbf{W})\mathbf{X}((\mathbf{I}-\mathbf{W})\mathbf{X})^T = \\
\|(\mathbf{I}-\mathbf{W})\mathbf{X}\|^2
\end{aligned}
\tag{3.2}
$$

□

La proposición anterior nos pone E en una forma más conveniente para los cálculos. El siguiente resultado nos relaciona el error producido al usar patrones centrados $\mathbf{x} = \mathbf{y} - \overline{\mathbf{y}}$, $\overline{\mathbf{x}} = \mathbf{0}$ respecto al de los originales $\mathbf{y}$:

Proposición 3.3.2. *$E = \left\langle \|\mathbf{y}-\mathbf{W}\mathbf{y}\|^2 \right\rangle$ es mínimo si $\overline{\mathbf{y}} = \mathbf{0}$.*

Demostración. Para $\mathbf{W}$ fija,

$$
\begin{aligned}
E_{\mathbf{y}}(\mathbf{W}) = \left\langle \|\mathbf{y}-\mathbf{W}\mathbf{y}\|^2 \right\rangle = \\
\left\langle \|\mathbf{x}+\overline{\mathbf{y}}-\mathbf{W}(\mathbf{x}+\overline{\mathbf{y}})\|^2 \right\rangle = \left\langle \|\mathbf{x}-\mathbf{W}\mathbf{x}+\overline{\mathbf{y}}-\mathbf{W}\overline{\mathbf{y}}\|^2 \right\rangle = \\
\left\langle \|\mathbf{x}-\mathbf{W}\mathbf{x}\|^2 + \|\overline{\mathbf{y}}-\mathbf{W}\overline{\mathbf{y}}\|^2 - 2((\mathbf{I}-\mathbf{W})\mathbf{x})^T(\mathbf{I}-\mathbf{W})\overline{\mathbf{y}} \right\rangle = \\
E_{\mathbf{x}}(\mathbf{W}) + pE_{\overline{\mathbf{y}}}(\mathbf{W}) - 2\left\langle \mathbf{x}^T \right\rangle(\mathbf{I}-\mathbf{W})^T(\mathbf{I}-\mathbf{W})\overline{\mathbf{y}} = \\
E_{\mathbf{x}}(\mathbf{W}) + pE_{\overline{\mathbf{y}}}(\mathbf{W})
\end{aligned}
\tag{3.3}
$$

donde $E_{\overline{\mathbf{y}}}(\mathbf{W}) = \|\mathbf{y}-\mathbf{W}\mathbf{y}\|^2$. Sea $\mathbf{W}_{\mathbf{y}}$ aquélla $\mathbf{W}$ (posiblemente sujeta a restricciones) que minimiza $E_{\mathbf{y}}$. Entonces:

$$
\min_{\mathbf{W}} E_{\mathbf{y}} = E_{\mathbf{y}}(\mathbf{W}_{\mathbf{y}}) = E_{\mathbf{x}}(\mathbf{W}_{\mathbf{y}}) + pE_{\overline{\mathbf{y}}}(\mathbf{W}_{\mathbf{y}}) \geq E_{\mathbf{x}}(\mathbf{W}_{\mathbf{x}}) + pE_{\overline{\mathbf{y}}}(\mathbf{W}_{\mathbf{y}}) = \min_{\mathbf{W}} E_{\mathbf{x}} + pE_{\overline{\mathbf{y}}}(\mathbf{W}_{\mathbf{y}})
$$

Por tanto $\min_{\mathbf{W}} E_{\mathbf{y}} \geq \min_{\mathbf{W}} E_{\mathbf{x}} + pE_{\overline{\mathbf{y}}}(\mathbf{W}_{\mathbf{y}})$. □

Tenemos, pues, garantizado que si usamos patrones centrados obtendremos el mínimo error posible, con una diferencia de al menos $pE_{\overline{\mathbf{y}}}(\mathbf{W}_{\mathbf{y}})$. No ocurre lo mismo si usamos patrones normalizados, porque el error en un patrón dado crece o decrece dependiendo de si su norma es menor o mayor que 1, respectivamente. El error total puede, por tanto, crecer o decrecer, según sea la distribución de las normas de los patrones. Para los patrones no normalizados empleados en este trabajo, las medias de las normas fueron: para $n = 640$, $\overline{\|\mathbf{y}\|} \approx 13,34$ y $\overline{\|\mathbf{x}\|} \approx 2,43$, y para $n = 1440$, $\overline{\|\mathbf{y}\|} \approx 20,03$ y $\overline{\|\mathbf{x}\|} \approx 3,70$. En este caso, por tanto, el error E será más pequeño si se usan patrones normalizados.

Ahora, siguiendo a Baldi y Hornik, podemos ya dar una descripción de los puntos estacionarios de E. Supondremos que $\mathbf{X}\mathbf{X}^T$ es de rango completo, $\operatorname{rg}\mathbf{X}\mathbf{X}^T = n$:

- $E(\mathbf{A},\mathbf{B})$ presenta un único mínimo global y local en $\mathbf{W} = \mathbf{A}\mathbf{B}$ que verifica:
  - $\mathbf{A} = \mathbf{U}_h\mathbf{C}$, donde $\mathbf{C}_{h\times h}$ es cualquier matriz no singular y $\mathbf{U}_h = (\mathbf{u}_1,\dots,\mathbf{u}_h)$, siendo $\mathbf{u}_i$ el autovector normalizado de $\mathbf{X}\mathbf{X}^T$ asociado al autovalor λ_i, con $\lambda_1 \geq \cdots \geq \lambda_n$.
  - $\mathbf{B} = \mathbf{A}^+ = \mathbf{C}^{-1}\mathbf{U}_h^T$
  - $\mathbf{W} = \mathbf{A}\mathbf{B} = \mathbf{A}\mathbf{A}^+ = \mathbf{U}_h\mathbf{U}_h^T = \Pi_{L(\mathbf{A})} = \Pi_{L(\mathbf{u}_1,\dots,\mathbf{u}_h)}$
  - El valor mínimo de E, obtenido en $\mathbf{W} = \mathbf{U}_h\mathbf{U}_h^T$, es:

$$
E = \operatorname{tr}\mathbf{X}\mathbf{X}^T - \sum_{i=1}^{h}\lambda_i = \|\mathbf{X}\|^2 - \sum_{i=1}^{h}\lambda_i = \sum_{i=1}^{n}\lambda_i - \sum_{i=1}^{h}\lambda_i \tag{3.4}
$$

- $E(\mathbf{A},\mathbf{B})$ presenta puntos de silla para cualquier otro conjunto de autovectores en $\mathbf{U}_h$ (distinto del formado por los h principales).

Observemos que:

- A pesar de que E no es una función convexa, no tiene mínimos locales.

- Si bien $\mathbf{W} = \mathbf{U}_h\mathbf{U}_h^T$ es única, $\mathbf{A}$ y $\mathbf{B}$ no tienen por qué serlo. De hecho hay infinitas matrices $\mathbf{A}$ y $\mathbf{B}$ que verifican $\mathbf{W} = \mathbf{AB}$; basta tomar $\mathbf{A} = \mathbf{U}_h\mathbf{C}$ y $\mathbf{B} = \mathbf{C}^{-1}\mathbf{U}_h^T$, con $\mathbf{C}_{h\times h}$ invertible[9].

- Si $\operatorname{rg}\mathbf{XX}^T = n_1 < n$ (caso degenerado), el mínimo de E sigue siendo el de la ecuación (3.4), pero $\mathbf{W}$ deja de ser única si $n \geq h > n_1$, ya que los autovalores $\lambda_{n_1+1}, \ldots, \lambda_n$ son 0 y no producen varianza. En este caso, no tiene sentido hacer $h > n_1$, ya que $E = 0$ para $h = n_1$.

  Los casos degenerados aparecen muy raramente (dado el ruido experimental y la precisión limitada del ordenador)[10]; $\operatorname{rg}\mathbf{XX}^T < n$ sólo si los patrones son linealmente independientes (proposición 1.4.1). Además, siempre pueden perturbarse ligeramente los elementos de $\mathbf{XX}^T$ para hacerla no singular. En cualquier caso, una $\mathbf{XX}^T$ degenerada indica redundancia en los patrones $\mathbf{x}_i$ (pueden eliminarse los autovectores asociados a $\lambda = 0$ sin pérdida alguna de información).

  Como se vio en la proposición 1.4.1, si los patrones son centrados, $\operatorname{rg}\mathbf{XX}^T < n$. Si además el número de patrones p es menor que n, $\operatorname{rg}\mathbf{XX}^T < p$; éste será nuestro caso en la parte experimental del trabajo. Es decir, el reducido número de patrones p $(= 85)$ empleado en las simulaciones convierte a la matriz $\mathbf{XX}^T$ en degenerada.

- El error ante un nuevo patrón $\mathbf{z}$ es su distancia cuadrática al h-autoespacio $L(\mathbf{u}_1, \ldots, \mathbf{u}_h)$:

  $$E(\mathbf{z}) = \|\mathbf{z} - \Pi_{L(\mathbf{u}_1,\ldots,\mathbf{u}_h)}\mathbf{z}\|^2$$

  La capacidad de generalización de la red queda definida, pues, por la distancia euclídea del nuevo patrón al h-autoespacio.

- Es claro, pues, que el proceso de minimización de $E(\mathbf{A}, \mathbf{B})$ que tiene lugar durante el aprendizaje de la red Ξ es equivalente al análisis de componentes principales hasta el orden h de los patrones $\mathbf{X}$. La única diferencia es que la matriz $\mathbf{A}$ obtenida no tiene por qué coincidir (de hecho, en general no lo hará nunca, salvo para $h = 1$) con la matriz $\mathbf{U}_h$, sino que $L(\mathbf{A}) = L(\mathbf{U}_h)$; es decir, ambas generan el mismo subespacio, pero mientras que $\{\mathbf{u}_1, \ldots, \mathbf{u}_h\}$ es una base ortonormal de autovectores de $\mathbf{XX}^T$ de este subespacio, $\{\mathbf{a}_1, \ldots, \mathbf{a}_h\}$ es una base a secas[11]. No obstante, veremos en los resultados experimentales que $\mathbf{A}$ *tiende* a proporcionar una base ortonormal (aunque no de autovectores).

  En el caso particular $h = 1$, tanto el ACP como el aprendizaje de la red Ξ producen un autovector, y más concretamente el autovector principal. En este sentido el resultado es el mismo que el obtenido por Oja [38] usando aprendizaje hebbiano.

- Como se indicó en la sección 2.2.5, los métodos del gradiente pueden estancarse en un punto de silla; sin embargo, de acuerdo con las simulaciones realizadas por Fleming y Cottrell [15] y otros autores (incluyendo las realizadas en este trabajo), tal eventualidad parece muy improbable. Además, si se usa la retropropagación patrón a patrón (es decir, si se desciende en cada dirección en lugar de en modo *batch*, todas de golpe), el método ya no es estrictamente del gradiente (aunque esto tampoco garantiza que no se estanque).

  Resaltemos nuevamente que el método iterativo empleado por la RNA, la retropropagación, es muy lento, por ser sólo de primer orden. Dado que además el problema de optimización tratado es casi equivalente al de obtener los autovalores y autovectores de una matriz simétrica definida positiva, $\mathbf{XX}^T$, pueden usarse otros métodos numéricos especializados en esta tarea, más rápidos y eficientes (por ejemplo, los de Givens o Householder o el método QR, todos de orden $\mathcal{O}(n^3)$; para un tratamiento más completo, véase el libro clásico de Wilkinson [59]). La ventaja de la retropropagación es que es aplicable a redes no lineales, para las que no se tiene un conocimiento detallado de la estructura y propiedades de las soluciones óptimas —aunque en este trabajo todas las redes empleadas son lineales—.

- Finalmente, observemos que si fijamos $\mathbf{B}$ (o $\mathbf{A}$) con $\operatorname{rg}\mathbf{B} = h$, E es una forma cuadrática estrictamente convexa en $\mathbf{A}$ (resp. $\mathbf{B}$) con un único mínimo en $\mathbf{A} = \mathbf{B}^+$ (resp. $\mathbf{B} = \mathbf{A}^+$).

[9] Por ejemplo, intercambiar columnas en $\mathbf{A}$ y filas en $\mathbf{B}$ equivale a que $\mathbf{C}$ sea una matriz de permutación,

[10] Además, si para encontrar $\mathbf{W}$ se usa un método iterativo (por ejemplo, retropropagación) y se utilizan matrices iniciales $\mathbf{A}$ y $\mathbf{B}$ aleatorias, es posible demostrar que dichas matrices serán inicialmente de rango completo con probabilidad 1.

[11] Dada la simetría de la red Ξ, parece lógico que lo que halle sea una base en la que todos los vectores sean igualmente importantes; no hay nada en la red que privilegie a una unidad oculta sobre las demás.

3.3.2 Algoritmo acelerado para la red Ξ

Si alteramos el aprendizaje por retropropagación para que en todo momento $\mathbf{B} = \mathbf{A}^T$, el proceso resulta más eficiente, pues nos ahorramos el espacio de almacenamiento y las correcciones de la mitad de los pesos, los correspondientes a una de las matrices, $\mathbf{B}$ o $\mathbf{A}$. Es obvio que la convergencia se produce para $\mathbf{A}^T = \mathbf{A}^+ = \mathbf{B}$ y $\mathbf{W} = \mathbf{A}\mathbf{A}^+ = \mathbf{A}\mathbf{A}^T$.

Además, si $\operatorname{rg}\mathbf{A} = h$ ($= \operatorname{rg}\mathbf{A}^T = \operatorname{rg}\mathbf{A}\mathbf{A}^T$), entonces por la observación 1.4.5 $\mathbf{A}^+ = (\mathbf{A}^T\mathbf{A})^{-1}\mathbf{A}^T$ y $\mathbf{A}^+\mathbf{A} = (\mathbf{A}^T\mathbf{A})^{-1}\mathbf{A}^T\mathbf{A} = \mathbf{I}$, es decir, $\mathbf{A}$ da una base ortonormal del h-autoespacio. Si $\operatorname{rg}\mathbf{A} < h$ no tiene por qué serlo.

En este caso tampoco coinciden $\mathbf{A}$ y $\mathbf{U}_h$ en general, ya que $\mathbf{A} = \mathbf{U}_h\mathbf{C}$ con $\mathbf{C}$ ortogonal ($\mathbf{C}^{-1} = \mathbf{C}^T$) también es solución.

Baldi [3] describe este algoritmo y demuestra que su convergencia es exponencial. A continuación damos algunos detalles sobre el mismo.

En cada iteración, el descenso de gradiente se aplica sólo a una de las matrices, $\mathbf{A}$ ó $\mathbf{B}$, y la otra se actualiza trasponiendo la primera, con lo que evitamos retropropagar un nivel de pesos. Concisamente, el paso inicial ($k = 0$) consiste en:

$$\begin{aligned} \mathbf{A}_{(0)} &= \text{aleatorio} \\ \mathbf{B}_{(0)} &= \mathbf{A}_{(0)}^T \end{aligned}$$

y el paso $k+1$ (en modo *batch*, es decir, tras pasar todos los patrones por la red) en:

$$\mathbf{A}_{(k+1)} = \mathbf{A}_{(k)} - \eta\frac{\partial E}{\partial \operatorname{vec}\mathbf{A}} \tag{3.5a}$$

$$\mathbf{B}_{(k+1)} = \mathbf{A}_{(k+1)}^T \tag{3.5b}$$

Intercambiando $\mathbf{A}$ por $\mathbf{B}$ se tiene el algoritmo simétrico. Baldi conjetura que pudiera ser mejor alternar el paso del gradiente: una iteración con respecto a $\mathbf{A}$, la siguiente con respecto a $\mathbf{B}$, etc.

(3.5) puede reescribirse como

$$\mathbf{A}_{(k+1)} = \mathbf{A}_{(k)} + \eta(\mathbf{I} - \mathbf{W}_{(k)})\mathbf{X}\mathbf{X}^T\mathbf{A}_{(k)} \tag{3.6a}$$

$$\mathbf{B}_{(k+1)} = \mathbf{B}_{(k)} + \eta\mathbf{B}_{(k)}\mathbf{X}\mathbf{X}^T(\mathbf{I} - \mathbf{W}_{(k)}) \tag{3.6b}$$

donde $\mathbf{W}_{(k)} = \mathbf{A}_{(k)}\mathbf{B}_{(k)} \longrightarrow \mathbf{W} = \mathbf{U}_{\mathcal{I}}\mathbf{U}_{\mathcal{I}}^T$. En el límite, $\mathbf{W}\mathbf{u}_i = \mathbf{0}$ si $i \notin \mathcal{I}$ y $\mathbf{W}\mathbf{u}_i = \mathbf{u}_i$ si $i \in \mathcal{I}$, o lo que es lo mismo, un autovector de $\mathbf{X}\mathbf{X}^T$ asociado al autovalor λ lo es de $\mathbf{W}$ asociado a 0 ó 1. Supongamos que, en una iteración k, el autovector $\mathbf{u}$ de $\mathbf{X}\mathbf{X}^T$ asociado a λ es ya autovector de $\mathbf{W}_{(k)}$ asociado a $\mu_{(k)}$. Entonces, multiplicando (3.6a) por (3.6b) se obtiene:

$$\mathbf{W}_{(k+1)} = \mathbf{A}_{(k+1)}\mathbf{B}_{(k+1)} = \mathbf{W}_{(k)} + 2\eta(\mathbf{I} - \mathbf{W}_{(k)})\mathbf{X}\mathbf{X}^T\mathbf{W}_{(k)} + \eta^2(\mathbf{I} - \mathbf{W}_{(k)})\mathbf{X}\mathbf{X}^T\mathbf{W}_{(k)}\mathbf{X}\mathbf{X}^T(\mathbf{I} - \mathbf{W}_{(k)})$$

luego

$$\mathbf{W}_{(k+1)}\mathbf{u} = \mu_{(k)}\mathbf{u} + 2\eta(1 - \mu_{(k)})\lambda\mu_{(k)}\mathbf{u} + \eta^2(1 - \mu_{(k)})\lambda\mu_{(k)}\lambda(1 - \mu_{(k)})\mathbf{u} = \mu_{(k)}\left[1 + \eta\lambda(1 - \mu_{(k)})\right]^2\mathbf{u}$$

con lo que $\mathbf{u}$ es autovector de $\mathbf{W}_{(k+1)}$ asociado al autovalor $\mu_{(k+1)} = \mu_{(k)}\left[1 + \eta\lambda(1 - \mu_{(k)})\right]^2$. $\mu_{(k+1)}$ debe tender a 0 ó a 1, para lo cual[12] $\eta < \frac{1}{2\lambda}$. Luego, al menos en las últimas iteraciones, el factor de aprendizaje debe ser menor que $\frac{1}{2\lambda_{\max}}$, para satisfacer a todos los autovalores.

Baldi sugiere que este algoritmo puede ser más propenso a encallar en puntos de silla si se acerca a ellos, pues una vez que se aprende un autovector no principal no se puede quitar de la matriz $\mathbf{A}$. Por ello, es conveniente probar el algoritmo con simulaciones, haciendo hincapié en el punto inicial de iteración, el factor de aprendizaje, el efecto de usar el gradiente exacto (*batch*) o no, y otras modificaciones típicas del algoritmo de retropropagación, como los términos de inercia (*momentum terms*), etc.

Finalmente, Baldi y Hornik [4] señalan que, en el caso de una única unidad oculta ($h = 1$), la matriz $\mathbf{B}_{h\times n} = \mathbf{B}_{1\times n}$ es realmente un vector fila y podemos llamar $\mathbf{w} = \mathbf{B}^T$. Entonces, trasponiendo la ecuación (3.6b) se tiene:

$$\mathbf{w}_{(k+1)} = \mathbf{w}_{(k)} + \eta(\mathbf{I} - \mathbf{w}\mathbf{w}^T)\mathbf{X}\mathbf{X}^T\mathbf{w} \Leftrightarrow \Delta\mathbf{w} = \eta(\mathbf{I} - \mathbf{w}\mathbf{w}^T)\mathbf{X}\mathbf{X}^T\mathbf{w}$$

que es exactamente la regla de Oja en modo *batch*, ecuación (3.9). Luego para $h = 1$ ambos algoritmos coinciden.

Nosotros emplearemos los algoritmos generales en las simulaciones (la retropropagación y el *quickprop*), dejando para un trabajo posterior la implementación del algoritmo acelerado.

[12] Esto se demuestra analizando la función $f(x) = x\left[1 + \eta\lambda(1 - x)\right]^2$ sobre $x \in \mathbb{R}$.

3.4 Redes de neuronas artificiales para la extracción de componentes principales

Como hemos visto en la sección 3.3.1, las redes Ξ llevan a cabo durante su aprendizaje autosupervisado un proceso muy similar al ACP; la única diferencia consiste en que la base hallada por la red Ξ no es de autovectores de la matriz de covarianzas, y es ortonormal solamente si se usa el algoritmo modificado que hace $\mathbf{W} = \mathbf{A}\mathbf{A}^+ = \mathbf{A}\mathbf{A}^T$. Además, se observa empíricamente [15] que las varianzas según las direcciones de los vectores de la base obtenida tienden a tener una distribución aproximadamente uniforme, es decir, cada dirección explica aproximadamente la misma cantidad de varianza. Recordemos que en ACP puro, las varianzas decrecen monótonamente y son iguales a los autovalores de la matriz de covarianzas. Esta circunstancia puede ser o no una desventaja, según se mire. La distribución uniforme de varianzas concede a cada vector de la base (o, análogamente, a cada unidad oculta de la red) la misma importancia, con lo que la red está equilibrada, lo cual es una característica deseable en algunos casos. Por el contrario, hay aplicaciones, como la codificación de longitud variable[13], para las que es deseable tener una distribución de varianzas lo menos uniforme posible. Otra ventaja de la distribución no uniforme es que se puede saber cómo es de importante una unidad (mirando su varianza) y decidir si dejarla o no.

También hay que tener en cuenta que ciertas aplicaciones requieren no los CPs sino los componentes menores (aquéllos asociados a autovectores pequeños). Oja [38] cita algunas. En particular, O'Toole *et al.* [39] sugieren que los componentes menores son portadores de información de detalle (de alta frecuencia), por lo que pueden ser más apropiados que los CPs para la identificación entre individuos de un mismo tipo (p. ej. para diferenciar entre dos caras de personas distintas), mientras que los principales sirven para reconocer una cara, pues reconstruyen su forma y rasgos generales.

Existen, sin embargo, varios tipos distintos de RNAs que, por medio de una arquitectura distinta o de una ley de aprendizaje distinta, son capaces de realizar un ACP puro. Algunas de estas redes se describen brevemente a continuación[14]; se pueden encontrar más modelos en Oja [38] y Dony y Haykin [13].

3.4.1 Regla de Oja

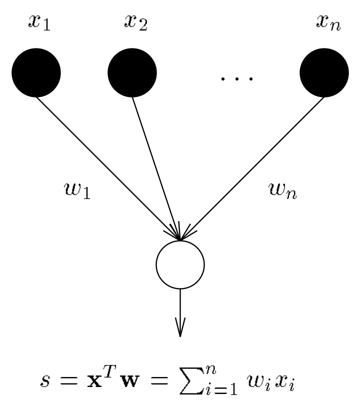

Figura 3.3: Red lineal de una sola capa con una única unidad de salida.

Supongamos que, tal como se ve en la fig. 3.3, tenemos una red que consta de una única unidad lineal. Si dotamos a la red de aprendizaje hebbiano y llamamos t al índice de iteraciones y η al factor de aprendizaje,

$$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} + \eta s^{(t)}\mathbf{x}^{(t)}$$

[13] En la cual se asignan más bits a los signos del alfabeto (en nuestro caso vectores de la base) que aparecen más frecuentemente en el mensaje (en nuestro caso que tienen una mayor varianza), con lo cual se consigue una mayor índice de compresión. Un ejemplo utilizando RNAs se da en el artículo de Sanger [50, págs. 467–469].

[14] Para todas estas redes consideraremos sólo patrones centrados $\mathbf{x}$. Si no se desea tener una fase inicial de preprocesamiento en la que se centran los vectores originales $\mathbf{y}$ restándoles su media $\overline{\mathbf{y}}$, la teoría puede reformularse introduciendo unidades lineales sesgadas (es decir, con una función de activación de la forma $\sum_{i=1}^{n} y_i w_i + w_0$), según muestran Bourlard y Kamp [9].

los pesos tienden a crecer sin limitación (la regla es inestable, tal como se vio en la sección 2.3.1). Esto puede evitarse normalizando los pesos en cada paso:

$$\mathbf{w}^{(t+1)} = \frac{\mathbf{w}^{(t)} + \eta s^{(t)}\mathbf{x}^{(t)}}{||\mathbf{w}^{(t)} + \eta s^{(t)}\mathbf{x}^{(t)}||} = \mathbf{w}^{(t)} + \eta s^{(t)}(\mathbf{x}^{(t)} - s^{(t)}\mathbf{w}^{(t)}) + \mathcal{O}(\eta^2) \tag{3.7}$$

donde se ha desarrollado en serie hasta el primer orden la fracción anterior. Esta ecuación nos da ya la regla de Oja:

$$\Delta w_i = \eta s(x_i - s w_i)$$

o, en forma vectorial,

$$\Delta \mathbf{w} = \eta s(\mathbf{x} - s\mathbf{w}) = \eta(\mathbf{x}\mathbf{x}^T\mathbf{w} - (\mathbf{w}^T\mathbf{x}\mathbf{x}^T\mathbf{w})\mathbf{w}) \tag{3.8}$$

Es posible probar que tanto la fracción de la ecuación (3.7) como la propia regla de Oja convergen (para η pequeño) al primer CP de los vectores $\mathbf{x}$. En efecto: en la posición de equilibrio se verificará $\langle \Delta\mathbf{w} \rangle = \mathbf{0} \Rightarrow \mathbf{X}\mathbf{X}^T\mathbf{w} = (\mathbf{w}^T\mathbf{X}\mathbf{X}^T\mathbf{w})\mathbf{w}$, o, lo que es lo mismo, $\mathbf{w}$ es autovector de $\mathbf{X}\mathbf{X}^T$ asociado al autovalor $\lambda = \mathbf{w}^T\mathbf{X}\mathbf{X}^T\mathbf{w} = ||\mathbf{X}^T\mathbf{w}||^2$. Por otro lado, $\mathbf{w}^T\mathbf{X}\mathbf{X}^T\mathbf{w} = \mathbf{w}^T\lambda\mathbf{w} = \lambda = \lambda||\mathbf{w}||^2$, luego $||\mathbf{w}||^2 = 1$, lo que prueba que los pesos no se disparan (permanecen en la esfera unidad). En [25, págs. 202–204] se prueba que, además, $\lambda = \lambda_{\max}$.

En resumen: la regla de Oja obtiene el autovector principal de la matriz de covarianzas (igual al de $\mathbf{X}\mathbf{X}^T$, por la tabla 1.1), es decir, hace el ACP puro hasta el primer componente.

De la ecuación (3.8), y sumando $\Delta\mathbf{w}$ para todos los patrones, obtenemos la forma de la regla de Oja en modo *batch*:

$$\langle \Delta\mathbf{w} \rangle = \eta(\langle \mathbf{x}\mathbf{x}^T \rangle \mathbf{w} - \mathbf{w}\mathbf{w}^T \langle \mathbf{x}\mathbf{x}^T \rangle \mathbf{w}) = \eta(\mathbf{I} - \mathbf{w}\mathbf{w}^T)\mathbf{X}\mathbf{X}^T\mathbf{w} \tag{3.9}$$

Un hecho destacable es que, contrariamente al algoritmo de retropropagación, la regla de Oja no puede interpretarse como el gradiente de ninguna función de error (o energía) E. Es decir, no existe $E : \mathbb{R}^n \longrightarrow \mathbb{R}$ tal que $\Delta w_i = -\eta\partial E/\partial w_i$.

Las reglas que vienen a continuación están todas basadas en la de Oja para calcular el primer CP y, a partir de él, obtener los demás:

- Ya sea usando aprendizaje antihebbiano,
- Ya restando de los datos originales la proyección sobre el primer CP y volviendo a introducir el resto en la red (esto se repite $h-1$ veces, hasta tener los h CPs), es decir, conjugando la regla de Oja con el proceso de ortogonalización de Gram-Schmidt.

3.4.2 Regla de Oja para h unidades

La regla de Oja puede ampliarse a h unidades de salida (y una sola capa) haciendo

$$\Delta w_{ij} = \eta s_i \left(x_j - \sum_{k=1}^{n} s_k w_{kj} \right) \tag{3.10}$$

donde ahora $s_k = \mathbf{x}^T\mathbf{w}_k$ o $\mathbf{s} = \mathbf{W}^T\mathbf{x}$, con $\mathbf{W}_{n\times h} = (\mathbf{w}_1, \ldots, \mathbf{w}_h) = (w_{ij})$. La nueva regla ya no es local, ya que la corrección del peso w_{ij} requiere más información que la disponible en la entrada j y la salida i. Puede demostrarse que la regla (3.10) converge a una base ortonormal $\mathbf{W} = \{\mathbf{w}_1, \ldots, \mathbf{w}_h\}$ del subespacio generado por los h primeros CPs, pero que —al igual que en el caso de las redes Ξ— no coincide en general con los primeros h autovectores de $\mathbf{X}\mathbf{X}^T$. La varianza también tiende a distribuirse uniformemente en promedio.

3.4.3 Aprendizaje hebbiano generalizado: la regla de Sanger

La regla de Sanger [50], que él llama *Generalized Hebbian Algorithm* (GHA), es muy parecida a la de Oja[15] para h unidades; solamente varía el límite superior del sumatorio:

$$\Delta w_{ij} = \eta s_i \left(x_j - \sum_{k=1}^{i} s_k w_{kj} \right) \tag{3.11}$$

[15] Obsérvese que, tanto la regla de Oja para h unidades, como la de Sanger, como las que vienen a continuación, se reducen a la regla de Oja en el caso de una sola unidad.

Sanger demuestra que su regla (que, como la de Oja para h unidades, tampoco es local) converge a los primeros h autovectores de la matriz de covarianzas ordenadamente: $\mathbf{w}_i \to \pm\mathbf{u}_i$, $1 \leq i \leq h$ y que es completamente equivalente a la KLT.

Sanger sugiere también una reformulación de la ecuación (3.11) que ya es local:

$$\Delta w_{ij} = \eta s_i \left[\left(x_j - \sum_{k=1}^{i-1} s_k w_{kj} \right) - s_i w_{ij} \right] \tag{3.12}$$

equivalente a la regla de Oja (3.8) pero sustituyendo x_j por $x_j - \sum_{k=1}^{i-1} s_k w_{kj}$; es decir, utilizando entradas modificadas.

La regla de Sanger puede extenderse también a funciones de activación no lineales. Sanger afirma que, en la práctica, el tiempo de entrenamiento es aproximadamente proporcional al número de salidas h.

3.4.4 RNA de Földiák

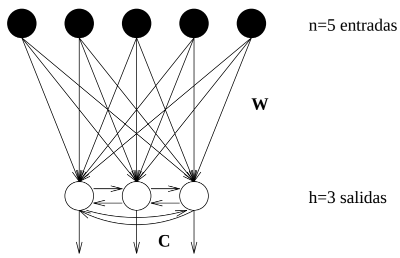

Figura 3.4: Red de Földiák.

Földiák [16] propone una RNA con n entradas y h salidas, con la estructura de la figura 3.4. Los pesos de las unidades de entrada a las de salida, $\mathbf{W}$, siguen la ley de Oja, mientras que los pesos entre las unidades de salida, $\mathbf{C}$ (llamados *decorreladores*) siguen la ley antihebbiana:

$$\Delta w_{ij} = \eta_1 s_i (x_j - w_{ij} s_i) \tag{3.13a}$$

$$\Delta c_{ij} = -\eta_2 s_i s_j \tag{3.13b}$$

Los pesos decorreladores c_{ij} valen 0 al principio y su función es la de decorrelar las variables s_i durante el aprendizaje; tienden a 0 conforme la red va convergiendo. La red halla una base de vectores del h-autoespacio, no necesariamente coincidente con los autovectores principales; su resultado es, pues, equivalente al de la red Ξ y al de la regla de Oja para h unidades. El entrenamiento es no supervisado y las reglas de modificación de pesos son completamente locales.

La salida de la red es:

$$s_i = \sum_{j=1}^{n} x_j w_{ij} + \sum_{j=1}^{h} c_{ij} s_j$$

o, en forma matricial:

$$\mathbf{s} = \mathbf{Wx} + \mathbf{Cs} \Rightarrow \mathbf{s} = (\mathbf{I} - \mathbf{C})^{-1}\mathbf{Wx}$$

donde $\mathbf{W}$ y $\mathbf{C}$ son de órdenes $h \times n$ y $h \times h$, respectivamente. η_1 y η_2 deben tomarse pequeñas (Földiák hace $\eta_1 = \eta_2 = 0.02$).

3.4.5 Adaptive Principal component EXtraction (APEX)

Kung y Diamantaras [30] proponen un modelo de RNA, al que llaman APEX, que es capaz de extraer los primeros CPs de la matriz de covarianzas, al igual que el método de Sanger, con las siguientes ventajas adicionales:

- Es posible añadir o quitar unidades de la RNA para obtener más o menos CPs sin necesidad de volver a entrenar las demás unidades.
- Se conoce un valor óptimo para el factor de aprendizaje η.

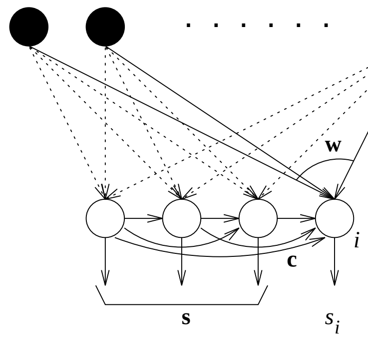

Figura 3.5: Esquema de la red APEX.

- Demuestran que la RNA converge exponencialmente[16].
- El algoritmo es paralelizable, pudiéndose obtener múltiples CPs de manera simultánea.
- Permite resolver también el problema del ACP *restringido*, en el que —como en el ACP puro— se desea maximizar la varianza de manera incorrelada pero los vectores obtenidos deben ser ortogonales a un subespacio prefijado (es decir, dicho subespacio está prohibido).

La red APEX es parecida a la de Földiák, pero las conexiones laterales entre las unidades de salida se reducen: la unidad h recibe entradas de las unidades previas $1, \ldots, h-1$ solamente (ver fig. 3.5). Las reglas de aprendizaje, basadas de nuevo en la de Oja y en la antihebbiana, son:

$$\Delta \mathbf{w} = \eta s_i (\mathbf{x} - s_i \mathbf{w}) \tag{3.14a}$$
$$\Delta \mathbf{c} = -\eta s_i (\mathbf{s} - s_i \mathbf{c}) \tag{3.14b}$$

donde $\mathbf{s} = \mathbf{W}\mathbf{x}$, $s_i = \mathbf{w}^T\mathbf{x} + \mathbf{c}^T\mathbf{s}$, $\mathbf{x} = (x_1, \ldots, x_n)^T$, $\mathbf{s} = (s_1, \ldots, s_{i-1})^T$ y $\mathbf{W}$ es la matriz de pesos para las primeras $i-1$ unidades. $\mathbf{c}$ es el vector de pesos laterales que van de las primeras $i-1$ unidades a la i-ésima. El aprendizaje es, pues, hebbiano restringido para $\mathbf{w}$, ecuación (3.14a), y antihebbiano restringido para $\mathbf{c}$, ecuación (3.14b). Como en la red de Földiák, la función de las conexiones laterales $\mathbf{c}$ es ortogonalizar o decorrelar los componentes. Conforme avanza el entrenamiento y los componentes se van haciendo ortogonales, $\mathbf{c} \to \mathbf{0}$. $\mathbf{w}$ y $\mathbf{c}$ se inicializan con valores aleatorios antes de comenzar el proceso iterativo. Para un esbozo de la demostración de la convergencia del método APEX, véase [30].

Una desventaja del método es que la precisión de los CPs obtenidos va decreciendo conforme i crece, ya que para obtener el componente i-ésimo, el método se basa en los CPs $1, \ldots, i-1$ previamente obtenidos[17]. De acuerdo con Kung y Diamantaras, los primeros 8–16 CPs se obtienen con una precisión razonable sin un tiempo excesivo de entrenamiento.

Como se indicó antes, Kung y Diamantaras obtienen el valor óptimo para el factor de aprendizaje, conocimiento del que se carece en muchos otros métodos. Resulta ser

$$\eta_t = \frac{1}{\sum_{j=t-M+1}^{t} s_j^2}$$

donde t es el índice de iteraciones y M es la *ventana de recuerdo* e indica cuántos valores de s_j anteriores al actual tener en cuenta; M la fija el usuario.

[16] La convergencia exponencial ocurre cuando el error en la iteración t es proporcional al de la iteración $t-1$:

$$\epsilon^{(t)} = \eta\epsilon^{(t-1)} \Rightarrow \epsilon^{(t)} = \eta^t \epsilon_0$$

Lo cual equivale a decir que el método es de primer orden.

[17] Esta desventaja es común a todos los métodos de deflacción [59], que aplican unas operaciones para obtener un autovector o autovalor y reducen la matriz de orden $n \times n$ a $(n-1) \times (n-1)$; a esta matriz se le aplica recursivamente el proceso. Los errores (debidos al redondeo, o a concluir una serie de iteraciones antes de haber obtenido una convergencia exacta) se van acumulando y pueden hacerse inaceptables tras pocos pasos en la recursión. Lo mismo le ocurre al proceso de ortogonalización de Gram-Schmidt.

Capítulo 4

Experimentos, parte I: Extracción de características

Con este capítulo damos paso a las simulaciones realizadas con diversas redes de compresión, utilizando el programa *SNNS*. En él nos limitamos a demostrar empíricamente lo afirmado en la sección3.3.1, es decir, la coincidencia de los subespacios generados por los vectores de pesos de la red y por los primeros autovectores de la matriz de covarianzas. La red de compresión funciona, pues, como un dispositivo que recibe un vector de n componentes reales y devuelve un vector de características de $h \ll n$ componentes reales, que representa —con mayor o menor acierto— al vector original y que, gracias a su menor dimensión, puede ser más apropiado para entrar en un proceso de clasificación.

4.1 Descripción de los patrones usados

Se dispone de un total de 6 fotos de la oreja izquierda para 17 individuos, tal como se muestra en la figura 2.7. Como se indicó en la sección 2.4.1, para la fase de extracción de características se empleó un conjunto de entrenamiento TS formado por $\frac{5}{6}$ del total de patrones, es decir, 5 fotos por individuo (elegidas al azar); las 17 fotos restantes se emplean para validar la red, es decir, conforman el VTS1. Adicionalmente se cuenta con otras 17 imágenes que conforman un segundo conjunto de validación VTS2. Dado que nos interesa llevar a la red lo más cerca posible de su mínimo para comprobar que alcanza los CPs, debemos entrenar la red tanto como sea necesario para que la curva de error E se estabilice. Por ello no usamos el conjunto de prueba de entrenamiento, TTS.

A partir de la misma serie original de fotos (fichero `/orejas/pgm/orejapgm.tgz`) se crearon dos conjuntos de patrones, uno con imágenes de 20×32 y el otro de 30×48. Ambos conjuntos recibieron el mismo tratamiento: 1) análisis espectral completo de $\mathbf{X}\mathbf{X}^T$; 2) entrenamiento de la red, para distintos números de unidades ocultas h; y 3) resultados.

4.2 Caso 20×32

4.2.1 Análisis espectral de $\mathbf{X}\mathbf{X}^T$

El análisis espectral de $\mathbf{X}\mathbf{X}^T$ sirve como base para la comparación con los resultados de la red Ξ. Este análisis se realizó sobre la matriz $\mathbf{X}^T\mathbf{X}$, para reducir el tiempo de cálculo (como consecuencia de la proposición 1.4.1); el rango de $\mathbf{X}^T\mathbf{X}$ y de $\mathbf{X}\mathbf{X}^T$ resulta ser 84 en ambos casos (no es completo para $\mathbf{X}^T\mathbf{X}$ ya que los vectores están centrados; ver la observación 1.4.1); pero mientras la dimensión de $\mathbf{X}\mathbf{X}^T$ es 640, la de $\mathbf{X}^T\mathbf{X}$ es tan sólo 85. $\mathbf{X}\mathbf{X}^T$ tiene, pues, 84 autovalores positivos y el autovalor $\lambda = 0$ con multiplicidad $640 - 84 = 556$. Para los cálculos se usó *Mathematica*; las imágenes, en formato PGM, se pasaron con el *shellscript* `pgmtomat` (ver la sección D.4) a una matriz $\mathbf{X}$ en formato de *Mathematica*, sobre la cual se hizo el análisis de CPs (después de haber centrado los vectores $\mathbf{y}$, restándoles su media $\overline{\mathbf{y}}$).

En suma, el análisis espectral de $\mathbf{X}\mathbf{X}^T$ nos lleva a calcular los autovalores $\lambda_1 \geq \cdots \geq \lambda_n$ y la matriz $\mathbf{U} = (\mathbf{u}_1, \ldots, \mathbf{u}_n)$ de la sección 3.3.1, así como el error E en función del número de unidades ocultas h, fórmula (3.4).

La figura 4.1 da los autovalores de $\mathbf{X}\mathbf{X}^T$ (recordemos que, en análisis de CPs, se conoce a estas figuras como gráfico *scree*, tal y como se indicó en la sección 3.1); y la 4.2, el error cuadrático E en función del

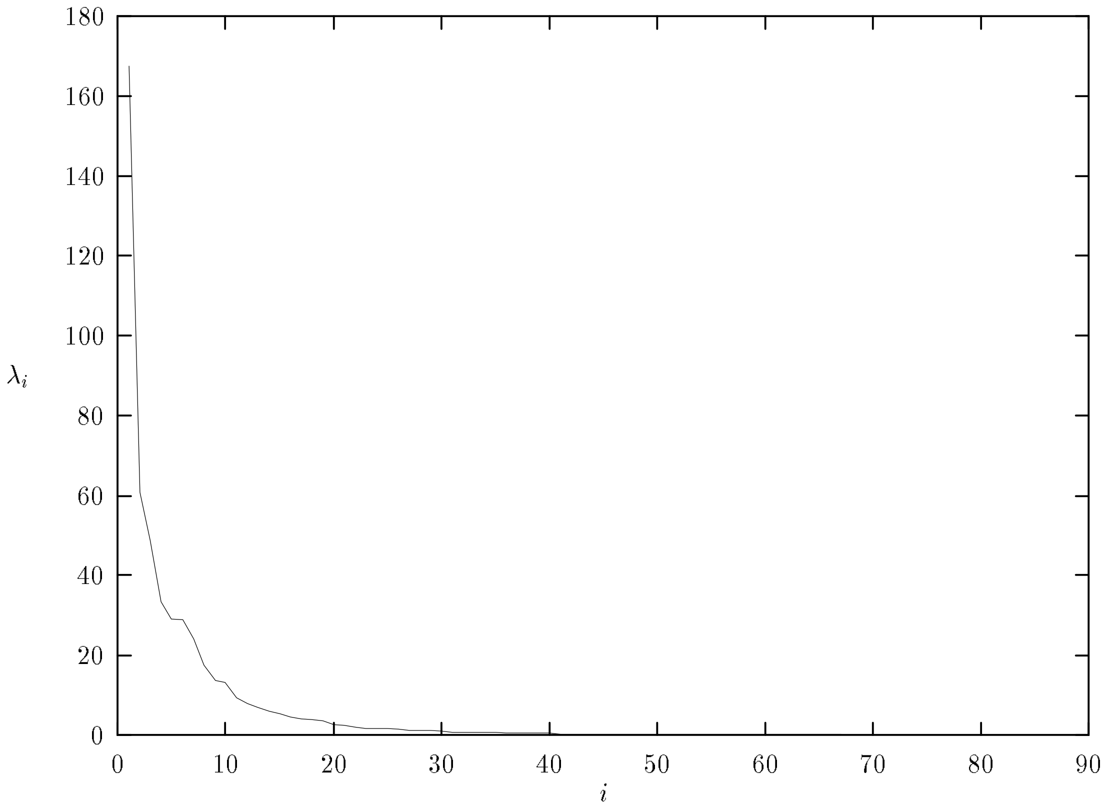

Figura 4.1: Autovalores de $\mathbf{XX}^T$ para el caso 20×32.

número de unidades ocultas, según la fórmula (3.4). Los puntos de corte indicados son los valores de E para $h = 1, 5, 10, 20, 30, 50$. Como vemos, queda justificado mantener unos pocos autovalores, o lo que es lo mismo, unos pocos componentes principales, ya que el error decrece rápidamente al principio de la curva.

Las figuras[1] 4.3, 4.4 y 4.5 dibujan los 85 vectores proyectados en los planos formados por los pares de componentes principales $(\mathbf{u}_1, \mathbf{u}_2)$, $(\mathbf{u}_1, \mathbf{u}_{11})$ y $(\mathbf{u}_1, \mathbf{u}_{84})$; el origen de coordenadas es el centro de masas $\overline{\mathbf{y}}$. Vemos que los puntos se reparten de manera aproximadamente normal y que $\mathbf{u}_1$ presenta la mayor dispersión, seguido por $\mathbf{u}_2$, hasta llegar al último autovector, $\mathbf{u}_{84}$, cuya dispersión es muy pequeña (nótese el cambio de escala vertical en la figura 4.5).

Una vez conocidos los autovalores no nulos de $\mathbf{XX}^T$ (iguales a los de $\mathbf{X}^T\mathbf{X}$), las proposiciones 1.4.3 y 1.4.2 nos permiten obtener los números de condición de las diversas matrices: $c(\mathbf{\Sigma}) = c(\mathbf{XX}^T) = c(\mathbf{X}^T\mathbf{X}) \approx 8331$, $c(\mathbf{X}) \approx 91$, ya que el intervalo de variación de los autovalores de $\mathbf{XX}^T$ es $[\lambda_n, \lambda_1] \approx [0.02, 168]$. Estos números están lejos del inverso de la precisión del ordenador $(3 \cdot 10^7)$, por lo que todas las matrices están bien condicionadas y soportan los métodos numéricos sin que se pierda precisión.

La figura 4.6 muestra una representación pictórica de los autovectores de $\mathbf{XX}^T$. Esta representación se obtuvo normalizando linealmente el intervalo de variación de dichos autovectores a 256 tonos de grises, empleando la fórmula

$$u'_{ij} = \frac{b-a}{M-m}(u_{ij} - m) + a \qquad M = \max_{i,j}\{u_{ij}\} \qquad m = \min_{i,j}\{u_{ij}\} \tag{4.1}$$

o, en forma matricial,

$$\mathbf{U}' = k\mathbf{U} + \mathbf{K} \qquad k = \frac{b-a}{M-m} \qquad k_{ij} = \frac{aM - bm}{M-m}$$

que normaliza el conjunto de valores $\{u_{ij}\}$, inicialmente sobre el dominio $[m, M]$, al $[a, b]$. El hacer la normalización global (es decir, m y M recorren todo el conjunto de autovectores y no sólo uno de ellos) permite comparar globalmente la distribución de intensidades.

[1] Los valores de los puntos (x, y) de estas figuras se obtuvieron con *Mathematica* mediante un comando de la forma `Xt.Transpose[{Ut[[i]],Ut[[j]]}]`, donde `i` y `j` apuntan a las CPs, contenidas en la lista de autovectores `Ut`, y `Xt` es otra lista que contiene los vectores centrados `x`.

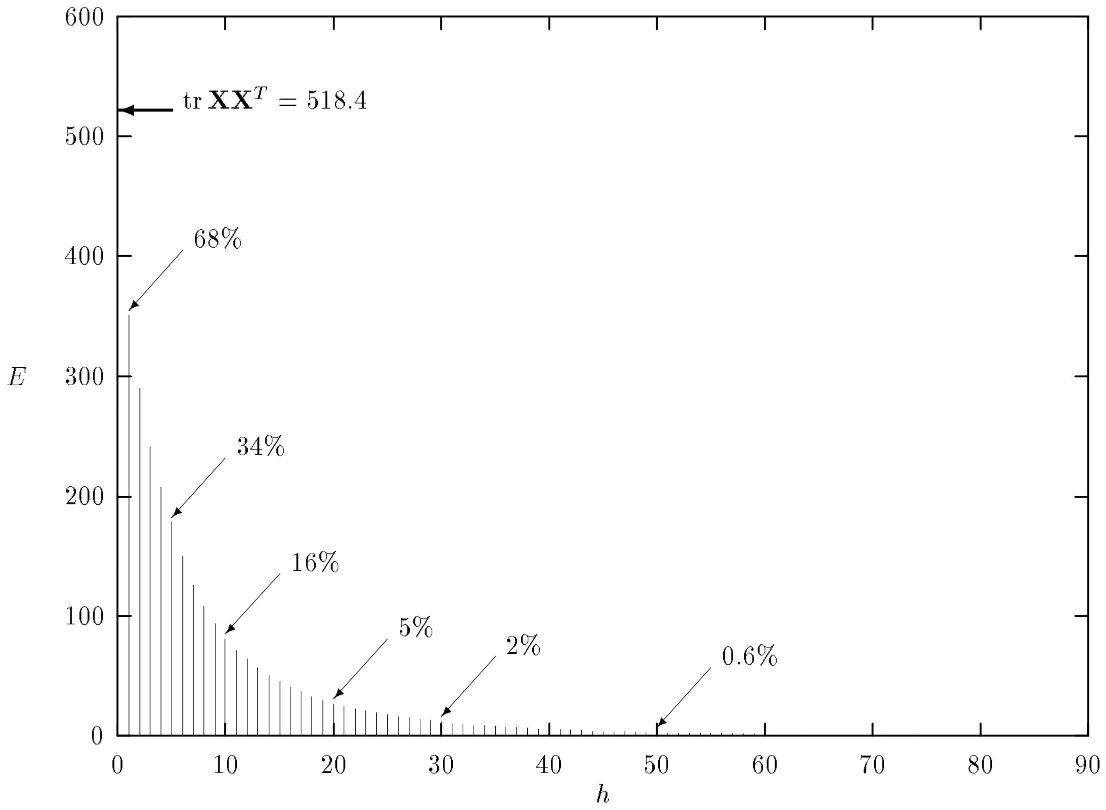

Figura 4.2: Errores cuadráticos E para el caso 20×32.

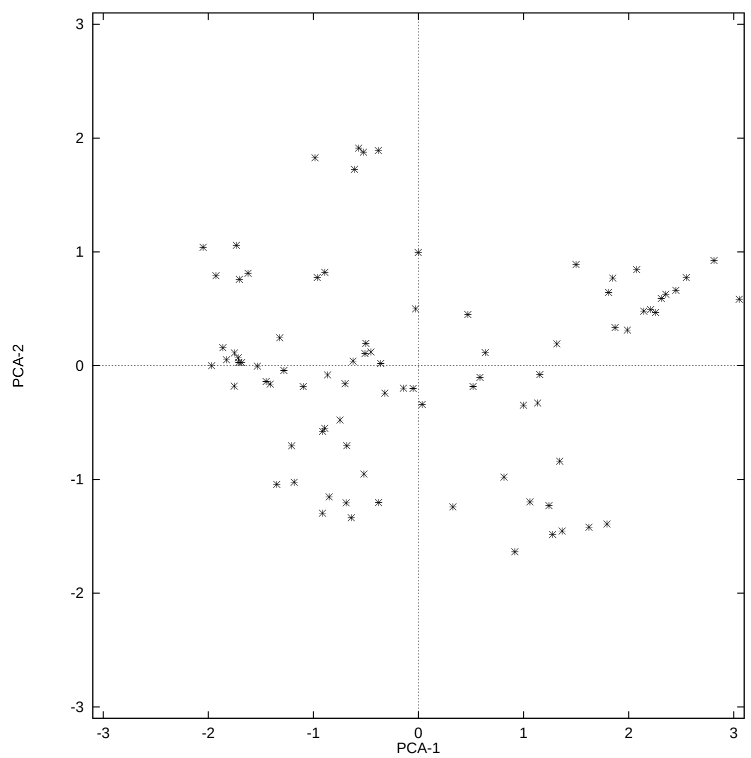

Figura 4.3: Proyección de los vectores centrados en el plano de los componentes principales $\mathbf{u}_1$ y $\mathbf{u}_2$ (caso 20×32).

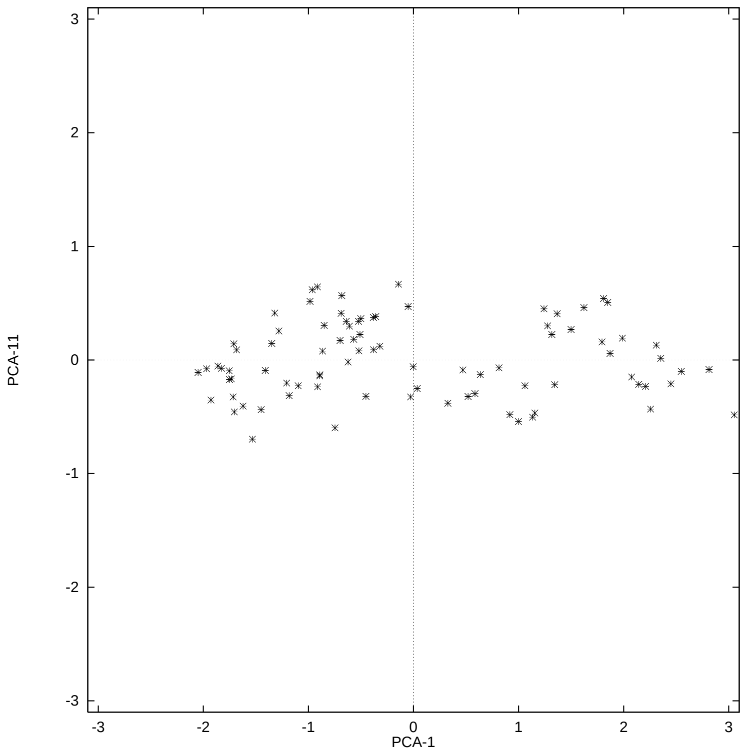

Figura 4.4: Proyección de los vectores centrados en el plano de los componentes principales $\mathbf{u}_1$ y $\mathbf{u}_{11}$ (caso 20×32).

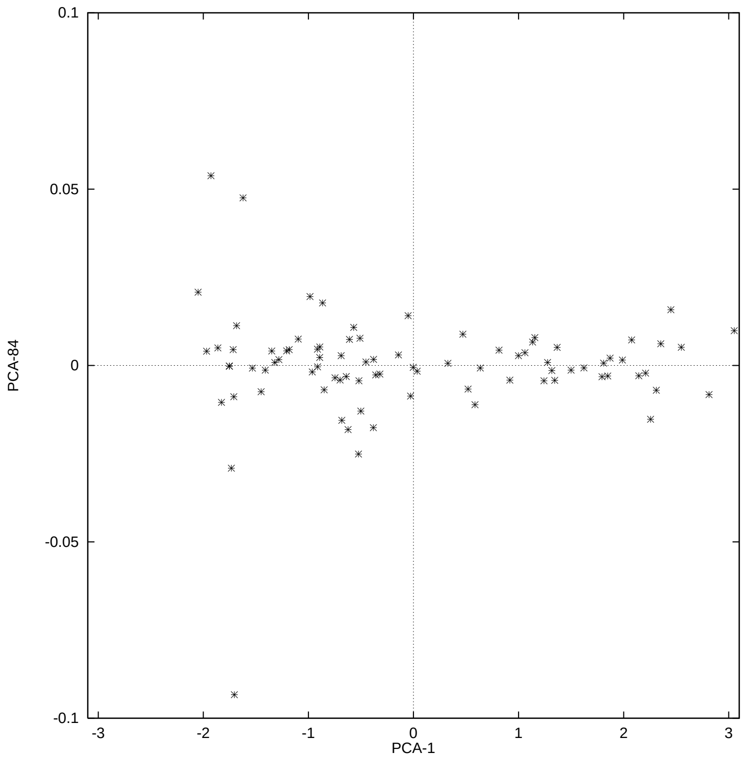

Figura 4.5: Proyección de los vectores centrados en el plano de los componentes principales $\mathbf{u}_1$ y $\mathbf{u}_{84}$ (caso 20×32). Nótese el cambio de escala vertical.

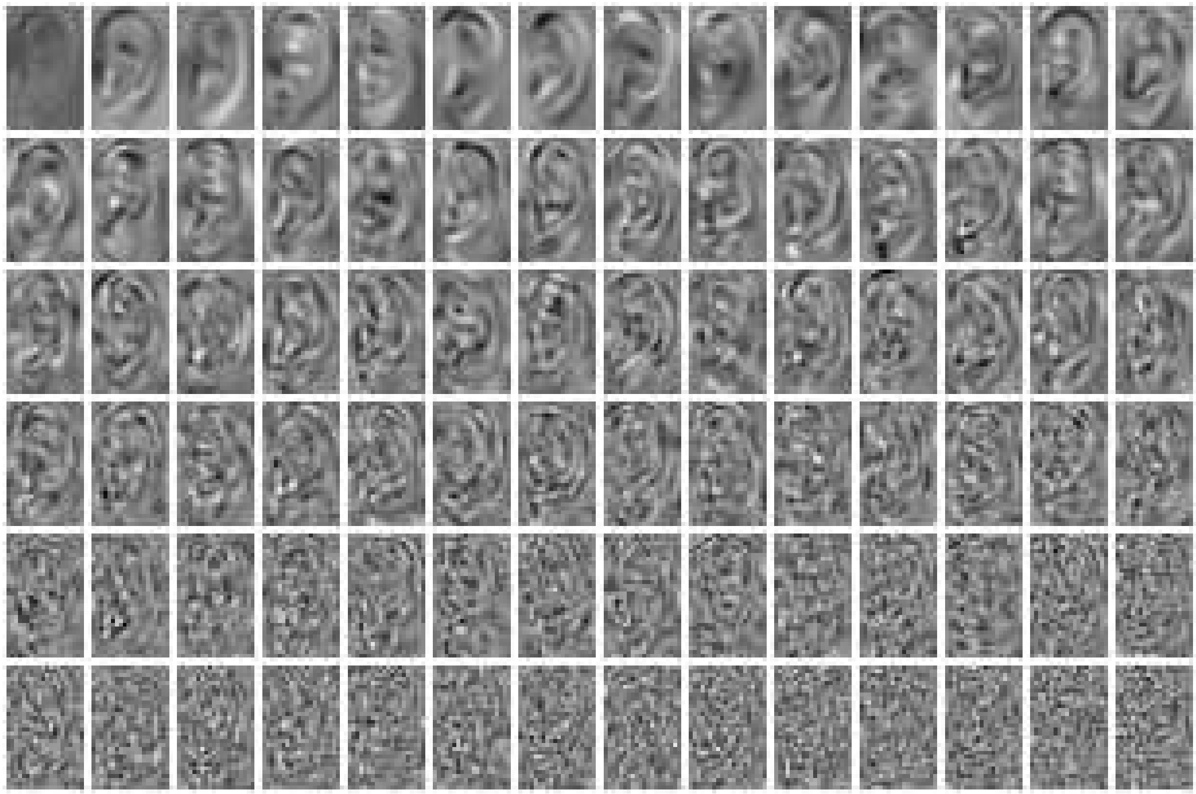

Figura 4.6: *Holones* de los autovectores principales $\mathbf{u}_i$ para el caso 20×32.

Fleming y Cottrell [15] llaman a las representaciones anteriores *holones*, en su estudio aplicado a imágenes faciales. De acuerdo con el *McGraw-Hill's Dictionary of Science* [37], en reconocimiento de caracteres se entiende por *marcas holísticas* (*holistic marks*) "aquel conjunto de caracteres que reside dentro de una lectora de caracteres y que —teóricamente— es capaz de representar todos los posibles caracteres de entrada." Esta definición puede aplicarse sin más al conjunto de autovectores mencionado.

Obsérvese cómo los primeros autovectores tienen una clara apariencia de oreja izquierda mientras los últimos parecen ruido blanco. Los primeros autovectores dan la forma general de la oreja y los últimos dan información de detalle; es decir, los primeros resultan útiles para la reconstrucción aproximada del objeto y su reconocimiento como oreja, pero los últimos pueden conllevar información útil para la identificación de uno de los patrones en particular [52, 39].

En particular, nótese que el primer autovector (el autovector dominante) resulta muy homogéneo y con una intensidad bastante oscura. Ello se debe a que la red obtuvo ese autovector con todos sus componentes negativos, por lo que la normalización (4.1) las coloca en la parte baja del intervalo $[0, 255]$, que es la más oscura. Si el autovector se multiplica por -1, seguiría siendo autovector (mantendría la misma apariencia) pero saldría mas brillante; puede comprobarse este hecho en la figura 4.30, del caso 30×48, en el que la red obtuvo un $\mathbf{u}_1$ de componentes positivos.

La uniformidad del autovector dominante es un fenómeno general, puesto que partimos de vectores $\mathbf{y}$ que tienden a tener su componente principal aproximadamente a lo largo de la hiperbisectriz del hiperoctante positivo (o negativo, si tomamos el sentido opuesto). Los componentes del autovector dominante varían en un intervalo muy estrecho, son todos positivos o todos negativos (según el signo que se considere) y su valor absoluto es pequeño[2]. Concretamente, el intervalo de variación es $[0.0026, 0.088]$ para nuestro caso, con un recorrido de $0.088 - 0.0026 \approx 0.086$. Los demás autovectores presentan un intervalo de variación mayor, pues tienen componentes positivos y negativos; por ejemplo, para el segundo autovector este intervalo es $[-0.12, 0.083]$, con un recorrido de $0.083 + 0.12 \approx 0.2$ (más del doble del de $\mathbf{u}_1$), con lo cual presenta píxeles muy oscuros y muy brillantes (mayor rango dinámico).

En cualquier caso, no debe olvidarse que la normalización (4.1) no tiene ningún efecto en la red ni en los autovectores; su única función es permitir "ver" en tonos de gris cada autovector.

[2] Como su norma vale 1 y todos los componentes son aproximadamente iguales, en media cada componente valdrá $1/\sqrt{640} \approx 0.04$ en valor absoluto

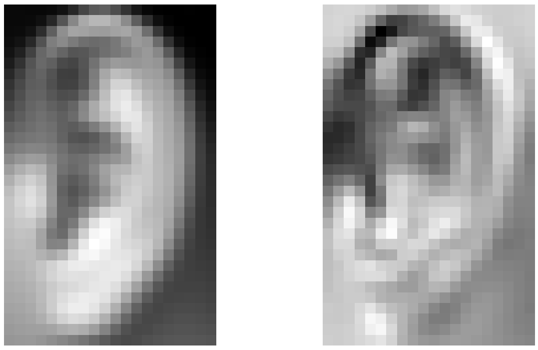

Figura 4.7: Media $\overline{\mathbf{y}}$ para el caso 20×32 (a la izquierda) y autovector principal $\mathbf{u}_1$ (a la derecha, normalizado a 256 tonos de gris).

La figura 4.7 muestra el vector media[3] $\overline{\mathbf{y}}$ de los patrones originales $\mathbf{y}$ (no centrados). Su aspecto borroso es debido a que el operador media es un filtro de paso bajo. A su lado se muestra el primer autovector (que, esta vez, ha sido normalizado pero de manera no global, para aparecer menos oscuro; es decir, M y m en la fórmula (4.1) recorren sólo el vector $\mathbf{u}_1$).

4.2.2 Medidas empleadas sobre la red

Partiendo de la base fijada por el análisis espectral, óptimo en el sentido de mínimos cuadrados, procedemos ahora a estudiar las características de las bases encontradas por la red Ξ. Sobre estas bases de vectores se van a evaluar una serie de medidas relacionadas con las siguientes características:

- Ortonormalidad de la base obtenida, es decir:
  - Distribución de las normas de cada vector de la base
  - Distribución de los ángulos (o de los productos escalares) entre los mismos

  Por supuesto, dada una base cualquiera, siempre puede obtenerse una ortonormal del mismo subespacio a partir de ella aplicando el procedimiento de Gram-Schmidt (o, de una manera más eficiente desde el punto de vista numérico, haciendo una descomposición en valores singulares de la matriz de dicha base, ver [43]), pero ello exige un paso adicional, una vez entrenada la red. Además se trata de una operación no local, ya que requiere conocer los valores de todos los pesos a la vez.

  Se mostrarán gráficos de "ortonormalidad" en los que el eje x da la norma media y el y el ángulo medio, y cada punto se encuadra en un intervalo de confianza.

- Grado de parecido del subespacio generado por $\mathbf{A}$ y $\mathbf{B}$ con el generado por los h primeros autovalores de $\mathbf{X}\mathbf{X}^T$. Para ello se tomarán las medidas siguientes (ver fig. 4.8):

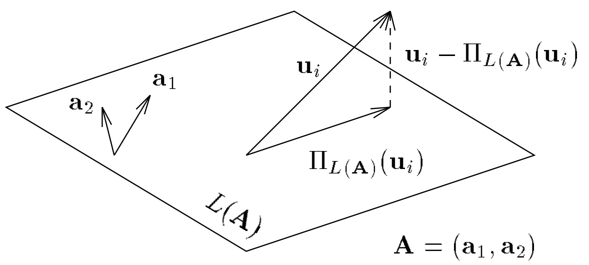

Figura 4.8: Proyección sobre el subespacio $L(\mathbf{A})$ y error generado.

  - La norma de la proyección de cada autovector de $\mathbf{X}\mathbf{X}^T$ sobre $L(\mathbf{A})$ y $L(\mathbf{B})$: $\|\Pi_{L(\mathbf{A})}(\mathbf{u}_i)\|$.
  - La norma del error entre dicha proyección y el propio autovector proyectado: $\|\mathbf{u}_i - \Pi_{L(\mathbf{A})}(\mathbf{u}_i)\|$.

[3]Nótese que el error para el vector media de los patrones del conjunto TS, $\overline{\mathbf{y}}$, es siempre $E_{\overline{\mathbf{y}}} = 0$, porque dicho vector se transforma en el $\mathbf{0}$ antes de entrar en la red.

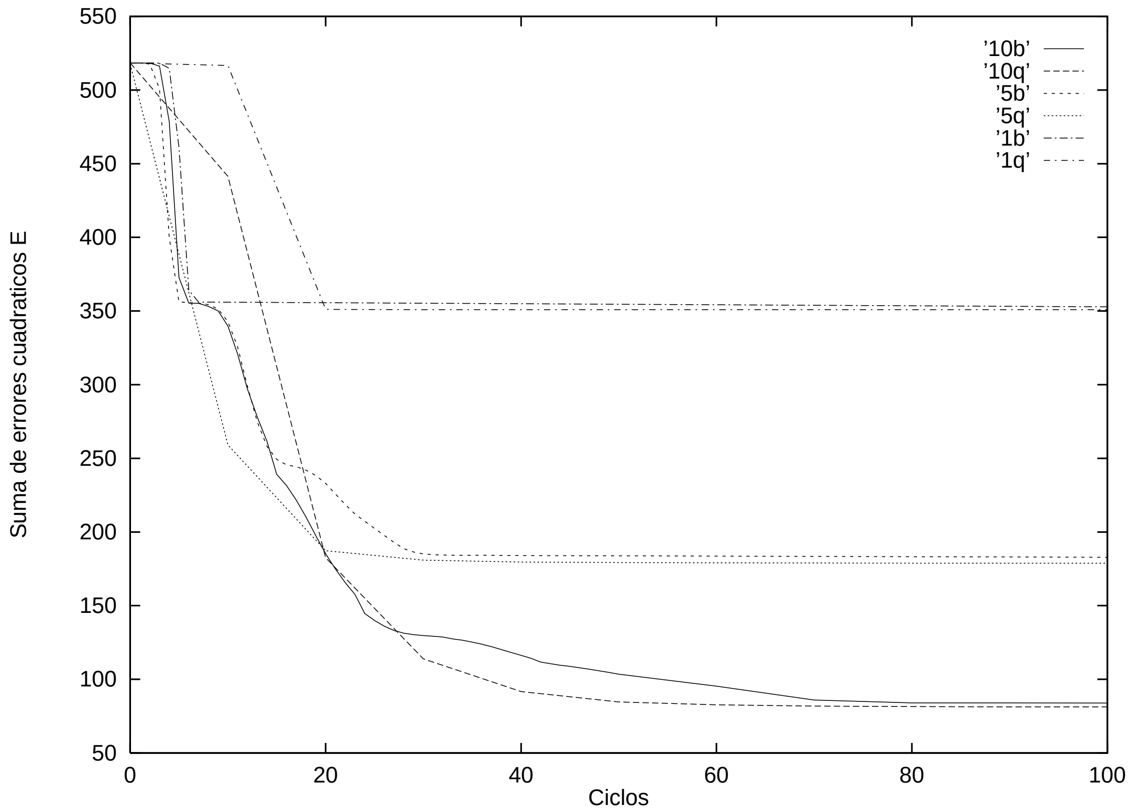

Figura 4.9: Caso 20×32: curvas de aprendizaje para $h = 1$, 5 y 10, con los algoritmos de retropropagación y *quickprop*.

- Grado de parecido de $\mathbf{B}$ y $\mathbf{A}^+$: se medirá como $\|\mathbf{B} - \mathbf{A}^+\|$. La matriz pseudoinversa de $\mathbf{A}$, $\mathbf{A}^+$, se calcula separadamente con *Mathematica*. En el mínimo de E, $\mathbf{A}^+ = \mathbf{B}$ y $\|\mathbf{B} - \mathbf{A}^+\| = 0$.

- Distribución de las varianzas a lo largo de las direcciones de $\mathbf{A}$ y de $\mathbf{B}$: la varianza multiplicada por el número de patrones p en la dirección del vector $\mathbf{v}$ puede calcularse como (cf. proposición 1.4.7): $p\,\mathrm{var}_{\mathbf{v}}(\mathbf{y}) = \mathbf{v}_1^T \mathbf{X}\mathbf{X}^T \mathbf{v}_1 = \|\mathbf{X}^T \mathbf{v}_1\|^2$, donde $\mathbf{v}_1$ es un vector unitario en la dirección de $\mathbf{v}$. Sabemos además que, si $\mathbf{v}$ es autovector de $\mathbf{X}\mathbf{X}^T$ asociado al autovalor λ, $p\,\mathrm{var}_{\mathbf{v}}(\mathbf{x}) = \mathbf{v}_1^T \mathbf{X}\mathbf{X}^T \mathbf{v}_1 = \mathbf{v}_1^T(\lambda \mathbf{v}_1) = \lambda$, es decir, las p-varianzas en las direcciones principales coinciden con los autovalores asociados (y evidentemente ocurre lo mismo con $\mathbf{\Sigma} = \frac{1}{p}\mathbf{X}\mathbf{X}^T$ y $\frac{\lambda}{p}$), como ya sabíamos del análisis de componentes principales.

Estas medidas se tomarán no sólo de la base final, sino también de etapas intermedias del entrenamiento, en las que el error aún no ha alcanzado el mínimo, lo cual permitirá ver la evolución de las matrices $\mathbf{A}$ y $\mathbf{B}$.

El proceso seguido para obtener dichas medidas consiste en convertir el fichero generado por el simulador *SNNS* (que contiene los pesos de la red) en un fichero de *Mathematica*, usando el *shellscript* `nettomat` (ver la sección D.4), y ejecutar *Mathematica* sobre dicho fichero con las funciones y programas que aparecen en la sección D.4. Los gráficos correspondientes se obtuvieron pasando las medidas mencionadas a *gnuplot*.

Procedimiento seguido durante el aprendizaje

La figura 4.9 muestra el valor de E en función del número de ciclos iterados.

Un **ciclo** o **iteración** consiste en una pasada completa de todos los patrones por la red, desordenados en cada ciclo (*shuffled*) o no, con las modificaciones correspondientes de los pesos (que pueden tener lugar en modo *batch* o en modo *on-line*). Nosotros usamos patrones desordenados y modificación *on-line*.

El aprendizaje se repitió para redes con $h = 1$, 5 y 10 unidades ocultas, empleando en cada caso los dos algoritmos de aprendizaje mencionados previamente: retropropagación (*backprop*) y *quickprop*. En dicha figura se muestran las curvas de aprendizaje para todas las combinaciones. La leyenda de la figura indica qué valor de h y qué algoritmo corresponden a la curva que sea; por ejemplo, `10b` indica $h = 10$ y retropropagación.

En cada caso, el aprendizaje se continuó hasta alcanzar al menos 4 decimales de precisión en el valor mínimo de E (que conocemos para cada caso, de la fórmula (3.4)). La razón es obtener la mayor precisión posible[4] en los vectores determinados por la red, para compararlos con los autovectores de $\mathbf{XX}^T$. En un caso práctico, sin embargo, basta detener el aprendizaje cuando se vea que la curva se estabiliza: en la figura 4.9 este punto estaría entre 20 y 80 iteraciones, dependiendo de h y del algoritmo de aprendizaje.

Obsérvese que sólo se muestra la parte de las curvas correspondiente a los 100 primeros ciclos, para poder observar mejor la bajada abrupta del error en las primeras iteraciones. Todas las curvas alcanzan un valor muy próximo a su mínimo en menos de 80 iteraciones; a partir de ahí E sigue decreciendo, pero a un ritmo muy lento. Por ejemplo, la curva `1b` requirió unas 2300 iteraciones en alcanzar 5 decimales de precisión (error absoluto: $5 \cdot 10^{-6}$; error relativo: 10^{-8}) para E; en la iteración número 100 el error absoluto era de 1.76 y el relativo de 0.005.

Cada red se inicializó aleatoriamente al principio de la siguiente manera: el valor de cada peso sigue una distribución uniforme $[-\epsilon, \epsilon]$ donde ϵ es un valor lo suficientemente pequeño como para que la convergencia sea rápida y, sobre todo, para evitar el desbordamiento en coma flotante durante el aprendizaje. El apéndice A.4 justifica nuestra elección en detalle. Nosotros tomamos $\epsilon = 0.001$ para todos los valores de h, tanto con redes de 20×32 como de 30×48.

Como factor de aprendizaje se tomó $\eta = 0.001$ por lo general; en algún caso de 20×32 se usó $\eta = 0.01$. Este valor lo cambiábamos dinámicamente de manera que, cuando el valor actual de η (0.001 al principio) ya no podía reducir más el error E, dividíamos η por 10 y continuábamos el aprendizaje; no obstante, llegados al primer cambio de η (que normalmente tenía lugar tras varios miles de iteraciones), se alcanzaba la convergencia a 4–5 decimales en unas pocas decenas de iteraciones más.

Las curvas muestran que el algoritmo *quickprop* aventaja al de retropropagación a secas a las pocas iteraciones: tras 10–20 ciclos en los que la pendiente de E varía rápidamente, la curva de *quickprop* está siempre por debajo de la de la retropropagación, prácticamente ya en el valor mínimo de E. Esto también se aprecia, de manera mucho más clara, en la figura 4.32, del caso 30×48; para $h = 20$, *quickprop* consigue en 70 iteraciones lo que la retropropagación en 800.

Ahora ya podemos ver, para cada valor de h, los resultados obtenidos.

4.2.3 Análisis de los resultados obtenidos por las redes

Caso $h = 1$

Este caso es especial, porque, como ya sabemos, es el único en el que la base hallada por la red Ξ (que consta de un único vector) coincide exactamente con el análisis de CPs. Tanto la matriz $\mathbf{A}$ como la $\mathbf{B}$ son un autovector dominante (columna o fila, respectivamente), que en general no está normalizado. Los comentarios siguientes se dan para $\mathbf{A}$, pero son igualmente válidos para $\mathbf{B}$ (excepción hecha de lo mencionado en la sección 4.2.4).

Los valores de $||\mathbf{u}_1 - \Pi_{L(\mathbf{A})}\mathbf{u}_1||$ y $||\Pi_{L(\mathbf{A})}\mathbf{u}_1||$ son 0.000305 y 1, respectivamente, lo que indica que $\mathbf{u}_1$ y el vector fila $\mathbf{A}$ están en la misma dirección. Para cualquier otro autovector $\mathbf{u}_i$ de $\mathbf{XX}^T$, esos valores son de 0 y 1 hasta 4 decimales de precisión; es decir, el vector fila $\mathbf{A}$ es ortogonal a los demás autovectores.

La p-varianza en la dirección de $\mathbf{A}$ resulta valer 167.509, muy cercana al autovalor dominante de $\mathbf{XX}^T$, $\lambda_1 = 167.509220820296$ (calculado con *Mathematica*).

$||\mathbf{B} - \mathbf{A}^+|| = 0.026$, luego $\mathbf{B}$ es prácticamente igual a la pseudoinversa de $\mathbf{A}$.

Finalmente, para la red entrenada con retropropagación, la norma del vector columna $\mathbf{A}$ resulta ser, tras 2300 iteraciones, 0.252, y la del vector fila $\mathbf{B}$ 3.96; su producto vale 0.997, es decir, aproximadamente 1 (como debe ser si $\mathbf{B} = \mathbf{A}^+$). Para la misma red con *quickprop*, las normas (tras 200 ciclos) valen 1.06 y 0.955 y su producto 1.012. Obviamente no es necesario que las normas de $\mathbf{A}$ y $\mathbf{B}$ (considerados vectores) valgan 1 siempre, aunque pueda ser deseable. Durante otras simulaciones de prueba se observó que las normas tendían a quedarse cerca de 1.

Lo anterior se refería siempre a valores de medidas en el punto de convergencia del aprendizaje (o muy cerca de él). Ahora nos fijaremos en puntos intermedios, para ver cómo evolucionan estas medidas.

Sólo se tomaron medidas intermedias para el aprendizaje por retropropagación[5].

[4] Es importante no olvidar que (véase [43, pág. 387]), debido al redondeo interno del ordenador, no debe esperarse –en general– de ningún algoritmo iterativo de minimización una precisión en la solución mayor que aproximadamente la mitad de dígitos significativos que ofrezca el ancho de palabra del procesador. Para precisión simple (que es la que emplea el simulador utilizado, *SNNS*), esto supone que no podremos obtener más de 4 ó 5.

[5] Por dos razones: a) estas mismas medidas para *quickprop* no añaden información nueva; b) resultan costosas de calcular: para cada una hay que "congelar" la red (cuyo fichero asociado puede ser muy grande: de 119873 bytes para el caso 20×32,

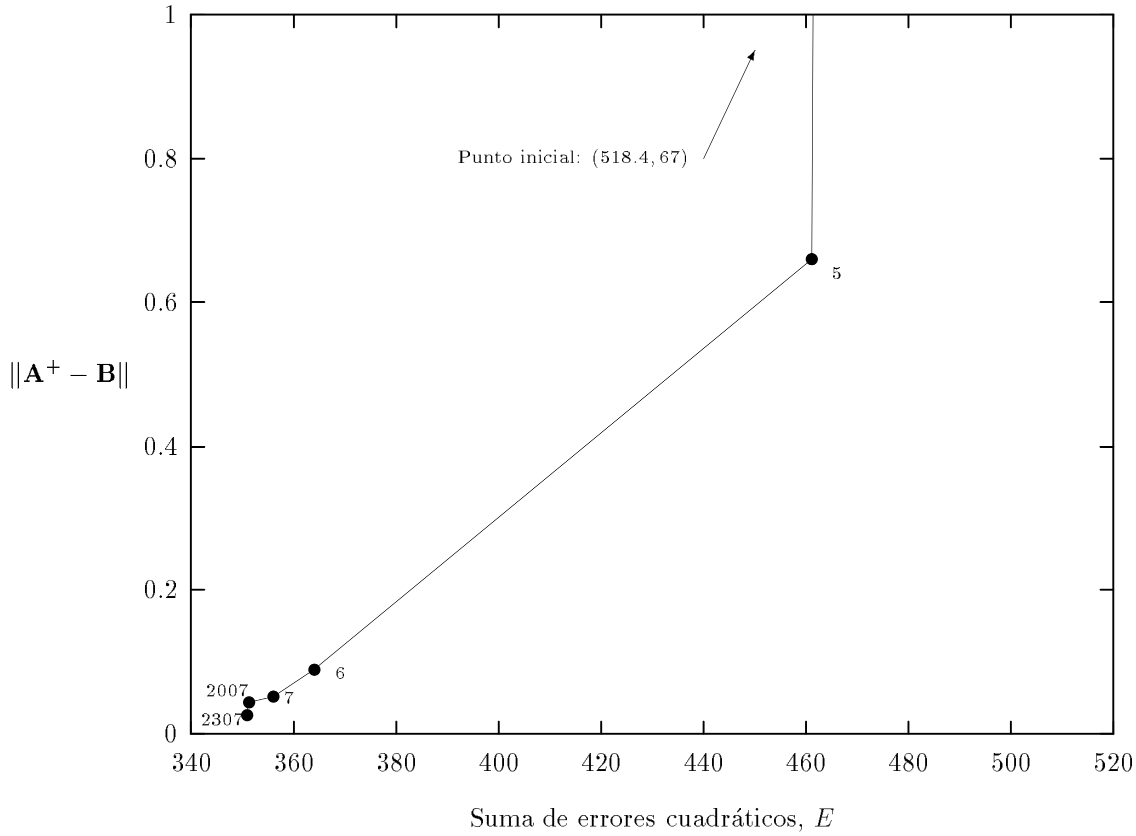

Figura 4.10: Caso 20×32, $h = 1$: evolución de $||\mathbf{B} - \mathbf{A}^+||$.

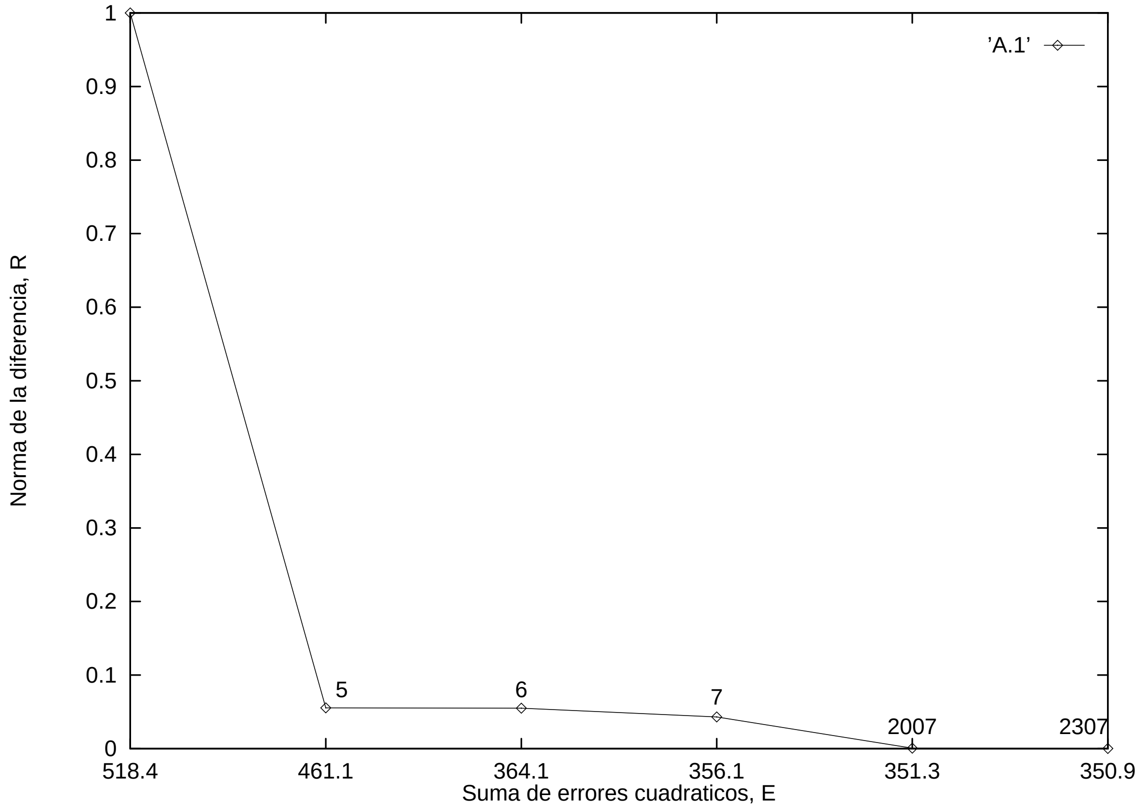

Figura 4.11: Caso 20×32, $h = 1$: evolución de $||\mathbf{u}_1 - \Pi_{L(\mathbf{A})}\mathbf{u}_1||$ durante el aprendizaje.

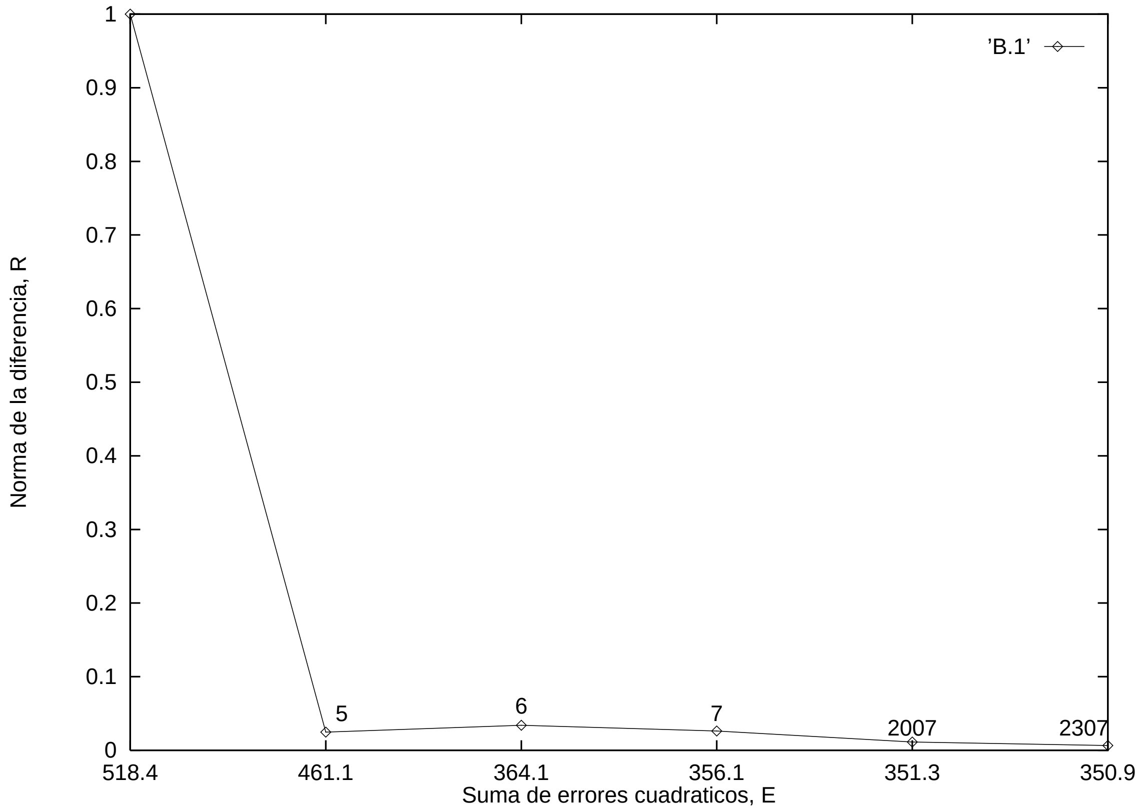

Figura 4.12: Caso 20×32, $h = 1$: evolución de $||\mathbf{u}_1 - \Pi_{L(\mathbf{B})}\mathbf{u}_1||$ durante el aprendizaje.

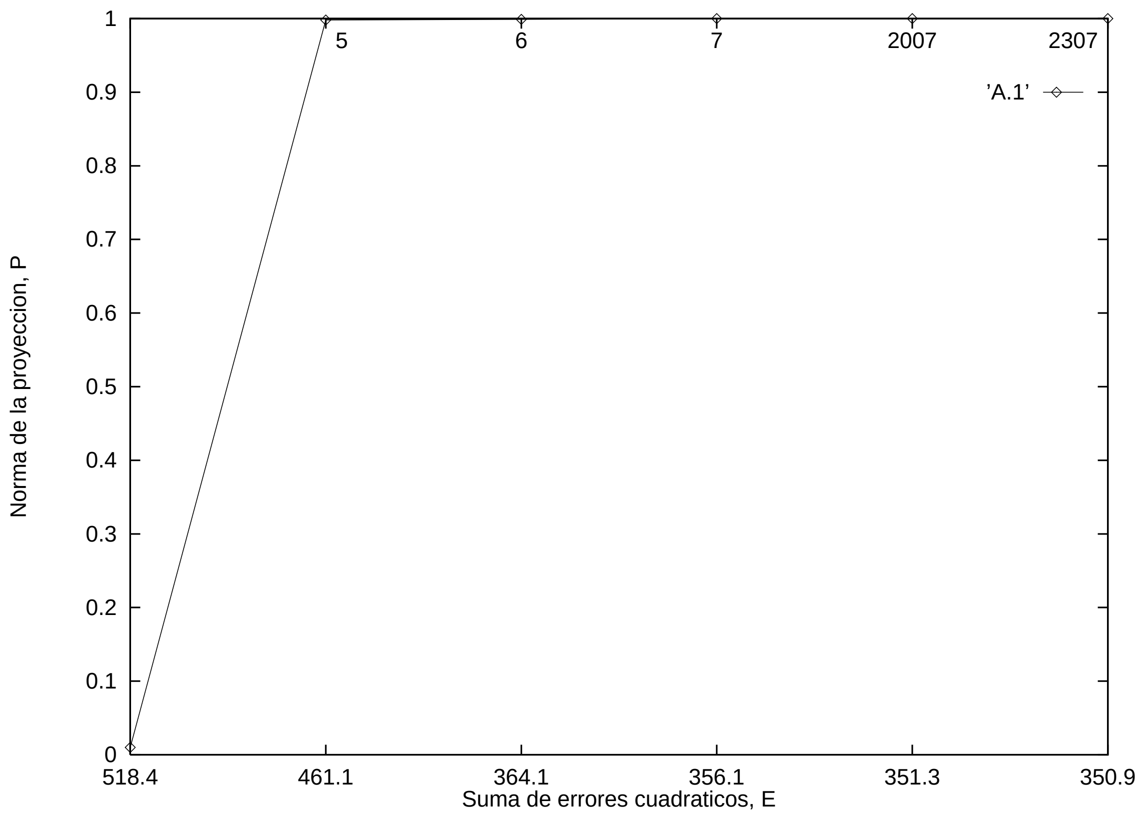

Figura 4.13: Caso 20×32, $h = 1$: evolución de $||\Pi_{L(\mathbf{A})}\mathbf{u}_1||$ durante el aprendizaje.

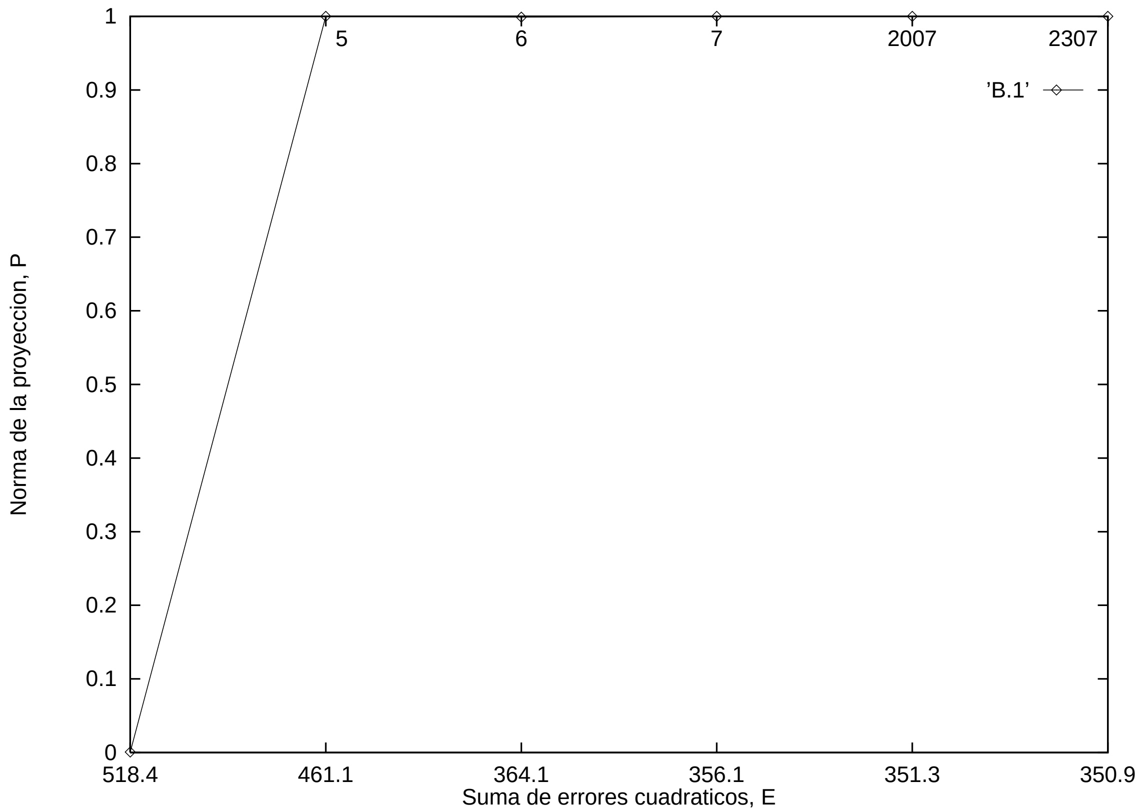

Figura 4.14: Caso 20×32, $h = 1$: evolución de $||\Pi_{L(\mathbf{B})}\mathbf{u}_1||$ durante el aprendizaje.

La figura 4.10 muestra que $\mathbf{B}$ es más parecida a $\mathbf{A}^+$ conforme más se avanza en el aprendizaje.

Las figuras 4.11 y 4.12 dan los valores dinámicos de $R = ||\mathbf{u}_1 - \Pi_{L(\mathbf{A})}\mathbf{u}_1||$ y $||\mathbf{u}_1 - \Pi_{L(\mathbf{B})}\mathbf{u}_1||$, respectivamente, y las 4.13 y 4.14 los de $P = ||\Pi_{L(\mathbf{A})}\mathbf{u}_1||$ y $||\Pi_{L(\mathbf{A})}\mathbf{u}_1||$, respectivamente. En todos los casos, el eje x da el valor de E y sobre el punto se indica el número de ciclos que se llevaban hasta ese valor de E. Obsérvese que la escala del eje x no es lineal, para que el gráfico no quede embarullado, y que va en orden decreciente de E.

En 5 iteraciones el valor alcanzado para R y P se puede considerar suficiente para el uso práctico.

La figura 4.15 muestra la variación de la norma de los vectores columna $\mathbf{A}$ y fila $\mathbf{B}$. Este gráfico lleva en el eje x la norma media y en el y el ángulo medio entre vectores de una matriz. Para $h = 1$ el eje y no es significativo, pues el ángulo es el mismo (0 grados). Las barras de error horizontales y verticales (en la figura 4.15 sólo horizontales) tienen por longitud total el doble de la desviación típica de la distribución de normas y ángulos, para cada eje respectivo. Los puntos correspondientes a $\mathbf{A}$ van unidos por líneas de un trazo y los de $\mathbf{B}$ por líneas de un trazo distinto. Se indican los puntos iniciales (correspondientes a las matrices aleatorias iniciales $\mathbf{A}$ y $\mathbf{B}$) y finales, cuando se detuvo el entrenamiento.

Nótese que, en lugar de poner los puntos sobre la ordenada $y = 0°$, los ponemos sobre $y = 90°$; esto es para dar uniformidad a los diagramas análogos de los demás casos con $h > 1$, en los que los ángulos están muy próximos a $90°$.

Para concluir, damos los *holones* obtenidos por la red. La figura 4.16 da la evolución de los mismos en el aprendizaje por retropropagación por matriz (columnas) y valor de E o del número de ciclos (filas; el número de ciclos crece hacia abajo). La 4.17 da los holones finales para el aprendizaje por *quickprop*. Si el lector se fija, verá que el autovector final obtenido por las redes es igual al representado en la figura 4.7 pero con la tonalidad cambiada: esto es debido al signo (cada vector es igual al otro multiplicado por -1, pero ambos siguen siendo autovectores unitarios).

Caso $h = 5$

Aquí se aplican los mismos comentarios del caso anterior, más los siguientes:

$h = 1$, a 1207597 para el 30×48, $h = 20$), pasarla a *Mathematica*, ejecutar los programas del apéndice D.4 y, finalmente, pasarlos a *gnuplot* (con un preproceso manual previo) para generar los gráficos deseados.

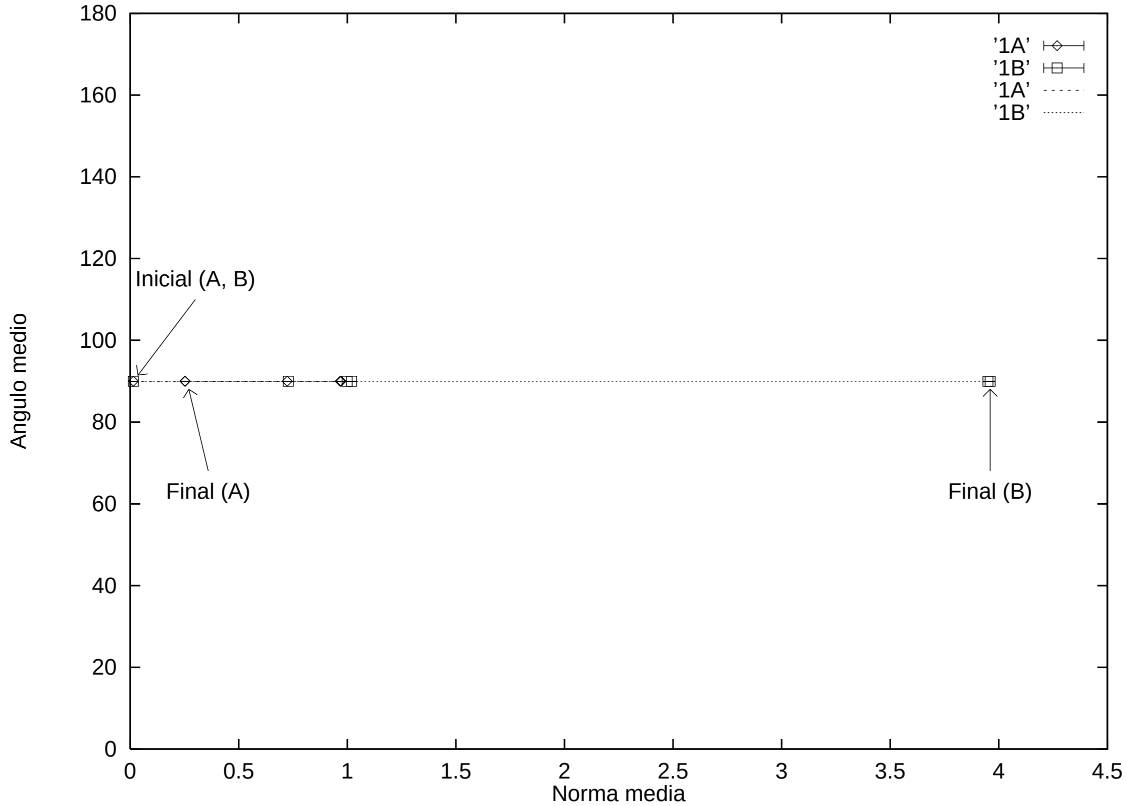

Figura 4.15: Caso 20×32, $h = 1$: evolución de las normas de los vectores $\mathbf{A}$ (columna) y $\mathbf{B}$ (fila) durante el aprendizaje.

- Obsérvese cómo, en las figuras 4.19 y 4.20, los autovectores principales $\mathbf{u}_1, \ldots, \mathbf{u}_5$ van siendo aproximados por la base de la red (sea $\mathbf{A}$ o $\mathbf{B}$) de manera más o menos *ordenada*: el autovector principal $\mathbf{u}_1$ es el primero en quedar bien representado, después el segundo, etc. Sólo tras 3000 iteraciones quedan bien representados los 5. El valor de E es ya prácticamente mínimo ño desde las primeras iteraciones, porque los primeros 3–4 autovectores pesan mucho y quedan bien representados enseguida. Lo mismo ocurre puede deducirse de las figuras 4.21 y 4.22.

- Las bases $\mathbf{A} = (\mathbf{a}_1, \ldots, \mathbf{a}_5)$ y $\mathbf{B} = (\mathbf{b}_1, \ldots, \mathbf{b}_5)$ pueden considerarse ortogonales desde el principio (en la primera decena de iteraciones los ángulos están muy repartidos, pero enseguida se concentran alrededor de 90°). Pero sus normas no tienden todas a 1, aunque sí cumplen (tras la convergencia) $||\mathbf{a}_i||||\mathbf{b}_i|| \approx 1$, ya que en ese momento $\mathbf{B} \approx \mathbf{A}^+$. Por ello, si las normas de $\mathbf{A}$ se quedan cerca de 0.3, las de $\mathbf{B}$ lo hacen cerca de $1/0.3 \approx 3.3$ (a pesar de que las normas medias de ambas se "pasean" durante el aprendizaje alrededor de 1); ver fig, 4.23.

- Las figuras 4.24 y 4.25 muestran que los holones también parecen orejas aun cuando $h > 1$ y la base ya no es de autovectores (ningún vector $\mathbf{a}_i$ o $\mathbf{b}_i$ coincide con alguno de los autovectores $\mathbf{u}_i$, excepto quizá en alguna ocasión).

- Ahora aparece una figura nueva, la 4.26. En ella se da la distribución de p-varianzas direcciones a lo largo de cada uno de los vectores de las bases $\mathbf{A}$ y $\mathbf{B}$. Obsérvese que esta distribución es más o menos uniforme; las varianzas no siguen la distribución decreciente de la figura 4.1, guiada por los autovalores de $\mathbf{X}\mathbf{X}^T$. Esto demuestra que en las redes Ξ, cada unidad oculta tiene aproximadamente la misma importancia (en términos de varianza, pero también de norma de su vector asociado), como no podía ser de otra manera por la simetría de la red: ninguna unidad tiene nada que la haga distinta de las demás.

Caso $h = 10$

Este caso no añade ya nada nuevo a lo anteriormente dicho. Al final del capítulo se incluyen, no obstante, las mismas figuras que para $h = 1$ y $h = 5$.

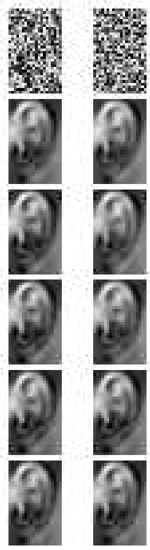

Figura 4.16: Caso 20×32, $h = 1$: evolución de los *holones* durante el aprendizaje por retropropagación. La columna izquierda corresponde a $\mathbf{B}$ y la derecha a $\mathbf{A}$. La primera fila a las matrices iniciales y la última a las finales.

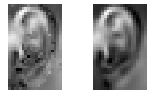

Figura 4.17: Caso 20×32, $h = 1$: *holones* al final del aprendizaje. La columna izquierda corresponde a $\mathbf{B}$ y la derecha a $\mathbf{A}$.

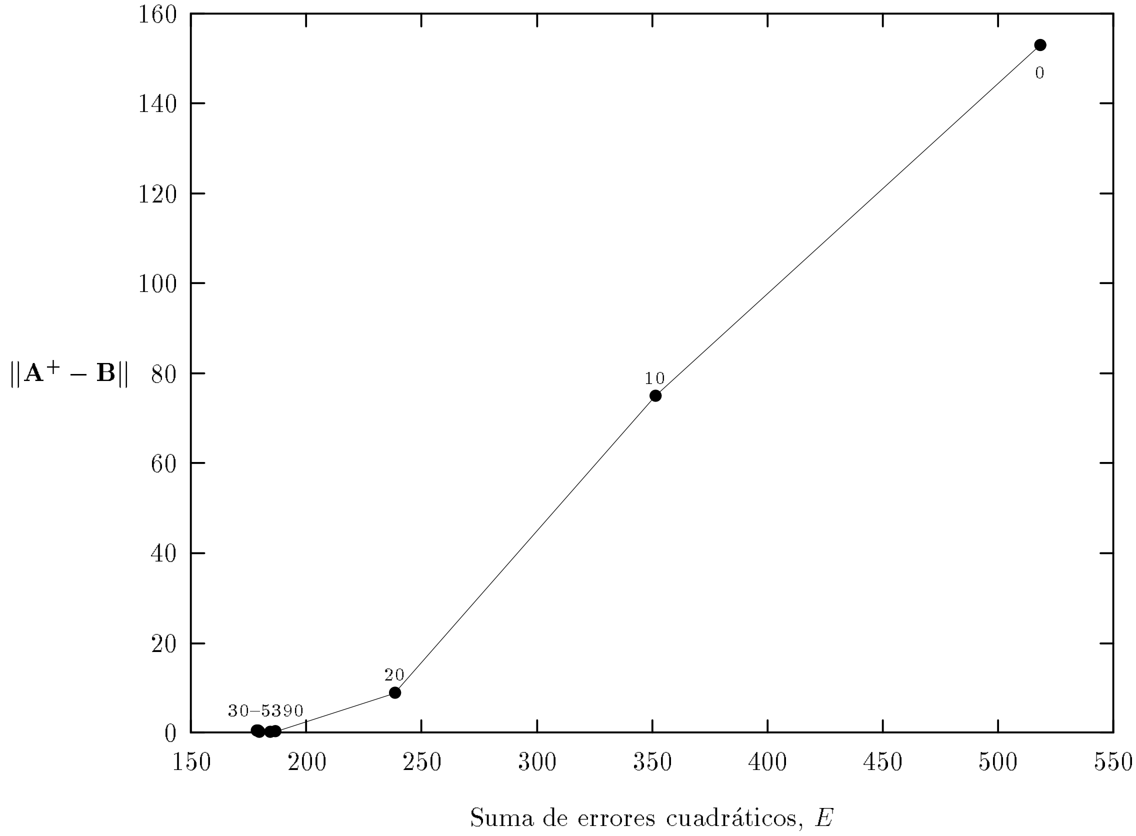

Figura 4.18: Caso 20×32, $h = 5$: evolución de $||\mathbf{B} - \mathbf{A}^+||$.

Como resumen de la distribución de las normas y los ángulos entre vectores para cada una de las bases **A** y **B** de cada valor de h y algoritmo de aprendizaje, se muestra el gráfico de la figura 4.27. En él las barras de error son asimétricas y sus brazos actúan como intervalo de variación, es decir, el valor mínimo está en el extremo izquierdo (o inferior) y el máximo en el derecho (o superior), con el promedio en el centro. Vemos que las bases salen prácticamente ortogonales, pero no ortonormales en todos los casos (aunque no se alejen demasiado del valor 1).

4.2.4 Retraso de B respecto de A en la convergencia

Si se observan con cuidado las figuras 4.17, 4.25 y 4.47 (y también las figuras del caso 30×48), se verá que el bloque izquierdo, correspondiente a la matriz **B**, sale "peor" que el derecho, correspondiente a **A**. De hecho, si se toman las medidas $||\mathbf{u}_i - \Pi_{L(\mathbf{B})}\mathbf{u}_i||$ y $||\Pi_{L(\mathbf{B})}\mathbf{u}_i||$, se verá que no aproximan bien el h-autoespacio. Las figuras mencionas corresponden todas al algoritmo *quickprop*, pero este fenómeno se da también para la retropropagación normal.

Es decir, la segunda capa de pesos de la red Ξ, la que corresponde a la matriz **A**, converge enseguida, mientras que la primera, matriz **B**, tarda mucho más. Sin embargo, cuando **A** se acerca a su valor límite, el error E está también muy cerca del suyo, a pesar de que **B** aún no se ha acercado a su propio límite (que, evidentemente, es $\mathbf{B} = \mathbf{A}^+$).

La causa de este *retraso* se debe a la precisión limitada de los cálculos (el simulador utilizado, *SNNS*, trabaja en precisión simple; es decir, se cuenta con unos 7 dígitos significativos): cuando el error E es muy pequeño, los valores δ_i de la ecuación (2.5) que son retropropagados hacia las capas previas se ven multiplicados por los valores de los pesos w_{ij}, que también son muy pequeños, y por el factor de aprendizaje, usualmente muy pequeño también. Entonces, las correcciones Δw_{ij} pueden llegar a ser despreciables comparadas con los pesos w_{ij} (debido a la truncación de decimales) y no afectar a los mismos. Este fenómeno aparece muchas veces cuando se está cerca del mínimo de E.

Por otro lado, el hecho de que los valores δ_i se hagan más y más pequeños en cada capa que se retropropaga hace que los niveles más próximos a la entrada evolucionen más despacio (sus correcciones son más pequeñas) que los que están más cerca de la salida.

Las figuras 4.16, 4.24 y 4.46 no muestran este retraso porque en ellas se emplearon muchas más

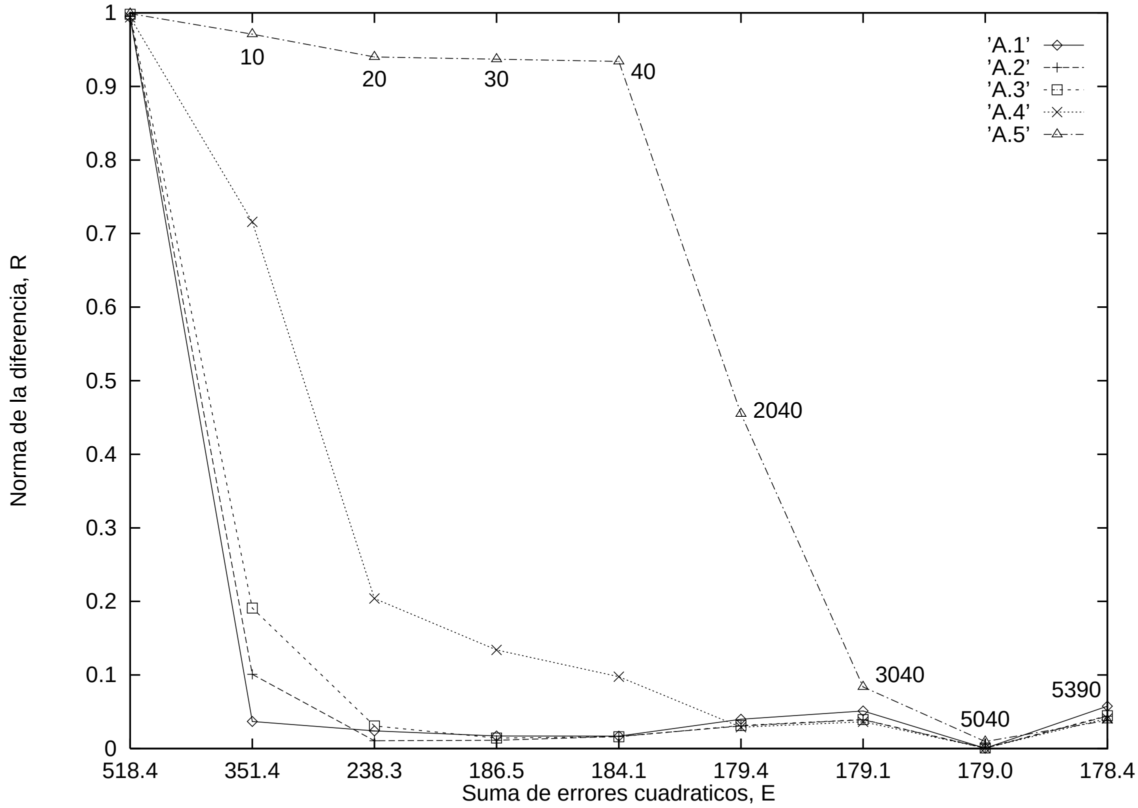

Figura 4.19: Caso 20×32, $h = 5$: evolución de $\|\mathbf{u}_i - \Pi_{L(\mathbf{A})}\mathbf{u}_i\|$, $i = 1, \ldots, 5$ durante el aprendizaje.

iteraciones que en las de *quickprop*. Pero fácilmente se comprueba que también en la retropropagación simple, si se detiene el entrenamiento pronto (pero cuando ya se tiene un E pequeño), $\mathbf{B}$ sale peor que $\mathbf{A}$.

Observemos que, si se emplea el algoritmo acelerado de la sección 3.3.2, el problema anterior desaparece: la segunda capa de pesos converge enseguida y la primera lo hace a la vez, pues una es la traspuesta de la otra.

4.3 Caso 30×48

4.3.1 Análisis espectral de $\mathbf{XX}^T$

Las figuras 4.28, 4.29, 4.30 y 4.31 muestran las distribuciones de autovalores y de errores cuadráticos y las representaciones pictóricas (*holones*) de los autovectores y de la media y del autovector dominante. Los mismos comentarios de la sección 4.2 se aplican aquí. Evidentemente la imagen contiene más información, a costa del mayor tiempo de proceso y de espacio de almacenamiento que conlleva aumentar la resolución.

Los números de condición de las matrices consideradas son $c(\mathbf{\Sigma}) = c(\mathbf{XX}^T) = c(\mathbf{X}^T\mathbf{X}) \approx 5454$, $c(\mathbf{X}) \approx 74$, ya que el intervalo de variación de los autovalores de $\mathbf{XX}^T$ es $[\lambda_n, \lambda_1] \approx [0.07, 379]$. Estos números son incluso menores que los obtenidos en el caso 20×32. Todas las matrices están bien condicionadas, pues.

4.3.2 Resultados

La figura 4.32 demuestra la superioridad de *quickprop* frente a la retropropagación simple. También queda claro que conforme el tamaño de la red crece, son necesarias más iteraciones para llevar a E cerca de su mínimo.

Como antes, sólo se muestra la primera parte de la figura. El entrenamiento se prolongó bastante más, para obtener 4 decimales de precisión en E. Esto no es necesario en un caso práctico, por supuesto. 100 ciclos con *quickprop* son más que suficientes, según dicha figura, para cualquiera de los 3 casos.

En todos los casos los parámetros de red usados fueron los siguientes: $\eta = 0.001$ para el factor de aprendizaje y $[-0.001, 0.001]$ para la distribución uniforme inicial de los pesos.

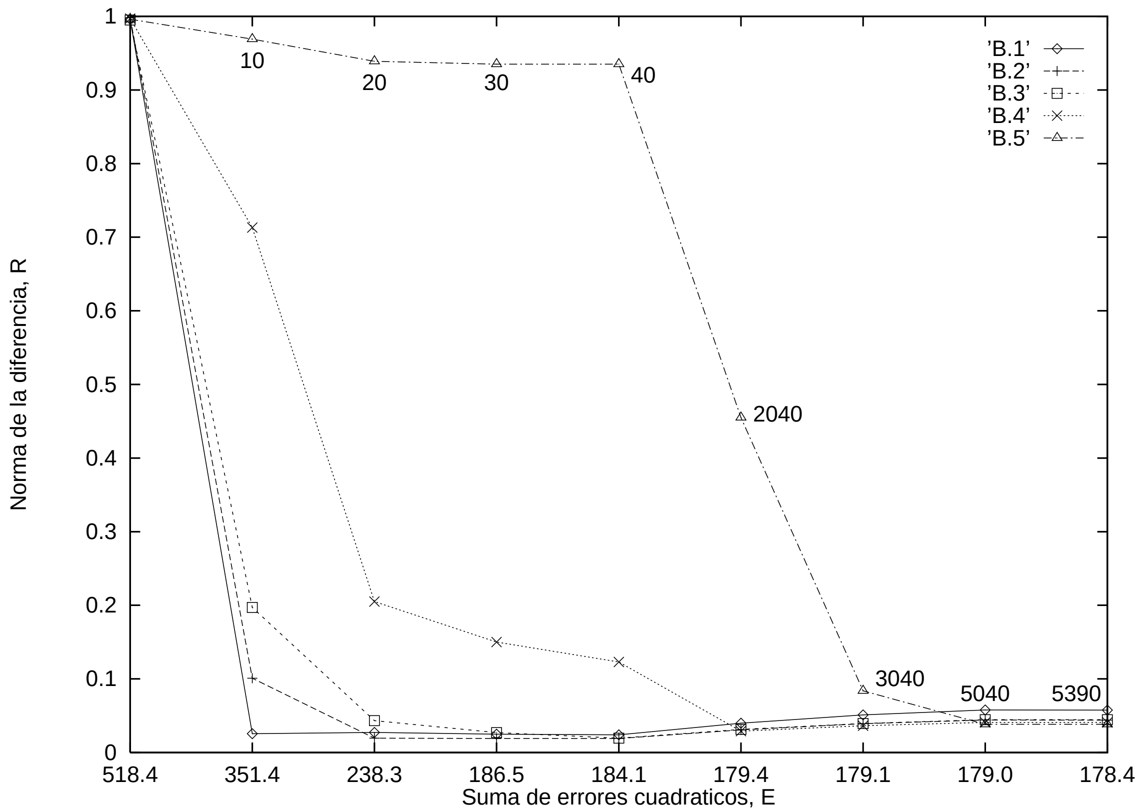

Figura 4.20: Caso 20×32, $h = 5$: evolución de $||\mathbf{u}_i - \Pi_{L(\mathbf{B})}\mathbf{u}_i||$, $i = 1, \ldots, 5$ durante el aprendizaje.

En las figuras 4.33 y 4.34 vuelve a pasar lo que en las análogas del caso 20×32 (4.16, 4.16): el autovector principal calculado por la red Ξ lleva el signo cambiado respecto al obtenido por *Mathematica* (fig. 4.31).

Igualmente llama la atención el contraste, en las figuras correspondientes a los holones obtenidos por *quickprop*, entre la matriz $\mathbf{A}$ y la $\mathbf{B}$: la primera ha convergido, la segunda no. Las razones se dieron en la sección 4.2.4. De nuevo parece que la "culpa" es de *quickprop*, pero no es así: el autor consiguió exactamente los mismos defectos en $\mathbf{B}$ deteniendo la retropropagación prematuramente (aunque ya muy cerca del valor mínimo de E).

Las p-varianzas tienden a repartirse uniformemente, como en casos anteriores, salvo por algún pico que otro. Con retropropagación, la p-varianza en el caso $h = 1$ fue de 378.773 para $\mathbf{A}$ y de 377.67 para $\mathbf{B}$; el autovalor principal de $\mathbf{X}\mathbf{X}^T$ vale 378.77263759. Con *quickprop* la p-varianza fue de 378.773 para $\mathbf{A}$ y de 364.263 para $\mathbf{B}$, bastante más lejos del valor exacto, lo que prueba que $\mathbf{B}$ no ha convergido del todo. Los valores de $R = ||\mathbf{u}_i - \Pi_{L(\mathbf{B})}\mathbf{u}_i||$ y $P = ||\Pi_{L(\mathbf{B})}\mathbf{u}_i||$ dados en la figura 4.37 refuerzan este hecho; para $\mathbf{B}$, dichos valores se encuentran lejos de los ideales $R = 0$ y $P = 1$ (no así para $\mathbf{A}$).

La figura 4.38 muestra una gran uniformidad en la geometría de las bases: son prácticamente ortogonales, y su dispersión en normas es menor que en el caso 20×32.

El resto de las figuras se muestran al final del capítulo.

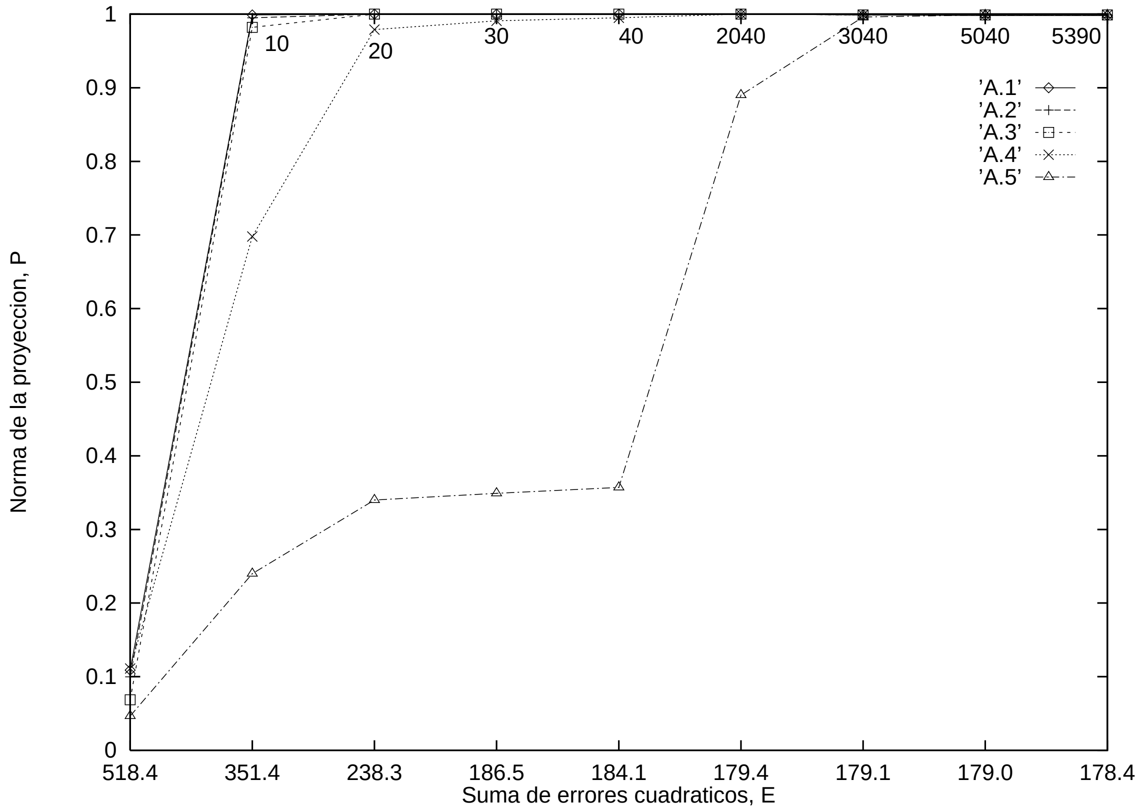

Figura 4.21: Caso 20×32, $h = 5$: evolución de $||\Pi_{L(\mathbf{A})}\mathbf{u}_i||$, $i = 1, \ldots, 5$ durante el aprendizaje.

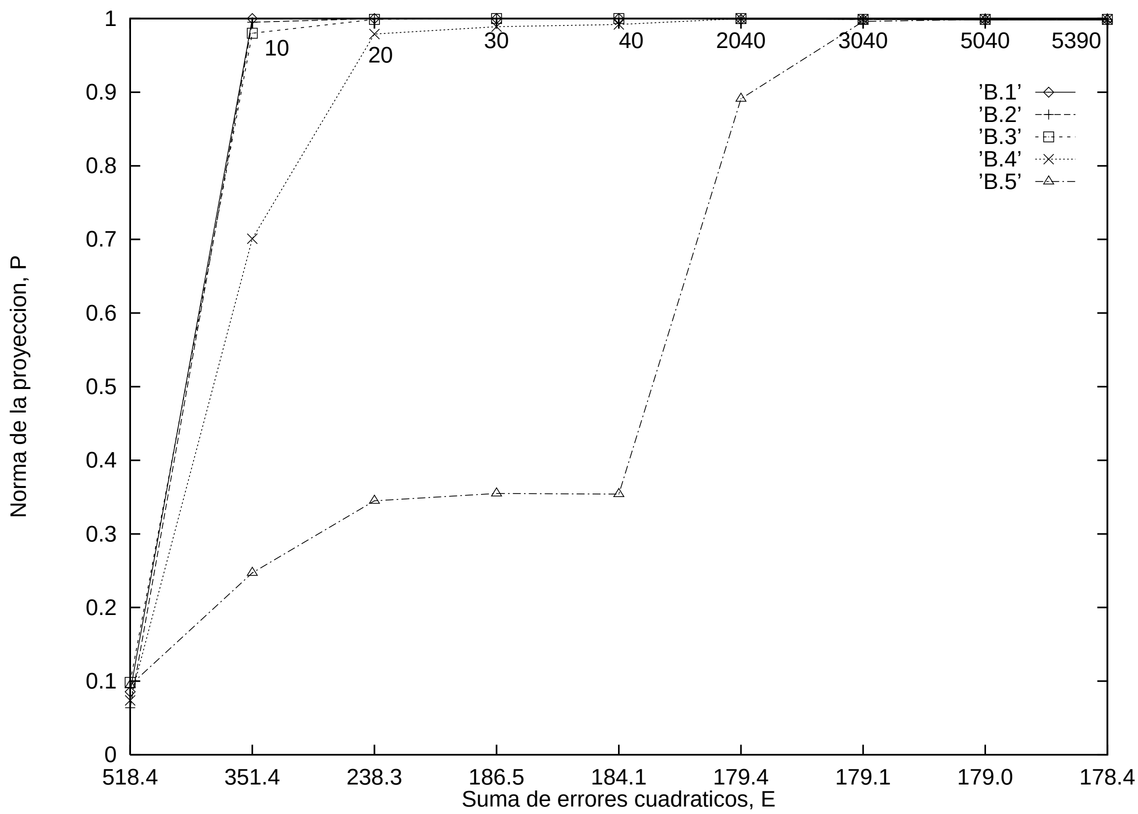

Figura 4.22: Caso 20×32, $h = 5$: evolución de $||\Pi_{L(\mathbf{B})}\mathbf{u}_i||$, $i = 1, \ldots, 5$ durante el aprendizaje.

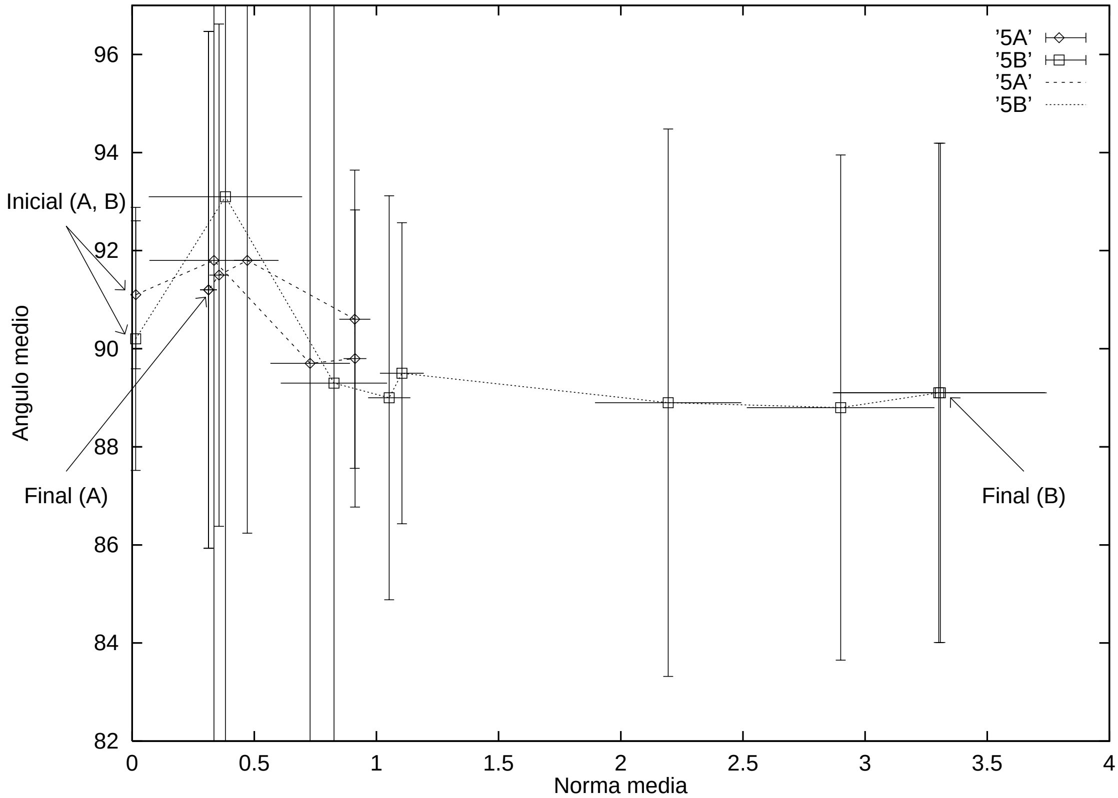

Figura 4.23: Caso 20×32, $h = 5$: evolución de las normas de los vectores $\mathbf{a}_i$, $i = 1, \ldots, 5$ (columna) y $\mathbf{b}_i$, $i = 1, \ldots, 5$ (fila) durante el aprendizaje.

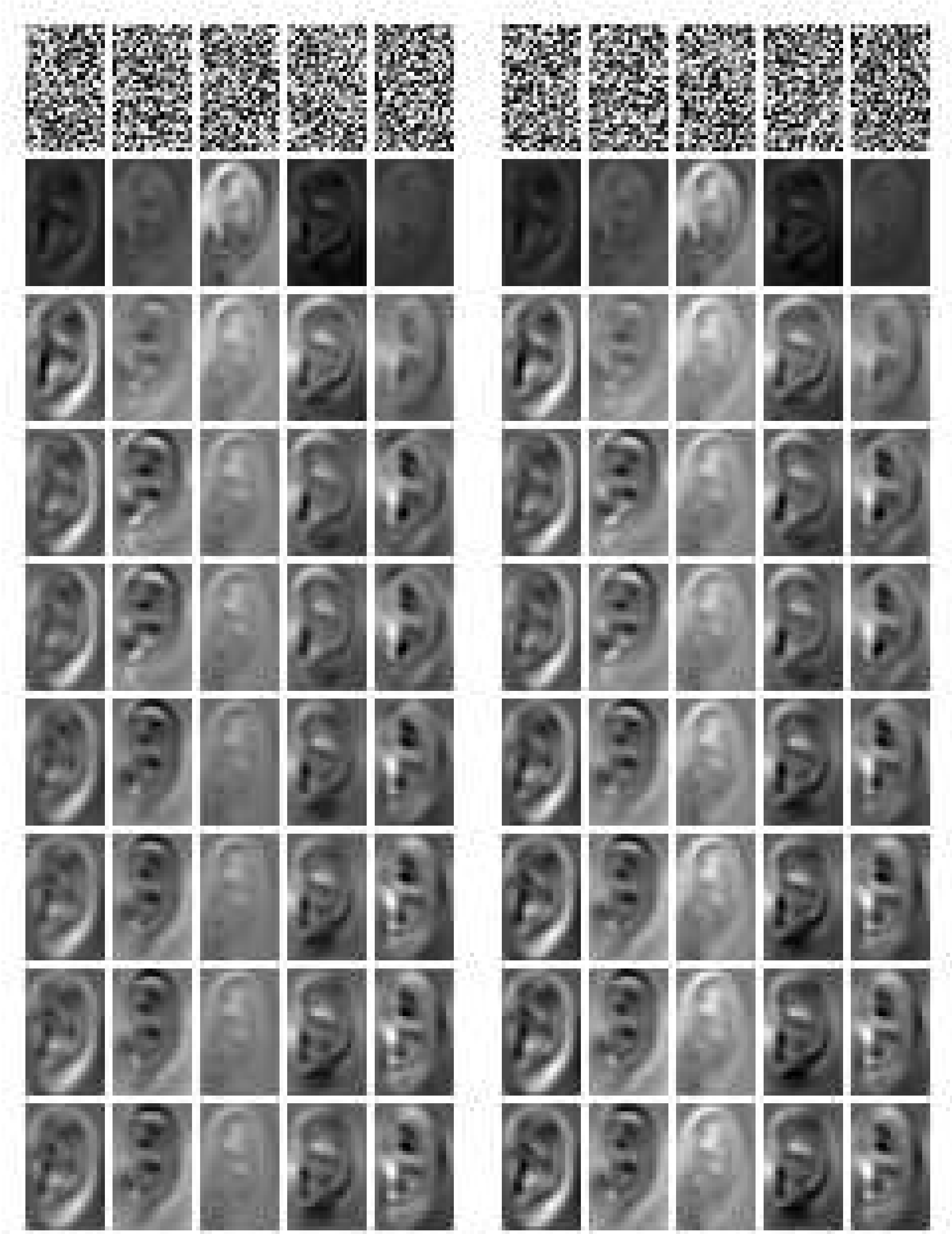

Figura 4.24: Caso 20×32, $h = 5$: evolución de los *holones* durante el aprendizaje por retropropagación. El bloque izquierdo corresponde a $\mathbf{B}$ y el derecho a $\mathbf{A}$. La primera fila a las matrices iniciales y la última a las finales.

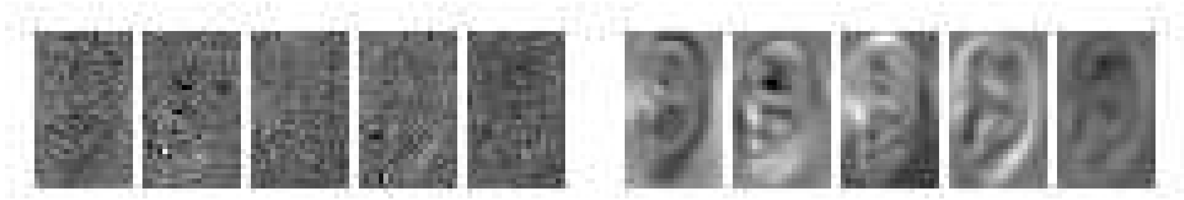

Figura 4.25: Caso 20×32, $h = 5$: *holones* al final del aprendizaje. El bloque izquierdo corresponde a **B** y el derecho a **A**.

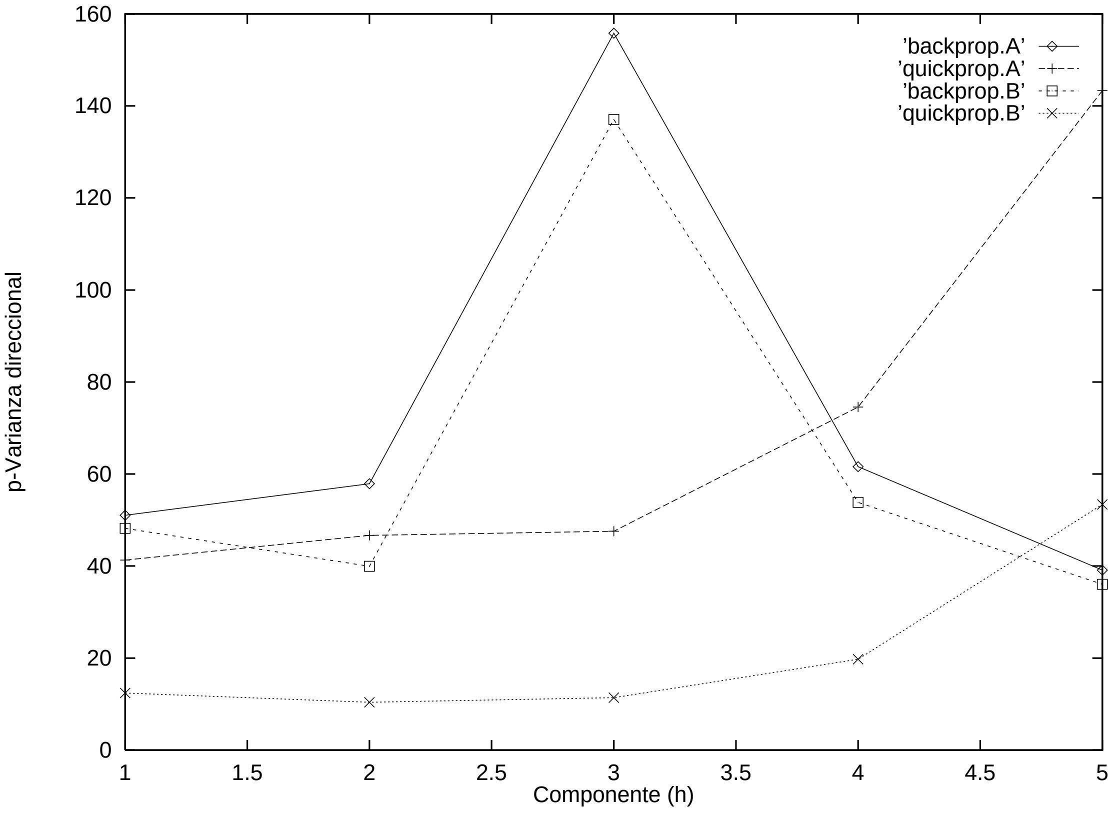

Figura 4.26: Caso 20×32, $h = 5$: distribución de p-varianzas en las bases finales, para las dos matrices **A** y **B** y los dos algoritmos de aprendizaje, retropropagación y *quickprop*.

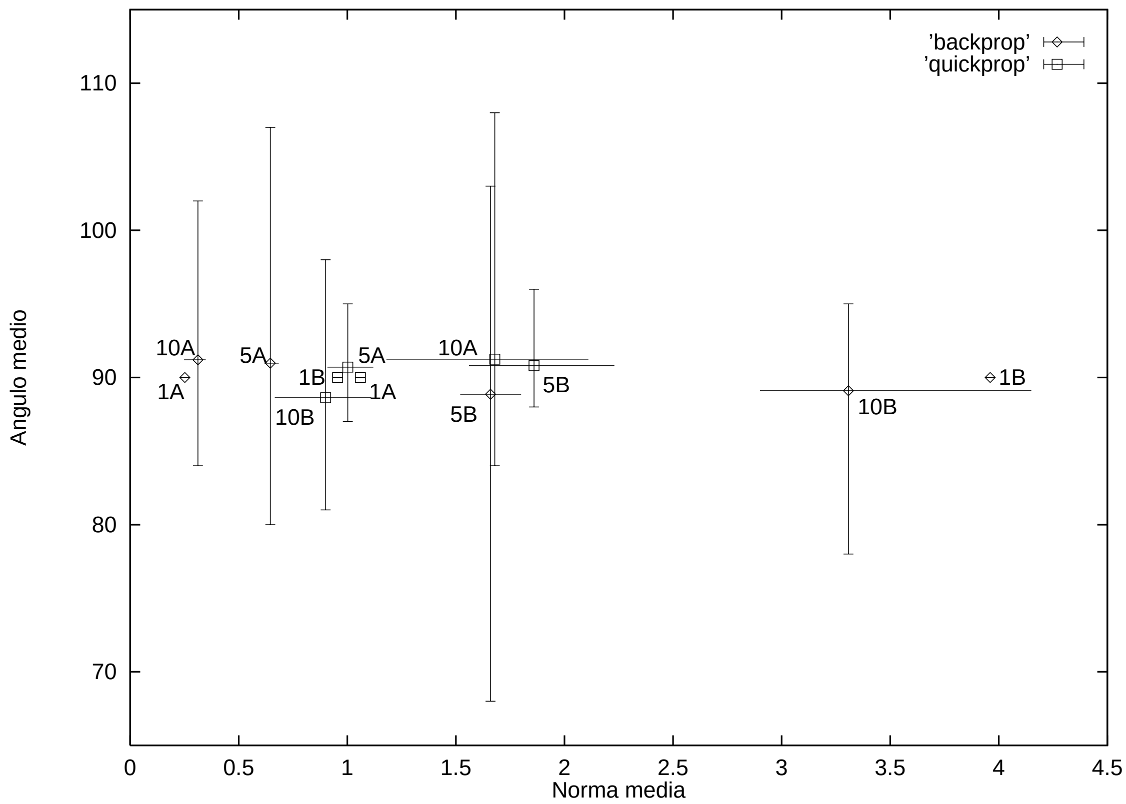

Figura 4.27: Caso 20×32: diagrama de "ortonormalidad" de las bases **A** y **B** en los distintos casos estudiados.

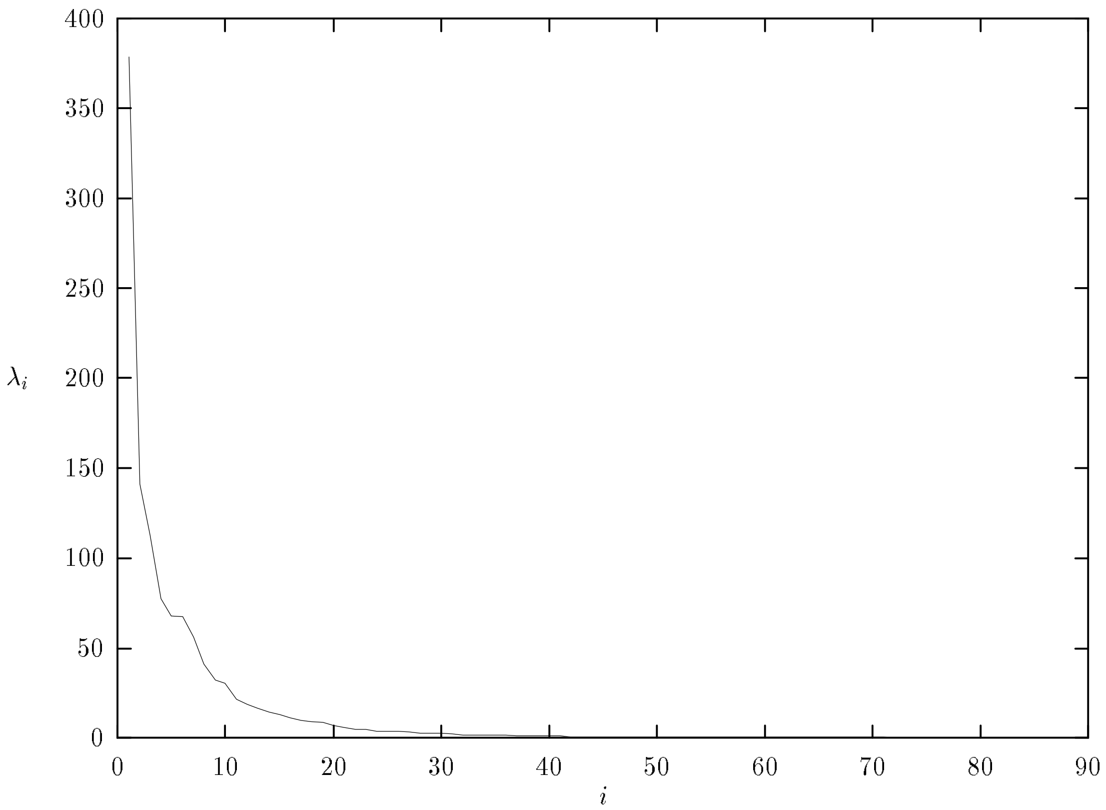

Figura 4.28: Autovalores de $\mathbf{XX}^T$ para el caso 30×48.

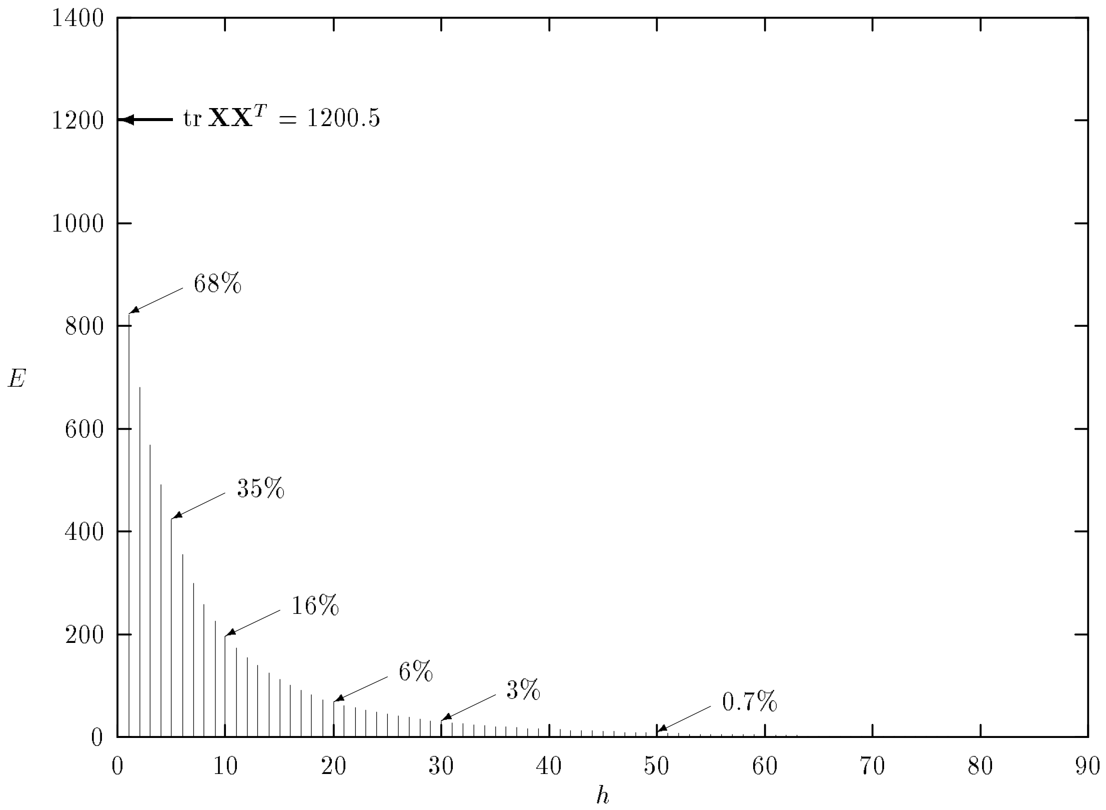

Figura 4.29: Errores cuadráticos E para el caso 30×48.

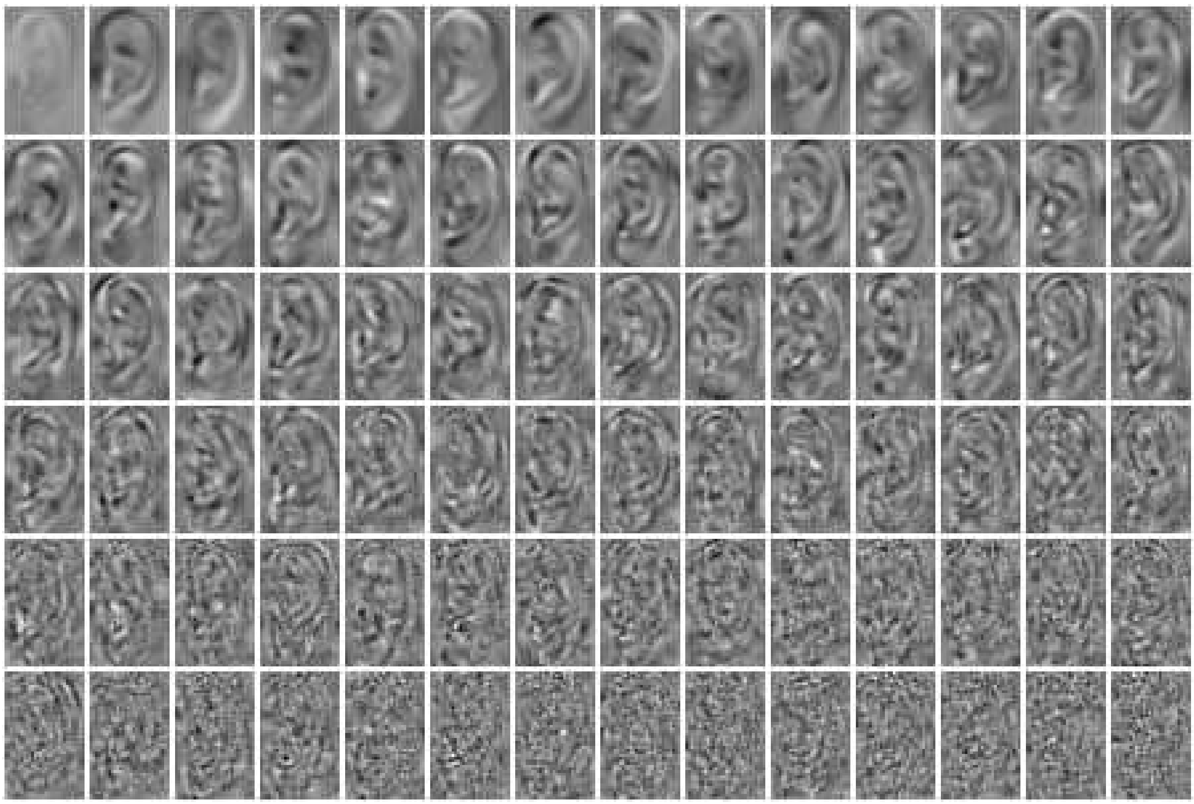

Figura 4.30: *Holones* de los autovectores principales $\mathbf{u}_i$ para el caso 30×48.

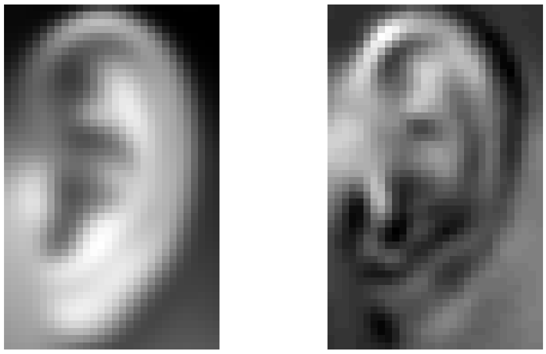

Figura 4.31: Media $\overline{\mathbf{y}}$ para el caso 30×48 (a la izquierda) y autovector principal $\mathbf{u}_1$ (a la derecha, normalizado a 256 tonos de gris).

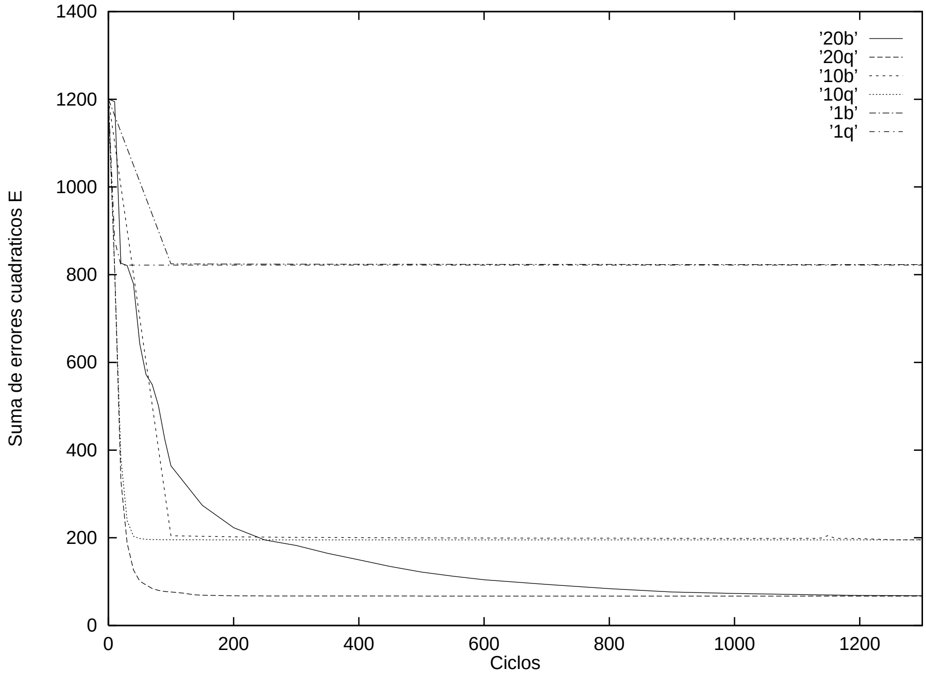

Figura 4.32: Caso 30×48: curvas de aprendizaje para $h = 1$, 10 y 20, con los algoritmos de retropropagación y *quickprop*.

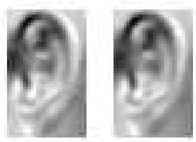

Figura 4.33: Caso 30×48, $h = 1$: *holones* al final del aprendizaje con retropropagación. El bloque izquierdo corresponde a $\mathbf{B}$ y el derecho a $\mathbf{A}$.

Figura 4.34: Caso 30×48, $h = 1$: *holones* al final del aprendizaje con *quickprop*. El bloque izquierdo corresponde a $\mathbf{B}$ y el derecho a $\mathbf{A}$.

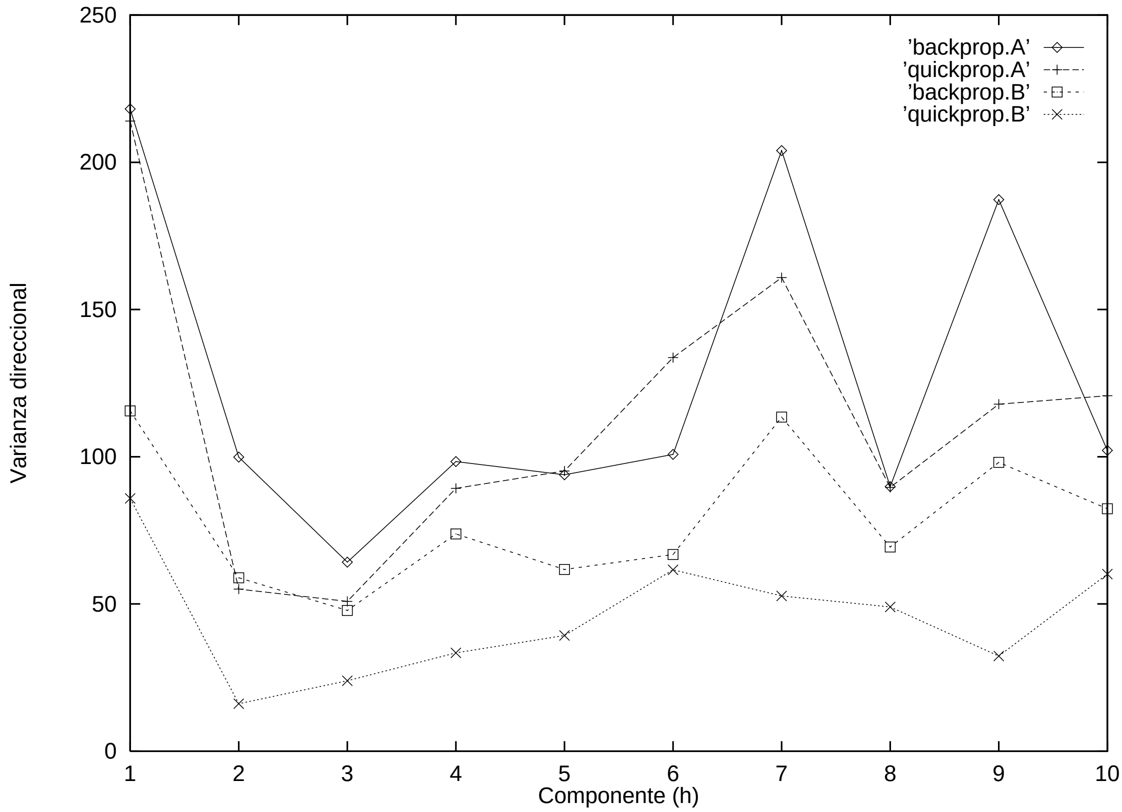

Figura 4.35: Caso 30×48, $h = 10$: distribución de p-varianzas en las bases finales, para las dos matrices **A** y **B** y los dos algoritmos de aprendizaje, retropropagación y *quickprop*.

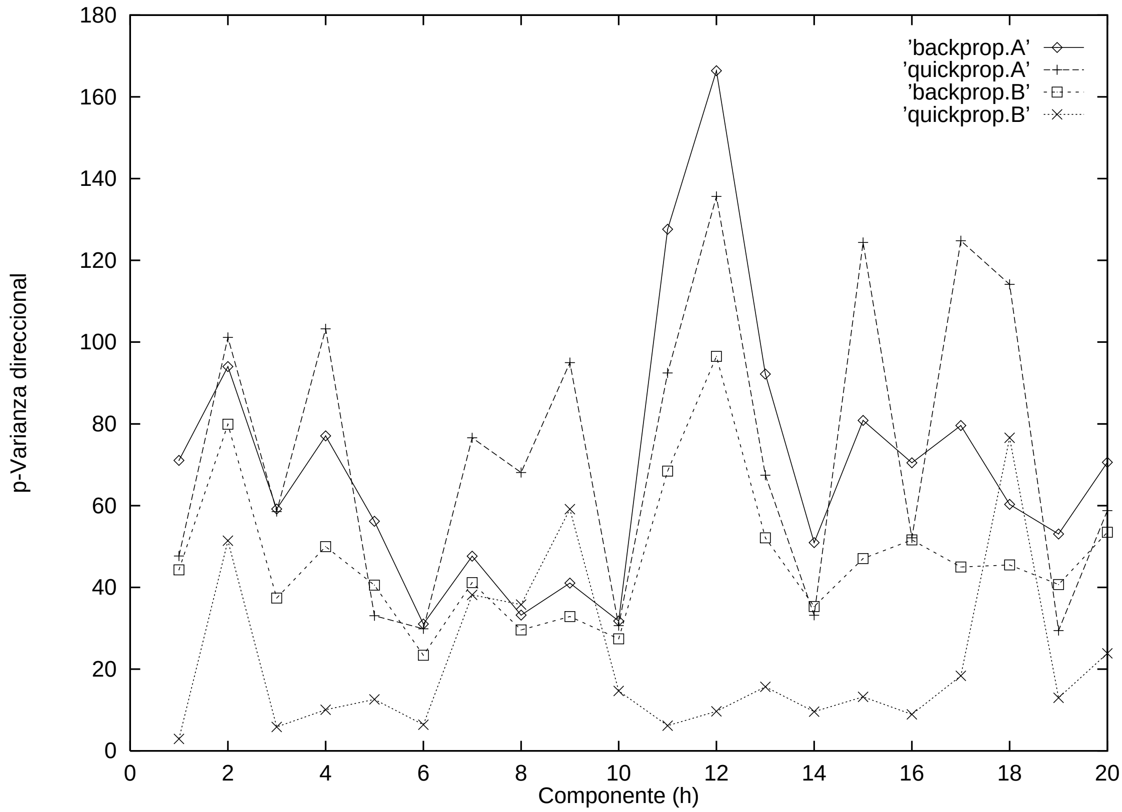

Figura 4.36: Caso 30×48, $h = 20$: distribución de p-varianzas en las bases finales, para las dos matrices **A** y **B** y los dos algoritmos de aprendizaje, retropropagación y *quickprop*.

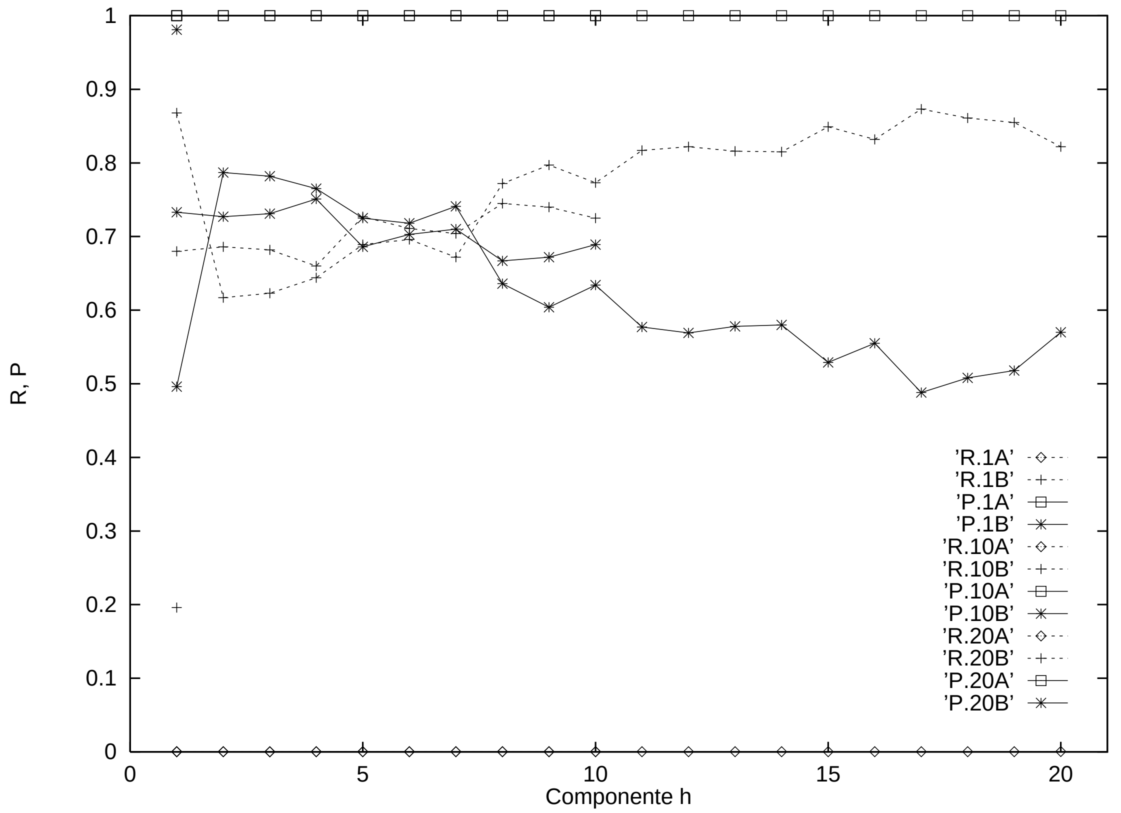

Figura 4.37: Caso 30×48: valores de $R = \|\mathbf{u}_i - \Pi_{L(\mathbf{B})}\mathbf{u}_i\|$ y $P = \|\Pi_{L(\mathbf{B})}\mathbf{u}_i\|$ para las redes entrenadas usando pocas iteraciones (con *quickprop*).

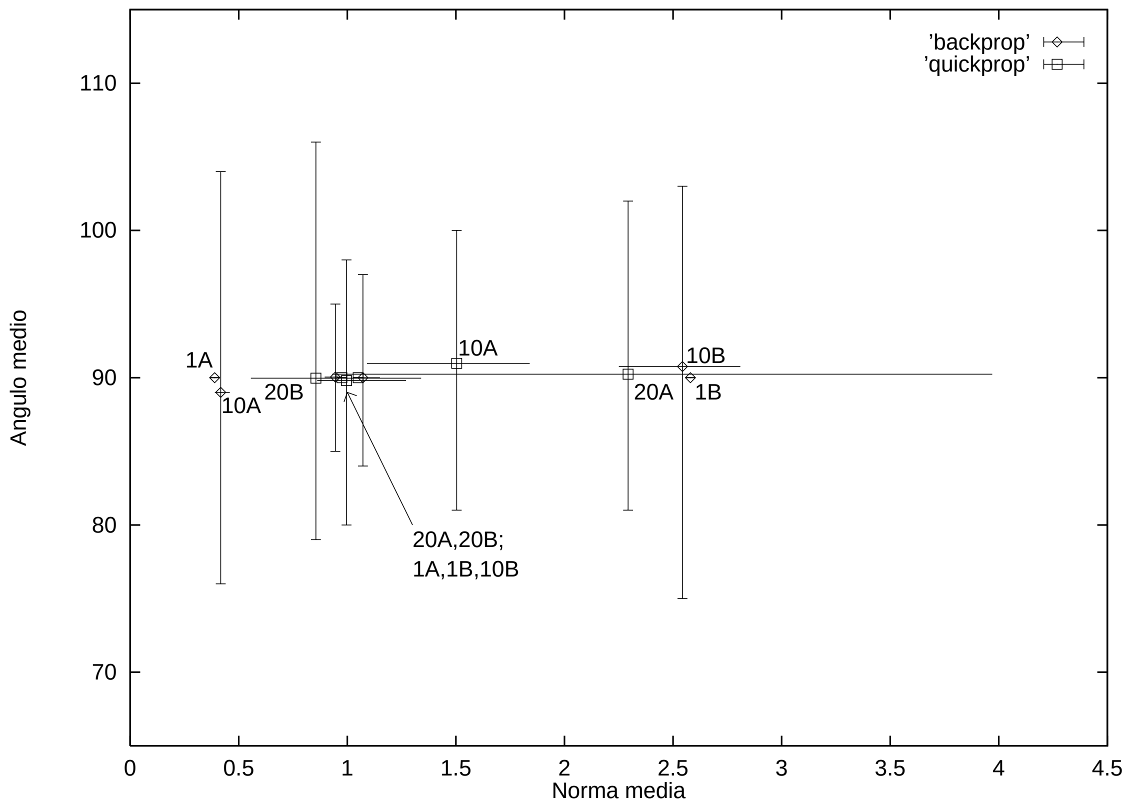

Figura 4.38: Caso 30×48: diagrama de "ortonormalidad" de las bases **A** y **B** en los distintos casos estudiados.

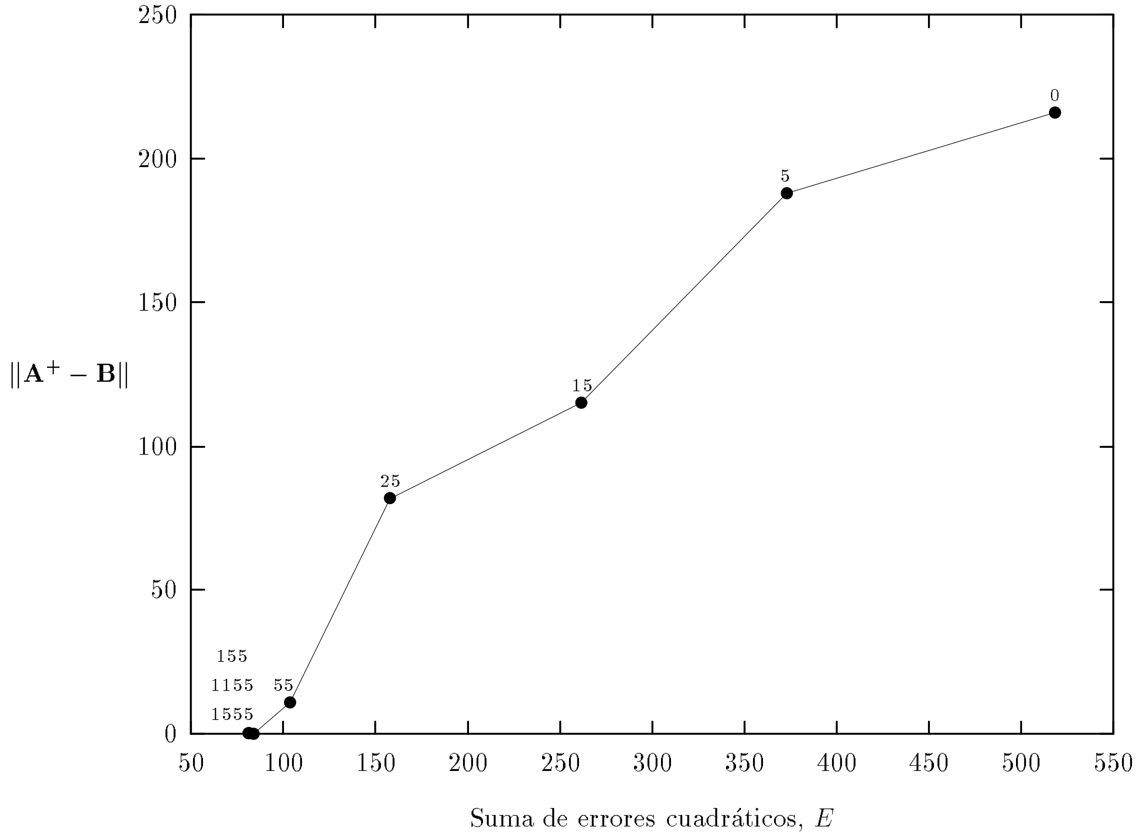

Figura 4.39: Caso 20×32, $h = 10$: evolución de $||\mathbf{B} - \mathbf{A}^+||$.

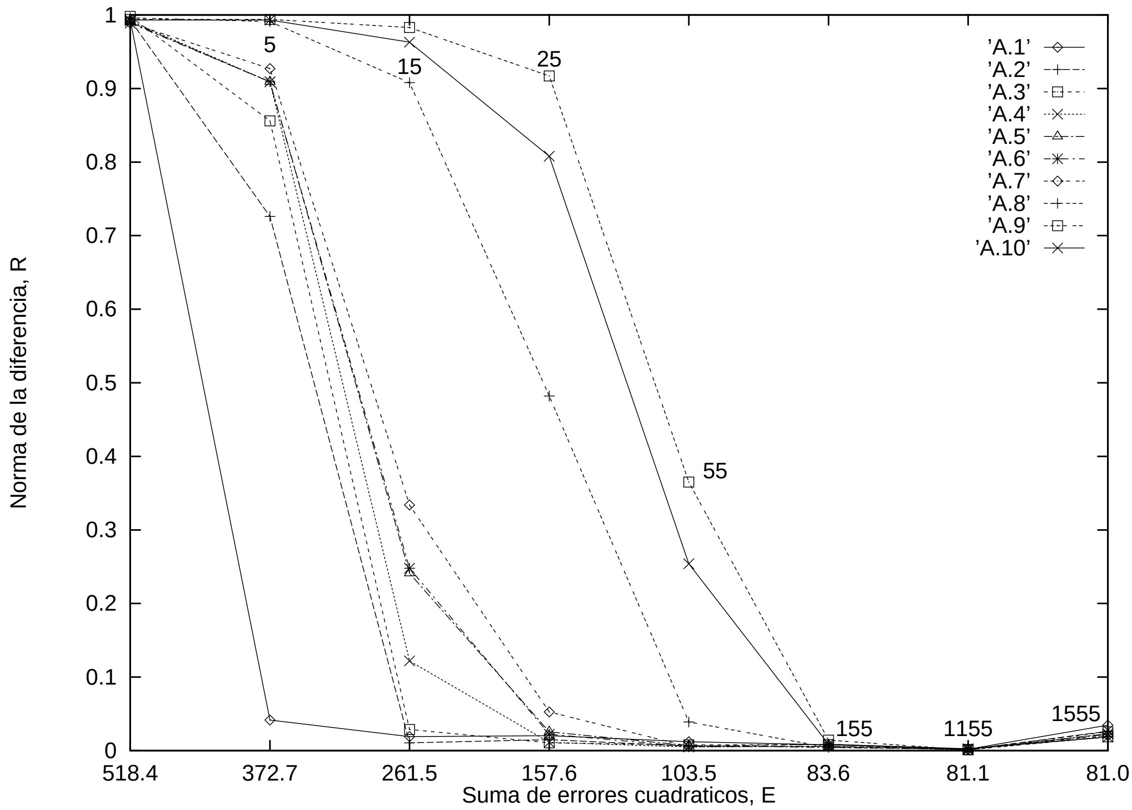

Figura 4.40: Caso 20×32, $h = 10$: evolución de $||\mathbf{u}_i - \Pi_{L(\mathbf{A})}\mathbf{u}_i||$, $i = 1, \ldots, 10$ durante el aprendizaje.

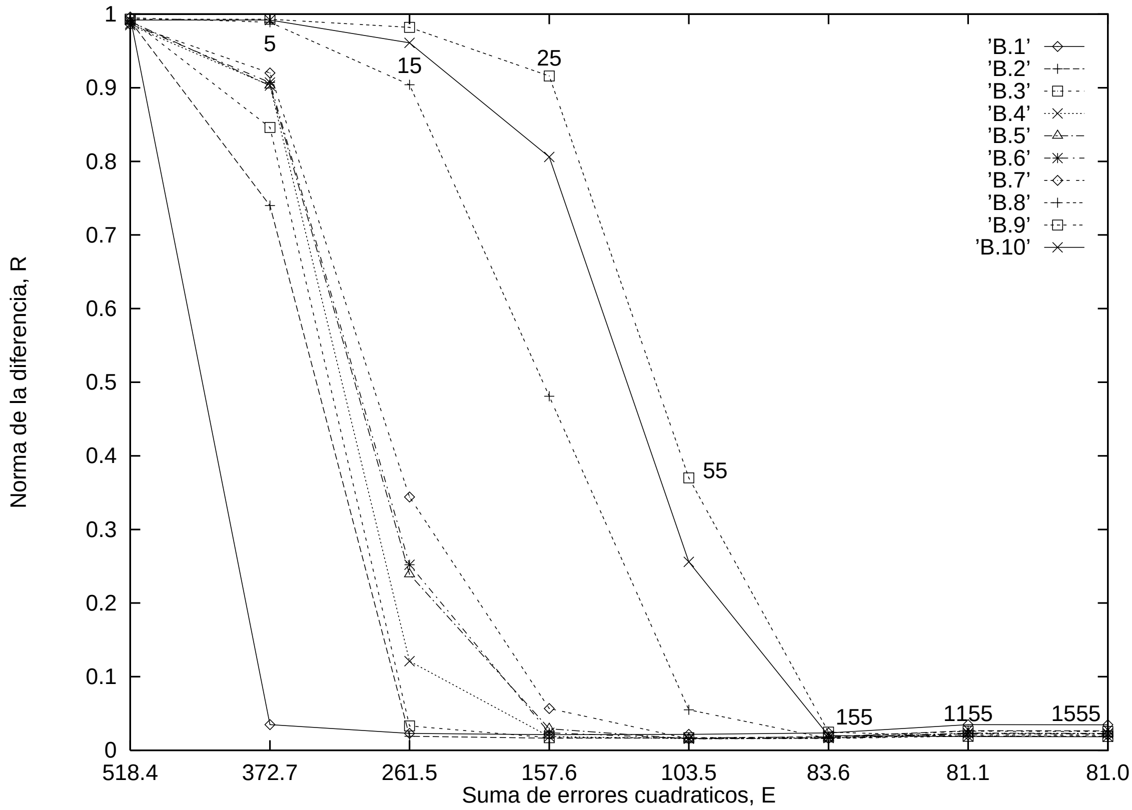

Figura 4.41: Caso 20×32, $h = 10$: evolución de $\|\mathbf{u}_i - \Pi_{L(\mathbf{B})}\mathbf{u}_i\|$, $i = 1, \ldots, 10$ durante el aprendizaje.

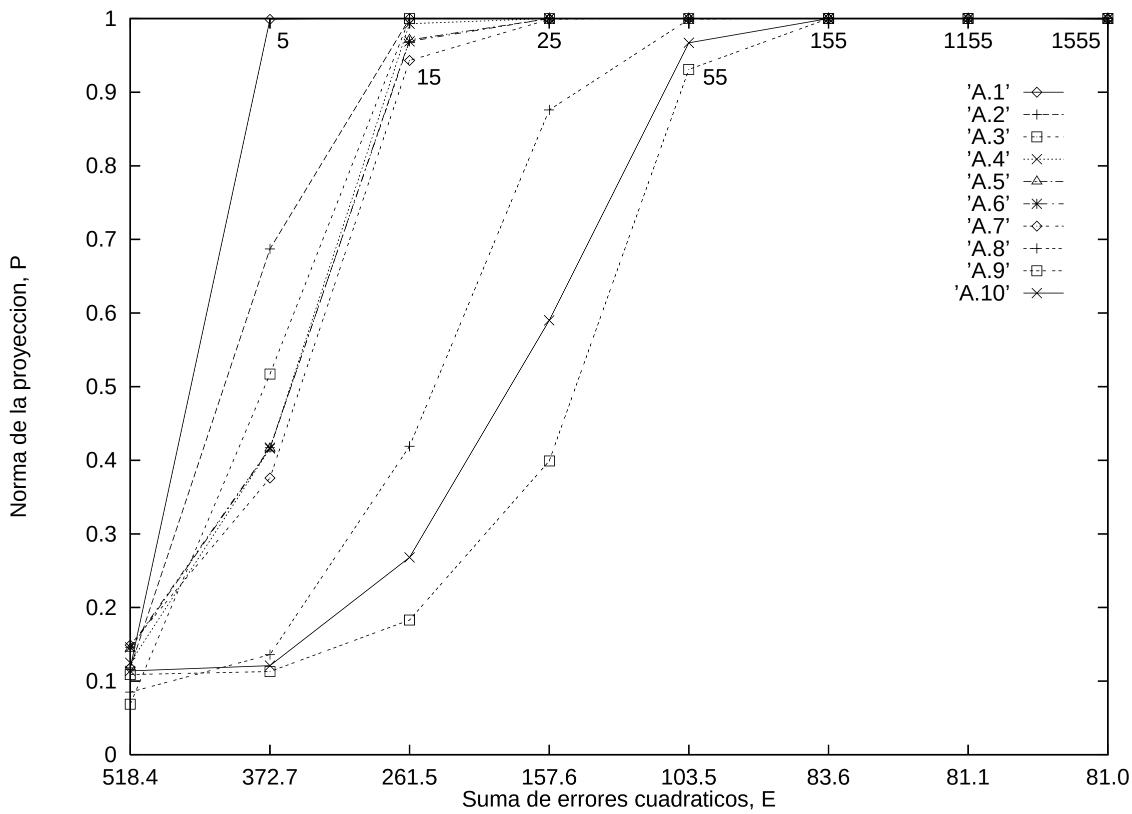

Figura 4.42: Caso 20×32, $h = 10$: evolución de $\|\Pi_{L(\mathbf{A})}\mathbf{u}_i\|$, $i = 1, \ldots, 10$ durante el aprendizaje.

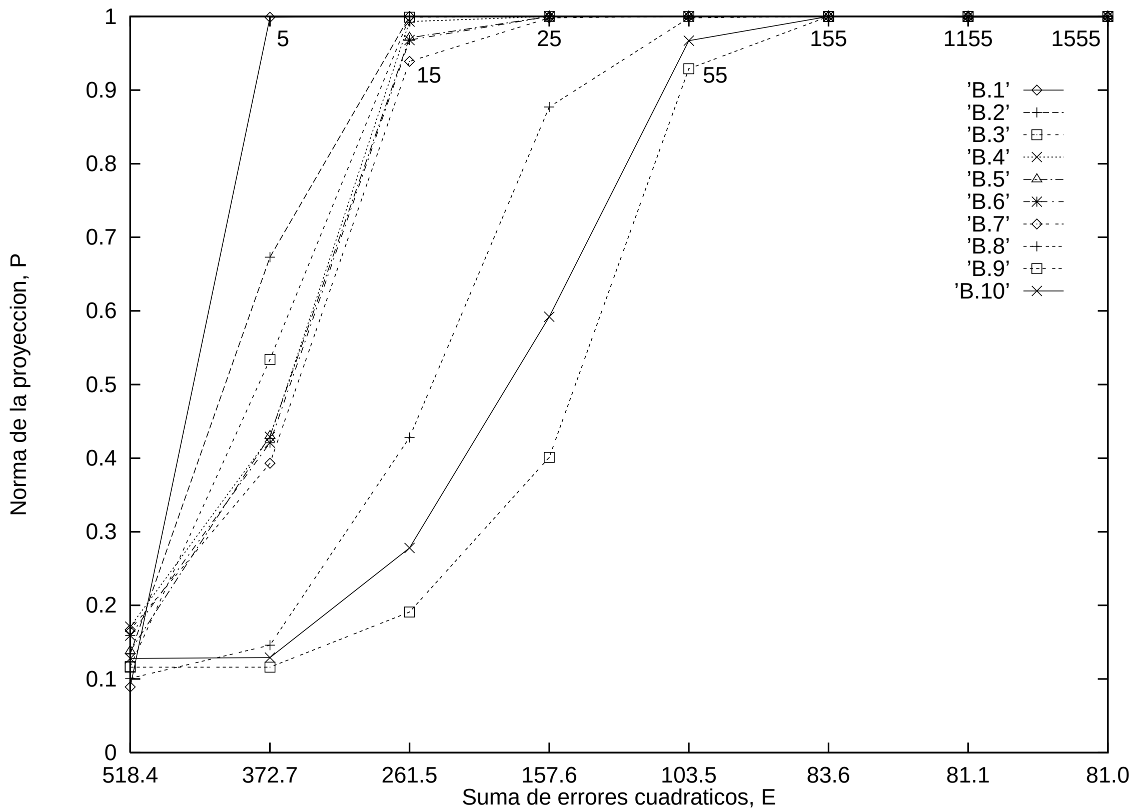

Figura 4.43: Caso 20×32, $h = 10$: evolución de $\|\Pi_{L(\mathbf{B})}\mathbf{u}_i\|$, $i = 1, \ldots, 10$ durante el aprendizaje.

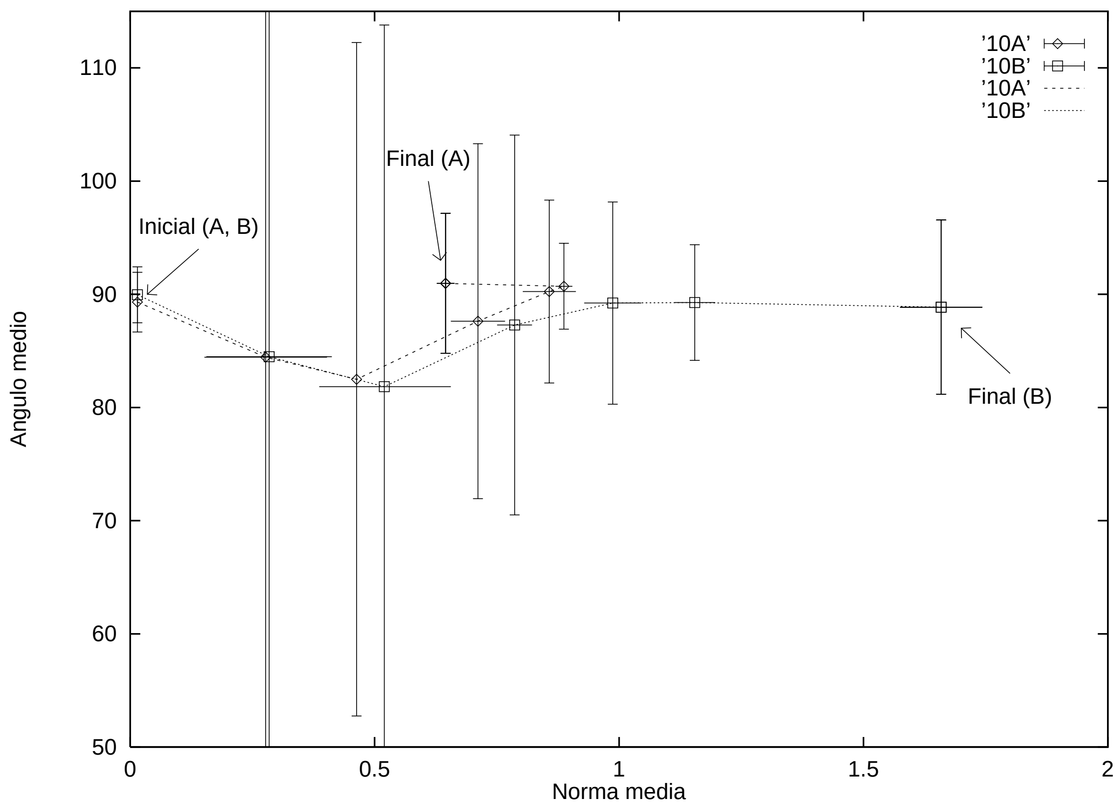

Figura 4.44: Caso 20×32, $h = 10$: evolución de las normas de los vectores $\mathbf{a}_i$, $i = 1, \ldots, 10$ (columna) y $\mathbf{b}_i$, $i = 1, \ldots, 10$ (fila) durante el aprendizaje.

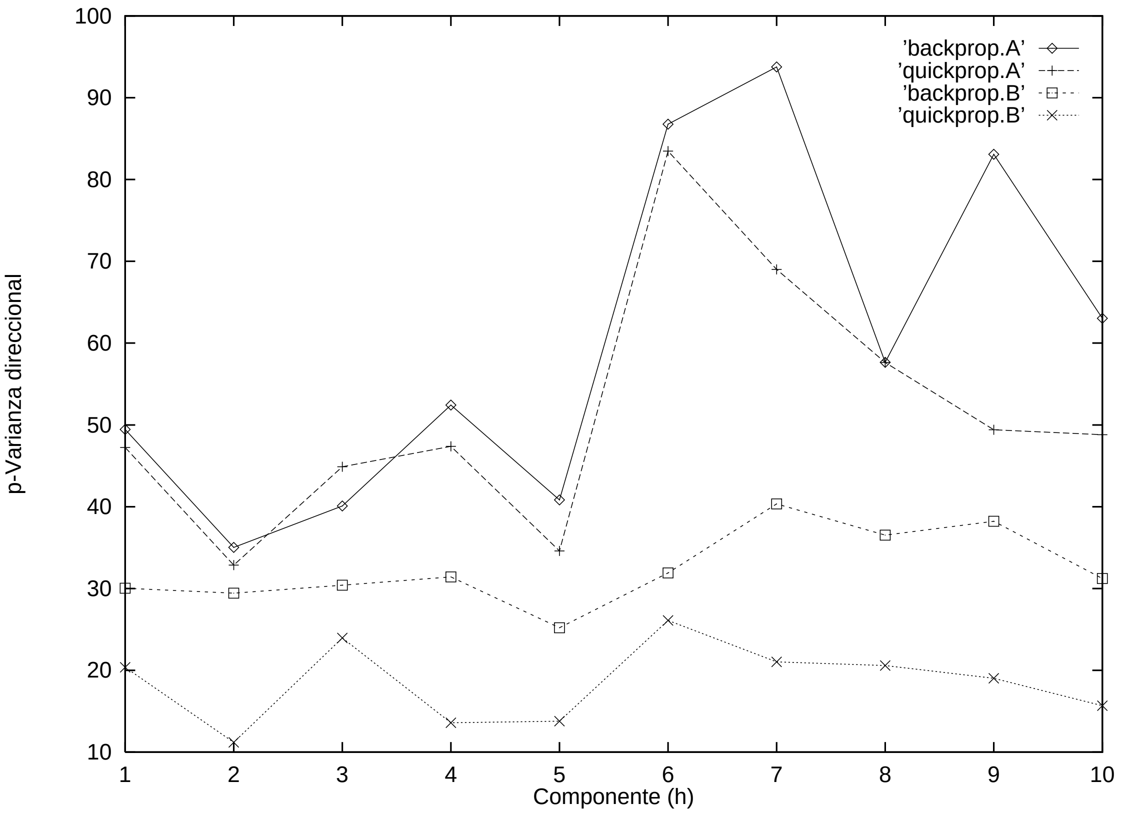

Figura 4.45: Caso 20×32, $h = 10$: distribución de p-varianzas en las bases finales, para las dos matrices **A** y **B** y los dos algoritmos de aprendizaje, retropropagación y *quickprop*.

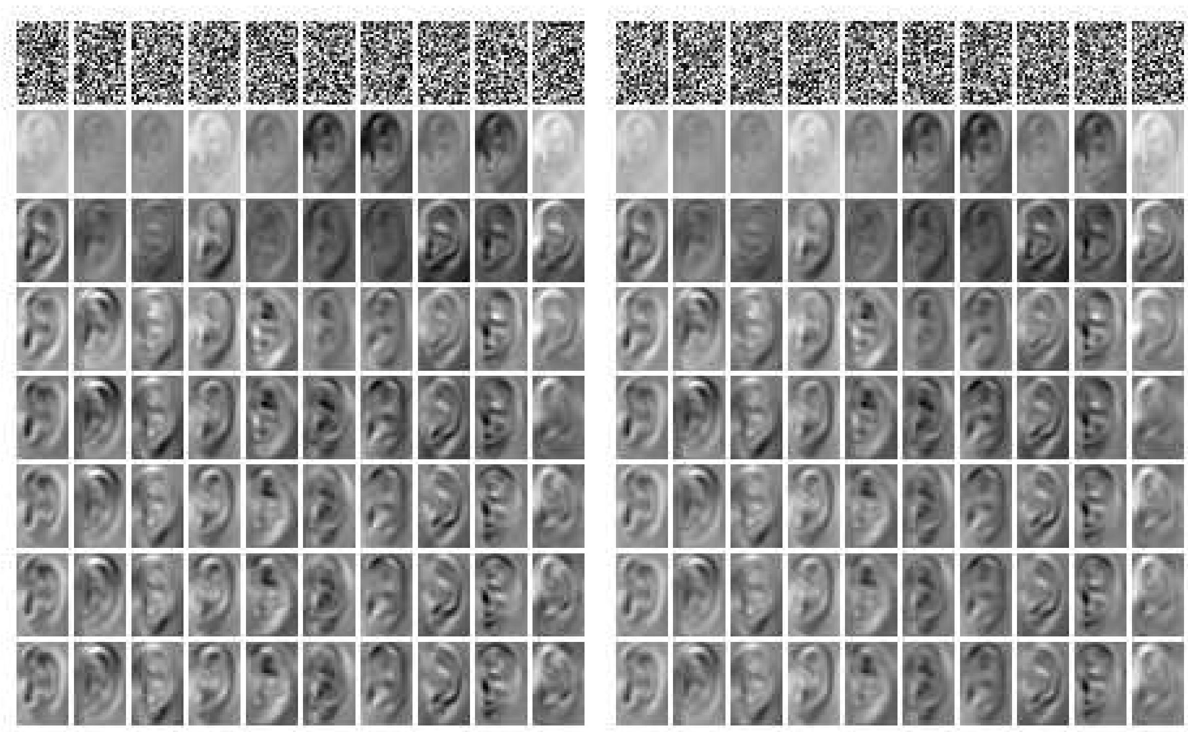

Figura 4.46: Caso 20×32, $h = 10$: evolución de los *holones* durante el aprendizaje con retropropagación. El bloque izquierdo corresponde a **B** y el derecho a **A**. La primera fila a las matrices iniciales y la última a las finales.

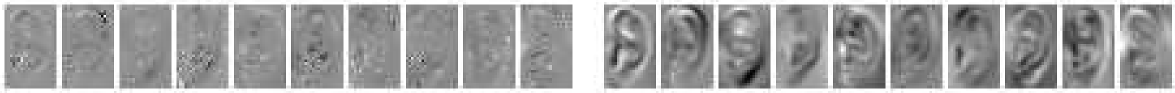

Figura 4.47: Caso 20×32, $h = 10$: *holones* al final del aprendizaje con *quickprop*. El bloque izquierdo corresponde a **B** y el derecho a **A**.

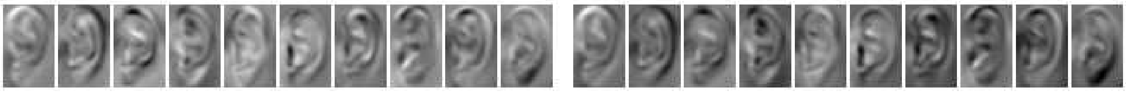

Figura 4.48: Caso 30×48, $h = 10$: *holones* al final del aprendizaje con retropropagación. El bloque izquierdo corresponde a **B** y el derecho a **A**.

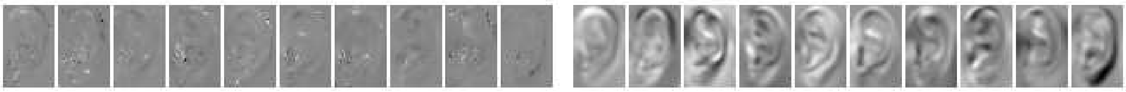

Figura 4.49: Caso 30×48, $h = 10$: *holones* al final del aprendizaje con *quickprop*. El bloque izquierdo corresponde a **B** y el derecho a **A**.

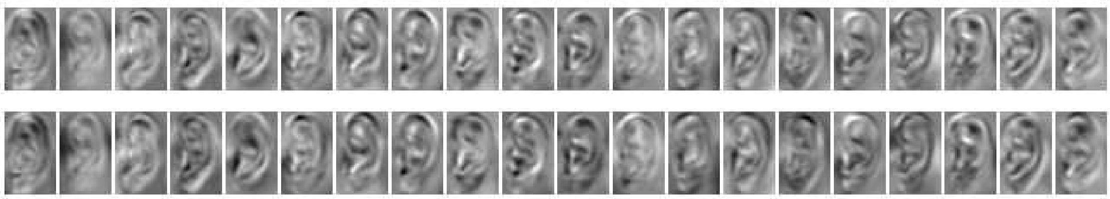

Figura 4.50: Caso 30×48, $h = 20$: *holones* al final del aprendizaje con retropropagación. La fila de arriba corresponde a **B** y la de abajo a **A**.

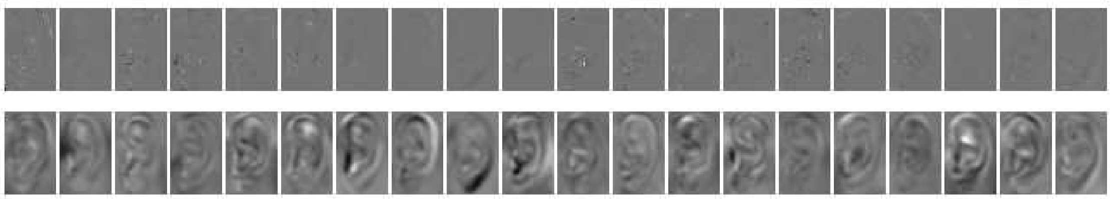

Figura 4.51: Caso 30×48, $h = 20$: *holones* al final del aprendizaje con *quickprop*. La fila de arriba corresponde a **B** y la de abajo a **A**.

Capítulo 5

Experimentos, parte II: Aplicación al reconocimiento

5.1 La regla de rechazo

La arquitectura de redes de compresión estudiada permite, a través del error E generado por la red ante un nuevo patrón, decidir si éste es o no de la clase de patrones con la que la red fue entrenada. En nuestro caso, dado que la red fue entrenada con imágenes de orejas, el nuevo patrón dará un error pequeño si es o se parece a una oreja y si, por el contrario, no se parece, es de esperar que el error sea grande.

Este hecho nos permite fijar un valor umbral E_0 para el error E, tal que si el error ante el nuevo patrón[1] $\mathbf{y}$, $E_{\mathbf{y}}$, es mayor que ese umbral, el objeto representado por él no será una oreja; si es menor, lo consideraremos una oreja. Este procedimiento es muy heurístico y su validez debe ser comprobada en la práctica, sometiéndolo a varios patrones de distintos tipos.

Podemos expresar la regla anterior como:

$$E_{\mathbf{y}} = \|\mathbf{y} - \mathbf{W}\mathbf{y}\|^2 = \begin{cases} > E_0 \Rightarrow \text{No es reconocido (rechazar)} \\ < E_0 \Rightarrow \text{Sí es reconocido (aceptar)} \end{cases} \tag{5.1}$$

En la literatura técnica encontramos distintos enfoques a este respecto:

- Kim y Lee [28] implementan este mismo enfoque para el reconocimiento de voz mediante RNAs. Llaman a la regla anterior **de rechazo**. Ellos introducen una fase basada en reglas dentro de su sistema de reconocimiento de voz, mediante la cual ciertos patrones son desechados.
- Fleming y Cottrell [15] emplean otro procedimiento. Ellos usan una red de compresión como la de la figura 2.2 pero no lineal (usan como función de activación la sigmoide), entrenada sobre un conjunto TS que contiene 231 imágenes: 204 de la clase que se desea reconocer (caras) y 27 que no son de esa clase (por ejemplo, fotos de paisajes, de la pantalla de un PC, etc.). Las activaciones de las unidades ocultas para cada patrón se utilizan (ver fig. 6.1) como patrones de entrada para una segunda red, llamada de reconocimiento e identificación, que es un perceptrón simple, de una sola capa. Este perceptrón tiene varias unidades de salida: una para cada individuo de la clase reconocida y una única unidad cuya salida indica si el nuevo patrón es reconocido o no; es decir, los pesos que influyen en esa unidad son entrenados para dar una salida alta para los 204 patrones "cara" y una salida baja para los otros 27 patrones "no-cara."

  Este procedimiento tiene dos desventajas:

  - No se puede pretender cubrir todas las ocurrencias de "no-caras" con tan sólo 27 patrones, más aún si se han dedicado muchos más (204) para cubrir el conjunto "caras." Esto trae como consecuencia que un patrón con aspecto de cara será probablemente reconocido por la segunda red (dará una salida alta); pero ante un patrón que no sea una cara pero que tampoco se parezca a uno de los 27 casos que conoce, la salida de la unidad podrá ser alta o baja.
  - Requiere entrenar ambas redes con patrones adicionales que representen el complemento de la clase que se desea reconocer.

[1] Un nuevo patrón es realmente $\mathbf{y} - \overline{\mathbf{x}}$, ya que a todo patrón original se le resta la media de los patrones del conjunto TS antes de entrar en la red.

El problema del rechazo de un patrón es general a todas las RNAs: la RNA intenta siempre asignar un patrón a una clase, incluso si no pertenece a ninguna. Crear una clase en la que encajar todos los patrones no reconocidos requiere entrenar a la red con ejemplos que no se desee que reconozca, como ocurre en el trabajo de Fleming y Cottrell.

5.1.1 Interpretación geométrica de la regla de rechazo

La regla de rechazo (5.1) separa el espacio $\mathbb{R}^n$ de los patrones en dos regiones: la de los reconocidos y la de los no reconocidos, y viene dada por la distancia mínima del patrón al subespacio de los h primeros CPs (la norma euclídea de la proyección ortogonal del patrón sobre el subespacio), como ya sabíamos de la sección 3.3.1. La figura 5.1 representa la situación para $n = 2$ y $h = 1$. El espacio $\mathbb{R}^n$ queda partido por un subespacio de dimensión h (recta en la figura), a cuyos "lados" hay sendas regiones de grosor $\sqrt{E_0}$ que representan las regiones reconocidas (rayadas en la figura).

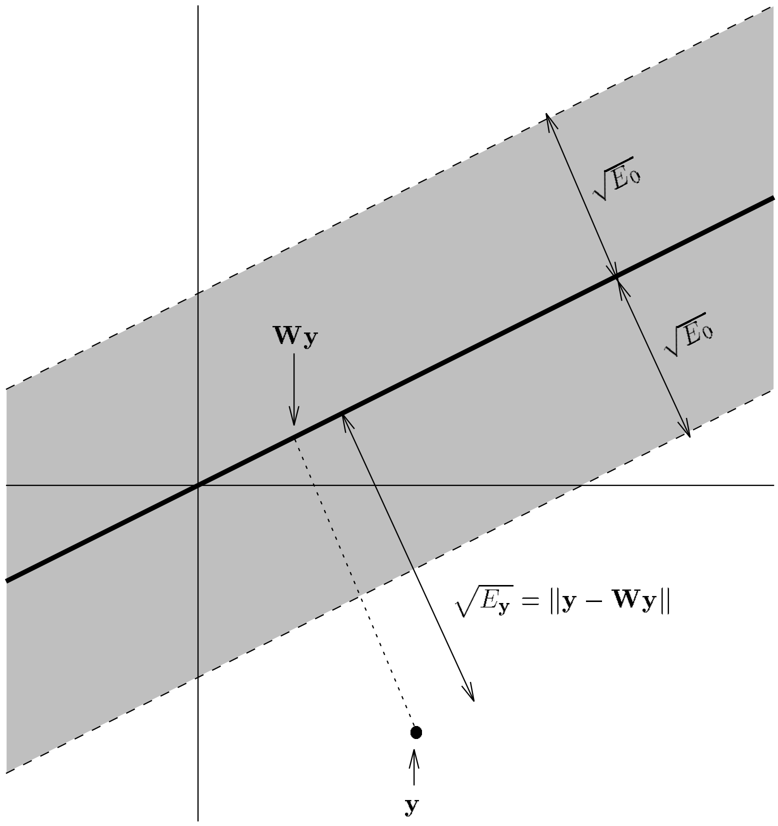

Figura 5.1: Región reconocida por la regla de rechazo y su complemento.

5.1.2 Ventajas y desventajas de la regla de rechazo

La regla de rechazo presenta la ventaja de una implementación fácil y económica, ya que no requiere ningún elemento adicional, ni tampoco patrones adicionales para entrenar la red de compresión ni ninguna otra.

Sin embargo, fijar el umbral es difícil: si está demasiado cerca del error máximo en los patrones del conjunto TS o del VTS, posiblemente rechazará patrones que, siendo de la clase deseada, difieran de los de dichos conjuntos; si por el contrario dicho valor se fija con demasiada holgura, la regla aceptará patrones que realmente no sean de dicha clase.

5.1.3 Elección del valor del umbral de rechazo

Para fijar el valor del umbral de rechazo podemos seguir varias estrategias, siempre a posteriori (una vez conocidos los errores $\{E_{\mathbf{y}_i}\}$, $i = 1, \ldots, p$). Se puede tomar, por ejemplo, igual al máximo error más una pequeña distancia de seguridad:

$$E_0 = \max_{i=1,\ldots,p} \{E_{\mathbf{y}_i}\} + k \quad k > 0$$

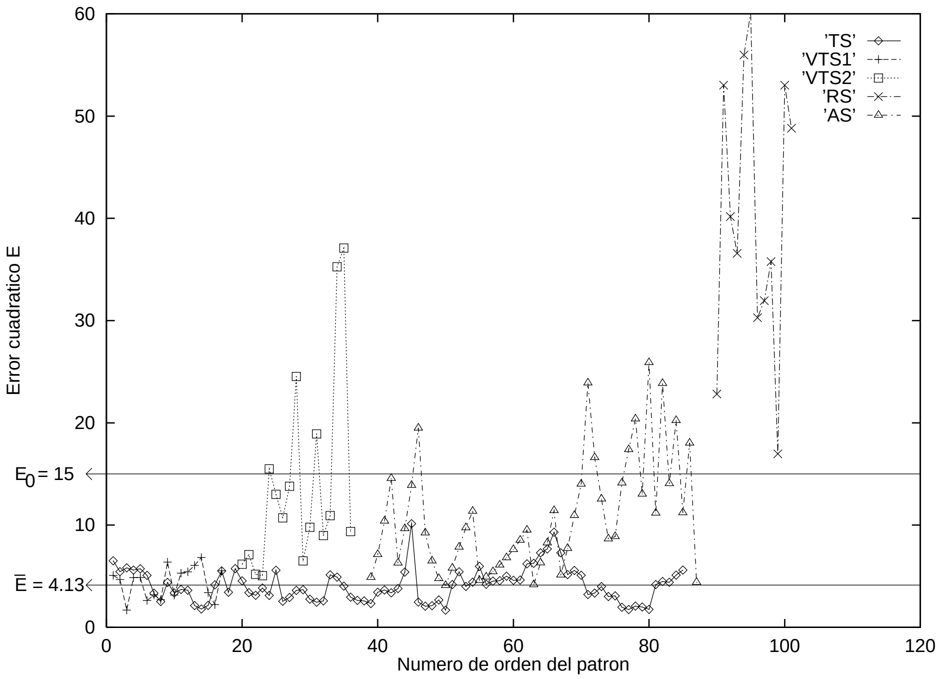

Figura 5.2: Caso 20×32, $h = 1$: errores para los conjuntos TS, VTS1, VTS2, AS y RS, valor medio del error para el conjunto TS ($\overline{E}$) y posible umbral de rechazo (E_0).

O bien, suponiendo que los errores se distribuyen normalmente con media $\overline{E}$ y varianza σ_E^2, hacer:

$$E_0 = \overline{E} + k\sigma_E \quad k > 0$$

$k = 1$ da una probabilidad del 84% de reconocer correctamente todos los patrones; $k = 2$, del 98%.

Un valor conservador para E (es decir, pequeño) puede ser mejor: rechazará algunos patrones erróneamente, pero evitará reconocimientos falsos.

5.1.4 Valor del umbral de rechazo para nuestros patrones

De nuevo, los resultados obtenidos son para los dos casos, 20×32 y 30×48, y utilizando el algoritmo de retropropagación. Para el *quickprop* no hay ninguna diferencia.

Para tener una perspectiva conjunta de los errores en los diferentes conjuntos de patrones (TS, VTS1, VTS2, AS, RS), se van a representar a la vez, en una misma figura, el error en cada patrón de cada conjunto. Los patrones llevan la siguiente numeración arbitraria para determinar su posición sobre el eje x: TS 1–85 (85 patrones), VTS1 1–17 (17 patrones), VTS2 20–36 (17 patrones), AS 39–87 (49 patrones) y RS 90–101 (12 patrones). Con esta numeración, podemos ver las partes interesantes de las distribuciones de error sin que se tapen mucho unas a otras. Cada distribución lleva, además, un trazo distinto.

Habrá una figura como la descrita para cada red. A la vista de la figura, puede observarse por dónde se mueve cada conjunto e incluso fijar a ojo un valor para el umbral de rechazo E_0.

Caso 20×32

La figura 5.2 muestra varias cosas:

- La distribución del error $E_{\mathbf{y}}$ para cada patrón del conjunto TS es bastante uniforme. Esto se debe a que todas las imágenes empleadas comparten las mismas condiciones (fondo, iluminación, orientación, etc.), las cuales son además muy homogéneas. Por supuesto, TS es el conjunto que presenta el error medio más bajo, ya que E se minimizó precisamente para él, y vale $\overline{E} = 4.13$.

- La distribución para el VTS1 es también muy uniforme y está prácticamente al mismo nivel de E que la del TS, debido a que tanto los patrones del TS como los del VTS1 forman parte de la misma serie inicial de imágenes. Podemos decir que el conjunto VTS1 valida la red obtenida, pues su error es similar al del TS.

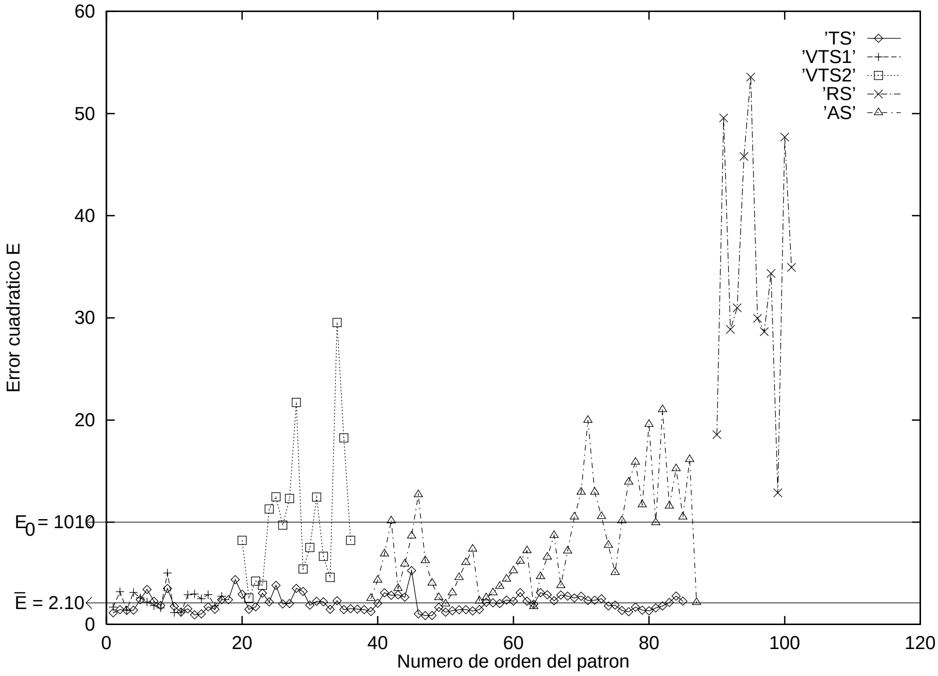

Figura 5.3: Caso 20×32, $h = 5$: errores para los conjuntos TS, VTS1, VTS2, AS y RS, valor medio del error para el conjunto TS ($\overline{E}$) y posible umbral de rechazo (E_0).

- La distribución para el VTS2 ya no es tan uniforme. Los 4 primeros patrones están al mismo nivel de E que el TS y el VTS1 porque (véase la fig. 2.8; son los 4 patrones de la izquierda de la primera fila) son los patrones sobrantes del proceso de captación, cuyas características son esencialmente iguales a las del TS y VTS1. Sin embargo, para los 13 patrones restantes el error es mayor, variando aproximadamente entre 10 y 20 (salvo casos extremos por arriba y por abajo).

  La consecuencia importante en este caso es que la red no ha sido capaz de generalizar correctamente a los patrones del conjunto VTS2; esto no debe extrañarnos, dado que los patrones de dicho conjunto son bastante distintos a los del TS. Evidentemente, si queremos que la red generalice, debemos darle patrones que cubran todas las posibilidades.

- La distribución para el conjunto AS presenta una forma de dientes de sierra notable. Esto se debe a que, conforme aumentamos la deformación de un patrón el error crece; al pasar a otro tipo de deformación, la curva empieza desde abajo de nuevo, para volver a crecer a medida que esta nueva deformación se hace más intensa, etc. Es decir, la curva sigue a los patrones mostrados en la fig. 2.10. La sección 5.2 analiza cada tipo de deformación en detalle.

- Finalmente, la distribución con mayor valor medio del error es, obviamente, la del RS, es decir, las fotos no de orejas. Dicha distribución presenta muchos picos y valles, ya que las imágenes que lo componen no tienen ninguna relación entre sí. Observemos que el menor valor del error, $E = 17$, lo obtiene un punto que se destaca considerablemente de los demás: corresponde a la imagen de ruido blanco (ver fig. 2.9). Es decir, parece que una imagen con distribución uniforme a lo largo del intervalo de grises disponible está más cerca de ser reconocida por la red que una imagen del mundo real que no sea una oreja. Esto nos da una pista para fijar el umbral de rechazo: debe estar por debajo del valor de E anterior, pero por encima del del TS. Por fortuna, tenemos suficiente margen para fijarlo, como vemos en la fig. 5.2; $E_0 = 15$ es perfectamente válido. Este umbral haría que la red reconociera todo el conjunto TS, todo el VTS1, la mayor parte del VTS2 y del AS, pero ni un solo patrón del RS, lo cual es bastante satisfactorio.

Las dos figuras siguientes, 5.3 y 5.4, representan las mismas distribuciones de error para los casos $h = 5$ y $h = 10$. Hemos mantenido la misma escala en el eje y para hacer más evidente la disminución del error medio en cada uno de los conjuntos de patrones (no sólo en TS y VTS1) conforme crece h. El error medio en el TS pasa a valer 2.10 y 0.95, para $h = 5$ y $h = 10$, respectivamente. Consecuentemente, es necesario reducir el valor del umbral; $E_0 = 10$ para $h = 5$ y $E_0 = 7$ para $h = 10$ son válidos: TS y VTS1

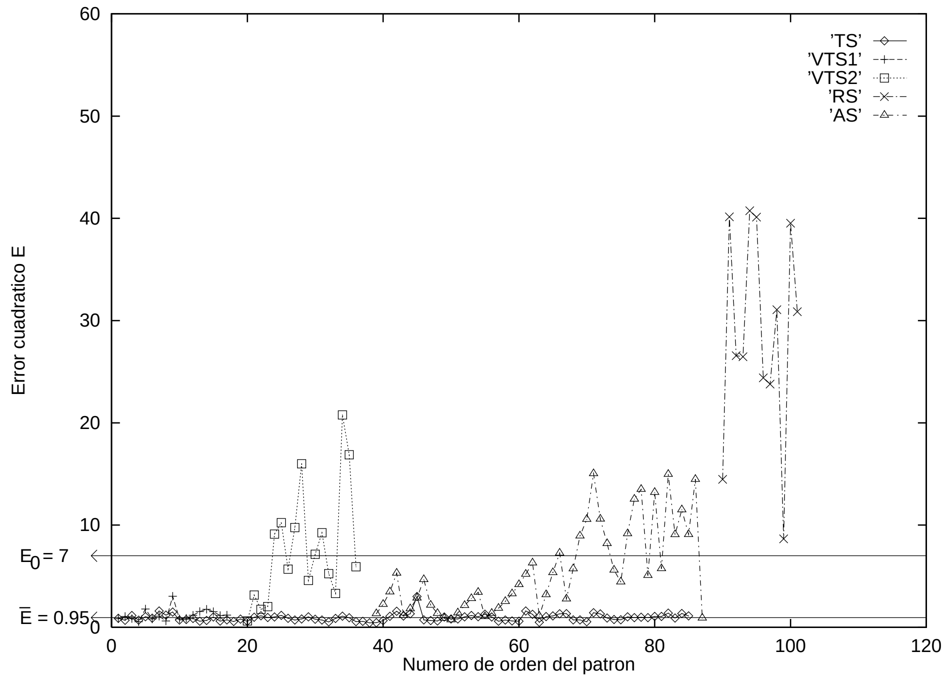

Figura 5.4: Caso 20×32, $h = 10$: errores para los conjuntos TS, VTS1, VTS2, AS y RS, valor medio del error para el conjunto TS ($\overline{E}$) y posible umbral de rechazo (E_0).

son reconocidos totalmente, AS y VTS2 en gran parte, y RS nada en absoluto. Por lo demás, los mismos comentarios de antes se pueden aplicar aquí, puesto que las figuras son esencialmente iguales (solamente se han aplastado un poco en la vertical).

Caso 30×48

Las figuras para el caso 30×48 ($h = 1$, $h = 10$ y $h = 20$) se muestran al final del capítulo sin más comentarios, por ser completamente análogas a las del caso 20×32.

5.2 Respuesta de la red ante patrones transformados

Ahora nos fijamos en cómo varía el error $E_{\mathbf{y}}$ ante un patrón que está en el conjunto TS —es decir, conocido por la red— pero que ha sido alterado mediante una transformación. Para ello haremos uso del conjunto AS, creado específicamente para esta tarea. Como se explicó en la sección 2.4.1, partiendo de un patrón fijo del TS, se le aplicaron 8 tipos de transformaciones, cada una de ellas en distinto grado. La figura 2.10 representa los patrones obtenidos. La tabla 5.1 da cuantitativamente el grado de alteración impuesto (obsérvese paralelamente a la fig. 2.10).

Las dos primeras transformaciones afectan a cada píxel de la imagen de la misma manera, en función de su intensidad; es decir, son transformaciones globales sobre la paleta de colores de la imagen. Las tres últimas, por el contrario, *mueven* píxeles de un lado a otro; son transformaciones geométricas.

En la figura 5.5 se dan los errores, por transformación, para cada patrón. El eje x da la numeración por columnas de la tabla 5.1. Como referencia, se fijan las alturas correspondientes al error del patrón original, que llamamos *error base*, $E_b = 4.411$, y al umbral de rechazo fijado anteriormente para este caso, $E_0 = 15$.

A la vista de la figura 5.5 podemos decir que:

- En el intervalo de variación de cada transformación (excepto para la traslación), el error resultante varía de manera aproximadamente lineal. Esto es particularmente notable en la adición de ruido.
- Casi sin excepción, cada alteración (por pequeña que sea) lleva consigo un aumento del error sobre el error base.
- La red soporta bastante bien la adición de ruido: incluso con un 40% añadido de ruido blanco normalizado a $[-255, 255]$, que deforma considerablemente la imagen —la cual, para una persona,

Tabla 5.1: Transformaciones correspondientes a la figura 2.10.

Fila	Transformación	Col. 1	Col. 2	Col. 3	Col. 4	Col. 5	Col. 6	Col. 7	Col. 8
1	Adición de intensidad	-40%	-30%	-20%	-10%	$+10\%$	$+20\%$	$+30\%$	$+40\%$
2	Multiplicación de intensidad	60%	70%	80%	90%	110%	120%	130%	140%
3	Adición de ruido	5%	10%	15%	20%	25%	30%	35%	40%
4	Rotación	$20°$	$15°$	$10°$	$5°$	$-5°$	$-10°$	$-15°$	$-20°$
5	Homotecia (escala)	60%	70%	80%	90%	110%	120%	130%	140%
6	Traslación	x:+20%	x:+10%	x:-10%	x:-20%	y:-20%	y:-10%	y:+10%	y:+20%

sigue siendo reconocible—, el error no sobrepasa el valor 10 (dentro del conjunto TS, algún patrón presenta un error cercano a 10). Dado que el umbral de rechazo es 15, esta categoría de imágenes sería reconocida por la red.

Esto es, por otro lado, consecuencia de un hecho observado anteriormente: el ruido blanco es, dentro de las imágenes no conocidas por la red, la que menor error producía[2]. Por eso, si $\mathbf{x}$ es el patrón original y $\mathbf{x}^* = \mathbf{x} + \mathbf{r}$ el ruidoso, tendremos que:

$$E_{\mathbf{x}^*} = \|\mathbf{x}^* - \mathbf{W}\mathbf{x}^*\|^2 = \|\mathbf{x} - \mathbf{W}\mathbf{x} + \mathbf{r} - \mathbf{W}\mathbf{r}\|^2 = E_{\mathbf{x}} + E_{\mathbf{r}} + 2((\mathbf{I}-\mathbf{W})\mathbf{x})^T(\mathbf{I}-\mathbf{W})\mathbf{r} = E_{\mathbf{x}} + E_{\mathbf{r}} + 2\mathbf{x}^T(\mathbf{I}-\mathbf{W})\mathbf{r} \quad (5.2)$$

donde se ha hecho uso de las propiedades de la proyección $\mathbf{W}$, y $E_{\mathbf{x}} = \|\mathbf{x}-\mathbf{W}\mathbf{x}\|^2$ y $E_{\mathbf{r}} = \|\mathbf{r}-\mathbf{W}\mathbf{r}\|^2$. El tercer término de (5.2) es de signo variable, según las direcciones y sentidos respectivos de $\mathbf{x}$ y $\mathbf{r}$; el segundo es, como decimos, pequeño (si $\|\mathbf{r}\|$ es pequeño). En suma, $E_{\mathbf{x}^*}$ queda relativamente cerca de $E_{\mathbf{x}}$.

- La red soporta aceptablemente la multiplicación en intensidad y la rotación: en todos los casos el patrón seguiría siendo reconocido. Notemos también que en estos dos casos, la alteración del patrón no es tan drástica como, por ejemplo, en la adición de intensidad (la imagen llega a saturarse) o la homotecia. La adición de intensidad se soporta algo peor, pero —salvo los extremos— no impide el reconocimiento.
- La red no soporta en absoluto transformaciones de escala o desplazamiento. Los errores, incluso para pequeñas alteraciones, se hacen intolerables.

En resumen, presenta bastante flexibilidad ante patrones ruidosos o con su intensidad modificada de manera global (no excesivamente), así como rotaciones de pequeño ángulo. Para homotecias y traslaciones, el error se dispara. Estas conclusiones coinciden con los resultados de otros autores, cuyas redes fallan casi siempre catastróficamente ante transformaciones geométricas de la imagen.

No se incluyen las figuras correspondientes al resto de combinaciones (20×32 y 30×48, $h = 1, 5, 10, 20$), porque son esencialmente iguales a la 5.5 (pero con E más pequeño).

5.2.1 Invarianza a transformaciones en la intensidad

El uso de patrones normalizados centrados (primero se les normaliza y luego se les resta su media), proporciona invarianza a transformaciones multiplicativas en la intensidad, ya que un nuevo patrón $\mathbf{z}^* = k\mathbf{z}$, $k \in \mathbb{R}$ se convierte, antes de entrar en la RNA (u otro dispositivo) en:

$$\mathbf{z}^* = k\mathbf{z} \xrightarrow{\text{Normalización}} \frac{\mathbf{z}^*}{\|\mathbf{z}^*\|} = \frac{\mathbf{z}}{\|\mathbf{z}\|} \xrightarrow{\text{Centrado}} \frac{\mathbf{z}}{\|\mathbf{z}\|} - \overline{\left(\frac{\mathbf{y}}{\|\mathbf{y}\|}\right)}$$

[2] Una posible explicación de este hecho es que los autovectores de orden grande se aproximan al ruido blanco; véase la figura 4.6.

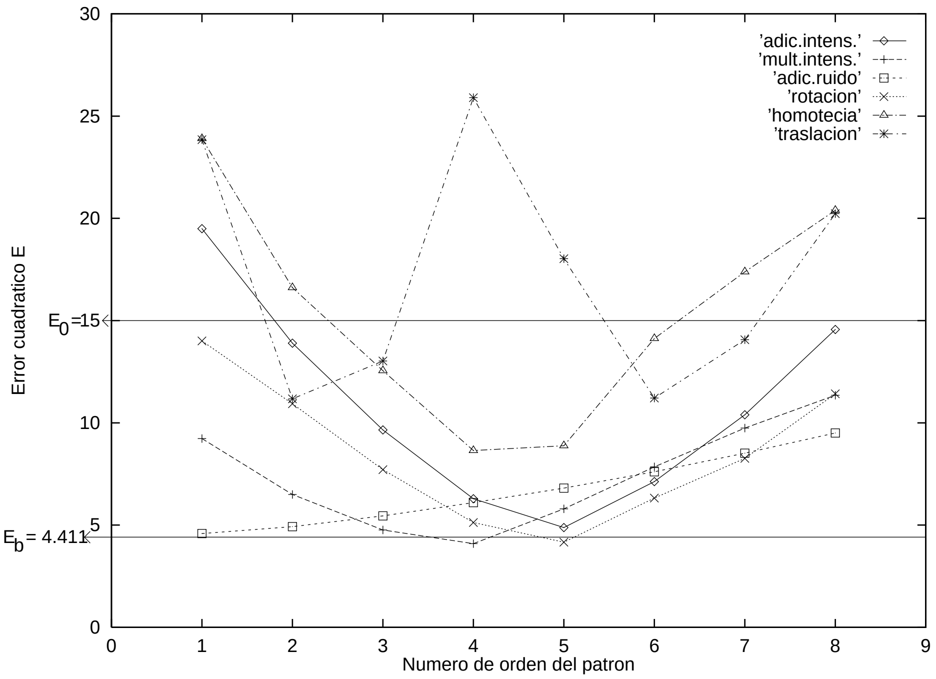

Figura 5.5: Caso 20×32, $h = 1$: errores para el conjunto AS, error base (E_b) y umbral de rechazo fijado anteriormente (E_0).

que es el mismo vector en el que se transforma el original $\mathbf{z}$; es decir, la normalización divide el espacio $\mathbb{R}^n$ de vectores en clases de equivalencia cuyo representante es el vector unitario en la dirección que sea. En la ecuación anterior, $\mathbf{y}$ representa los vectores del conjunto TS.

Nótese que lo anterior es sólo cierto si la imagen transformada no se satura; es decir, si el valor máximo de intensidad es I, la multiplicación $\mathbf{z}^* = k\mathbf{z}$ se truncará a I si lo sobrepasa. El uso de imágenes saturadas debe evitarse, ya que pierden calidad.

La invarianza a transformaciones aditivas es más difícil de conseguir. Podemos representar una transformación aditivo-multiplicativa como

$$\mathbf{z} \longrightarrow k_1\mathbf{z} + k_2\mathbf{1} \quad k_1, k_2 \in \mathbb{R} \quad \mathbf{1} = (1, 1, \ldots, 1)^T \tag{5.3}$$

Es decir, se le suman al vector $\mathbf{z}$ k_2 tonos de grises en cada píxel. Si consideramos k_2 pequeño (si no, la imagen representada por $\mathbf{z}$ se haría irreconocible incluso para una persona), la ecuación (5.3) representa una franja del plano que pasa por los puntos $\mathbf{z}$, $\mathbf{0}$ y $\mathbf{1}$. Una aplicación lineal de matriz $\mathbf{W}$ que autoasocie $\mathbf{z}$ consigo mismo no puede ser invariante a la transformación (5.3):

$$\mathbf{W}(k_1\mathbf{z} + k_2\mathbf{1}) = k_1\mathbf{W}\mathbf{z} + k_2\mathbf{W}\mathbf{1} = k_1\mathbf{z} + k_2\mathbf{W}\mathbf{1}$$

Podríamos eliminar el factor k_1 normalizando $\mathbf{z}$, pero no el debido a $\mathbf{1}$. Además, dado que en la práctica el primer autovector de la matriz $\mathbf{X}\mathbf{X}^T$ tiende a estar en la misma dirección que $\mathbf{1}$, el término $\mathbf{W}\mathbf{1}$ será aproximadamente $\mathbf{1}$.

Puede comprobarse fácilmente que la normalización no supone ninguna ventaja para otras transformaciones, como traslación, homotecia o rotación. En cuanto a la función de error E, ya se dijo en la sección 3.3.1 que la normalización puede dar lugar a una valor mayor o menor de E, dependiendo de la distribución particular de las normas de los vectores originales $\mathbf{y}$. No obstante, en el caso particular de patrones empleados por nosotros, se vio que dicho error decrecería algo.

5.3 La red de compresión como memoria autoasociativa

Ya hemos visto que la primera capa de conexiones de la red Ξ construye una representación del patrón en la capa de unidades ocultas y que la segunda capa reconstruye lo mejor que puede (en el sentido de mínimos cuadrados para el conjunto TS) el patrón de entrada. Por tanto, si éste pertenece al conjunto de entrenamiento TS o es similar a alguno de los patrones de este conjunto —que son los que conoce la red—, la red producirá en su nivel de unidades de salida una "buena" reconstrucción (dependiendo del número de unidades ocultas, h) de dicho patrón. Si no le es conocido a la red, el patrón generado por ésta seguirá teniendo apariencia de oreja, porque pertenece al subespacio de los primeros h CPs (es decir, es combinación lineal de los primeros h autovectores de $\mathbf{X}\mathbf{X}^T$), pero claro, no se parecerá al original.

En este sentido la red actúa de manera similar a como lo hace una memoria autoasociativa: si se le presenta a la entrada un patrón conocido, aunque esté ligeramente alterado (por ruido, por obstrucción de una zona de la imagen, etc.), tratará de reconstruirlo a su salida. Para comprobarlo en la práctica, hemos seleccionado 24 imágenes de resolución 30×48, que aparecen en la figura 5.6. La red empleada contiene 20 unidades en su capa oculta. En la figura 5.6, las filas impares contienen las imágenes originales y las pares las imágenes producidas por la red. Para obtener cada imagen de salida $\mathbf{y}^*$ a partir de la de entrada $\mathbf{y}$ se sigue el proceso:

$$\mathbf{y} \xrightarrow{\text{Sistema c.d.m. del TS}} \mathbf{x} = \mathbf{y} - \overline{\mathbf{y}}_{TS} \xrightarrow{\text{Red } \Xi} \mathbf{x}^* = \mathbf{W}\mathbf{x} = \mathbf{A}\mathbf{B}\mathbf{x} \xrightarrow{\text{Sistema original}} \mathbf{y}^* = \mathbf{x}^* + \overline{\mathbf{y}}_{TS}$$

dado que, como de costumbre, la red se entrena con vectores centrados. Además, las imágenes $\mathbf{y}^*$ resultantes se normalizan globalmente mediante la fórmula (4.1), para permitir su representación en 256 tonos de gris.

En la figura 5.7 se dan los errores de cada patrón (numerados de izquierda a derecha y de arriba abajo sobre la fig. 5.6). Las líneas horizontales dan los errores medios para los conjuntos TS, VTS1, VTS2 y RS, $E = \frac{1}{p}\left\langle \|\mathbf{x} - \mathbf{W}\mathbf{x}\|^2 \right\rangle$; la suma $\langle\cdot\rangle$ va extendida a los patrones de cada conjunto y $p = 85, 17, 17, 12$, respectivamente.

A continuación damos unos comentarios sobre las imágenes reconstruidas, patrón por patrón:

- 1: este patrón es el mismo que se usó como base para las transformaciones del conjunto AS. Vemos que la red lo reconstruye casi perfectamente, como es de esperar, pues pertenece al TS. Aparece algo más oscuro debido a que el proceso de normalización es global al resto de patrones.
- 2: es el patrón 1, al que se le ha restado un 40% de intensidad; pertenece al AS (ver fig. 2.10). La red lo reconstruye bastante bien.
- 3: es el patrón 1 multiplicado por 1.4; pertenece al AS (ver fig. 2.10). La red también lo reconstruye bastante bien.
- 4: es el patrón 1, al que se le ha añadido un 40% de ruido; pertenece al AS (ver fig. 2.10). La red consigue eliminar casi todas las "motas" introducidas por el ruido y el patrón reconstruido es esencialmente igual al original.
- 5: es el patrón 1 rotado 20° en sentido antihorario; pertenece al AS (ver fig. 2.10). Al contrario que en los casos anteriores, la reconstrucción ya no es buena, aunque el patrón rotado queda ligeramente delineado. Como en la sección 5.2, la red soporta bien las transformaciones sobre la paleta de grises y la adición de ruido, pero no las geométricas.
- 6–9: es el patrón 1 con una parte obstruida (de intensidad 0), ocupando un 50% del área de la imagen. La red reconstruye la parte obstruida aunque con un cierto error. Se puede observar que hay ciertas partes que la red reconstruye peor, es decir, que parecen conllevar más informacion: la parte de abajo y la parte izquierda. Otros autores han observado efectos similares con caras, llegando a la conclusión de que en ellas la parte de los ojos es la que más información conlleva.
- 10: este patrón (la oreja de una mujer, con un pendiente, con baja calidad de imagen) pertenece al conjunto VTS2 (ver fig. 2.8). La red lo reconstruye mal. De nuevo, este patrón es bastante distinto a los del conjunto TS, con lo que a la red se le escapa; no le es posible generalizar a dicho patrón partiendo de los que conoce.
- 11–24: estos patrones representan imágenes dispares, ninguno es de una oreja. Excepto los dos últimos, son todos del conjunto RS (ver fig. 2.9). La salida generada por la red tiene apariencia de oreja —un tanto monstruosa en ocasiones—, como siempre (por las razones indicadas antes) y no se parece, por tanto, al patrón de entrada.

Figura 5.6: Imágenes de entrada y su reconstrucción por la red Ξ.

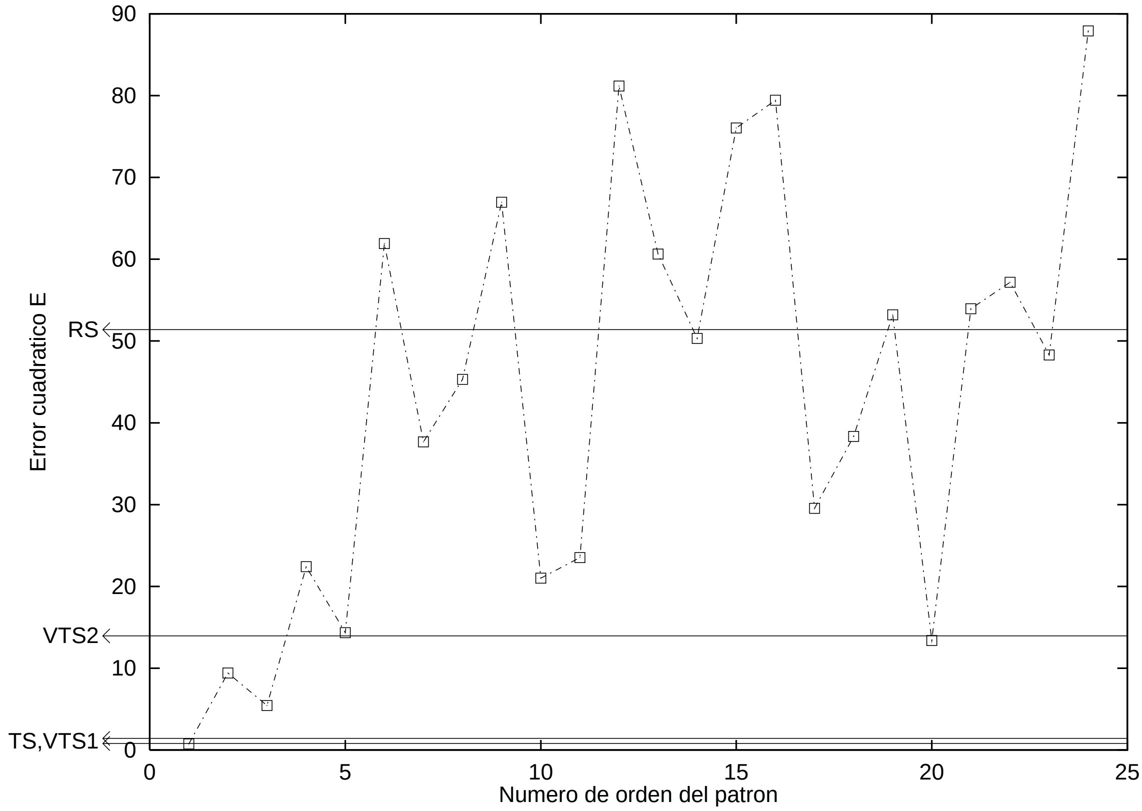

Figura 5.7: Errores para los distintos conjuntos (TS, VTS1, VTS2, RS) y para los patrones mostrados en la figura 5.6. La red empleada contenía 20 unidades ocultas, para imágenes de 30×48.

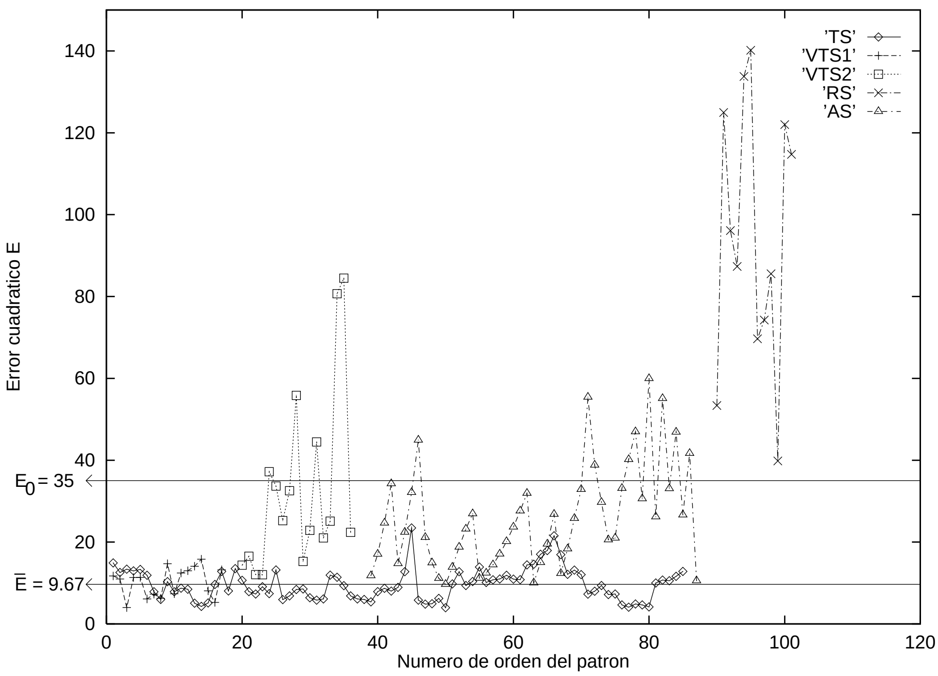

Figura 5.8: Caso 30×48, $h = 1$: errores para los conjuntos TS, VTS1, VTS2, AS y RS, valor medio del error para el conjunto TS ($\overline{E}$) y posible umbral de rechazo (E_0).

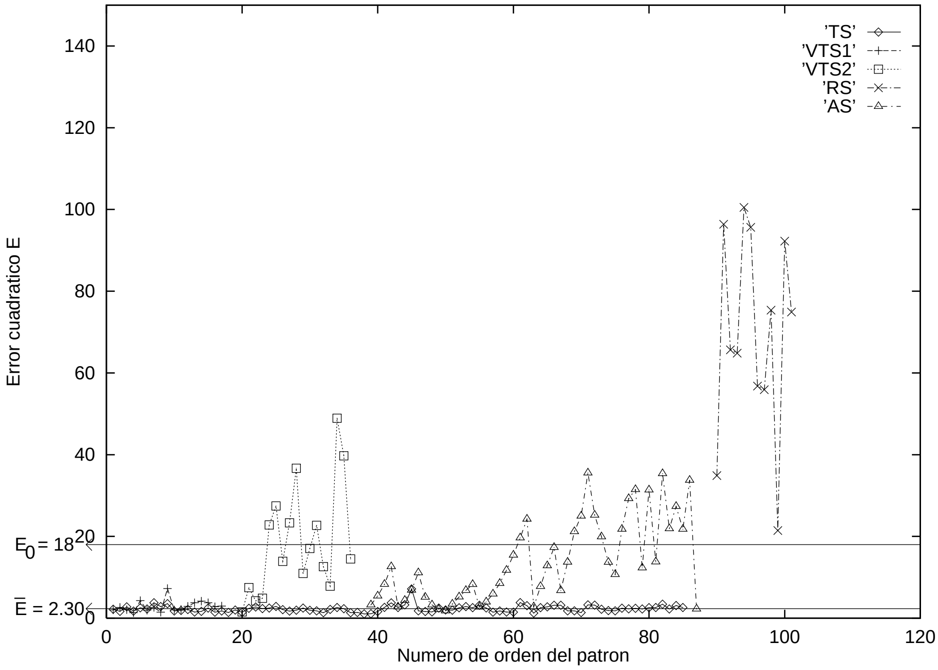

Figura 5.9: Caso 30×48, $h = 10$: errores para los conjuntos TS, VTS1, VTS2, AS y RS, valor medio del error para el conjunto TS ($\overline{E}$) y posible umbral de rechazo (E_0).

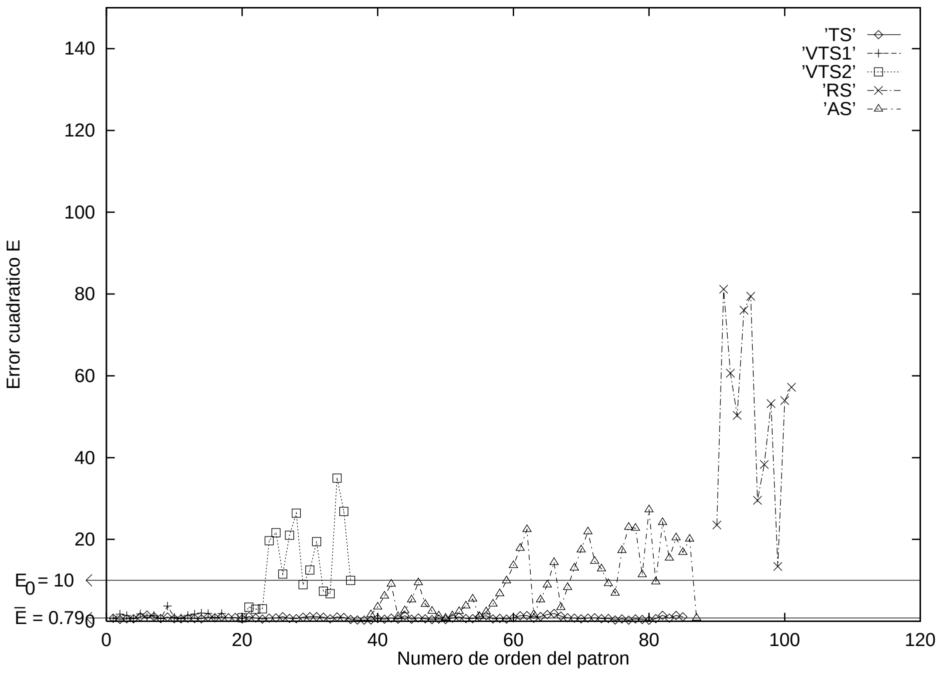

Figura 5.10: Caso 30×48, $h = 20$: errores para los conjuntos TS, VTS1, VTS2, AS y RS, valor medio del error para el conjunto TS ($\overline{E}$) y posible umbral de rechazo (E_0).

Capítulo 6

Conclusiones y perspectivas

6.1 Conclusiones del trabajo

6.1.1 Resultados generales

El trabajo desarrollado tiene dos vertientes: la del problema que aborda, la extracción de características como primera fase de un sistema de reconocimiento e identificación personal a través de imágenes de la oreja y también de la cara (que es un subconjunto de los problemas asociados a un sistema de procesamiento de imágenes faciales); y la del método empleado para abordar dicho problema, las RNAs.

Se ha dado una justificación teórica de cierto rigor, incluyendo la mayor parte de las demostraciones sobre el comportamiento de dichas redes; cosa que no suele ser habitual en muchas de las aplicaciones de las RNAs que aparecen en la literatura técnica de este campo, en las que las RNAs se consideran como *cajas negras* y se obtienen resultados a base de ir probando distintos valores para la constante de aprendizaje, el número de niveles, etc.[1] Esta justificación ha permitido:

- Relacionar el proceso de aprendizaje de la red de compresión con una técnica bien conocida en estadística, el análisis de componentes principales.
- Demostrar que el mínimo de la función de error de la red nos deja una base del subespacio de los primeros CPs en los pesos de la red, así como los componentes de la proyección de un patrón sobre dicho subespacio en las activaciones de las unidades ocultas (que constituyen el vector de características asociado a dicho patrón).

Desde otro punto de vista, se ha señalado la relación del aprendizaje de la red de compresión con la transformada de Karhunen-Loève, equivalente al análisis de CPs.

La fase de reconocimiento ha sido implementada, a partir del error en la reconstrucción por la red del patrón de entrada, mediante una simple regla de rechazo: el patrón pertenece a la clase considerada (oreja, cara) si y solamente si dicho error es menor que un cierto valor umbral, fijado a posteriori (o, equivalentemente, si la distancia de dicho patrón al subespacio de los primeros CPs es menor o igual que dicho umbral). La implementación de esta regla es trivial, y los resultados pueden considerarse buenos (sobre todo teniendo en cuenta lo económico del método). En general, parece posible escoger un valor para el umbral que resuelve satisfactoriamente el compromiso entre asegurar el reconocimiento de los patrones conocidos y rechazar los que abiertamente no son de la clase estudiada.

6.1.2 Comparación con otros métodos

Existen, sin embargo, muchas otras técnicas en análisis numérico que resuelven el mismo problema del análisis de CPs, esto es, la obtención de los autovalores no nulos y de sus autovectores asociados de una matriz real simétrica semidefinida positiva, la matriz de covarianzas de los patrones: por ejemplo, los métodos de Householder, de Givens o el algoritmo QR, todos ellos de orden $\mathcal{O}(n^3)$, que gozan de gran aceptación. Sobre un ordenador de un solo procesador, con arquitectura von Neuman, el aprendizaje por

[1] Sin embargo, hay algunos autores, como Hecht-Nielsen [24, pág. 328], que sugieren que, en el futuro, conforme crezca la complejidad de las RNAs y su análisis se haga más difícil, el estudio teórico de sus algoritmos dejará de ser una cuestión importante para tener un mero interés académico; los sistemas serán capaces de obtener sus propios algoritmos. Nosotros discrepamos rotundamente con esta postura.

retropropagación resulta muy lento[2] comparado con estos métodos, a pesar de que la superficie de error es simple (cuasicuadrática) y no presenta mínimos locales: se trata de un método de primer orden. La ventaja de la RNA tiene lugar *cuando el orden de la matriz de covarianzas es muy grande y sólo quieren hallarse los primeros CPs*, porque el tamaño de dicha matriz hace prohibitivo su cálculo explícito; en el caso anterior, que no es en absoluto grande, ya que son imágenes de $30 \times 48 = 1440$ píxeles, la matriz sería de orden 1440×1440 y tendría 2073600 elementos (realmente sólo la mitad, pues es simétrica) lo cual supone $4 \times 2073600 \approx 8$ megabytes en simple precisión. Si el número de patrones es mayor o igual que 1440, como ocurrirá en un caso real, el uso de los métodos matriciales mencionados se hace muy difícil. Por el contrario, la RNA no ocupa ningún espacio extra, aparte del requerido por los vectores de la base y por los patrones (al menos en el algoritmo acelerado, que sólo guarda la mitad de los pesos), el cual no puede reducirse, y es por tanto perfectamente aplicable.

Además, es muy importante observar que no es necesario llevar a la red muy cerca de su mínimo —como se ha hecho en este trabajo, con fines didácticos—. Las simulaciones realizadas demuestran que basta un número no muy elevado de iteraciones (aproximadamente 1000 con retropropagación y 100 con el algoritmo *quickprop*, para patrones de dimensión 1440 y 20 unidades ocultas) para que los pesos de la segunda capa alcancen prácticamente su valor límite; en este caso, el tiempo de entrenamiento se reduce a unos pocos minutos para las redes estudiadas. Desde esta perspectiva, las RNAs lineales de compresión presentan una alternativa eficiente a los métodos matriciales tradicionales para el cálculo de los componentes principales y, por tanto, para la extracción de características de las imágenes; tanto más si se considera el alto grado de paralelismo latente de la RNA, que en general no se ve igualado por el de versiones paralelas de dichos métodos.

6.1.3 Ventajas e inconvenientes de los enfoques presentados

Una ventaja evidente del empleo de imágenes de la oreja en lugar de usar imágenes faciales es la menor resolución necesaria para las primeras (por el menor tamaño de la oreja). No obstante, es necesario ver si los resultados de una segunda fase (de identificación) mejoran o igualan a los obtenidos usando imágenes faciales.

En cualquier caso, debe tenerse en cuenta que el conjunto de datos empleado es **muy pequeño** y por tanto no se puede sacar ninguna conclusión a más alto nivel; es decir, tanto los métodos desarrollados en páginas anteriores como muchos otros existentes (salvo alguna excepción) aún no han sido probados con bases de caras (u orejas) de varios miles de individuos.

Asimismo, hay que destacar la degradación que sufre el sistema cuando las imágenes de entrada no están en las condiciones normalizadas que caracterizan a las usadas durante el entrenamiento. Esto es un obstáculo para el funcionamiento robusto y flexible del sistema, ya que no siempre será posible disponer de imágenes de calidad óptima, con el objeto perfectamente centrado y derecho en la misma. La invarianza a ciertas trasformaciones comunes exige probablemente un preprocesamiento de los patrones.

6.1.4 Sobre la dimensión mínima del subespacio asociado a una clase

Mencionemos finalmente una conclusión interesante, que concierne a la dimensión del subespacio[3] de orejas o caras obtenido a partir de los patrones de entrada. Esta conclusión está relacionada con la resolución mínima que debe tener una imagen —o, de otra manera, con la cantidad de información mínima que debe tener— para poderse interpretar como oreja o cara. Sirovich y Kirby, en su aplicación del análisis de CPs al reconocimiento facial [52], señalan que dicho análisis permite estimar la dimensión mínima del subespacio de todas las caras posibles, dado un número significativo de patrones. En efecto, un espacio de 2^{14} dimensiones (en su caso, pues usaron imágenes de $128 \times 128 = 2^{14}$ píxeles) es suficiente para construir una imagen de manera aceptable, pero el análisis de CPs reduce dicha dimensión (concretamente a 100 en su caso). Luego ésta sería la dimensión del subespacio de todas las caras, o, dicho de otra manera, 100 números reales serían suficientes para identificar unívocamente una cara en el universo de todas las caras. El análisis de CPs da, pues, una estimación del número de dimensiones del conjunto (fractal) en el que encaja el espacio de todas las caras. Sin embargo, el número de patrones empleados por Sirovich y Kirby

[2] El autor calculó los 84 autovalores no nulos y sus autovectores para la matriz de covarianzas de los 85 patrones disponibles, empleando la matriz $\mathbf{X}^T\mathbf{X}$, de orden 85×85 (proposición 1.4.1). El cómputo tardó, empleando *Mathematica* 2.0 sobre un PC 486 a 66 MHz con 16M de RAM, tan sólo unos 5 minutos (en el caso de patrones de dimensión 1440). Empleando una estación Sun S.P.I. (que es aproximadamente 2.5 veces más rápida, de acuerdo con los datos de la tabla D.1), una red Ξ con 20 unidades ocultas con retropropagación simple tardó varias horas en generar la base del subespacio de los primeros 20 autovectores (requirió 4000 iteraciones en reducir el error a una distancia de 0.7 de su mínimo). La misma red con el algoritmo *quickprop* empleó una media hora en acercarse a esa misma distancia (200 iteraciones).

[3] Siempre suponiendo que la clase en cuestión tiene propiedades de espacio vectorial, lo cual no es cierto en general.

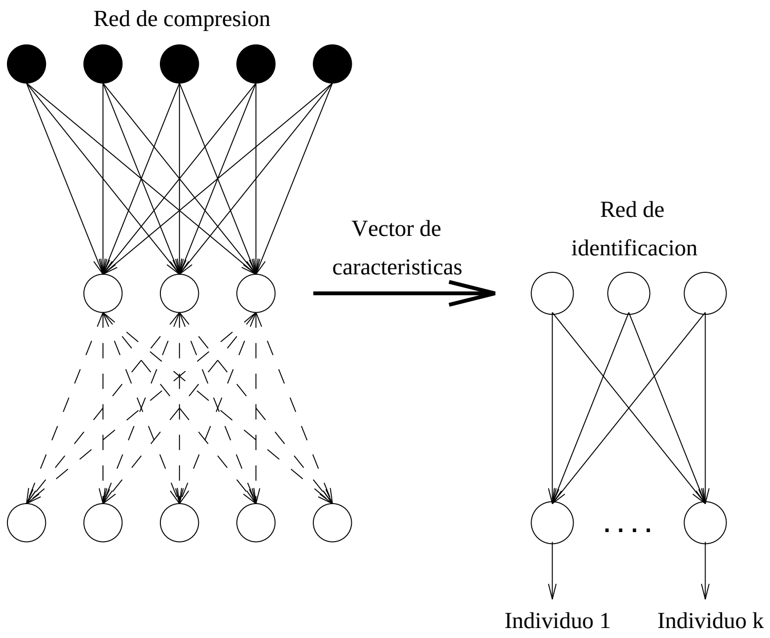

Figura 6.1: Uso de la red de compresión como fase de extracción de características de un sistema de identificación. La segunda red se encarga de la identificación propiamente dicha.

(115) es demasiado pequeño como para poder, siquiera a grosso modo, fijar tal dimensión. Además, en ella no se tendrían en cuenta los patrones transformados por rotación, etc. Algo similar nos ocurre a nosotros con los datos que hemos podido emplear, necesariamente limitados en número y en resolución por dos razones: la dificultad de obtenerlos (aunque últimamente han ido apareciendo algunas bases de datos de imágenes faciales, véase la sección B.3) y el elevado tiempo de proceso, que se incrementa con el número de patrones y con su resolución.

6.2 Desarrollos futuros

Como se dijo en la introducción, el procesamiento facial (sea usando enfoques conexionistas o de otro tipo) se encuentra en su infancia y son muchas las direcciones en las que se está progresando. En lo que concierne a este trabajo en particular, las siguientes continuaciones son claras:

- Utilizar la red de compresión como fase de extracción de características de un sistema de identificación. Una segunda fase podría emplear otra RNA, por ejemplo un perceptrón (de una sola capa o de varias) para —entrenado sobre dichas características— realizar la identificación (ver fig. 6.1). Es decir, las activaciones de las unidades ocultas obtenidas para cada patrón en la primera red (de compresión) constituyen el conjunto de entrenamiento TS de la segunda red (de identificación[4]).

  Evidentemente, no es necesario que la fase de identificación sea llevada a cabo por una RNA, aunque esto da uniformidad al sistema. Existen otros métodos de la teoría de reconocimiento de patrones que pueden realizar esta tarea: clasificador por distancia euclídea, bayesiano, etc. (véase [54], por ejemplo). Dado que el perceptrón de una sola capa tan sólo es capaz de determinar un hiperplano discriminante, sólo funcionará correctamente con clases linealmente separables. En casos más complejos podrá ser necesario emplear perceptrones de 2 ó 3 capas u otro método capaz de encontrar regiones de decisión más generales (cóncavas, no conexas, etc.).

- El trabajo ha aplicado los métodos descritos solamente a imágenes de la oreja. No hay nada que impida usar la misma arquitectura de RNA para las imágenes de caras. De hecho, las imágenes están ya tomadas (ver el apéndice C) y basta repetir el proceso indicado en el capítulo 4. A priori, parece que uno debería poder obtener resultados para las caras muy similares a los expuestos en dicha sección para las orejas.

[4] Fleming y Cottrell [15] implementaron un sistema de estas características con buenos resultados.

- Es esencial, tanto con la arquitectura de RNA empleada como con otras, usar un conjunto de patrones realista, tanto en resolución de imagen como en número de individuos y número de imágenes por individuo[5]:

  - Resolución: posiblemente exista una resolución óptima que permita el reconocimiento y la identificación y a la vez sea lo menor posible, para que la RNA (u otro método que se emplee) sea lo más compacta y eficiente posible. Samal e Iyengar [48] sugieren que, para reconocer una imagen como cara, son suficientes entre 16×16 y 32×32 píxeles; para su identificación es necesaria una resolución mayor. Lo mismo ocurre para el número de niveles de gris necesarios, aunque en este caso dicho valor sea menos crítico (256 e incluso menos parecen más que suficientes para cualquier aplicación). En cualquier caso, las resoluciones empleadas en este trabajo (20×32 y 30×48) son posiblemente demasiado pequeñas, incluso para orejas.
  - Un número elevado de individuos es importante para el reconocimiento. Sin embargo, algunas aplicaciones de identificación no requieren un número grande; por ejemplo, para un sistema de control de acceso personal a unas instalaciones pueden ser suficientes 20 ó 30 individuos. No obstante, con vistas a evaluar un método de identificación es necesario probarlo con un número muy de caras (Chellappa *et al.* [10] sugieren entre 5000 y 50000, dependiendo de la aplicación).
  - Mucho más importante es el número de imágenes por individuo[6]: en un espacio de miles de dimensiones, no se puede pretender aproximar la región en la que se mueven las instancias de un individuo dado con unos pocos patrones (5 en el presente trabajo), porque muchas posibilidades se nos escapan —el espacio está prácticamente vacío. La obtención de datos se hace, consecuentemente, más costosa, y el tiempo de entrenamiento de la red mayor. El tamaño de la red no varía, sin embargo. Estas mismas consideraciones son aplicables al punto anterior sobre el número de individuos.

- Gracias a la comprensión teórica que se tiene del análisis de componentes principales (ver la sección 3.1), es suficiente construir la distribución de los autovalores de la matriz de covarianzas para predecir los resultados (el error) en función del número de componentes que se deseen conservar. Sería interesante, desde un punto de vista general, conocer dicha distribución para diversas "familias" de imágenes (orejas, caras, etc.) y diversas resoluciones[7]. Por supuesto que, para resoluciones elevadas (p. ej. 128×128) y muchos patrones, el problema completo (que requiere obtener *todos* los autovalores y autovectores) se hace inatacable computacionalmente. De cualquier manera, de los datos usados en este trabajo y en otros (Fleming y Cottrell [15], O'Toole *et al.* [39], etc.) se desprende el hecho de que la curva de autovalores desciende rápidamente al principio.

- Estudiar (usando, p. ej., una de las redes descritas en la sección 3.4, capaces de extraer componentes principales aislados, o directamente obteniendo todos los autovectores de la matriz de covarianzas) el conjunto de autovectores en cuanto a: capacidad discriminante —O'Toole *et al.* [39] indican que, para la identificación, los primeros CPs no son los más útiles—, significado (O'Toole *et al.* [39], Valentin *et al.* [56]) —el primer autovector parece ir asociado a la forma de la cara; el segundo, al sexo y a la raza; etc. (O'Toole *et al.* [39], Valentin *et al.* [56])— y otras posibles características. Para ello será necesario utilizar imágenes de mayor resolución; O'Toole *et al.* [39] emplearon una memoria autoasociativa de imágenes de 151×225, con 16 niveles de gris.

- Emplear otras arquitecturas de RNAs en la fase de extracción de características (compresión). En este trabajo sólo se han empleado redes lineales completas de dos niveles, pero hay muchas variaciones posibles que merece la pena considerar:

  - Redes no lineales, con distintas funciones de activación. *SNNS* dispone de muchos tipos de funciones de activación y permite al usuario añadir los suyos propios en lenguaje C de manera sencilla,

[5]Éste es un problema que ha plagado todas las investigaciones realizadas hasta ahora, dada la dificultad de obtener un número grande de fotos y de entrenar la red asociada. Por ejemplo, Fleming y Cottrell [15] utilizaron 64 fotos de 17 sujetos distintos, Kaufman y Breeding [27] 120 de 10, Samaria [49] 400 de 40, Petkov *et al.* [40] 205 de 30, O'Toole *et al.* [39] 159, Sirovich y Kirby [52] 115 y Liu y Lee [35] 100. El número de patrones está, pues, en torno a la centena, el cual es a todas luces muy pequeño.

[6]Si se están empleando RNAs o cadenas de Markov; para el enfoque tradicional, basado en rasgos geométricos, basta una imagen —de buena calidad y sin elementos (pelo, gafas, etc.) que obstruyan puntos clave— por individuo.

[7]Se ha demostrado que, para matrices simétricas aleatorias, el primer autovalor es mucho mayor que el segundo prácticamente siempre.

– Redes incompletas, en las que cada nivel no está completamente conectado al siguiente. El patrón disperso de interconexión puede fijarse al principio, haciendo conexiones por bloques solapados; este tipo de arquitectura es bastante usado en procesamiento de imágenes y permite una reducción importante del número de conexiones, pero es difícil dar una justificación rigurosa de su comportamiento. Particionando la imagen original de $M \times N$ píxeles en bloques de $m \times n$, con $1 \leq m \leq M$, $1 \leq n \leq N$, como en la fig. 6.2, se obtienen diversas configuraciones. Por ejemplo, si no hay solapamiento, habrá en total MN/mn bloques[8]:

  ∗ $m = n = 1$: MN bloques de 1×1.
  ∗ $m = M$, $n = N$: 1 bloque de $M \times N$.
  ∗ $m = 1$, $n = N$: M filas de $1 \times N$.
  ∗ $m = M$, $n = 1$: N columnas de $M \times 1$.

  El solapamiento ayuda a correlar píxeles adyacentes, ya que fija el grado de acoplamiento entre bloques, y puede representarse por medio del desplazamiento del bloque $m \times n$ al irse moviendo sobre la imagen original. Si el solapamiento es lo más fino posible (desplazamiento de 1 píxel), habrá $(M - m + 1)(N - n + 1)$ bloques; solapamientos menos finos generarán un número de bloques menor.

  En cualquier caso, cada píxel de cada bloque actúa como una entrada de la RNA, pero estas entradas no llegan a cada unidad del nivel siguiente, sino que cada una recibe solamente las de un bloque.

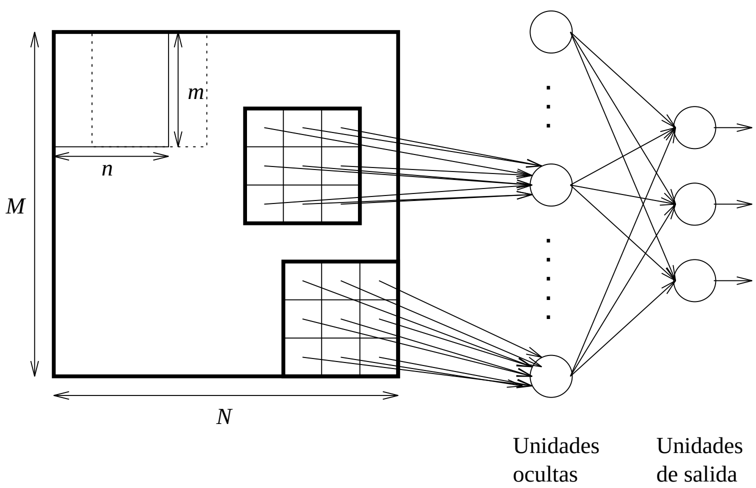

Figura 6.2: Partición por bloques de $m \times n$ de una imagen original de $M \times N$.

  Liu y Lee [35] aplican a cada bloque de la imagen una pequeña red Ξ y luego reúnen las salidas; llaman a este método "de ventanas."

  Por otro lado, también se puede alterar dinámicamente la estructura de la red, empleando técnicas de poda (*pruning*) y decaimiento de pesos (*weight decay*) para ir eliminando unidades y conexiones durante el aprendizaje (ver [25, págs. 156–162], entre otros). *SNNS* implementa ambas técnicas.

– Redes con más capas intermedias. Baldi [3] da algunos resultados teóricos al respecto.

– Otro tipo de arquitecturas, no de alimentación hacia adelante: los mapas de rasgos autoorganizativos de Kohonen [25, 24, 17], por ejemplo.

Los *surveys* [48, 56, 10] citan muchos otros tipos de enfoques (no sólo conexionistas) al problema.

- Implementación efectiva de un sistema integrado que haga uso del reconocimiento e identificación personal. Un ejemplo sería un sistema de control de acceso como el de la figura 6.3, que incorpore captación de datos (del sujeto que intenta acceder a la instalación), identificación positiva o negativa del mismo y en su caso apertura de la puerta u otra acción predeterminada.

[8] En [36] se utiliza una RNA reconocedora de caracteres sobre bloques (filas y columnas) de la imagen para comparar varios simuladores de RNAs.

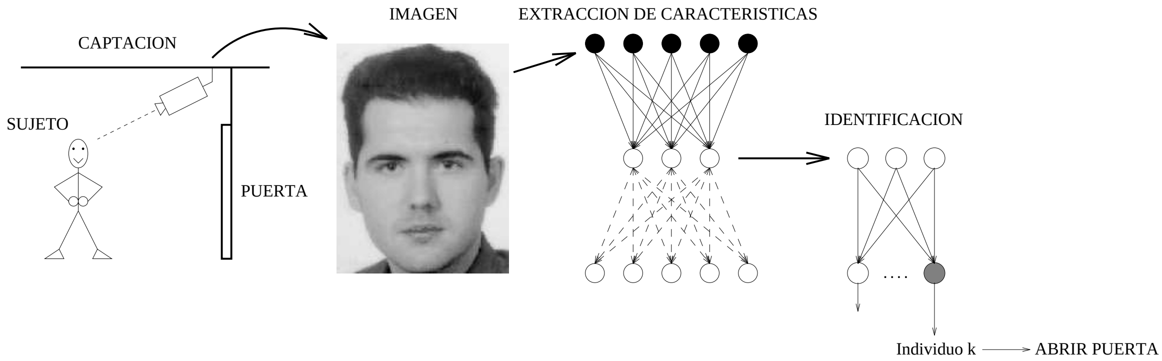

Figura 6.3: Representación esquemática de un dispositivo de control de acceso por medio de un sistema de identificación facial.

- Análisis e implementación de otros enfoques de reconocimiento e identificación, distintos del de las RNAs: las cadenas de Markov [49], convolución de la imagen con funciones de Gabor [40], etc. Los *surveys* [48, 56, 10] dan información sobre estos y otros enfoques.

- Al igual que para las caras se han hecho bastantes estudios sobre qué rasgos usar a priori (distancia interocular, ángulo de la frente, etc.), lo mismo es posible para las orejas. Para esto es desde luego necesario cierto conocimiento del dominio (anatomía de la oreja, distribución estadística de las distintas variedades de orejas, dentro de una misma raza o fuera de ella, etc.) para seleccionar aquellas características o rasgos que conlleven un mayor poder discriminante.

- Dotar al sistema de invarianza a ciertas transformaciones: traslación, homotecia (escala) y rotación, modificaciones en la intensidad (aditivas, multiplicativas u otras) y mayor robustez frente al ruido. Una de las mayores desventajas que presentan las RNAs es su falta de flexibilidad para reconocer patrones transformados. Una solución es aumentar el conjunto de entrenamiento con variaciones de los patrones originales (p. ej., para obtener invarianza a rotaciones, incluir en el conjunto de entrenamiento cada patrón rotado $0, 15, 30, \ldots, 345°$), pero entonces el número de patrones crece enormemente y con él el tiempo de entrenamiento de la RNA. Además, si todas las instancias posibles le son dadas a la red en su entrenamiento, no existe generalización, que es la propiedad más preciada de la red. Wang [57] ha puesto de relieve este profundo problema de las RNAs, actualmente sin solución satisfactoria.

  Otra posibilidad es obtener la invarianza externamente a la red, mediante un preprocesamiento de los datos. Ya se demostró en la sección 5.2.1 que el uso de patrones normalizados da invarianza a transformaciones multiplicativas (pero no aditivas) de la intensidad. Igualmente puede conseguirse invarianza a las demás transformaciones proporcionando a la red imágenes en unas condiciones normalizadas de posición, tamaño, orientación y luminosidad mediante el uso de técnicas comunes de procesamiento de imágenes (descriptores de Fourier, etc.). Tanto las caras como las orejas presentan un contorno aproximadamente elíptico que facilita su localización y segmentación —si se destacan suficientemente del fondo— y la obtención de sus elementos geométricos (los ejes principales y el centro de masas), por lo cual dicha normalización es factible mediante una transformación afín de parámetros adecuados (ver fig. 6.4):

  $$\begin{pmatrix} x' \\ y' \end{pmatrix} = k \begin{pmatrix} \cos\alpha & -\operatorname{sen}\alpha \\ \operatorname{sen}\alpha & \cos\alpha \end{pmatrix} \begin{pmatrix} x-a \\ y-b \end{pmatrix}$$

  El problema del ruido puede eliminarse pasando los patrones por un filtro paso bajo antes de entrar en la red. No obstante, éste es quizá el problema menos acuciante, pues ya vimos en la sección 5.2 que la red respondía aceptablemente a niveles de ruido de hasta el 40%.

  Liu y Lee [35] realizan un preprocesamiento de las imágenes faciales que utilizan (no dicen si de manera automática) para colocar la cara en posición vertical. También normalizan las imágenes en intensidad.

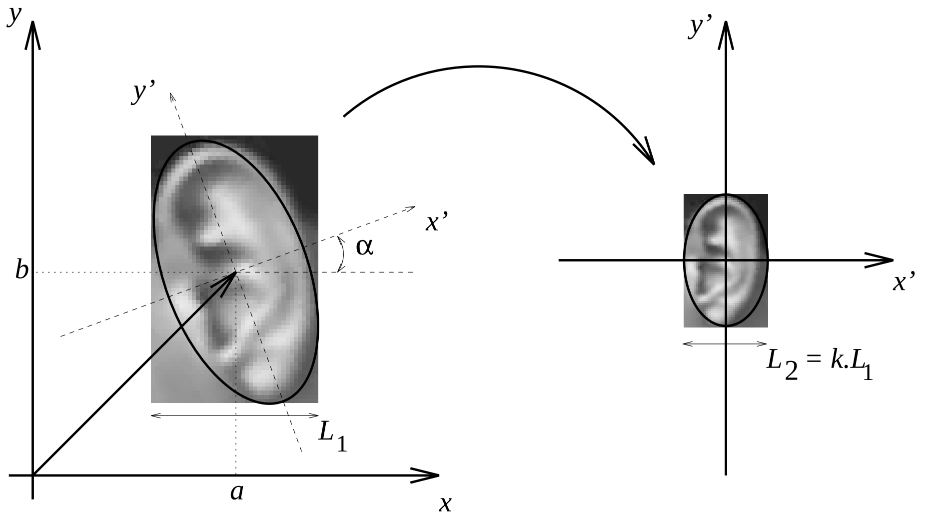

Figura 6.4: Normalización de una imagen a partir de su centro de masas y de sus ejes principales.

Apéndice A

Demostraciones adicionales

Este apéndice detalla algunas demostraciones y observaciones de carácter matemático que, por su complejidad o carácter secundario, hemos preferido no incluir en el texto principal.

A.1 Clasificación de la forma $E(\mathbf{W})$

Este apéndice requiere conocimiento de la teoría de clasificación de formas cuadráticas y superficies de segundo grado; véase, por ejemplo, [6, págs. 243–253] o [11, págs. 281–344].

A.1.1 Formas cuadráticas mayor y menor de la función de error

Desarrollando $E(\mathbf{W})$ para $\mathbf{W}$ no restringida, es decir, todos sus elementos w_{ij} son independientes entre sí:

$$\begin{aligned}
E = \left\langle \|\mathbf{x} - \mathbf{W}\mathbf{x}\|^2 \right\rangle = \\
\left\langle \sum_i \left[x_i^2 + \sum_j w_{ij}^2 x_j^2 + 2 \sum_{j<k} w_{ij} x_j w_{ik} x_k - 2 x_i \sum_j w_{ij} x_j \right] \right\rangle = \\
\sum_i \left[\left\langle x_i^2 \right\rangle + \sum_j w_{ij}^2 \left\langle x_j^2 \right\rangle + 2 \sum_{j<k} w_{ij} w_{ik} \left\langle x_j x_k \right\rangle - 2 \sum_j w_{ij} \left\langle x_i x_j \right\rangle \right] = \\
\operatorname{tr} \mathbf{X}\mathbf{X}^T + \sum_{ij} w_{ij}^2 \chi_{jj} + 2 \sum_i \sum_{j<k} w_{ij} w_{ik} \chi_{jk} - 2 \sum_{ij} w_{ij} \chi_{ij} = \\
\operatorname{tr} \mathbf{X}\mathbf{X}^T + \sum_i \sum_j w_{ij} \sum_k \chi_{jk} w_{ik} - 2 \sum_{ij} w_{ij} \chi_{ij} = \\
\operatorname{tr} \mathbf{X}\mathbf{X}^T + (\operatorname{vec} \mathbf{W})^T \mathbf{Q} (\operatorname{vec} \mathbf{W}) - 2 (\operatorname{vec} \mathbf{W})^T (\operatorname{vec} \mathbf{X}\mathbf{X}^T)
\end{aligned} \tag{A.1}$$

donde $\mathbf{X}\mathbf{X}^T = (\chi_{ij})$, $\operatorname{tr} \mathbf{X}\mathbf{X}^T = \|\mathbf{X}\|^2$ y $(\operatorname{vec} \mathbf{W})^T \mathbf{Q} (\operatorname{vec} \mathbf{W})$ es la forma cuadrática menor de E, de matriz $\mathbf{Q}_{n^2 \times n^2}$. $\mathbf{Q}$ es diagonal por bloques, y cada bloque es la matriz $\mathbf{X}\mathbf{X}^T$ (ver fig. A.1). Si llamamos $\mathcal{W}$ a la concatenación de w_0 y $\operatorname{vec} \mathbf{W}$ y ampliamos $\mathbf{Q}$ a la matriz $\mathcal{Q}_{n^2+1 \times n^2+1}$ en la manera indicada en la fig. A.3, se tiene:

$$\begin{aligned}
\mathcal{W}^T \mathcal{Q} \mathcal{W} = \\
w_0 (w_0 \operatorname{tr} \mathbf{X}\mathbf{X}^T - (\operatorname{vec} \mathbf{X}\mathbf{X}^T)^T (\operatorname{vec} \mathbf{W})) - w_0 (\operatorname{vec} \mathbf{X}\mathbf{X}^T)^T (\operatorname{vec} \mathbf{W}) + (\operatorname{vec} \mathbf{W})^T \mathbf{Q} (\operatorname{vec} \mathbf{W}) = \\
w_0^2 \operatorname{tr} \mathbf{X}\mathbf{X}^T - 2 w_0 (\operatorname{vec} \mathbf{W})^T (\operatorname{vec} \mathbf{X}\mathbf{X}^T) + (\operatorname{vec} \mathbf{W})^T \mathbf{Q} (\operatorname{vec} \mathbf{W})
\end{aligned} \tag{A.2}$$

que es la forma cuadrática mayor de E; haciendo $w_0 = 1$ se obtiene E y haciendo $w_0 = 0$ se obtiene la forma cuadrática menor.

A.1.2 Propiedades de $\mathbf{Q}$ y $\mathcal{Q}$

- $\mathbf{Q}$ es semidefinida positiva pues $\mathbf{X}\mathbf{X}^T$ lo es (por la proposición 1.4.1).
- $\mathbf{Q}$ tiene los mismos autovalores que $\mathbf{X}\mathbf{X}^T$, cada uno repetido n veces, ya que $p_{\mathbf{Q}}(\lambda) = (p_{\mathbf{X}\mathbf{X}^T}(\lambda))^n$

$$\mathbf{Q}_{n^2\times n^2} = \begin{pmatrix} \mathbf{X}\mathbf{X}^T & & \\ & \ddots & \\ & & \mathbf{X}\mathbf{X}^T \end{pmatrix} \qquad \operatorname{vec}\mathbf{W} = \begin{pmatrix} w_{11} \\ \vdots \\ w_{1n} \\ \vdots \\ w_{n1} \\ \vdots \\ w_{nn} \end{pmatrix}$$

Figura A.1: Matriz $\mathbf{Q}$ de la forma cuadrática menor de E y vector $\operatorname{vec}\mathbf{W}$ asociado.

$$\text{Bloque } i \text{ de } \mathbf{Q} \left\{ \begin{array}{c} \begin{array}{cccccc} & \chi_{11} & & & & \chi_{1n} \\ & & \ddots & & & \\ j \rightarrow & \boxed{\chi_{j1} \quad \cdots \quad \chi_{jk} \quad \cdots \quad \chi_{jn}} & & & & \\ & & \ddots & & & \\ & \chi_{n1} & & & & \chi_{nn} \\ & \longleftarrow & k & \longrightarrow & & \end{array} \end{array} \right. \longleftrightarrow \left. \begin{array}{cc} \boxed{\begin{array}{c} w_{i1} \\ \vdots \\ w_{1k} \\ \vdots \\ w_{in} \end{array}} & \begin{array}{c} \uparrow \\ k \\ \downarrow \end{array} \end{array} \right\} \text{Fila } i \text{ de } \mathbf{W}$$

Figura A.2: Multiplicación $\sum_i \sum_j w_{ij} \sum_k \chi_{jk} w_{ik} = (\operatorname{vec}\mathbf{W})^T \mathbf{Q} (\operatorname{vec}\mathbf{W})$.

- $\operatorname{rg}\mathbf{Q} = n \operatorname{rg}\mathbf{X}\mathbf{X}^T$, $\operatorname{sig}\mathbf{Q} = n \operatorname{sig}\mathbf{X}\mathbf{X}^T = (n \operatorname{rg}\mathbf{X}\mathbf{X}^T, 0)$.
- $\operatorname{rg}\mathcal{Q} = \operatorname{rg}\mathbf{Q}$, ya que la primera fila de $\mathcal{Q}$ se obtiene a partir de $\mathbf{Q}$ sumando la primera fila del bloque 1, la segunda del 2,..., la n-ésima del n y multiplicando el resultado por -1.
- $p_{\mathcal{Q}}(\lambda) = \lambda p_{\mathbf{Q}}(\lambda)$, pues los autovalores de $\mathbf{Q}$ lo son de $\mathcal{Q}$:

$$\mathbf{Q}(\operatorname{vec}\mathbf{W}) = \lambda(\operatorname{vec}\mathbf{W}) \Rightarrow \mathcal{Q}\begin{pmatrix} w_0 \\ \operatorname{vec}\mathbf{W} \end{pmatrix} = w_0 \begin{pmatrix} \operatorname{tr}\mathbf{X}\mathbf{X}^T \\ \operatorname{vec}\mathbf{X}\mathbf{X}^T \end{pmatrix} + \begin{pmatrix} -(\operatorname{vec}\mathbf{W})^T(\operatorname{vec}\mathbf{X}\mathbf{X}^T) \\ \mathbf{Q}(\operatorname{vec}\mathbf{W}) \end{pmatrix} =$$
$$(\text{haciendo } w_0 = 0) = \lambda \begin{pmatrix} \operatorname{tr}\mathbf{W} \\ \operatorname{vec}\mathbf{W} \end{pmatrix} = \lambda \begin{pmatrix} 0 \\ \operatorname{vec}\mathbf{W} \end{pmatrix} \quad \text{(A.3)}$$

  ya que podemos hacer $\operatorname{tr}\mathbf{W} = 0$ multiplicando por constantes adecuadas las filas de $\mathbf{W}$ (que seguirán siendo autovectores de $\mathbf{X}\mathbf{X}^T$).
- $\operatorname{sig}\mathcal{Q} = \operatorname{sig}\mathbf{Q}$.

A.1.3 Clasificación de $E(\mathbf{W})$

Por tanto, $\operatorname{rg}\mathcal{Q} = \operatorname{rg}\mathbf{Q}$ y $\operatorname{sig}\mathcal{Q} = \operatorname{sig}\mathbf{Q} = (\operatorname{rg}\mathbf{Q}, 0)$. De acuerdo con la teoría de superficies de segundo grado, $E = 0$ es:

- Un hipercono imaginario si $\operatorname{rg}\mathbf{X}\mathbf{X}^T = n \Leftrightarrow \operatorname{rg}\mathbf{Q} = n^2$. Es decir, existe un conjunto de variables $\Omega = \{\omega_{11}, \ldots, \omega_{nn}\}$, obtenido mediante la siguiente transformación afín de las variables originales:

$$\operatorname{vec}\Omega = \mathbf{P}^T(\operatorname{vec}\mathbf{W}) - \mathbf{d} \quad \text{(A.4)}$$

  donde $\mathbf{P}_{n^2+1}$ es una matriz ortogonal de autovectores de $\mathbf{Q}$ y $\mathbf{d}_{n^2+1}$ un vector dependiente sólo de elementos de $\mathbf{X}\mathbf{X}^T$. En el conjunto de variables Ω, E toma la forma:

$$E = \sum_{i=1}^{n}\sum_{j=1}^{n} \lambda_j \omega_{ij}^2 = 0$$

  También puede considerarse un hiperelipsoide de semiejes E/λ_j si $E > 0$:

$$\sum_{i=1}^{n}\sum_{j=1}^{n} \frac{\omega_{ij}^2}{E/\lambda_j} = 1$$

$$
\mathcal{Q} = \begin{pmatrix}
\operatorname{tr}\mathbf{XX}^T & -\chi_{11} & \cdots & -\chi_{1n} & \cdots & -\chi_{11} & \cdots & -\chi_{nn} \\
-\chi_{11} & \chi_{11} & \cdots & \chi_{1n} & & & & \\
\vdots & \vdots & \ddots & \vdots & & & & \\
-\chi_{1n} & \chi_{n1} & \cdots & \chi_{nn} & & & & \\
\vdots & & & & \ddots & & & \\
-\chi_{n1} & & & & & \chi_{11} & \cdots & \chi_{1n} \\
\vdots & & & & & \vdots & \ddots & \vdots \\
-\chi_{nn} & & & & & \chi_{n1} & \cdots & \chi_{nn}
\end{pmatrix}
\qquad
\mathcal{W} = \begin{pmatrix} w_0 \\ w_{11} \\ \vdots \\ w_{1n} \\ \vdots \\ w_{n1} \\ \vdots \\ w_{nn} \end{pmatrix}
$$

Figura A.3: Matriz $\mathcal{Q}$ de la forma cuadrática mayor de E y vector $\mathcal{W}$ asociado.

- Planos imaginarios si $\operatorname{rg}\mathbf{XX}^T = n_1 < n$ (casos degenerados):

$$E = \sum_{i=1}^{n}\sum_{j=1}^{n_1} \lambda_j \omega_{ij}^2 = 0$$

O un hipercilindro elíptico si $E > 0$:

$$\sum_{i=1}^{n}\sum_{j=1}^{n_1} \frac{\omega_{ij}^2}{E/\lambda_j} = 1$$

En ambos casos, $\min_{\mathbf{W}} E = 0$ y se obtiene en $\mathbf{W} = \mathbf{I}$.

Si $\mathbf{W}$ está restringida (porque se le obligue a tener $\operatorname{rg}\mathbf{W} < n$), el mínimo deja de ser 0 y se alcanza en otro lugar (en general). Éste es el caso de $\mathbf{W} = \mathbf{AB}$, con $\operatorname{rg}\mathbf{W} \leq p$ si $\mathbf{A}$ y $\mathbf{B}$ son de órdenes $n \times p$ y $p \times n$, respectivamente, que se analiza en la sección 3.3.1.

A.2 Demostración de Baldi y Hornik sobre la superficie de error $E(\mathbf{A},\mathbf{B})$

No se dará la demostración detallada, que el lector puede encontrar en el artículo original de Baldi y Hornik [5] —que completa los resultados obtenidos por Bourlard y Kamp [9] sobre la autoasociación lineal, basándose en la descomposición en valores singulares—, pero sí se delineará. La demostración se basa en los cuatro hechos siguientes:

1. Para $\mathbf{A}_{n\times h}$ fija, $E(\mathbf{A},\mathbf{B})$ es convexa en los coeficientes de $\mathbf{B}$ y tiene su mínimo en cualquier $\mathbf{B}$ que satisfaga $\mathbf{A}^T\mathbf{ABXX}^T = \mathbf{A}^T\mathbf{XX}^T$. Si $\operatorname{rg}\mathbf{XX}^T = n$ y $\operatorname{rg}\mathbf{A} = h$, E es estrictamente convexa y tiene un único mínimo en $\mathbf{B} = \mathbf{A}^+ = (\mathbf{A}^T\mathbf{A})^{-1}\mathbf{A}$.

2. Para $\mathbf{B}_{h\times n}$ fija, $E(\mathbf{A},\mathbf{B})$ es convexa en los coeficientes de $\mathbf{A}$ y tiene su mínimo en cualquier $\mathbf{A}$ que satisfaga $\mathbf{ABXX}^T\mathbf{B}^T = \mathbf{XX}^T\mathbf{B}^T$. Si $\operatorname{rg}\mathbf{XX}^T = n$ y $\operatorname{rg}\mathbf{B} = h$, E es estrictamente convexa y tiene un único mínimo en $\mathbf{A} = \mathbf{B}^+ = \mathbf{XX}^T\mathbf{B}^T(\mathbf{BXX}^T\mathbf{B}^T)^{-1}$.

3. Supongamos $\operatorname{rg}\mathbf{XX}^T = n$. Si $\mathbf{A}$ y $\mathbf{B}$ definen un punto estacionario de E, es decir:

$$\nabla E(\mathbf{A},\mathbf{B}) = 0 \Leftrightarrow \frac{\partial E}{\partial a_{ij}} = \frac{\partial E}{\partial b_{ji}} = 0 \qquad 1 \leq i \leq n \quad 1 \leq j \leq h$$

entonces la matriz global $\mathbf{W} = \mathbf{AB}$ tiene la forma $\mathbf{W} = \Pi_{L(\mathbf{A})}$ y $\mathbf{A}$ satisface $\Pi_{L(\mathbf{A})}\mathbf{XX}^T = \Pi_{L(\mathbf{A})}\mathbf{XX}^T\Pi_{L(\mathbf{A})} = \mathbf{XX}^T\Pi_{L(\mathbf{A})}$. Si $\operatorname{rg}\mathbf{A} = h$, $\mathbf{A}$ y $\mathbf{B}$ definen un punto estacionario de E si y sólo si:

$$\left\{ \begin{array}{c} \Pi_{L(\mathbf{A})}\mathbf{XX}^T = \Pi_{L(\mathbf{A})}\mathbf{XX}^T\Pi_{L(\mathbf{A})} = \mathbf{XX}^T\Pi_{L(\mathbf{A})} \\ \mathbf{B} = \mathbf{A}^+ \end{array} \right\} \Leftrightarrow$$
$$\left\{ \begin{array}{c} \Pi_{L(\mathbf{A})}\mathbf{XX}^T = \Pi_{L(\mathbf{A})}\mathbf{XX}^T\Pi_{L(\mathbf{A})} = \mathbf{XX}^T\Pi_{L(\mathbf{A})} \\ \mathbf{W} = \Pi_{L(\mathbf{A})} \end{array} \right\} \tag{A.5}$$

4. Supongamos $\operatorname{rg} \mathbf{X}\mathbf{X}^T = n$ con $\lambda_1 > \cdots > \lambda_n$ autovalores distintos. Si por $\mathcal{I} = \{i_1, \ldots, i_h\}$, para $1 \leq i_1 < \cdots < i_h \leq n$, denotamos un conjunto de h índices y $\mathbf{U}_\mathcal{I} = (\mathbf{u}_{i_1}, \ldots, \mathbf{u}_{i_h})$, con $\{\mathbf{u}_{i_1}, \ldots, \mathbf{u}_{i_h}\}$ base ortonormal de autovectores de $\mathbf{X}\mathbf{X}^T$ asociados a los autovalores $\lambda_{i_1}, \ldots, \lambda_{i_h}$, respectivamente. Entonces $\mathbf{A}$ y $\mathbf{B}$ definen un punto estacionario de E sii existe una matriz $\mathbf{C}_{h\times h}$ invertible y $\mathbf{A} = \mathbf{U}_\mathcal{I}\mathbf{C}$ y $\mathbf{B} = \mathbf{C}^{-1}\mathbf{U}_\mathcal{I}^T$. En este caso, $\mathbf{W} = \mathbf{A}\mathbf{B} = \mathbf{A}\mathbf{A}^+ = \mathbf{U}_\mathcal{I}\mathbf{U}_\mathcal{I}^T = \Pi_{L(\mathbf{U}_\mathcal{I})}$ y $E(\mathbf{A}, \mathbf{B}) = \operatorname{tr} \mathbf{X}\mathbf{X}^T - \sum_{i\in\mathcal{I}} \lambda_i$, que es la ecuación (3.4) de la sección 3.3.1. La matriz $\mathbf{W}$ asociada al conjunto de índices $\{1, 2, \ldots, h\}$ es el único mínimo (global y local) de E. Los restantes $\binom{n}{h} - 1$ conjuntos de h índices corresponden a puntos de silla. Cualquier otro punto estacionario definido por matrices $\mathbf{A}$ y $\mathbf{B}$ de rango menor que h es también un punto de silla y equivale a una proyección ortogonal en un subespacio generado por $q < h$ autovectores.

La demostración de la ecuación (3.4) es sencilla. En efecto, si $\mathbf{W} = \mathbf{U}_\mathcal{I}\mathbf{U}_\mathcal{I}^T$ (que es simétrica, evidentemente), se tiene:

$$\begin{aligned} E = \|(\mathbf{I} - \mathbf{W})\mathbf{X}\|^2 = \operatorname{tr}(\mathbf{I} - \mathbf{W})\mathbf{X}\mathbf{X}^T(\mathbf{I} - \mathbf{W}) = \operatorname{tr}(\mathbf{I} - \mathbf{W})\mathbf{U}\mathbf{\Lambda}\mathbf{U}^T(\mathbf{I} - \mathbf{W}) = \\ \operatorname{tr}(\mathbf{U}\mathbf{\Lambda}\mathbf{U}^T - \mathbf{W}\mathbf{U}\mathbf{\Lambda}\mathbf{U}^T - \mathbf{U}\mathbf{\Lambda}\mathbf{U}^T\mathbf{W} + \mathbf{W}\mathbf{U}\mathbf{\Lambda}\mathbf{U}^T\mathbf{W}) \end{aligned} \quad \text{(A.6)}$$

donde se ha descompuesto $\mathbf{X}\mathbf{X}^T = \mathbf{U}\mathbf{\Lambda}\mathbf{U}^T$ en virtud del teorema espectral. Ahora, ya que (utilizando la propiedad (1.2) de la traza) $\operatorname{tr} \mathbf{X}\mathbf{X}^T = \operatorname{tr} \mathbf{U}\mathbf{\Lambda}\mathbf{U}^T = \operatorname{tr} \mathbf{\Lambda}\mathbf{U}^T\mathbf{U} = \operatorname{tr} \mathbf{\Lambda}$, $\operatorname{tr} \mathbf{W}\mathbf{U}\mathbf{\Lambda}\mathbf{U}^T = \operatorname{tr} \mathbf{U}\mathbf{\Lambda}\mathbf{U}^T\mathbf{W}$ y $\operatorname{tr} \mathbf{W}\mathbf{U}\mathbf{\Lambda}\mathbf{U}^T\mathbf{W} = \operatorname{tr} \mathbf{U}\mathbf{\Lambda}\mathbf{U}^T\mathbf{W}\mathbf{W} = \operatorname{tr} \mathbf{U}\mathbf{\Lambda}\mathbf{U}^T\mathbf{W}$, la expresión anterior se simplifica de la siguiente manera:

$$E = \operatorname{tr} \mathbf{\Lambda} - \operatorname{tr} \mathbf{U}\mathbf{\Lambda}\mathbf{U}^T\mathbf{W} = \operatorname{tr} \mathbf{X}\mathbf{X}^T - \sum_{i\in\mathcal{I}} \lambda_i = \|\mathbf{X}\|^2 - \sum_{i\in\mathcal{I}} \lambda_i = \sum_{i=1}^{n} \lambda_i - \sum_{i\in\mathcal{I}} \lambda_i$$

ya que $\operatorname{tr} \mathbf{U}\mathbf{\Lambda}\mathbf{U}^T\mathbf{W} = \operatorname{tr} \mathbf{U}\mathbf{\Lambda}\mathbf{U}^T\mathbf{U}_\mathcal{I}\mathbf{U}_\mathcal{I}^T = \operatorname{tr} \mathbf{\Lambda}\mathbf{U}^T\mathbf{U}_\mathcal{I}\mathbf{U}_\mathcal{I}^T\mathbf{U} = \sum_{i\in\mathcal{I}} \lambda_i$.

Baldi extiende en parte estos resultados a redes perceptrón de más de dos niveles en [3]. En estas redes, el cuello de botella viene marcado por el nivel (o los niveles) de menor anchura, es decir, de menos unidades. Las características principales de E no cambian: múltiples puntos de silla, ausencia de mínimos locales y un mínimo global que satisface $\mathbf{W} = \mathbf{U}_\mathcal{I}\mathbf{U}_\mathcal{I}^T$.

A.3 Sobre las medidas de error en espacios de dimensión muy grande

En este apéndice se pretende poner de manifiesto que las medidas de error que usamos acostumbradamente en espacios vectoriales de dimensión "manejable" (1, 2 ó 3) pueden dar resultados sorprendentes en espacios de dimensión mucho mayor. El efecto mostrado consiste en que dos vectores cuyos componentes difieren en un valor muy pequeño pueden ser muy parecidos en un espacio de dimensión pequeña pero muy distintos en uno de dimensión grande.

Sea el vector de dimensión n

$$\mathbf{v} = \left(1 - \binom{n-1}{2}\epsilon^2, \underbrace{\epsilon, \ldots, \epsilon}_{n-1}\right)^T$$

donde $\epsilon \ll 1$ y $\binom{n-1}{2}\epsilon^2 \ll 1$. La norma del vector $\mathbf{v}$ es 1 hasta orden 4, ya que:

$$\|\mathbf{v}\| = \sqrt{\left(1 - \binom{n-1}{2}\epsilon^2\right)^2 + (n-1)\epsilon^2} = \sqrt{1 + \left(\binom{n-1}{2}\epsilon^2\right)^2} = 1 + \mathcal{O}(\epsilon^4)$$

Sea $\{\mathbf{e}_i\}_{i=1,\ldots,n}$ la base canónica de $\mathbb{R}^n$, con $\|\mathbf{e}_i\| = 1\ \forall i$. Utilizaremos las dos medidas siguientes, análogas a las de la sección 4.2.2:

- Error relativo entre $\mathbf{e}_i$ y su proyección sobre $\mathbf{v}$, definido como

$$E_r(\mathbf{e}_i) = \frac{\|\mathbf{e}_i - \Pi_\mathbf{v}\mathbf{e}_i\|}{\|\mathbf{e}_i\|} \approx \|\mathbf{e}_i - (\mathbf{v}^T\mathbf{e}_i)\mathbf{v}\| \approx \begin{cases} \sqrt{n-1}\,\epsilon, & i = 1 \\ 1 - \frac{1}{2}\epsilon^2, & i \neq 1 \end{cases}$$

  Vemos que el error para $\mathbf{e}_1$ es proporcional a $\sqrt{n-1}$, luego para n grande dejará de ser despreciable (será mucho más grande que los componentes ϵ de $\mathbf{v}$).

- Norma de la proyección de $\mathbf{e}_i$ sobre $\mathbf{v}$, definida como

$$||\Pi_{\mathbf{v}}(\mathbf{e}_i)|| \approx ||(\mathbf{v}^T\mathbf{e}_i)\mathbf{v}|| = (\mathbf{v},\mathbf{e}_i) \approx \begin{cases} 1-\left(\frac{n-1}{2}\right)\epsilon^2, & i=1 \\ \epsilon, & i \neq 1 \end{cases}$$

  El caso es parecido al anterior, con un término proporcional a $n-1$ en el valor para $\mathbf{e}_1$.

La figura A.4 muestra esquemáticamente los segmentos error relativo y proyección.

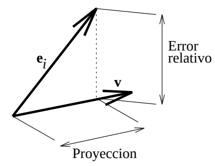

Figura A.4: Segmentos error relativo y proyección.

La tabla A.1 prueba lo anterior para algunos valores de n y ϵ. Vemos en ella que, a pesar de que $\mathbf{v}$ es prácticamente igual a $\mathbf{e}_1$, los valores de $E_r(\mathbf{e}_1)$ y $||\Pi_{\mathbf{v}}(\mathbf{e}_1)||$ no están tan cerca de 0 y 1, respectivamente, como uno podría esperar.

Tabla A.1: Módulos de los segmentos error relativo y proyección para algunos vectores seleccionados.

n	ϵ	$E_r(\mathbf{e}_1)$	$E_r(\mathbf{e}_i)$	$\|\|\Pi_{\mathbf{v}}(\mathbf{e}_1)\|\|$	$\|\|\Pi_{\mathbf{v}}(\mathbf{e}_i)\|\|$	$\mathbf{v}$
9	0.1	0.28	0.995	0.96	0.1	$(0.96, 0.1, \ldots, 0.1)^T$
101	0.1	1	0.995	0.5	0.1	$(0.5, 0.1, \ldots, 0.1)^T$
101	0.01	0.1	0.99995	0.995	0.01	$(0.995, 0.01, \ldots, 0.01)^T$

A.4 Elección de los pesos iniciales

Recordemos que, para la función de error E considerada en este trabajo, si bien el punto de inicio del aprendizaje es irrelevante porque sólo hay un mínimo, desde el punto de vista computacional existe la posibilidad de desbordamiento en alguno de los cálculos intermedios. Si esto ocurre, el simulador de RNAs dará —dependiendo de la máquina empleada[1]— resultados erróneos.

En este apéndice damos una cota para los pesos iniciales que garantiza, al menos en las primeras iteraciones de la retropropagación (dependiendo del valor de la constante de aprendizaje η), que no se producirá desbordamiento en las operaciones.

Llamaremos M al mayor número real representable en el ordenador. Como siempre, p será el número de patrones, n la dimensión de los mismos (igual al número de unidades en los niveles de entrada o salida) y h el número de unidades ocultas. Supondremos el caso peor: todas las entradas toman su valor máximo $g=1$ (pues el intervalo de variación de los píxeles está normalizado a $[0,1]$) y todos los pesos tienen igual valor w. Las operaciones relevantes son:

- Primer nivel (oculto): la salida de las unidades ocultas es $f\left(\sum_{i=1}^n gw\right) = f(nw)$.
- Segundo nivel (de salida): la salida de las unidades es $s = f\left(\sum_{i=1}^h f(nw)w\right) = f(hwf(nw))$.
- Error: $E = \left\langle n(g-s)^2 \right\rangle = pn(1-s)^2$.

donde f es la función de activación de las unidades. Según el tipo de f se presentan dos casos:

- f acotada: $\operatorname{im} f \subset [-a,a]$, con $a \ll M$; es el caso de la función sigmoide, por ejemplo. Entonces sólo podrán producir desbordamiento:

[1] Así, usando el simulador *SNNS* en una estación Sun S.P.I., que sigue el estándar IEEE P754 de coma flotante, el desbordamiento produce el resultado NaN (*Not a Number*), que inutiliza algunos valores pero el programa prosigue, permitiendo al usuario detener el proceso, salvar la red, etc. Sin embargo, en un PC/486 bajo Linux 1.2.8, se produce una excepción y *SNNS* muere.

- $|nw| \leq M \Rightarrow w \leq \frac{M}{n}$
- $|hwf(nw)| \leq M \Rightarrow w \leq \frac{M}{ha}$
- $E = pn(1-s)^2 \leq M$ no da ninguna condición para w porque s está acotada.

Por tanto, $w \leq \frac{M}{\max\{n,ha\}}$. Dado el valor[2] de M en comparación con los de a, n o h, prácticamente no hay limitación para w. El hecho clave es que la salida de las unidades está limitada en cada nivel.

- f es la identidad (éste es nuestro caso en este trabajo, porque tratamos con redes lineales). Ahora hay que tener en cuenta que, aunque al principio los pesos sean pequeños, en pasos subsiguientes (dependiendo del valor de η) pueden dispararse, porque f no los controla. La condición que interesa fijar es que la salida s de cada unidad sea *parecida* a la salida deseada, 1 en el caso peor. Podemos poner, por ejemplo, que $|1-s| \lesssim 1 \Rightarrow s \lesssim 2 \Rightarrow w \lesssim \sqrt{\frac{2}{nh}}$. Esta condición no depende de M ni de p. Para $n = 1440$ y $h = 20$ da $|w| \lesssim 0,008$ y para $n = 640$ y $h = 10$ da $|w| \lesssim 0,02$. En las simulaciones se usaron los valores respectivos de $0,001$ y $0,01$, con buenos resultados; además, se comprobó que valores un orden de magnitud superiores producían desbordamiento en las primeras iteraciones.

  Por otro lado, si es de esperar que la base obtenida por la red sea aproximadamente ortonormal (hemos visto que *tiende* a serlo, dentro de ciertos márgenes), cada vector tendrá componentes del orden $\frac{1}{\sqrt{n}}$ en valor absoluto (así ocurre, de hecho), valor del mismo orden que la cota anterior (para los valores de h utilizados).

[2]En precisión simple (4 bytes) es aproximadamente 10^{38}; en precisión doble (8 bytes) 10^{308}.

Apéndice B

Captación y preparación inicial de los datos

B.1 Captación de las imágenes

La figura B.1 muestra la disposición de los distintos elementos para la captación de las imágenes. Se empleó una cámara de tonos de grises, modelo Kappa CF 4, con enfoque y apertura del diafragma regulables (focal 16 mm, ∅ 25.5 mm y número F 1.4–16). Dicha cámara envía las imágenes, en tiempo pseudorreal —limitado por la velocidad del ordenador—, a un PC 486 (a través de una tarjeta digitalizadora de vídeo con conexión RCA). Las imágenes pueden mostrarse en una ventana (ventana *preview*) con el programa *Imager for VIDEO NT* v1.52 Jan 15th 1995, de VITEC MULTIMEDIA. en entorno Windows. Cuando se desea tomar una imagen, se acciona con el ratón el botón *still image* y el programa congela la imagen en una ventana. Pueden capturarse de esta manera tantas imágenes como se desee, aunque conviene no tomar más de 4 ó 5 de una vez, porque si no el proceso de grabación en disco se ralentiza notablemente. Finalmente, se almacenan las imágenes deseadas en uno de los formatos ofrecidos por el programa.

Para cada sujeto, primero se tomaban 6 fotos de frente y a continuación, haciéndole sentarse de lado, 6 de perfil. Siempre se usó el perfil izquierdo. Se contó con la colaboración de 17 individuos, todos ellos estudiantes o profesores de la Facultad, y también amigos del autor de este trabajo; teniéndose, pues, un total de 102 imágenes frontales de la cara y otras 102 de perfil.

Durante la toma de imágenes, el sujeto se sentaba frente a la cámara, cuya altura había que regular en cada caso para adaptarla a la del sujeto —de modo que la cara apareciera en el centro de la imagen digitalizada, más o menos—, y a aproximadamente metro y medio de distancia de la misma. En ese momento se le tomaban las 12 fotos una detrás de otra, sin interrupción (excepto para irlas grabando de 4 en 4, para no agotar la memoria del PC). Durante la toma se permitían —es más, eran deseables— ligeros cambios en la expresión y la orientación. Las condiciones de iluminación se mantuvieron aproximadamente uniformes para todos los casos, aunque dado que el proceso de captación se prolongó durante varios días, antes de iniciar cada sesión había que reacondicionar el montaje de los distintos elementos, lo que

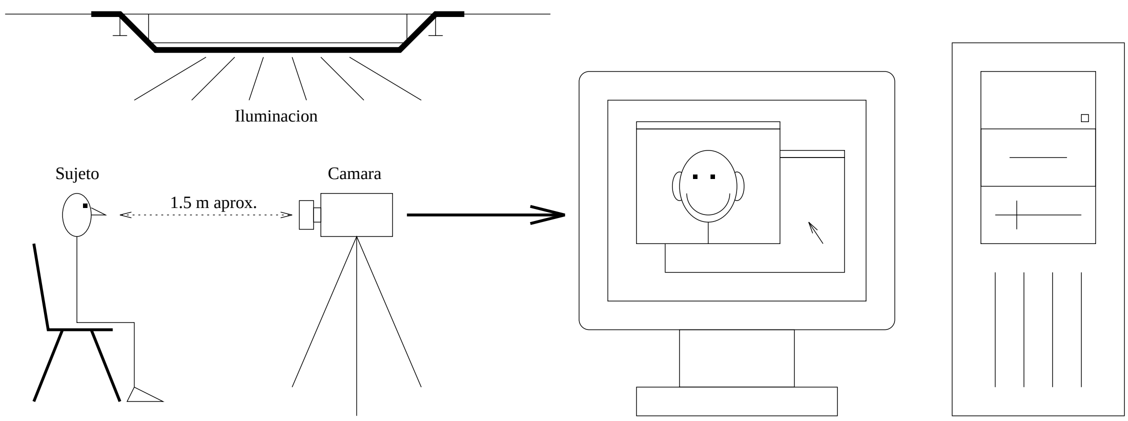

Figura B.1: Montaje para la captación de las imágenes.

introducía ligeras variaciones de un día para otro. Se intentó tener un fondo homogéneo, que disminuya el número de grados de libertad de la imagen, y que —caso de ser necesario— permitiera una fácil segmentación para extraer la cara de la imagen, eliminado el fondo. Para ello, se tomó como fondo una pared de yeso, que producía un color blanco mate. Además, siempre se situó al sujeto un poco retirado de la pared para evitar sombras en la misma.

La iluminación provenía exclusivamente de tubos fluorescentes, dado que la estancia (el Laboratorio de Tecnología Fotónica de la Facultad) carecía de ventanas. Esta iluminación, al incidir directamente sobre superficies lisas —como la frente— producía reflejos muy intensos que saturaban inmediatamente el rango dinámico de la cámara, bastante pobre. Con objeto de lograr un iluminación difusa que evitara estos reflejos directos se apantallaron los tubos fluorescentes cercanos; para ello se probaron distintos materiales, que no dejaran pasar ni demasiada luz ni que oscurecieran excesivamente la imagen. El resultado óptimo se obtuvo con un papel de tul grueso que venía como embalaje para un *plotter* que había en el laboratorio; se fijó este envoltorio por medio de papel adhesivo y chinchetas al techo, tapando los mencionados tubos. Esto consiguió reducir en buena medida los nefastos reflejos, si bien a costa de oscurecer considerablemente la imagen que aparecía en pantalla. Esto no supuso ningún problema, tras realzar el brillo de las imágenes con el programa *xv*.

B.2 Procesamiento de las imágenes captadas

Se decidió almacenar las fotos con la mayor resolución que ofrecía el programa, 384×288 con 256 tonos de grises (8 bits/píxel), y guardarlos en formato JPEG con un nivel de compresión del 75%, lo que aseguraba una buena calidad de imagen sin un consumo excesivo de espacio en disco (20–30K por imagen). Estas imágenes se guardaron en formato JPEG de tonos de gris, otra vez con un 75% de compresión, con el comando[1] (en Linux):

```
djpeg -targa * | cjpeg -gr -op *
```

con lo que su tamaño se redujo a 5–8K. Estas imágenes son las que aparecen en el fichero `/jpg/jpg.tgz`.

A continuación, empleando el programa *xv* de John Bradley (versión 3.10), se procesaron manualmente todas las imágenes, incrementando su brillo mediante una transformación gamma de parámetro 1.5 aproximadamente, dado que el proceso de captación hacía que salieran muy oscuras por lo general. En algún caso particular se acentuó también el contraste. Finalmente, y de nuevo de manera manual con *xv*, se recortó la parte interesante de las imágenes anteriores (es decir, dejando sólo la parte comprendida entre la barbilla y la coronilla, y entre las orejas en las fotos de frente y la nariz y el extremo occipital de la cabeza en las de perfil) y se almacenaron estas imágenes en formato PGM, listo para ser procesado por el conjunto de programas `pbmplus`. Las resoluciones de estas imágenes varían ligeramente (debido a ligeras variaciones en el tamaño de la cabeza del sujeto y de su distancia a la cámara), pero se obligó (dejando un poco de espacio a los lados o arriba y abajo, en caso de ser necesario) a que todas mantuvieran una relación altura:anchura igual a 4/3 (que es el aspecto que presenta, en promedio, una cara). Estas imágenes son las que aparecen en el fichero `/caras/caras.tgz`.

Para obtener las imágenes de la oreja, se recortó manualmente la misma y su región circundante de cada una de las imágenes de perfil. Después, en cada caso hubo que rotar dicha imagen recortada un cierto número de grados para que quedara en posición recta (definida como aquélla que presenta la oreja de la figura 1.2, aproximadamente). Después se volvió a recortar para eliminar partes no interesantes y obtener un rectángulo lo más pequeño posible y de razón altura:anchura igual a 1.6 (que es el aspecto que presenta, en promedio, una oreja) que envolviera completamente a la oreja. El resultado se guardó en el fichero `/orejas/pgm/orejapgm.tgz`. Para realizar estas operaciones se usó nuevamente *xv*.

Tras este proceso, resulta ya muy fácil, usando unos *shellscripts* creados *ad hoc* por el autor, obtener un fichero de patrones en el formato de *SNNS*, listo para ser utilizado para el entrenamiento o prueba de una RNA, o bien pasar esas imágenes a *Mathematica* (por ejemplo, para calcular su media, centrarlas, y pasarlas desde *Mathematica* a *SNNS* con la función `MatToPat[]`; o bien para, desde *Mathematica*, generar los *holones* con la función `MatToPGM[]`). El *shellscript* que pasa un conjunto de ficheros PGM a un fichero `.pat` de *SNNS* se da en la secciónD.4. En esa misma sección se da también, entre otros, un listado del *shellscript* `pgmtomat`, que pasa un conjunto de ficheros PGM a una matriz de *Mathematica*.

Todo el procesamiento de las imágenes posterior a su captación tuvo lugar en un PC 486/66MHz con 16M RAM y sistema operativo Linux, empleando los programas citados: *xv* y `pbmplus`.

[1] `cjpeg` y `djpeg` son dos programas creados por el *Independent JPEG Group*, cuya función es crear un fichero en formato JPEG a partir de otro en otro formato y al revés, respectivamente.

B.3 Bases de imágenes faciales existentes

Gracias al apoyo de la comunidad de usuarios de Usenet, el autor ha podido conseguir una serie de datos y direcciones sobre bases de datos de dominio público de imágenes faciales y similares. Parte de esta información puede encontrarse en el FAQ (*Frequently Asked Questions*) [42] del grupo `comp.ai.neural-nets`. Asimismo, existe desde hace poco una página en la *World Wide Web* (WWW) dedicada íntegramente al procesamiento facial, la *Face Recognition Home Page* (administrada por P. Kruizinga [40]), que contiene gran cantidad de punteros a grupos de investigación (y sus correspondientes páginas de WWW), bases de datos, artículos técnicos, programas (comerciales y de dominio público), anuncios de congresos, bibliografía, etc.; su dirección es `http://www.cs.rug.nl/~peterkr/FACE/face.html`.

Actualmente están disponibles, al menos, las siguientes bases de datos:

- Olivetti Research Ltd. (ORL) face database: utilizada por Samaria [49]. Consta de 400 imágenes frontales de la cara de 40 sujetos distintos (10 de cada uno), empleados de Olivetti y estudiantes de la Universidad de Cambridge, de edades entre 18 y 81 años (la mayoría entre 20 y 35); 4 mujeres y 36 hombres. A cada individuo se le pidió mirar de frente a la cámara, sin restricción en la expresión. Sólo se toleraron un desplazamiento lateral e inclinación limitados. Las fotos se tomaron en momentos distintos y con condiciones de iluminación diferentes, pero el fondo se mantuvo siempre oscuro. Algunos sujetos aparecen con y sin gafas. Las imágenes fueron recortadas y escaladas manualmente a una resolución de 92×112 con 256 tonos de gris. La base de datos es accesible por ftp anónimo a `ftp.cam-orl.co.uk` o en la WWW como `http://www.cam-orl.co.uk/facedatabase.html`.

- MIT face database: establecida por Turk y Pentland [55], del *MIT Media Lab, Vision and Modeling Group* (`turk@teleos.com`). Contiene 2592 imágenes de 16 personas. Para cada persona hay 27 imágenes tomadas bajo distintas condiciones de iluminación, tamaño y posición a partir de una secuencia de vídeo. Está en `ftp://whitechapel.media.mit.edu/`, directorios `/pub/eigenfaces` y `/pub/images`.

- USENIX FaceSaver database: contiene imágenes de los asistentes a las conferencias Usenix celebradas hasta ahora. Son imágenes de condiciones no demasiado homogéneas de 5592 individuos distintos (salvo unos pocos duplicados), de 96×128 y 256 tonos de gris en la mayoría de los casos, almacenadas en el formato *FaceSaver*, que consiste en un fichero ASCII que contiene, además de los valores de los píxeles, información relativa al sujeto que sea (nombre, dirección, etc.). Este formato es legible por los programas del paquete `pbmplus`, por ejemplo. La base está disponible en UUNET: `ftp.uu.net://published/usenix/faces` (sitio original) y en diferentes *mirrors*: `ftp://src.doc.ic.ac.uk:/pub/packages/faces` (servidor del Imperial College, en Londres) y `ftp://cs.indiana.edu:/pub/faces` (Indiana).

- Una base de datos creada por Baluja y Rowley, que se dedican sobre todo al problema de detección y localización de caras. Son 42 imágenes en tonos de gris, en formato GIF, tomadas de diversas fuentes: cámaras CCD, fotografías digitalizadas (de periódicos, revistas, fotos propias ...) y dibujos a mano alzada. Las condiciones de captación son variadas (en cuanto a iluminación, contraste, calidad, etc.). En `http://www.ius.cs.cmu.edu/IUS/dylan_usr0/har/faces/test/index.html` pueden encontrarse estas imágenes; además, a través de `http://www.cs.cmu.edu/~baluja` se tiene acceso a una demostración de su sistema.

- Imagery of the VISION-LIST-ARCHIVE: es un extracto de la base del MIT, con 68 imágenes. Está en `ftp://ftp.teleos.com:/VISION-LIST-ARCHIVE/IMAGERY/FACES` (Teleos Research, Palo Alto, CA).

- De acuerdo con Chellappa *et al.* [10], el NIST (*National Institute of Standards and Technology*) ha hecho pública recientemente una base de fotografías de fichas policiales (*mugshots*) con un total de 3248 imágenes. Para más detalles, contactar con `craig@magi.ncsl.nist.gov`.

- Picons database: consta de imágenes de muy baja resolución en blanco y negro (1 bit por píxel), destinadas a ser usadas por programas de correo electrónico: cada imagen o icono va asociado a la dirección de correo de una persona; cuando se recibe correo de una persona que tiene una imagen asociada, ésta aparece en pantalla. Su mala calidad hace que no sean de mucha utilidad. Se encuentra en `ftp://ftp.cs.indiana.edu:/pub/faces/picons/db` y también tiene una página en la WWW: `http://www.cs.indiana.edu/ftp/faces/index.html`.

Muchos de los sitios anteriores ofrecen, además, otras informaciones de interés para el procesamiento de caras, tales como artículos, etc. Para más detalles, el lector debe acceder a las direcciones indicadas y leer los ficheros `README` correspondientes.

La obtención de imágenes de perfil (especialmente de múltiples imágenes por individuo) es más difícil. El autor tuvo que tomarlas personalmente, como se indicó en la sección B.1.

Dado que el autor piensa seguir trabajando en este campo, agradecerá cualquier información al respecto que le sea enviada. Su dirección de correo electrónico es `mcarreir@ls.fi.upm.es`.

Apéndice C

Estructura de directorios

La figura C.1 describe el árbol de directorios en el que se distribuyen las distintas redes, patrones, etc. con objeto de facilitar una posible continuación del trabajo.

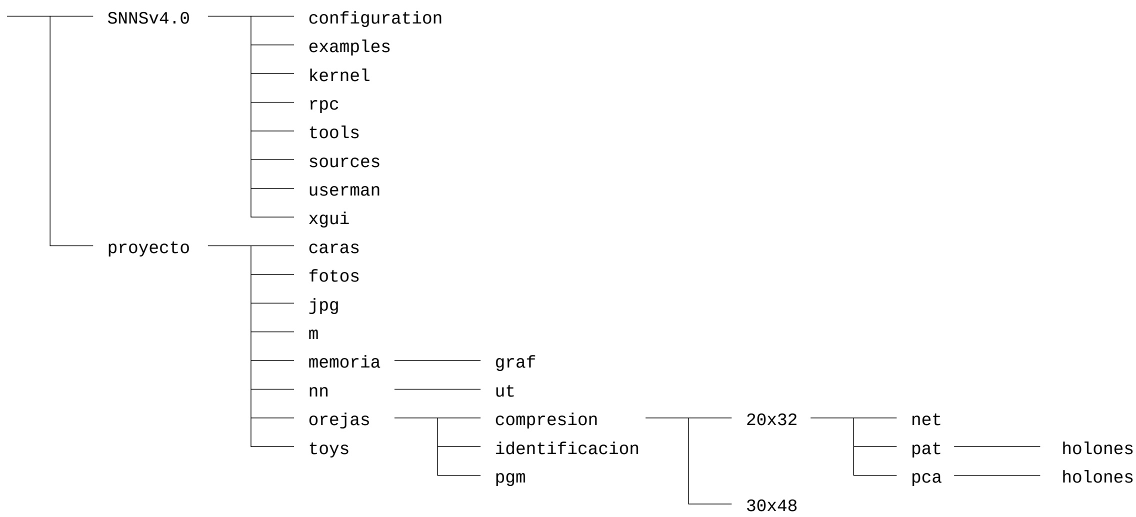

Figura C.1: Estructura de directorios que contienen las redes, patrones, etc.

Nótese que la estructura real puede diferir ligeramente: a veces de un directorio cuelgan más de los que aparecen en la figura, y otras el contenido del mismo está archivado con `tar` y comprimido con `gzip` (extensión `.tgz`). Para mantener la figura clara no se han indicado estos detalles, que por otro lado son fáciles de intuir sobre el terreno.

Los directorios y sus contenidos son:

- `SNNSv4.0`: contiene todo el simulador *SNNS* versión 4.0. La estructura indicada se crea al instalar el programa. En este directorio hay un enlace (*link*) `snns -> ./xgui/bin/pc_linux/xgui*` al ejecutable verdadero. Para más detalles, consúltese el manual de *SNNS* [62].
- Dentro del directorio `proyecto` se encuentran los siguientes:
  - `caras`: contiene el fichero `caraspgm.tgz` con las imágenes faciales captadas por el autor, en formato PGM.
  - `fotos`: contiene el fichero `fotospgm.tgz` con las imágenes no de orejas, empleadas para demostrar la capacidad de reconocimiento de la red Ξ con regla de rechazo.
  - `jpg`: contiene el fichero `jpg.tgz` con las imágenes faciales captadas por el autor, de frente y de perfil, en formato JPEG.
  - `m`: contiene los programas de *Mathematica* de la sección D.4.
  - `memoria`: contiene el texto fuente de la memoria del proyecto, escrita utilizando LaTeX 2_ε y $\mathcal{A}\mathcal{M}\mathcal{S}$-LaTeX 1.2 (OJO: no funciona con el LaTeX 2.09 de Leslie Lamport). Las figuras y gráficos (creados con `xfig`, *gnuplot* y *xv*) están en el subdirectorio `graf`.

- `nn`: contiene varios subdirectorios con información sobre RNAs y proesamiento de imágenes faciales, principalmente (referencias bibliográficas adicionales, correspondencia privada, artículos de **`comp.ai.neural-nets`** y otros *newsgroups*, extractos de la lista de distribución de *SNNS*, etc.). Particularmente importante es el subdirectorio `ut`, que contiene los *shellscripts* creados por el autor para transformar formatos entre *SNNS*, *Mathematica* y PGM.
- `orejas`: se subdivide en:
  - `compresion`: contiene un subdirectorio para cada resolución de red Ξ o de compresión empleada (20×32, 30×48). A su vez, cada uno contiene los ficheros de *SNNS* con la definición de la red (`net`), los conjuntos de patrones TS, TTS, VTS1, VTS2 y AS en el formato de *SNNS* (`pat`) y los resultados del análisis espectral de la matriz $\mathbf{XX}^T$ de los datos del TS (`pca`), junto con los *holones* (representaciones pictóricas de los vectores imagen) correspondientes a cada caso.
  - `identificacion`: actualmente vacío. Reservado para almacenar todo lo relativo a la segunda fase del sistema, la de identificación a partir de las características obtenidas por la red de compresión.
  - `pgm`: contiene el fichero `orejapgm.tgz`, con las imágenes de la oreja izquierda obtenidas a partir de las fotos de perfil.
- `toys`: contiene redes Ξ muy pequeñas (de $n = 3$ y $n = 9$) y sus ficheros de *Mathematica* asociados. Vienen bien para hacer pruebas rápidas.

Apéndice D

Programas utilizados

Este apéndice describe los programas empleados en la relización de este trabajo. Entre ellos se encuentran programas externos, empleados principalmente para simular las redes, procesar matemáticamente sus resultados y manejar gráficos, y programas de apoyo realizados por el autor.

D.1 Programa de cálculo numérico y simbólico: *Mathematica*

El programa *Mathematica*, creado por Stephen Wolfram y sus colaboradores, es un sistema interactivo de cálculo potente y versátil, que cuenta con multitud de usuarios desde su aparición en 1988. Está disponible para muchos sistemas; la versión utilizada en este trabajo es la 2.0, para MS-DOS.

Es capaz de trabajar con aritmética de precisión ilimitada y de hacer cálculos tanto numéricos (integrales numéricas, inversión de matrices, transformadas de Fourier, ajustes, interpolación, minimización, programación lineal, solución de ecuaciones, ecuaciones diferenciales, etc.) como simbólicos (expansión, factorización y simplificación de polinomios y expresiones racionales, soluciones algebraicas de polinomios y de sistemas de ecuaciones lineales, cálculo de primitivas, derivación simbólica, manipulación de series, límites, etc.) y gráficos (en dos y tres dimensiones, de líneas de nivel, de densidad, etc.). Se puede utilizar como calculadora numérica y simbólica, como herramienta de visualización de funciones y datos y como entorno de modelación y análisis de datos, entre otros.

Mathematica cuenta con un lenguaje de programación especialmente apto para representar expresiones matemáticas de todo tipo. Este lenguaje presenta rasgos de lenguaje procedural (con estructuras de bloques: secuencial, condicional, iterativa y recursiva), funcional (basado en listas, con funciones puras y operadores funcionales) y basado en reglas (gracias a su capacidad para realizar *pattern matching*).

Incluye una elevada cantidad de funciones predefinidas, que permiten crear otras con ayuda del lenguaje, como se muestra en los programas de abajo, usados por el autor para diversas operaciones de pre- y postprocesamiento de los patrones y bases de las RNAs discutidas en el cuerpo del texto.

Los programas del apéndice D.4 contienen definiciones útiles para la media, varianza, norma, proyección ortogonal, ortogonalización de Gram-Schmidt y salida en varios formatos, entre otras, así como para el análisis de componentes principales de la matriz de covarianzas y la obtención de gráficos y tablas sobre las redes Ξ simuladas. También se dan *shellscripts* para pasar un fichero de red de *SNNS* a una matriz de *Mathematica*.

D.2 Simulador de redes de neuronas artificiales: *SNNS*

Si bien el paralelismo inherente a las RNAs hace deseable el uso de ordenadores multiprocesador (sobre todo para redes de gran tamaño), la mayoría de los simuladores existentes funcionan sobre máquinas tradicionales, probablemente por razones de complejidad del diseño[1]. Para entrenar, probar y analizar las redes descritas en este trabajo se decidió utilizar el *Sttutgart Neural Network Simulator*, *SNNS*.

El simulador de RNAs *SNNS*, desarrollado por el Instituto de Sistemas Paralelos y Distribuidos de Altas Prestaciones de la Universidad de Stuttgart (Institut für Parallele und Verteilte Höchstleistungsrechner), es uno de los simuladores que cuenta con más aceptación por los usuarios en la actualidad, dadas sus favorables características. La versión 1.3 de *SNNS* recibió un premio del Ministerio Alemán de Educación y Ciencia en 1991.

[1] Sin embargo, la última versión del *SNNS*, la 4.0, incluye un módulo RPC (*Remote Procedure Call*) que permite trabajar sobre un grupo de estaciones Sun a la vez.

Las características principales del *SNNS* son:

- No es de dominio público, pero su uso está regulado por una licencia muy similar a la *GNU General Public License* de la *Free Software Foundation*. Está disponible por FTP anónimo en la red Internet.
- Existe una lista de distribución (*mailing list*) para ayudar a la comunicación entre los usuarios y los diseñadores del *SNNS*.
- Posee un interfaz gráfico de usuario sobre X Windows potente y fácil de usar, que facilita mucho la creación y modificación de una red, así como su entrenamiento y la presentación de resultados.
- Permite introducir nuevas funciones de aprendizaje, activación, etc. creadas por el usuario. Además se entrega todo el código fuente (en C).
- Lleva multitud de estructuras de red y de funciones (de activación, inicialización de pesos, etc.) incluidas por defecto. Entre las arquitecturas disponibles están los perceptrones multicapa y las memorias autoasociativas.
- El entrenamiento de una red es rápido.
- Dispone de herramientas gráficas de análisis: Analyzer, Graph, etc.
- La ventana de control permite fijar valores para diversos parámetros: el factor de aprendizaje η, el número de ciclos, las funciones de aprendizaje, actualización, salida e inicialización de pesos (y sus parámetros asociados), período de validación, así como varias opciones sobre el conjunto de patrones que se va a emplear, qué patrones, etc.
- Está disponible para un gran número de sistemas operativos, entre los que destacan Unix y Linux, por lo que puede funcionar en un PC 386 o superior con una cantidad aceptable de RAM (8M).
- Permite generar, con la utilidad `snns2c`, código fuente en C para incluir la funcionalidad de una red ya entrenada en un programa de usuario.
- Incluye una versión *batch*, `snnsbat`, que entrena una red en *background* tomando los parámetros de un fichero de configuración; es útil para redes muy grandes, que tarden mucho en entrenarse, y convenga dejarlas entrenándose por la noche o un fin de semana.

Existe un test, hecho por Lutzy y Dengel [36] —un poco antiguo ya, pues es de 1993— en el que, junto al *SNNS* v1.3 y v2.0, se comparan otros simuladores de RNAs: PlaNet v5.6, Pygmalion 2.0 y Rochester Connectionist Simulator 4.2. Según este test, *SNNS* da los mejores resultados de todos los simuladores analizados en prácticamente todos los aspectos de la comparación: velocidad, facilidad de uso, funciones disponibles, herramientas adicionales, interfaz, manejo de patrones, edición de redes, documentación disponible y apoyo técnico. La última versión disponible, la 4.0, incorpora mejoras sensibles a las anteriores.

En un PC 486/66MHz con 16M de RAM se tardó alrededor de 15 minutos en instalar el paquete completo, haciendo uso del programa de instalación que trae (que se limita a compilar el código fuente y crear los objetos y ejecutables necesarios).

SNNS Consta de:

- Un núcleo, que contiene las funciones necesarias para el diseño y simulación de una red, la gestión de memoria y otras operaciones.
- Un interfaz gráfico de usuario, que conecta el núcleo con el sistema de ventanas X para permitir la interacción con el usuario y la salida gráfica de resultados.

Desde su primera versión, *SNNS* ha ido incorporando gran cantidad de funciones para el aprendizaje, etc., lo que permite poner a punto una red más o menos típica enseguida. Sin embargo, y dado el ritmo al que surgen nuevos algoritmos, es cierto que faltan algunas. Una vía adecuada para preguntar si un algoritmo dado está o no implementado en *SNNS*, o si se va a implementar, o si algún usuario se lo ha hecho para sí mismo, es la lista de distribución.

SNNS puede obtenerse por ftp anónimo en `ftp.informatik.uni-stuttgart.de:/pub/SNNS`. Allí se encuentra el código fuente del mismo y su manual en PostScript. También dispone de una página en WWW: `http://vasarely.informatik.uni-stuttgart.de/snns/snns.html`. El manual [62] y los distintos ficheros `README` dan más información sobre la versión particular de que se trate.

D.2.1 Rendimiento de *SNNS* en diversos ordenadores

SNNS trae un pequeño programa llamado `netperf` que sus autores usan como test de velocidad del algoritmo de retropropagación, para calcular de manera aproximada la velocidad de propagación hacia adelante y hacia atrás (retropropagación de errores δ_i) para un ordenador dado. El programa ejecuta un número dado de iteraciones sobre la red que se le indique; como base se toma una red de ejemplo cuyo fichero fuente se distribuye con *SNNS*, la conocida `nettalk`. Esto nos permite comparar las velocidades relativas de los distintos equipos que tuvimos a nuestra disposición (en el Laboratorio de Metodologías y Lenguajes de la Facultad y el propio PC del autor, `alcorcon`); se resumen en la tabla D.1. Se midió el tiempo empleado por `netperf` Rev 2.1, usando `nettalk.net` con el primer patrón de `nettalk.pat`.

Tabla D.1: Rendimiento de *SNNS* 3.3, medido con el *benchmark* `netperf` Rev 2.1.

Máquina	Prop (CPS $\times 10^5$)	Backprop (WUPS $\times 10^5$)
`lml` (Sun S.P.I)	21.810	8.7238
`gedeon` (Sun Sparcstation 2)	9.7562	4.2267
`alcorcon` (PC 486DX2/66Mhz, 16M RAM)	8.3780	2.9453
`esther` (Sun Sparcstation IPC)	5.6466	2.4463
`judith` (Sun Sparcstation IPC)	5.6082	2.4319
`moises` (Sun Sparcstation 330)	5.4059	2.3157
`trucha` (PC 486DX/20MHz, 8M RAM y 13M swap)	2.0977	0.94174

Las unidades empleadas son “conexiones propagadas hacia adelante por segundo” (CPS, *Connections Per Second*) y “pesos actualizados por segundo en la retropropagación” (WUPS, *Weight Updates Per Second*).

La tabla anterior se obtuvo usando *SNNS* 3.3. Para *SNNS* 4.0 no hay prácticamente ninguna diferencia (las mejoras introducidas son principalmente de otro tipo).

La estación Sun S.P.I dispone en realidad de 4 procesadores. Las cifras anteriores son válidas para uno solo de ellos (al lanzar un nuevo proceso la *shell* mediante la función `fork()`, éste no se distribuye entre los 4 procesadores, sino que se carga en uno solamente. Es en este ordenador en el que se han entrenado la práctica totalidad de las redes usadas en este trabajo.

En la tabla D.2 damos el tiempo en segundos por iteración y por patrón, para la retropropagación y el algoritmo *quickprop* usando *SNNS* versión 4.0, y para las distintas combinaciones de redes de compresión usadas en el texto principal, en cuanto a número de unidades de entrada (dependiente de la resolución de la imagen) y número de unidades ocultas. El ordenador empleado es un PC 486DX2/66Mhz con 16M de RAM, que de acuerdo con Linux 1.2.8 trabaja a una velocidad aproximada de 33.22 BogoMips.

El número de conexiones es $2hn$, si h es el número de unidades ocultas y n el de unidades de entrada.

Tabla D.2: Tiempo en segundos por iteración y por patrón para diversos tamaños de red de compresión, con los dos algoritmos de aprendizaje usados (retropropagación y *quickprop*), para *SNNS* v4.0 sobre un PC 486DX2/66Mhz con 16M de RAM y Linux 1.2.8.

n	h	$2hn$	$t_{\text{backprop}}('')$	$t_{\text{quickprop}}('')$
$20 \times 32 = 640$	1	1280	0.0088	0.0086
	5	6400	0.025	0.025
	10	12800	0.048	0.047
$30 \times 48 = 1440$	1	2880	0.023	0.023
	10	28800	0.12	0.11
	20	57600	0.22	0.21

Estos tiempos se tomaron directamente usando un reloj y ejecutando un número elevado de iteraciones con los 85 patrones del conjunto TS, con lo que en ellos va también incluido el tiempo consumido por otros procesos que están en *background* (`xclock`, `crond`, etc.). Por ello, estos valores no son exactos del todo; sin embargo, son mucho más representativos de lo que el usuario se va a encontrar en el caso práctico de uso de *SNNS*.

D.2.2 Formatos `.net` y `.pat` de *SNNS*

SNNS guarda la definición de una red en un fichero de texto con extensión **`.net`** como el siguiente (corresponde al típico perceptrón que resuelve el problema del XOR):

```
SNNS network definition file V1.4-3D
generated at Mon Apr 25 15:58:28 1994

network name : xor
source files :
no. of units : 4
no. of connections : 5
no. of unit types : 2
no. of site types : 2


learning function : Quickprop
update function   : Topological_Order


unit default section :

act      | bias     | st | subnet | layer | act func     | out func
---------|----------|----|--------|-------|--------------|-------------
 0.00000 |  0.00000 | h  |      0 |     1 | Act_Logistic | Out_Identity
---------|----------|----|--------|-------|--------------|-------------


unit definition section :

no. | typeName | unitName | act      | bias     | st | position | act func | out func | sites
----|----------|----------|----------|----------|----|----------|----------|----------|-------
  1 |          | in_1     |  1.00000 |  0.00000 | i  | 3,5,0    |          |          |
  2 |          | in_2     |  1.00000 |  0.00000 | i  | 9,5,0    |          |          |
  3 |          | hidden   |  0.04728 | -3.08885 | h  | 6,3,0    |          |          |
  4 |          | result   |  0.10377 | -2.54932 | o  | 6,0,0    |          |          |
----|----------|----------|----------|----------|----|----------|----------|----------|-------


connection definition section :

target | site    | source:weight
-------|---------|------------------------------------------------------
     3 |         | 2:-4.83963, 1: 4.92521
     4 |         | 3:11.11523, 2: 4.53903, 1:-4.67122
-------|---------|------------------------------------------------------
```

El formato es bastante autoexplicativo. No obstante, no es necesario conocerlo para crear, entrenar y probar una red. Al autor le fue necesario para poder extraer los pesos (**`connection definition section`**) y pasárselos a *Mathematica*, por medio del *shellscript* **`nettomat`**.

El fichero **`.pat`** que contiene los patrones de la red tiene el siguiente aspecto:

```
SNNS pattern definition file V3.2
generated at Mon Apr 25 15:58:23 1994


No. of patterns : 4
No. of input units : 2
No. of output units : 1

# Input pattern 1:
0 0
# Output pattern 1:
0
# Input pattern 2:
0 1
```

```
# Output pattern 2:
1
# Input pattern 3:
1 0
# Output pattern 3:
1
# Input pattern 4:
1 1
# Output pattern 4:
0
```

Cada patrón lleva asociado el vector de entrada y el de salida. *SNNS* proporciona unos programitas para ayudar a automatizar la construcción de ficheros de patrones a partir de imágenes: `mkhead` crea la cabecera del mismo (dados el número de patrones, de unidades en entrada y de unidades de salida); `mkpat` crea un patrón a partir de un fichero binario que contenga los bytes de la imagen ordenados por filas y columnas (dada la anchura y la altura de la imagen, y suponiendo que es de 256 tonos de gris, es decir, 1 byte/píxel). `mkpat` normaliza el intervalo de grises $[0, 255]$ al $[0, 1]$.

Con ayuda del *shellscript* `pgmtopat` es muy fácil crear un fichero completo `.pat` a partir de un número dado de imágenes en formato PGM.

D.3 Programas de tratamiento de imágenes

Para el procesamiento de las imágenes captadas, se han empleado los siguientes programas (todos ellos disponibles en la red Internet):

- *xv* 3.10, de John Bradley. *xv* es un programa interactivo de X Windows, que incorpora las funciones típicas de procesamiento de imágenes de mapa de bits (*raster*): transformaciones geométricas (escala, rotación, recorte, etc.), modificación de la paleta de colores, filtros, etc. Permite leer y grabar las imágenes en varios formatos, particularmente JPEG, PGM y PostScript. Las figuras de los *holones* y otros mapas de bits han sido pasadas de JPEG o PGM a PostScript con *xv* y luego insertadas en el texto en LaTeX 2_ε con el comando `\includegraphics` del paquete `graphicx`.

  xv puede conseguirse por ftp anónimo en (entre otros sitios) `export.lcs.mit.edu:/contrib/xv*` y `ftp.cis.upenn.edu:/pub/xv*`.

- La colección de programas `pbmplus`, de Jef Poskanzer (versión de 1991). Incluye una serie de programas separados, no interactivos, cada uno de los cuales realiza una operación gráfica sencilla: pasar de un formato a otro, reducir la imagen, recortarla, etc. Reciben los parámetros necesarios en la línea de comandos, así como los nombres de los ficheros de entrada y salida; por defecto se toman la entrada y salida estándares, lo que permite conectar varios procesos mediante una tubería. Esto facilita la creación de *shellscripts* que realicen operaciones más complejas.

  Entre los programas más útiles para este trabajo están: `pnmcat`, que concatena vertical u horizontalmente imágenes (los *collages* de los capítulos 2 y 4 han sido construidos así); `pnmmargin`, que rodea la imagen de un borde de color y grosor dados; y `pnmscale`, que cambia de tamaño la imagen. Dado que *SNNS* tiene una utilidad, `mkpat`, que crea un fichero de patrones `.pat` a partir de un fichero binario que contenga la imagen como lista de bytes por columnas y por filas, esta operación se simplifica mucho con el formato PGM, que consta de una cabecera más los bytes de la imagen en ese orden.

  Puede conseguirse en `ftp.x.org:/contrib` o `ftp.ee.lbl.gov:/`, ficheros `pbmplus*.tar.Z`.

- Los programas `djpeg` y `cjpeg` del *Independent JPEG Group's JPEG software*, versión 5beta2 (20-Aug-94), para pasar del formato JPEG a otros varios (GIF, Targa, etc.) y al revés. Estos programas pueden obtenerse en `ftp.uu.net:/graphics/jpeg/jpegsrc.v5beta2.tar.gz`.

El procesamiento tuvo lugar en un PC 486/66MHz con 16M RAM bajo Linux 1.2.8.

D.4 Listados

D.4.1 Programas de *Mathematica*

`defines.m`

El programa siguiente contiene definiciones útiles: media, varianza, norma, proyección ortogonal, ortogonalización de Gram-Schmidt y salida en varios formatos, entre otras.

```
(* ---------------------------------------------------------------------------------- *)

(* Opciones del formato de salida de expresiones *)

Format[Continuation[n_]] := ""
Format[LineBreak[n_]] := ""
Format[Indent[n_]] := ""
SetOptions[MatrixForm, TableSpacing->{0}]

(* Necesario para Empareja y otras funciones recursivas que pueden aplicarse
   a listas de bastantes elementos *)

$RecursionLimit = $IterationLimit = 2000

(* Media de una lista l de reales o de vectores *)

Media[l_] := Apply[Plus,l] / Length[l]

(* Varianza de una lista l de reales o de vectores *)

Var[l_] := Media[l^2] - Media[l]^2

(* Covarianza de una lista l de vectores *)

(* Cov[l_] := Sum[Outer[Times,l[[i]],l[[i]]], {i,Length[l]}] / Length[l] -
              Outer[Times,Media[l],Media[l]] *)

Cov[l_] := Transpose[l].l / Length[l] - Outer[Times,Media[l],Media[l]]

(* "Empareja" las listas l1, l2 tomando elementos dos a dos de ellas:
   Empareja[{a,b,...,z},{A,B,...,Z}] -> {{a,A},{b,B},...,{z,Z}} *)

Empareja[l1_, l2_] :=
  If[Length[l1]==0,
        {},
        Prepend[Empareja[Rest[l1], Rest[l2]], {First[l1], First[l2]}]
  ]

(* A partir de la matriz cuadrada m obtiene su matriz triangular inferior en
   forma de lista plana (quita los elementos de la diagonal y de por encima
   de la diagonal *)

LinTriMat[m_, n_:0] :=
  If[Length[m]==0, {}, Join[Take[m[[1]],n],LinTriMat[Rest[m],n+1]]]

(* Normaliza linealmente la lista l de vectores al intervalo dado *)

NormInterval[l_, a_:0, b_:1] := Module[{M,m},
  m = Min[l]; M = Max[l]; (b-a)/(M-m) (l-m) + a
]

(* Eigensystem ordenado y de vectores normalizados de la matriz l *)

EigenOrd[l_] := Module[{aux},
  aux = Eigensystem[l];
  aux = Sort[Empareja[aux[[1]], aux[[2]]], (#2[[1]]<#1[[1]])&];
```

```
  {Map[First, aux], Map[Normaliza,Map[Last, aux]]}
]

(* Norma euclidea de un vector o matriz v *)

Norma[v_] := Sqrt[Apply[Plus,v^2,{0,1}]]

(* Norma infinito (del maximo) de un vector v *)

NormaInf[v_] := Max[Abs[v]]

(* Normaliza el vector o matriz v respecto a la norma f *)

Normaliza[v_, f_:Norma] := v / f[v]

(* Proyeccion del vector u sobre la lista de vectores l *)

(* Proyeccion[u_, l_] := Sum[u.l[[i]]/Norma[l[[i]]]^2 l[[i]], {i,Length[l]}] *)

Proyeccion[u_, l_] := u.Transpose[l].l

(* Vector error absoluto al proyectar el vector u sobre la lista de vectores l *)

VAbsErr[u_, l_] := u - Proyeccion[u,l]

(* Vector error relativo al proyectar el vector u sobre la lista de vectores l *)

VRelErr[u_, l_] := (u - Proyeccion[u,l])/Norma[u]
RestoR[u_, l_] := (u - Proyeccion[u,l])/Norma[Proyeccion[u,l]]

(* Ortonormalizacion Gram-Schmidt. Es mas eficiente hacer una descomposicion
   en valores singulares *)

GS[l_] := Module[{aux},
  If[Length[l]==1,
    {Normaliza[l[[1]]]},
    aux=GS[Rest[l]]; Prepend[aux, Normaliza[l[[1]]-Proyeccion[l[[1]],aux]]]
  ]]

(* Devuelve las normas de los vectores error relativo al proyectar cada vector
   de l1 sobre el subespacio generado por l2 *)

RelErrs[l1_, l2_] := Map[Norma[VRelErr[#,l2]]&, l1]
RestosR[l1_, l2_] := Map[Norma[RestoR[#,l2]]&, l1]

(* Devuelve las normas de las proyecciones de los vectores de l1 sobre el
   subespacio generado por l2 *)

Proys[l1_, l2_] := Map[Norma[Proyeccion[#,l2]]&, l1]

(* Matriz de Gram de la base l (puesta como lista de vectores: l=(v1,...,vn) *)

GramMatrix[l_] := l.Transpose[l]

(* Angulo en grados entre los vectores u y v *)

Angulo[u_, v_] := N[Re[ArcCos[u.v/(Norma[u] Norma[v])]/Degree]]

(* Matriz de angulos entre los vectores de la lista l1 y los de la lista l2:
   el elemento (i,j) contiene el angulo entre l1[i] y l2[j] (grados) *)

Angulos[l1_, l2_] := Table[Map[Angulo[l1[[i]],#]&,l2], {i,Length[l1]}]

(* Varianza de la lista de vectores l respecto a la direccion v *)
```

```
VarD[l_, v_] := Var[l.Normaliza[v]]

(* Suma de errores cuadraticos ||u-m1.m2.u||^2 para cada vector u de la lista
   l, siendo m1.m2 una matriz cuadrada *)

SSE[l_, m1_, m2_] := Apply[Plus,(l-l.Transpose[m1].m2)^2,{0,1}]

(* Pone cada componente de la lista l en un fichero f.n *)

PonFichs [l_, f_, n_:1] :=
  If[Length[l]==0,,Put[OutputForm[MatrixForm[l[[1]]]],f<>"."<>ToString[n]];
                   PonFichs[Rest[l],f,n+1]]

(* Multiplica cada elemento de la lista de reales l por k, truncando lo que
   se salga de [0,1]. Util si l contiene valores de intensidad de grises con
   un rango dinamico de 0 a 1, fuera del cual hay saturacion. Valido tambien
   si k es una lista como l (producto elemento a elemento) *)

MulIntensity[l_, k_] := Map[If[#>1,1,If[#<0,0,#]]&, k l]

(* Analogo, pero sumando k a cada elemento de l. Valido tambien si k es una
   lista como l (suma elemento a elemento) *)

AddIntensity[l_, k_] := Map[If[#>1,1,If[#<0,0,#]]&, k+l]

(* Pasa cada elemento de la lista l de vectores a un fichero PGM de texto
   llamado f.n. Es necesario normalizar primero dicha lista a un intervalo
   dentro de [0-255], de manera global a toda l. Normalizaciones posibles:
   - NormInterval[l,0,255]: util cuando el intervalo de variacion de l no
         esta dentro del [0,1].
   - 255 l: util si se sabe que todos los elementos estan en [0,1] y no se
         quiere deformar el rango de valores. *)

MatToPGM[l_, w_, h_, f_, n_:1] :=
  PonFichs[Map[Join[{P2,w,h,255},#]&,Floor[l]],f,n]

(* Pasa la lista de patrones (vectores de reales) l1 a f, un fichero .pat de
   SNNS, con dos copias por patron (input = output), truncando a d decimales
   y a~adiendo los nombres de la lista l2 como referencia *)

MatToPat[l1_, l2_, f_, d_:6, n_:1] := Module[{},
                    SetOptions[PaddedForm,ExponentFunction->(Null&)];
                    OpenWrite[f];
                    SetOptions[f,PageWidth->81];
                    MatToPat2[l1,l2,f,d,n];
                    Close[f]
                   ]

MatToPat2[l1_, l2_, f_, d_, n_:1] :=
  If[Length[l1]==0,, PutAppend[
      OutputForm["# Input "<>ToString[n]<>" ("<>l2[[1]]<>")"],
      OutputForm[PaddedForm[MatrixForm[{l1[[1]]}],{d+1,d}]],
      OutputForm["# Output "<>ToString[n]<>" ("<>l2[[1]]<>")"],
      OutputForm[PaddedForm[MatrixForm[{l1[[1]]}],{d+1,d}]],
    f];
    MatToPat2[Rest[l1],Rest[l2],f,d,n+1]
  ]

(* Formatea l a d decimales *)

PR[l_,d_] := OutputForm[NumberForm[Chop[l,10^-5],d,ExponentFunction->(Null&)]]

(* ------------------------------------------------------------------------------- *)
```

`pca.m`

Este listado muestra cómo calcular los autovalores y autovectores de la matriz de correlación $\mathbf{XX}^T$:

```
(* ------------------------------------------------------------------------------ *)
<< defines.m

(* Leer X *)

(*
   -------------------------------------------------------------------------
   Metodo directo: problema espectral de XXt
   -------------------------------------------------------------------------
*)

{Autovalores,Ut} = EigenOrd[N[Transpose[X].X]]

TrazaXXt = Apply[Plus,X^2,{0,1}]

(* Baldi & Hornik: E(A,B) = tr(XXt) - Sum{autovalores de XXt} *)

ListaSSE = TrazaXXt -
  Join[{0},Table[Sum[Autovalores[[j]],{j,i}],{i,Length[Autovalores]}]]

(* ListaSSE = Table[SSE[X,Take[Ut,i],Take[Ut,i]],{i,Length[Ut]}] *)

NormMediaX = Media[Map[Norma,X]]
Xm=Media[X]; Xc=Map[#-Xm&,X]
NormMediaXc = Media[Map[Norma,Xc]]
(* ------------------------------------------------------------------------------ *)
```

`pca1.m`

El listado siguiente muestra cómo calcular los autovalores no nulos y sus autovectores asociados para la matriz $\mathbf{XX}^T$, usando el método de la proposición 1.4.1 (a partir de $\mathbf{X}^T\mathbf{X}$):

```
(* ------------------------------------------------------------------------------ *)
<<defines.m

(* Leer X *)

(*
   --------------------------------------------------------------------------
   Metodo indirecto: problema espectral de XtX
   Valido cuando Length[X] < Length[X[[1]]], es decir, menos vectores que
   dimensiones
   --------------------------------------------------------------------------
*)

SL = EigenOrd[N[X.Transpose[X]]]

(* Autovalores no nulos de XXt = autovalores no nulos de XtX.
   Si Media[X]=0, 0 es autovalor de XtX con autovector (1,1,...,1)t, ya que
   X.(1,1,...,1)t=0. Pero X.(1,1,...,1)t=0 no puede tomarse como autovector
   de XXt, luego hay que descartar 0 de Autovalores y (1,1,...,1) de Ut *)

Autovalores=Drop[SL[[1]],-1]

(* Autovectores ui de XXt = X.autovectores de XtX *)

Ut = Map[Normaliza,Drop[SL[[2]],-1].X]

TrazaXXt = Apply[Plus,X^2,{0,1}]

(* Baldi & Hornik: E(A,B) = tr(XXt) - Sum{autovalores de XXt} *)
```

```
ListaSSE = TrazaXXt -
  Join[{0},Table[Sum[Autovalores[[j]],{j,i}],{i,Length[Autovalores]}]]

(* ListaSSE = Table[SSE[X,Take[Ut,i],Take[Ut,i]],{i,Length[Ut]}] *)

NormMediaX = Media[Map[Norma,X]]
Xm=Media[X]; Xc=Map[#-Xm&,X]
NormMediaXc = Media[Map[Norma,Xc]]
(* ------------------------------------------------------------------------------ *)
```

`result.m`

En este listado se muestra cómo obtener resultados sobre las bases obtenidas por la red Ξ. Las matrices `W1` y `W2` representan, respectivamente, las matrices **B** y **A** de la sección 3.3.1.

```
(* ------------------------------------------------------------------------------ *)
<<defines.m

(* Leer: X, Autovalores y Ut para Cov[X], W1 y W2; Xm, Xc *)
(* Diversas estadisticas sobre las bases de Cottrell (obtenidas por la red)
   y las bases de PCA (obtenidas por Mathematica), para los mismos datos X *)

W   = W2.W1
{R0,S0} = EigenOrd[N[W]]
F01 = Map[Norma,W]
F02 = Map[Norma,W-Transpose[W]]

R1  = Map[Norma, W1]
R2  = LinTriMat[Angulos[W1,W1]]
bon = GS[W1]
R3  = RelErrs[Ut,bon]
(* R4  = RestosR[Ut,bon] *)
R5  = Proys[Ut,bon]
R6  = Map[VarD[Xc,#]&, W1] Length[Xc]

W2t = Transpose[W2]
S1  = Map[Norma, W2t]
F1  = Norma[PseudoInverse[W1]-W2]
S2  = LinTriMat[Angulos[W2t,W2t]]
bon = GS[W2t]
S3  = RelErrs[Ut,bon]
(* S4  = RestosR[Ut,bon] *)
S5  = Proys[Ut,bon]
S6  = Map[VarD[Xc,#]&, W2t] Length[Xc]

<<Statistics`LinearRegression`
F2  = Table[Regress[Empareja[W1[[i]],W2t[[i]]],{1,x},x],{i,Length[W1]}]
F2  = Table[Join[Part[F2[[i,1,2,1]],{1,2},{1,2}],{F2[[i,2]]}],{i,Length[F2]}]

F3  = SSE[Xc,W1,W2t]
F4  = SSE[{Xm},W1,W2t]

OpenWrite["salida"]
SetOptions["salida",PageWidth->500]
Put[
OutputForm["*** W="],OutputForm[PaddedForm[MatrixForm[W],{5,4}]],
OutputForm["\n*** Autovalores de W"],PR[R0,3],
OutputForm["\n*** Autovectores de W"],OutputForm[PaddedForm[MatrixForm[S0],{4,3}]],
OutputForm["\n*** Normas de W"],PR[F01,2],
OutputForm["\n*** Simetria de W: Normas de W-Transpose[W]"],PR[F02,2],
OutputForm["\n*** ---------------------------------------------------------------"],
OutputForm["\n*** W1="],OutputForm[PaddedForm[MatrixForm[W1],{6,5}]],
OutputForm["\n*** Normas"],PR[R1,3],
OutputForm["\n*** Angulos"],OutputForm[Round[R2]],
```

```
OutputForm["\n*** RelErrs"],PR[R3,3],
(* OutputForm["\n*** RestosR"],PR[R4,3], *)
OutputForm["\n*** Proys"],PR[R5,3],
OutputForm["\n*** p-Varianzas"],PR[R6,6],
OutputForm["\n*** ------------------------------------------------------------"],
OutputForm["\n*** W2t="],OutputForm[PaddedForm[MatrixForm[W2t],{6,5}]],
OutputForm["\n*** Normas"],PR[S1,3],
OutputForm["\n*** Norma de PseudoInverse[W1]-W2"],PR[F1,3],
OutputForm["\n*** Angulos"],OutputForm[Round[S2]],
OutputForm["\n*** RelErrs"],PR[S3,3],
(* OutputForm["\n*** RestosR"],PR[S4,3], *)
OutputForm["\n*** Proys"],PR[S5,3],
OutputForm["\n*** p-Varianzas"],PR[S6,6],
OutputForm["\n*** ------------------------------------------------------------"],
OutputForm["\n*** Ajuste lineal W2t = k1 + k2 W1: { {{k1,SE},{k2,SE},{r2}},...}"],
                  OutputForm[N[Chop[F2],3]],
OutputForm["\n*** ------------------------------------------------------------"],
OutputForm["\n*** Suma de SSE(x, W2.W1.x)"],PR[F3,10],
OutputForm["\n*** SSE(Media[x], W2.W1.Media[x])"],PR[F4,6],
"salida"
]
Close["salida"]
(* ---------------------------------------------------------------------------- *)
```

D.4.2 *Shellscripts* de transformación de formatos

Para poder pasar fácilmente y de manera casi automática ficheros entre *Mathematica*, *SNNS* y `pbmplus` se emplearon una serie de *shellscripts* (para Unix o Linux), que se muestran a continuación.

Nótese que todos los *shellscripts* escriben a la salida estándar por defecto; conviene redirigirla a un fichero en la línea de comando. Esto se ha hecho así para permitir la construcción de tuberías (*pipes*) que conecten varios comandos.

Para ver con más detalle lo que hacen estos *shellscripts*, debe consultarse el manual del `pbmplus` [41] y del *SNNS* [62]. Otro libro recomendado es la guía *UNIX in a Nutshell*, de O'Reilly, que contiene información sobre `sed`, entre otros.

Nótese también que algunos de los directorios especificados pueden cambiar, dependiendo de la instalación particular de los paquetes (p. ej. `/home/neural/proyecto/nn/ut`).

`nettomat`

`nettomat` pasa un fichero `.net` con la descripción de una red Ξ a un fichero de *Mathematica* con las dos matrices `W1` y `W2` (**B** y **A** en la sección 3.3.1).

```
# Syntax: nettomat files...
nn_ut=/home/neural/proyecto/nn/ut
for abc in $@
do
  sed -f $nn_ut/nettomat1.sed $abc | sed -f $nn_ut/nettomat2.sed >
    `echo $abc | sed 's/\.net/\.mat/'`
done
echo Recuerda corregir },{ por W2={{
```

donde el fichero `nettomat1.sed` contiene las líneas:

```
1,/^-------|------/d
s/,[^:]*:/,/g
s/^.*|.*|.*:/\},\{/g
s/^.*://g
$s/.*/\}/
```

y el `nettomat2.sed`:

```
1s/},/W1={/
$s/}/}}/
```

Ambos contienen patrones para la utilidad estándar de Unix, `sed`, que es un editor de texto no interactivo. En el *shellscript* anterior se limita a hacer ciertas sustituciones en el fichero de entrada.

`pgmtomat`

El siguiente *shellscript* transforma una serie de ficheros PGM (que contienen cada uno una imagen de tonos de gris) en un fichero de *Mathematica* (que contiene una matriz, cuyas filas son los vectores imagen correspondientes a cada imagen):

```
# Sintax: pgmtomat width height files...
snns_tools=/home/neural/SNNSv4.0/tools/bin/pc_linux
nn_ut=/home/neural/proyecto/nn/ut
width=$1
height=$2
no_inputs=$[$1*$2]
shift
shift
# Keep file names
echo $@|sed ´s/ /\",\"/g´|sed ´s/^/names={\"/´|sed ´s/$/"}/´
echo data=
# File data
echo { >pgmtomat.tmp
echo { >>pgmtomat.tmp
pnmscale -width $width -height $height $1 | tail --bytes $no_inputs
    | $snns_tools/mkpat $no_inputs 1 >>pgmtomat.tmp
echo } >>pgmtomat.tmp
shift
for abc in $@
do
  echo ,{ >>pgmtomat.tmp
  pnmscale -width $width -height $height $abc | tail --bytes $no_inputs
    | $snns_tools/mkpat $no_inputs 1 >>pgmtomat.tmp
  echo } >>pgmtomat.tmp
done
echo } >>pgmtomat.tmp

sed -f $nn_ut/pgmtomat.sed pgmtomat.tmp | sed ´/[0-9]/s/ /,/g´
rm -f pgmtomat.tmp
```

El fichero `pgmtomat.sed` contiene:

```
/[0-9]/{
N
s/ *\n}/}/g
P
D
}
```

Bibliografía

[1] Akimoto, T., Suenaga, Y., Wallace, R. S.: "Automatic creation of 3D facial models." *IEEE Computer Graphics & Applications* **13**, No. 5, pp. 16–22 (Sep. 1993).

[2] "$\mathcal{AMS}$-LaTeX Version 1.2 User's Guide." American Mathematical Society (Jan. 1995).

[3] Baldi, P.: "Linear learning: landscapes and algorithms." In D. S. Touretzky (ed.), *Advances in Neural Information Processing Systems* **1**, pp. 65–72. San Mateo, CA: Morgan Kaufmann (1989).

[4] Baldi, P., Hornik, K.: "Learning in neural networks: A survey," to be published by the IEEE (1994). Available by anonymous ftp in the *Neuroprose* database at `archive.cis.ohio-state.edu:/pub/sci/neural/neuroprose/baldi.linear.ps.gz`.

[5] ———: "Neural networks and principal component analysis: Learning from examples without local minima." *Neural Networks* **2**, pp. 53–58 (1989).

[6] Beklémichev, D.: *Cours de géométrie analytique et d'algebre linéaire.* MIR (1984).

[7] Bjerhammar, A.: *Theory of Errors and Generalized Matrix Inverses.* Elsevier Scientific Publishing Company (1973).

[8] Bradley, J.: "*xv*: Interactive Image Display for the X Window System," Version 3.10 (1994). *xv* is © 1989, 1994 by John Bradley.

[9] Bourlard, H., Kamp, Y.: "Autoassociation by the multilayer perceptrons and singular value decomposition." *Biological Cybernetics* **59**, pp. 291–294 (1988).

[10] Chellappa, R., Wilson, C. L., Sirohey, S.: "Human and machine recognition of faces: A survey." *Proc. of the IEEE* **83**, No. 5, pp. 704–740 (May 1995).

[11] Costa González, A. F., Lafuente López, J.: *Geometrías lineales y grupos de transformaciones.* Cuadernos de la UNED, Madrid (1987).

[12] Cottrell, G. W., Munro, P. W., Zipser, D.: "Image compression by backpropagation: a demonstration of extensional programming." In N. E. Sharkey (ed.), *Advances in Cognitive Science* **2**. Norwood, NJ: Abbex (1988).

[13] Dony, R. D., Haykin, S.: "Neural network approaches to image compression." *Proc. of the IEEE* **83**, No. 2, pp. 288–303 (Feb. 1995).

[14] Dunteman, G. H.: *Principal Components Analysis.* Sage University Paper Series on Quantitative Applications in the Social Sciences, Series no. 07-069. Beverly Hills: Sage Publications (1989).

[15] Fleming, M. K., Cottrell, G. W.: "Categorization of faces using unsupervised feature extraction." *Proc. Int. J. Conf. on Neural Networks* **II**, pp. 65–70 (1990).

[16] Földiák, P.: "Adaptive network for optimal linear feature extraction." *Proc. Int. J. Conf. on Neural Networks* **I**, pp. 401–405 (1989).

[17] Freeman, J. A., Skapura, D. M.: *Neural Networks: Algorithms, applications, and programming techniques.* Addison-Wesley (1991).

[18] Frisch, Æ.: *Essential System Administration.* O'Reilly & Associates, Inc. (1991).

[19] García Pindado, M., Sánchez de Dios, J. L. *et al.*: *Estudios de Policía Cientíca: Identificación. Sección I: Identificación Personal. Sección II: Necroidentificación*, 2ª ed. División de Formación y Perfeccionamiento de la Dirección General de la Policía (1992).

[20] Gilly, D. and the staff of O'Reilly & Associates, Inc.: *UNIX in a Nutshell. A Desktop Quick Reference for System V & Solaris 2.0. (updated for SVR4).* O'Reilly & Associates, Inc. (1992).

[21] González, R. C., Wintz, P.: *Digital Image Processing*, 2nd. Ed. Addison-Wesley (1987).

[22] Harmon, L. D., Khan, M. K., Lasch, R., Ramig, P. F.: "Machine identification of human faces." *Pattern Recognition* **13**, No. 2, pp. 97–110 (1981).

[23] Harmon, L. D., Kuo, S. C., Ramig, P. F., Raudkivi, U.: "Identification of human face profiles by computer." *Pattern Recognition* **10**, pp. 301–312 (1978).

[24] Hecht-Nielsen, R.: *Neurocomputing.* Addison-Wesley (1990).

[25] Hertz, J., Krogh, A., Palmer, R. G.: *Introduction to the Theory of Neural Computation.* Addison-Wesley (1991).

[26] Himmelblau, D. M.: *Applied Nonlinear Programming.* McGraw-Hill (1972).

[27] Kaufman, G. J., Breeding, K. J.: "The automatic recognition of human faces from profile silhouettes." *IEEE Trans. on Systems, Man, and Cybernetics* **SMC-6**, No. 2, pp. 113–121 (Feb. 1976).

[28] Kim, D.-S., Lee, S.-Y.: "Intelligent judge neural network for speech recognition." *Neural Processing Letters* **1**, No. 1 (Sep. 1994).

[29] Kohonen, T.: *Associative Memories. A system theoretical approach.* Springer Verlag (1977).

[30] Kung, S. Y., Diamantaras, K. I., Taur, J. S.: "Adaptive Principal component EXtraction (APEX) and applications." *IEEE Trans. on Signal Processing* **42**, No. 5, pp. 1202–1217 (May 1994).

[31] Kwok, C.: "EEPIC: Extensions to epic and LaTeX Picture Environment Version 1.1," (2 Feb. 1988).

[32] Lamport, L.: *LaTeX. A Document Preparation System.* Addison-Wesley (1986).

[33] "LaTeX 2_ε for authors." LaTeX3 Project Team (10 Jun. 1995).

[34] Lippmann, R. P.: "An introduction to computing with neural nets." *IEEE ASSP Magazine* **4**, pp. 4–22 (Apr. 1987).

[35] Liu, J., Lee, C. M.: "Grouped window-based neural network approach to face recognition." *ICARCV'92 2nd. International Conf. on Automation, Robotics and Comp. Vision* **1**, pp. CV-20.3.1–CV-20.3.5 (1992).

[36] Lutzy, O., Dengel, A.: "A comparison of neural net simulators." *IEEE Expert* **8**, No. 4, pp. 43–51 (Aug. 1993).

[37] *McGraw-Hill Dictionary of Scientific and Technical Terms*, 4th. Ed. McGraw-Hill (1989).

[38] Oja, E.: "Principal components, minor components, and linear neural networks." *Neural Networks* **5**, pp. 927–935 (1992).

[39] O'Toole, A. J., Abdi, H., Deffenbacher, K. A., Valentin, D.: "Low-dimensional representation of faces in higher dimensions of the face space." *J. Opt. Soc. Am. A.* **10**, No. 3, pp. 405–411 (1993).

[40] Petkov, N., Kruizinga, P., Lourens, T.: "Face recognition on the connection machine CM-5." In *Parallel Computing: Trends and Applications*, pp. 185–192. G. R. Joubert, D. Trystram, F. J. Peters and D. J. Evans (eds.). Elsevier Science B. V. (1994).

[41] Poskanzer, J. *et al.*: "`pbmplus` (Portable Bitmap File Format) man pages" (1991). `pbmplus` is © 1989, 1991 by Jef Poskanzer.

[42] Prechelt, L. (administrador): "`comp.ai.neural-nets`: Frequently Asked Questions (FAQ)." La versión más moderna puede encontrarse en los grupos `comp.ai.neural-nets` y `comp.answers`, así como en la WWW: `http://wwwipd.ira.uka.de/ prechelt/FAQ/neural-net-faq.html`.

[43] Press, W. H., Teukolsky, S. A., Vetterling, W. T., Flannery, B. P.: *Numerical Recipes in Fortran: The Art of Scientific Computing*, 2nd. Ed. Cambridge University Press (1992).

[44] Riesco Sobré, A.: “La descripción policial.” *Revista Policía*, n$^{\underline{o}}$ 80, pp. 25–32 (jun. 1992).

[45] Rokicki, T.: “DVIPS: A TEXDriver.” Version 5.523. dvips is © 1986, 1994 Radical Eye Software.

[46] Rumelhart, D. E., MacClelland, J. L. and the PDP Research Group: *Parallel Distributed Computing: explorations in the microstructure of cognition. Vol. 1: Foundations.* MIT Press, Cambridge, Massachusetts (1986).

[47] ______: *Parallel Distributed Computing: explorations in the microstructure of cognition. Vol. 2: Psychological and biological models.* MIT Press, Cambridge, Massachusetts (1986).

[48] Samal, A., Iyengar, P. A.: “Automatic recognition and analysis of human faces and facial expressions: A survey.” *Pattern Recognition* **25**, No. 1, pp. 65–77 (1992).

[49] Samaria, F.: “Face segmentation for identification using hidden Markov models.” In *British Machine Vision Conference 1993.* BMVA Press (1993).

[50] Sanger, T. D.: “Optimal unsupervised learning in a single-layer linear feedforward neural network.” *Neural Networks* **2**, pp. 459–473 (1989).

[51] Sims, D.: “Biometric recognition: Our hands, eyes, and faces give us away.” *IEEE Computer Graphics and Applications* **14**, No. 5, pp. 14–15 (Sep. 1994).

[52] Sirovich, L., Kirby, M.: “Low-dimensional procedure for the identification of human faces.” *J. Opt. Soc. Am. A.* **4**, No. 3, pp. 519–524 (1987).

[53] Strang, G.: *Linear Algebra and ist Applications*, 3rd. Ed. Harcourt Brace Jovanovich (1988).

[54] Tou, J. T., González, R. C.: *Pattern Recognition Principles.* Addison-Wesley, 1974.

[55] Turk, M., Pentland, A.: “Eigenfaces for recognition.” *J. Cognitive Neurosci.* **3**, pp. 71–86 (1991).

[56] Valentin, D., Abdi, H., O’Toole, A. J., Cottrell, G. W., “Connectionist models of face processing: A survey.” *Pattern Recognition* **27**, No. 9, pp. 1209–1230 (1994).

[57] Wang, D.: “Pattern recognition: Neural networks in perspective.” *IEEE Expert* **8**, No. 4, pp. 52–60 (Aug. 1993).

[58] Welsh, M.: “Linux Installation and Getting Started.” Version 2.2.2 (11 Feb. 1995). The Linux Documentation Project.

[59] Wilkinson, J. H.: *The Algebraic Eigenvalue Problem.* Oxford University Press (1965).

[60] Williams, T., Kelley, C.: “GNUPLOT. An Interactive Plotting Program.” Version 3.5. *gnuplot* is © 1986–1993 Thomas Williams, Colin Kelley.

[61] Wolfram, S.: Mathematica. *A System for Doing Mathematics by Computer*, 2nd. Ed. Addison-Wesley, 1991.

[62] Zell, A. *et al.*: “Stuttgart Neural Network Simulator (*SNNS*) User Manual, Version 4.0.” Institute for Parallel and Distributed High Performance Systems (IPVR), University of Stuttgart, Report No. 6/95 (1995).