LA PSEUDOINVERSA EN EL PROCESO DE APRENDIZAJE DEL ASOCIADOR LINEAL

Nancy Dayana Díaz Toro

12 de septiembre de 2014

LA PSEUDOINVERSA EN EL PROCESO DE APRENDIZAJE DEL ASOCIADOR LINEAL

Nancy Dayana Díaz Toro

Trabajo de Investigación como requisito parcial para optar al título de Magister en Ciencias Matemáticas

Director:
Héctor Jairo Martínez Romero, Ph.D

UNIVERSIDAD DEL VALLE
FACULTAD DE CIENCIAS NATURALES Y EXACTAS
MAESTRÍA EN CIENCIAS MATEMÁTICAS
SANTIAGO DE CALI

12 de septiembre de 2014

Resumen

Muchos algoritmos de aprendizaje de una red neuronal artificial (RNA) requieren resolver problemas de mínimos cuadrados lineales a fin de obtener los pesos sinápticos.

Para la RNA conocida como Asociador Lineal, estudiamos algoritmos de aprendizaje y realizamos un análisis comparativo desde el punto de vista de la precisión y la complejidad de éstos algoritmos. En este sentido, hemos propuesto algoritmos para resolver el problema de mínimos cuadrados lineales presente en esta red para el caso de matrices con rango deficiente. Comparamos aquellos algoritmos que usan la matriz pseudoinversa teniendo en cuenta seis maneras diferentes de calcularla: Método de la Descomposición en Valores Singulares(DVS), método basado en la Factorización de Cholesky Generalizada, método basado en el proceso de Ortogonalización de Gramm-Schmidt, método de la Escalonada Reducida, método basado en las ideas de la Proyección del Gradiente, y el último es el método derivado del Teorema de Cayley-Hamilton.

Finalmente, a partir de los resultados obtenidos en el estudio comparativo, analizamos el comportamiento de la matriz pseudoinversa en el proceso de aprendizaje de las redes neuronales artificiales Hopfield y Funciones de Base Radial.

Palabras clave: Redes Neuronales Artificiales (RNA), algoritmos de aprendizaje, Asociador Lineal, Problema de mínimos cuadrados lineales (PMCL), matriz pseudoinversa, Hopfield, Funciones de Base Radial(FBR).

Abstract

Many learning algorithms of an artificial neural network (ANN) require the solution of linear least squares problems in order to obtain the synaptic weights.

For the ANN known as Linear associator, we study learning algorithms and perform a comparative analysis from the point of view of accuracy and complexity of these algorithms. In this sense, we have proposed algorithms to solve linear least squares problem present in this network for the case of rank deficient matrices. We compare those algorithms that use the pseudoinverse considering six ways to calculate the matrix: Method of the singular value descomposition, method based on the Generalized Cholesky Factorizacition, method based in the process of Gramm-Schmidt Orthogonalization, method of the row reduce echelon form, method based on the ideas of the Gradient Projection, and the latter is a method derived from the Cayley-Hamilton theorem.

Finally, we analyze, from the results of the comparative study, the behavior of the pseudoinverse in the learning process of Hopfield neural network and Radial Basis Functions.

Keywords: Artificial Neural Networks (ANN), learning algorithms, Linear Associator, linear least squares problem, pseudoinverse matrix, Hopfield Network, Radial Basis Function (RBF).

Índice general

Índice de figuras

Indice de tablas

Lista de algoritmos

Capítulo 1

Introducción

Las redes neuronales artificiales son consideradas como un acercamiento hacia la inteligencia humana. Cumplen con tareas de procesamiento como el reconocimiento de patrones, percepción, control, clasificación, etc. La idea que subyace en los sistemas neuronales artificiales como es descrito en [21] es que, para abordar el tipo de problema que el cerebro resuelve con eficiencia, puede resultar conveniente construir sistemas que copien, en cierto modo, la estructura y funcionamiento de las redes neuronales biológicas con el fin de alcanzar una eficiencia similar.

Una red neuronal artificial se parece al cerebro en dos aspectos [30]:

1. El conocimiento es adqurido a partir de experiencias de su ambiente a través de un proceso de aprendizaje.

2. Las fuerzas de conexión entre neuronas, conocidas como conexiones sinápticas, son utilizadas para almacenar conocimiento adquirido.

Para realizar el proceso de aprendizaje, una red neuronal artificial (RNA) utiliza un algoritmo mediante el cual calcula los pesos sinápticos de la red. Sin embargo, es posible también para una red neuronal, en su proceso de aprendizaje, modificar su propia topología lo que viene motivado por el hecho de que las neuronas del cerebro humano pueden morir y que nuevas conexiones sinápticas pueden crecer o desaparecer [30].

En este sentido, muchos algoritmos de aprendizaje neuronal requieren resolver problemas de mínimos cuadrados para estimar los pesos sinápticos [sección 5.1]. En casos particulares, la matriz pseudoinversa permite la solución

de tales problemas, especialmente cuando son problemas lineales de rango deficiente y se desea obtener la solución de norma mínima. En este trabajo, para resolver estos casos especiales, se analizarán, desde el punto de vista de la precisión y la rapidez, algoritmos que utilicen matrices pseudoinversas, para el Asociador Lineal [sección 4.5].

1.1. Objetivos

El objetivo de este trabajo es analizar y comparar algunos algoritmos de aprendizaje para el Asociador Lineal.

Los objetivos específicos son:

1. Analizar el proceso de entrenamiento de las redes neuronales artificiales.
2. Analizar y plantear el problema de mínimos cuadrados lineales y los métodos de solución, especialmente aquellos que involucren matrices pseudoinversas.
3. Determinar las ventajas y desventajas relativas de los algoritmos para el cálculo del conjunto óptimo de pesos sinápticos en el Asociador Lineal.
4. Intentar extender estos resultados a las redes Hopfield y Funciones de base radial.

1.2. Estructura del Documento

La distribución de los temas a abordar en este trabajo es la siguiente:

En el Capítulo 2, estudiamos el problema de mínimos cuadrados lineales y los procedimientos más utilizados para resolverlo teniendo en cuenta si el rango de la matriz es completo o deficiente.

En el Capítulo 3, describimos distintos métodos para calcular la pseudoinversa. El primero, calcula la pseudoinversa a partir de la descomposición en valores singulares; el segundo está basado en la Factorización de Cholesky Generalizada; el tercero, en el proceso de Ortogonalización de Gramm-Schmidt; el cuarto, en el método de la Escalonada Reducida; el quinto se basa en las

ideas de la Proyección del Gradiente; y el último es un procedimiento derivado del teorema de Cayley-Hamilton. Finalmente, presentamos un estudio experimental comparativo de los algoritmos que calculan la matriz pseudoinversa.

En el Capítulo 4, presentamos en primera instancia un breve resúmen histórico de las redes neuronales artificiales (RNA) y a continuación describimos los elementos que forman una neurona biológica así como la teoria básica del aprendizaje. Presentamos también aquellos aspectos más relevantes de un sistema neuronal artificial y estudiamos un modelo de red neuronal artificial denominado Asociador Lineal, incluyendo algunas ideas relacionadas con la regla de aprendizaje de Hebb y la regla de la pseudoinversa.

En el Capítulo 5, realizamos un análisis de la comparación de los algoritmos de aprendizaje para el Asociador Lineal que utilizan o no, la matriz pseudoinversa, determinando ventajas y desventajas relativas de éstos algoritmos en cada uno de los casos y concluiremos qué procedimientos resultan más eficientes desde el punto de vista de la precisión y la complejidad de los algoritmos. Además, presentamos un ejemplo de reconocimiento de caracteres como aplicaciónn del Asociador Lineal.

En el Capítulo 6, realizamos una extensión de los resultados obtenidos en el Capítulo 3 a las redes neuronales artificiales Hopfield y Funciones de Base Radial, y presentamos los resultados de ésta extensión.

En el Capítulo 7, presentamos las conclusiones del trabajo de investigación, así como posibles trabajos futuros que se pueden realizar teniendo como base los logros alcanzados.

El documento se termina con el apéndice y las referencias bibliográficas utilizadas. En el apéndice, incluimos propiedades de la pseudoinversa, códigos implementados en MATLAB de los algoritmos estudiados en los apartados anteriores y la función newrbe para la creación de las Funciones de Base Radial en MATLAB.

Capítulo 2

Problema de Mínimos Cuadrados Lineales (PMCL)

2.1. Introducción

Los sistemas de ecuaciones $A\mathbf{x} = \mathbf{b}$ con $A \in \mathbb{R}^{m\times n}$, $m > n$ y $\mathbf{b} \in \mathbb{R}^m$ y $rango(A|\mathbf{b}) \neq rango(A) \leq n$, no tienen solución, pero se les puede encontrar una pseudosolución siguiendo el criterio de encontrar $\mathbf{x} \in \mathbb{R}^n$ que minimice la norma $\|A\mathbf{x} - \mathbf{b}\|_2$, o escoger entre las pseudosoluciones existentes aquella cuya norma sea mínima. El hecho de que para dar solución a los problemas referidos se utilice el criterio de minimizar la norma euclídea de una manera u otra es lo que engloba y da nombre a los procedimientos para resolver esos problemas: mínimos cuadrados [7].

En este capítulo, presentaremos la descripción del problema de mínimos cuadrados lineales y estudiaremos algunos de los principales métodos que solucionan este problema. Previo a la presentación de los métodos para el PMCL, se introducirá un concepto de gran importancia para el estudio de los métodos numéricos: el condicionamiento numérico

2.2. Planteamiento del problema

En este trabajo, estamos interesados en estudiar algunos métodos para resolver el PMCL, el cuál se plantea en los siguientes términos:

Dada una matriz $A \in \mathbb{R}^{m\times n}$ *de rango* $r \leq \min(m,n)$ *y un vector* $\mathbf{b} \in \mathbb{R}^m$,
encontrar un vector $\mathbf{x} \in \mathbb{R}^n$ *que minimice* $\|A\mathbf{x} - \mathbf{b}\|_2$. (2.1)

Caracterización de la solución del PMCL[11]: Denotemos el conjunto de todas las soluciones para el PMCL como

$$\mathbf{X} = \{\mathbf{x} \in \mathbb{R}^n : \|A\mathbf{x} - \mathbf{b}\|_2 = min\} \tag{2.2}$$

Este conjunto satisface las siguientes propiedades:

1. $\mathbf{x} \in \mathbf{X} \Leftrightarrow A^T(\mathbf{b} - A\mathbf{x}) = 0$ (2.3a)
2. $\mathbf{X}$ es convexo (2.3b)
3. $\mathbf{X}$ tiene un único elemento $\mathbf{x}_{LS}$ con mínima norma-2 (2.3c)
4. $\mathbf{X} = \{\mathbf{x}_{LS}\} \Leftrightarrow \ rango\,(A) = n$ (2.3d)

Los procedimientos más utilizados para resolver el PMCL conllevan la reducción de la matriz A a alguna forma canónica mediante transformaciones ortogonales. La pseudoinversa también permite resolver el PMCL.

En el caso de que una matriz A no sea de rango completo, las transformaciones ortogonales requieren modificaciones adecuadas para encontrar una solución. De otro lado, la matriz pseudoinversa resuelve el PMCL, proporcionando $\mathbf{x}_{LS}$, la solución de mínima norma-2, independientemente del rango y de las dimensiones de la matriz.

2.3. La matriz pseudoinversa

La noción de pseudoinversa de Moore-Penrose fue introducida por primera vez por E.H. Moore alrededor de 1920, permaneciendo casi desconocida hasta la publicación de un artículo de R. Penrose en 1955, titulado "*A generalized inverse for matrices*" [23], artículo que lograría atraer la atención sobre el tema. Parece ser que Penrose no estaba enterado del trabajo de Moore cuando publicó su artículo. R. Rado, en 1956, mostraría que la definición de la inversa generalizada, dada por Penrose, es equivalente a la dada por Moore.

Hoy en día, existe una extensa literatura en torno a generalizaciones del concepto de inversa usual; la inversa generalizada, también conocida como **pseudoinversa** o **inversa de Moore-Penrose**, es apenas una entre una variedad de inversas generalizadas que hay asociadas a una matriz.

El concepto de pseudoinversa de Moore-Penrose, que debido a su característica de unicidad, es útil en el tratamiento de problemas de optimización, como la determinación de la solución de mínimos cuadrados lineales, especialmente cuando son de rango deficiente.

Definición de la pseudoinversa: *Sea $A \in \mathbb{R}^{m\times n}$. La matriz pseudoinversa $A^+ \in \mathbb{R}^{n\times m}$ está definida como la única matriz que cumple las cuatro condiciones de Moore-Penrose*:

1. $AA^+A = A$,
2. $A^+AA^+ = A^+$,
3. $(AA^+)^T = AA^+$, es decir, AA^+ es simétrica,
4. $(A^+A)^T = A^+A$, es decir, A^+A es simétrica.

Existen otras propiedades relevantes de la pseudoinversa, que pueden deducirse de las condiciones anteriores y que también mantienen una analogía con las propiedades de la inversa para el caso de matrices cuadradas.

Teorema 2.3.1. *[2] Sea $A \in \mathbb{R}^{m\times n}$. La pseudoinversa A^+ satisface las siguientes propiedades:*

1. $(A^+)^+ = A$
2. $(A^+)^T = \left(A^T\right)^+$
3. $(\alpha A)^+ = \alpha^{-1}A^+$ *para todo* $\alpha \in \mathbb{R}$ *tal que* $\alpha \neq 0$
4. $\left(A^TA\right)^+ = A^+\left(A^+\right)^T$
5. *Si* $AA^T = A^TA$, *entonces* $A^+A = AA^+$ *y* $(A^n)^+ = (A^+)^n$ *para todo* $n \in \mathbb{Z}$
6. $rango(A) = rango(A^T) = rango(A^+) = rango(A^+A) = traza(A^+A)$

La solución del PMCL se puede expresar, en términos de A^+ como $\mathbf{x} = A^+\mathbf{b}$, como lo indica el siguiente teorema.

Teorema 2.3.2. *Cualquier vector* $\mathbf{x}$ *es solución de mínimos cuadrados lineales de un sistema* $A\mathbf{x} = \mathbf{b}$*, y además,* $\mathbf{x} = A^+\mathbf{b}$ *tiene mínima norma, si y solo si, se cumple*

$$A\mathbf{x} = AA^+\mathbf{b}.$$

Demostración.

$$\begin{aligned}\|A\mathbf{x} - \mathbf{b}\|_2^2 &= \left\|A\mathbf{x} - AA^+\mathbf{b} + AA^+\mathbf{b} - \mathbf{b}\right\|_2^2 \\ &= \left\|A\mathbf{x} - AA^+\mathbf{b}\right\|_2^2 + \left\|AA^+\mathbf{b} - \mathbf{b}\right\|_2^2 + 2(A\mathbf{x} - AA^+\mathbf{b})^T(AA^+\mathbf{b} - \mathbf{b})\end{aligned}$$

Siendo

$$2\mathbf{x}^T A^T(AA^+ - I)\mathbf{b} = 2\mathbf{x}^T A^T(A^T AA^T - A^T)\mathbf{b} = 0$$

y

$$-2\mathbf{b}^T(AA^+)^T(AA^+ - I)\mathbf{b} = -2\mathbf{b}^T(AA^+AA^+ - AA^+)$$

como $A^+AA^+ = A^+$, lo de arriba vale cero, por tanto

$$\|A\mathbf{x} - \mathbf{b}\|_2^2 = \left\|A\mathbf{x} - AA^+\mathbf{b}\right\|_2^2 + \left\|AA^+\mathbf{b} - \mathbf{b}\right\|_2^2$$

□

2.4. Métodos numéricos clásicos

En esta sección, daremos una breve introducción a los métodos numéricos directos más conocidos para la resolución del PMCL que se encuentran en la literatura.

Previo a la presentación de los métodos para el PMCL, introducimos un concepto de gran importancia para el estudio de los métodos numéricos: el condicionamiento numérico.

El condicionamiento numérico de un problema se refiere a una medida de *sensibilidad* que puede tener la solución de dicho problema a pequeñas perturbaciones en los datos de entrada. Para el caso de los sistemas de ecuaciones

lineales, sea $M \in \mathbb{R}^{n\times n}$ y $\mathbf{b} \in \mathbb{R}^n$ y supongamos que se desea resolver el sistema

$$M\mathbf{x} = \mathbf{b}. \tag{2.4}$$

Si los valores numéricos de M y $\mathbf{b}$ provienen, por ejemplo, de algún experimento, estos serán susceptibles de tener alguna perturbación debido al redondedo u otros errores experimentales como instrumentos de medición mal calibrados, una lectura incorrecta por parte del experimentador, o la propagación de errores en mediciones indirectas, entre otros. Por lo tanto, en la práctica, en lugar de resolver el sistema (2.4) se tendrá que resolver el siguiente sistema asociado

$$(M + \Delta M)\,\mathbf{x} = \mathbf{b} + \Delta\mathbf{b} \tag{2.5}$$

donde ΔM y $\Delta\mathbf{b}$ son una matriz y un vector que contienen los errores experimentales o perturbaciones asociados a M y $\mathbf{b}$, respectivamente. Entonces, se dice que el sistema (2.4) está *bien condicionado* si el error relativo de la solución del sistema perturbado (2.5) respecto a la solución del sistema original es "pequeño". En caso contrario, se dice que el problema está *mal condicionado.*

Una medida para el condicionamiento de una matriz, en el caso general, es el número de condición, el cual se define a continuación.

Definición 2.4.1. *Sea $A \in \mathbb{R}^{n\times n}$, no sigular, y sea $\|\cdot\|$ una norma matricial inducida. El número de condición de A, el cual se denota como $k(A)$, se define como*

$$k(A) \equiv \|A\| \left\|A^{-1}\right\|$$

Para el caso de una matriz rectangular $A \in \mathbb{R}^{m\times n}$, el número de condición se define como

$$k(A) \equiv \|A\| \left\|A^{+}\right\|.$$

Observemos que para cualquier norma inducida, $k(A) \geq 1$. En efecto $1 = \|I\| = \|AA^{+}\| \leq \|A\|\,\|A^{+}\| = k(A)$.

El número de condición da una medida del condicionamiento de un sistema de ecuaciones lineales que involucre a la matriz A. Si $k(A) \approx 1$ (esto es, un número de condición *"pequeño"*), entonces la matriz está bien condicionada, mientras que un número de condición que sea mucho mayor a 1 (esto es, un

número de condición *"grande"*) indica que la matriz está mal condicionada, y por lo tanto, el resultado de un método numérico que involucre a dicha matriz podría ser poco confiable.

Además, destacamos que en el caso del PMCL asociado a una matriz A y un vector $\mathbf{b}$, el condicionamiento del problema no depende sólo de A, sino que depende además del valor de $\mathbf{b}$, por lo cual se puede obtener una *extensión* del número de condición para el PMCL tal como se define a continuación.

Definición 2.4.2. *Sea $A \in \mathbb{R}^{n\times n}$ y $\mathbf{b} \in \mathbb{R}^n$. Si $\mathbf{x}_{LS}$ es una solución al PMCL asociado a A y $\mathbf{b}$, entonces el número de condición del PMCL, usando la norma euclídea, viene dado por*

$$k_{LS}(A, \mathbf{b}) \equiv k(A)\left(1 + k(A)\frac{\|\mathbf{r}\|_2}{\|A\|_2 \|\mathbf{x}_{LS}\|_2}\right)$$

donde $\boldsymbol{r} = b - A\mathbf{x}_{LS}$.

PMCL para el caso de rango completo

2.4.1. Ecuaciones normales

Una forma de abordar el problema de mínimos cuadrados lineales es a través de las ecuaciones normales. El siguiente teorema garantiza que el problema de encontrar la pseudosolución del sistema $A\mathbf{x} = \mathbf{b}$, con $A \in \mathbb{R}^{mXn}$, $m > n$ y $rango(A) = n$ es equivalente a resolver un sistema lineal de ecuaciones de la forma

$$A^T A\mathbf{x} = A^T\mathbf{b}. \tag{2.6}$$

Teorema 2.4.1. *Dado el sistema de ecuaciones lineales $A\boldsymbol{x} = \boldsymbol{b}$, el vector $\mathbf{x} \in \mathbb{R}^n$ minimiza $\|A\mathbf{x} - \mathbf{b}\|_2$, si y solo si, $A^T A\mathbf{x} = A^T\mathbf{b}$.*

Demostración. Recordemos que los mínimos de $\|A\mathbf{x} - \mathbf{b}\|_2$ son los mismos que los de $\|A\mathbf{x} - \mathbf{b}\|_2^2$ y sean $\mathbf{x}$ y $\mathbf{z} \in \mathbb{R}^n$, y $\alpha \in \mathbb{R}$. Consideremos la igualdad

$$\|A(\mathbf{x} + \alpha\mathbf{z}) - \mathbf{b}\|_2^2 = \|A\mathbf{x} - \mathbf{b}\|_2^2 + \alpha^2 \|A\mathbf{z}\|_2^2 + 2\alpha\mathbf{z}^T A^T(A\mathbf{x} - \mathbf{b}) \tag{2.7}$$

Si $\mathbf{x}$ resuelve el problema de mínimos cuadrados, entonces se puede escribir

$$\|A\mathbf{x} - \mathbf{b}\|_2^2 < \|A(\mathbf{x} + \alpha\mathbf{z}) - \mathbf{b}\|_2^2$$

y por la igualdad (2.7), $\alpha^2 \|A\mathbf{z}\|_2^2 + 2\alpha \mathbf{z}^T A^T (A\mathbf{x} - \mathbf{b}) > 0$.

Como el primer término siempre es positivo y, α y $\mathbf{z}$ son arbitrarios, se deduce que $A^T(A\mathbf{x} - \mathbf{b}) = 0$, pues de no serlo, se podría encontrar un $\mathbf{z}$ y un α de forma que no se cumpliera lo anterior. Así, para $\mathbf{z} = -A^T(A\mathbf{x} - \mathbf{b})$ y haciendo α lo suficientemente pequeño, se tiene la desigualdad contradictoria siguiente

$$\|A(\mathbf{x} + \alpha\mathbf{z}) - \mathbf{b}\|_2 < \|A\mathbf{x} - \mathbf{b}\|_2 .$$

Podemos concluir que si $\mathbf{x}$ y $\mathbf{x} + \alpha\mathbf{z}$ son soluciones del PMCL, entonces $\mathbf{z} \in N(A)$ (espacio nulo de A).

□

El sistema de ecuaciones que define la ecuación $A^T(A\mathbf{x} - \mathbf{b}) = 0$ se denomina *ecuaciones normales*, y el vector solución

$$\mathbf{x} = (A^T A)^{-1} A^T \mathbf{b} \tag{2.8}$$

es único, si A es de rango completo.

Algoritmo 1 PMCL con Ecuaciones normales

Entrada: Dada la matriz $A \in \mathbb{R}^{m \times n}$ con $rango(A) = n$ y $b \in \mathbb{R}^m$.
Salida: Solución $\mathbf{x}$ del problema mín $\|A\mathbf{x} - \mathbf{b}\|_2$.

1: Formar la matriz de la ecuación normal $C = A^T A$ y el vector $\mathbf{d} = A^T \mathbf{b}$.
2: Calcular la factorización de Cholesky $C = GG^T$, con G triangular inferior.
3: Resolver los dos sistemas $G\mathbf{y} = \mathbf{d}$ y $G^T\mathbf{x} = \mathbf{y}$. El vector $\mathbf{x}$ es la solución deseada.

Este algoritmo requere $(m + n/3)n^2$ operaciones [11].

2.4.2. Factorización QR

Uno de los métodos más utilizados para resolver el PMCL es el de la descomposición ortogonal de la matriz A. Esta estrategia presenta buenas propiedades gracias a que las transformaciones ortogonales conservan la norma euclídea.

Existen varios métodos para calcular la factorización QR. Así, algunos se basan en transformaciones de Householder, otros en rotaciones de Givens y por ultimo están los métodos que realizan una ortogonalización vía Gram-Schmidt.

Transformaciones de Householder

Definición 2.4.3. *Se denomina transformación de Householder a una transformación lineal de* $\mathbb{R}^n$ *en* $\mathbb{R}^n$*, caracterizada por una matriz* $H \in \mathbb{R}^{n\times n}$ *de la forma*

$$H = I - 2\mathbf{w}\mathbf{w}^T,$$

donde $\mathbf{w} \in \mathbb{R}^n$ *es un vector unitario.*

Aplicar una transformación de Householder a un vector cualquiera $\mathbf{a}$ equivale a reflejarlo en el subespacio ortogonal al subespacio $Gen(\mathbf{w})$, asi

$$H\mathbf{a} = \mathbf{a} - 2(\mathbf{w}^T\mathbf{a})\mathbf{w},$$

donde $(\mathbf{w}^T\mathbf{a})\mathbf{w}$ es la proyección de $\mathbf{a}$ sobre $\mathbf{w}$. Geométricamente, se puede expresar como se describe en la Figura 2.1.

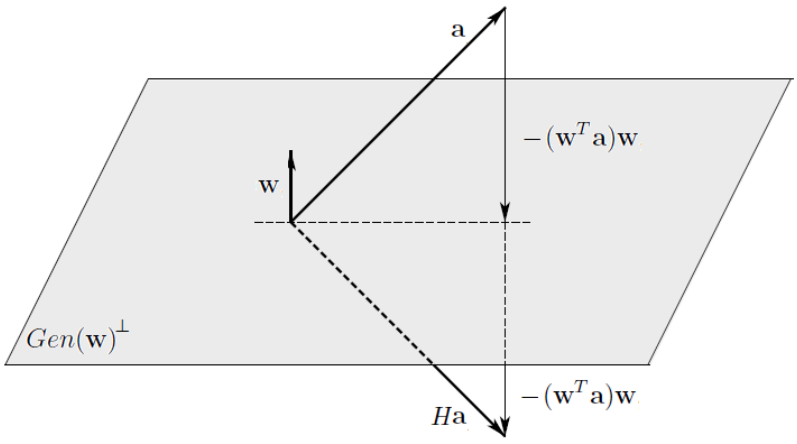


Figura 2.1: Transformación de Householder de vector $\mathbf{w}$ aplicada a $\mathbf{a}$

La importancia de las matrices Householder es que ellas pueden ser usadas para "introducir" ceros en un vector y, por lo tanto, pueden dar lugar a matrices triangulares.

Teorema 2.4.2. *(**Método de Householder**)[29]*
Supongamos que $A \in \mathbb{R}^{m\times n}$. Existe una sucesión $H_1, H_2, ..., H_n$ de a lo más n matrices de Householder de manera que

$$H_n, H_{n-1}, ..., H_1 A = R,$$

en donde R es una matriz triangular superior y tiene elementos no negativos en la diagonal principal; equivalentemente

$$A = QR,$$

donde $Q = H_1^{-1}, H_2^{-1}, ..., H_n^{-1}$ es una matriz ortogonal $m \times m$.

Demostración. Sea $A = [\mathbf{a}_1, \mathbf{a}_2, \cdots, \mathbf{a}_n]$. Buscamos una matriz de Householder $H_1 = \left(I - 2\mathbf{w}_1\mathbf{w}_1^T\right)$ que haga ceros la primera columna de A a excepción del primer elemento; es decir, se debe cumplir que $H_1\mathbf{a}_1 = c_1\mathbf{e}_1$ para un cierto c_1 y siendo $\mathbf{e}_1$ el vector $(1, 0, ..., 0)^T$, luego

$|c_1| = \|H_1\mathbf{a}_1\|_2 = \|\mathbf{a}_1\|_2$. Por definición de H_1, debe ser

$$H_1\mathbf{a}_1 = \mathbf{a}_1 - 2\mathbf{w}_1\mathbf{w}_1^T\mathbf{a}_1 = c_1\mathbf{e}_1.$$

Como la expresión $\mathbf{w}_1^T\mathbf{a}_1$ es un escalar, resulta que

$$\mathbf{w}_1 = K\mathbf{v}_1, \text{ donde } \mathbf{v}_1 = a_1 - c_1\mathbf{e}_1$$

y la constante K se determina de forma que $\|\mathbf{w}_1\|_2 = 1$. Como todavía queda libertad de elegir el signo de c_1, este lo escogemos como $-(sign\ a_{11})$, para evitar pérdida de cifras significativas en el cálculo de $\mathbf{v}_1 = \mathbf{a}_1 - c_1\mathbf{e}_1$, por lo tanto, $c_1 = -(sign\ a_{11})\|\mathbf{a}_1\|_2$ [22].

Para calcular la matriz H_1A, tenemos en cuenta que

$$H_1A = A - 2\mathbf{w}_1\mathbf{w}_1^TA = A - \beta\mathbf{v}_1(\mathbf{v}_1^TA), \text{ donde } \beta = \frac{2}{\mathbf{v}_1^T\mathbf{v}_1}.$$

Este factor β lo calculamos como sigue

$$\beta^{-1} = \frac{\mathbf{v}_1^T\mathbf{v}_1}{2} = \frac{1}{2}(A_1^TA_1 - 2ca_{11} + c_1^2) = -c_1(a_{11} - c_1).$$

Asi

$$H_1 A = [H_1\mathbf{a}_1, H_1\mathbf{a}_2, \cdots, H_1\mathbf{a}_n]$$
$$= \begin{bmatrix} c_1 & * & \cdots & * \\ 0 & * & \cdots & * \\ \vdots & \vdots & & \vdots \\ 0 & * & \cdots & * \end{bmatrix} = \begin{bmatrix} c_1 & * & \cdots & * \\ 0 & & & \\ \vdots & & K & \\ 0 & & & \end{bmatrix}. \tag{2.9}$$

En la segunda etapa, aplicamos el mismo procedimiento a la submatriz K de orden $(m-1)\times(n-1)$, lo cual proporciona un vector $\overline{\mathbf{w}}_2 \in \mathbb{R}^{m-1}$ y una matriz de Householder $\overline{H}_2 = I - 2\overline{\mathbf{w}}_2\overline{\mathbf{w}}_2^T$. Escribiendo $\mathbf{w}_2 = (0, \overline{\mathbf{w}}_2)^T$ y multiplicando (2.9) por $H_2 = I - 2\mathbf{w}_2\mathbf{w}_2^T$, obtenemos

$$H_2H_1A = H_2\begin{bmatrix} c_1 & * & \cdots & * \\ 0 & & & \\ \vdots & & K & \\ 0 & & & \end{bmatrix} = \begin{bmatrix} c_1 & * & \cdots & * \\ 0 & & & \\ \vdots & & \overline{H}_2K & \\ 0 & & & \end{bmatrix} = \begin{bmatrix} c_1 & * & * & \cdots & * \\ 0 & c_2 & * & \cdots & * \\ 0 & 0 & * & \vdots & * \\ \vdots & \vdots & \vdots & & \vdots \\ 0 & 0 & * & \cdots & * \end{bmatrix}.$$

Este proceso lo repetimos n veces ($n-1$, si $m=n$), de tal manera que

$$H_nH_{n-1}\cdots H_2H_1A = R,$$

donde R es una matriz triangular superior. Entonces $A = QR$, donde $Q^T = H_nH_{n-1}\cdots H_2H_1$ es una matriz ortogonal. □

Notemos que al construir un programa para este algoritmo, no es necesario calcular explícitamente las matrices H_i, ni la matriz Q; es suficiente retener, además de R, los valores de β_i (o c_i) y los vectores $\mathbf{v}_i$, los cuales contienen toda la información necesaria.

Algoritmo 2 Vector de Householder (house)

Entrada: Dado $\mathbf{x} \in \mathbb{R}^n$.

Salida: Esta función calcula $\mathbf{v} \in \mathbb{R}^n$ con $\mathbf{v}(1) = 1$ y $\beta \in \mathbb{R}^n$ de manera que $P = I - \beta vv^T$ es ortogonal y $P\mathbf{x} = \|\mathbf{x}\|_2 \mathbf{e}_1$.

$n = length(\mathbf{x})$
$\sigma = \mathbf{x}(2:n)^T \mathbf{x}(2:n)$
$v = \begin{bmatrix} 1 \\ \mathbf{x}(2:n) \end{bmatrix}$
si $\sigma = 0$ **entonces**
  $\beta = 0$
si no
  $\mu = \sqrt{\mathbf{x}(1)^2 + \sigma}$
  si $\mathbf{x}(1) \leq 0$ **entonces**
    $\mathbf{v}(1) = \mathbf{1} - \mu$
  si no
    $\mathbf{v}(1) = -\sigma/(\mathbf{x}(1) + \mu)$
  fin si
  $\beta = 2\mathbf{v}(1)^2/(\sigma + \mathbf{v}(1)^2)$
  $\mathbf{v} = \mathbf{v}/\mathbf{v(1)}$
fin si

Algoritmo 3 Ortogonalización de Householder

Entrada: Dada $A \in \mathbb{R}^{m \times n}$.

Salida: El algoritmo encuentra las matrices de Householder $H_1, \cdots, H_n$ tales que si $Q = H_1, \cdots, H_n$ se verifica que $Q^T A = R$ es triangular superior. La parte triangular superior de A es sobrescrita por la para triangular superior de R y las componentes $j+1 : m$ del j-ésimo vector de Householder son almacenadas en $A(j+1:m, j)$, $j < m$.

para $j = 1 : n$ **hacer**
  $[v, \beta] =$ **house**$(A(j:m, j))$
  $A(j:m, j:n) = (I_{m-j+1} - \beta vv^T)A(j:m, j:n)$
  si $j < m$ **entonces**
    $A(j+1:m, j) = v(2:m-j+1)$
  fin si
fin para

Este algoritmo requiere $2n^2(m - n/3)$ operaciones [11].

Transformaciones de Givens

Definición 2.4.4. *Se denomina transformación de Givens a la transformación lineal $G : \mathbb{R}^n \to \mathbb{R}^n$, caracterizada por una matriz $G(i,j) \in \mathbb{R}^{n\times n}$ de la forma*

$$G(i,j) = \begin{array}{c} \left[\begin{array}{ccccc} 1 & \vdots & & \vdots & \\ \cdots & c & \cdots & s & \cdots \\ & \vdots & & \vdots & \\ \cdots & -s & \cdots & c & \cdots \\ & \vdots & & \vdots & 1 \end{array}\right] \begin{array}{l} \\ \leftarrow i \\ \\ \leftarrow j \\ \\ \end{array} \\ \begin{array}{ccc} & i & \quad\quad j \end{array} \end{array},$$

donde $c^2 + s^2 = 1$.

Si se tiene una transformación de Givens de $\mathbb{R}^n$ en $\mathbb{R}^n$ representada por una matriz $G(i,j)$ de la forma antes definida, con $c = cos\theta$ y $s = sen\theta$, al aplicarla a un vector cualquiera $\mathbf{x} \in \mathbb{R}^n$, éste rota un ángulo θ en el subespacio que generan los vectores $\mathbf{e}_i$ y $\mathbf{e}_j$ de $\mathbb{R}^n$ (plano (i,j)), en efecto

$$G(i,j)\mathbf{x} = \left[\begin{array}{c} \mathbf{x}_1 \\ \vdots \\ \mathbf{x}_{i-1} \\ \mathbf{x}_i cos\theta + \mathbf{x}_j sen\theta \\ \mathbf{x}_{i+1} \\ \vdots \\ \mathbf{x}_{j-1} \\ -\mathbf{x}_i sen\theta + \mathbf{x}_j cos\theta \\ \mathbf{x}_{j+1} \\ \vdots \\ \mathbf{x}_n \end{array}\right] \begin{array}{l} \\ \\ \\ \leftarrow i \\ \\ \\ \\ \leftarrow j \\ \\ \\ \end{array}$$

Si deseamos hacer cero la componente j de un vector $\mathbf{x}$, escogemos un θ tal que $-\mathbf{x}_i sen\theta + \mathbf{x}_j cos\theta = 0$, es decir

$$\tan\theta = \frac{\mathbf{x}_j}{\mathbf{x}_i},$$

o, lo que es equivalente,

$$c = \cos\theta = \frac{\mathbf{x}_i}{\sqrt{\mathbf{x}_i^2 + \mathbf{x}_j^2}} \quad \text{y} \quad s = \operatorname{sen}\theta = \frac{\mathbf{x}_j}{\sqrt{\mathbf{x}_i^2 + \mathbf{x}_j^2}}.$$

La factorización QR se deduce fácilmente a partir de las rotaciones de Givens que convierten A, en n etapas, en una matriz triangular superior R. En la etapa j, hacemos, uno a uno, cero los componentes $j+1$ a m de la columna j. A continuación, ilustramos el procedimiento en una matriz de 4×3.

$$\begin{bmatrix} X & X & X \\ X & X & X \\ X & X & X \\ X & X & X \end{bmatrix} \overset{G(1,2)}{\rightarrow} \begin{bmatrix} X & X & X \\ 0 & X & X \\ X & X & X \\ X & X & X \end{bmatrix}$$

$$\overset{G(1,3)}{\rightarrow} \begin{bmatrix} X & X & X \\ 0 & X & X \\ 0 & X & X \\ X & X & X \end{bmatrix} \overset{G(1,4)}{\rightarrow} \begin{bmatrix} X & X & X \\ 0 & X & X \\ 0 & X & X \\ 0 & X & X \end{bmatrix}$$

$$\overset{G(2,3)}{\rightarrow} \begin{bmatrix} X & X & X \\ 0 & X & X \\ 0 & 0 & X \\ 0 & X & X \end{bmatrix} \overset{G(2,4)}{\rightarrow} \begin{bmatrix} X & X & X \\ 0 & X & X \\ 0 & 0 & X \\ 0 & 0 & X \end{bmatrix} \overset{G(3,4)}{\rightarrow} \begin{bmatrix} X & X & X \\ 0 & X & X \\ 0 & 0 & X \\ 0 & 0 & 0 \end{bmatrix}$$

Claramente, si $G(p,q)$ denota la rotación de Givens, entonces $Q^T A = R$ es triangular superior y

$$Q^T = G(3,4)G(2,4)G(2,3)G(1,4)G(1,3)G(1,2).$$

Algoritmo 4 Función:[c,s]=**givens**(a,b)

Entrada: Dado dos escalares a, b.

Salida: Esta función calcula $c = cos(\theta)$ y $s = sen(\theta)$ de manera que $\begin{bmatrix} c & s \\ -s & c \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} r \\ 0 \end{bmatrix}$

si $b = 0$ **entonces**
  $c = 1$; $s = 0$
si no
  si $|b| > |a|$ **entonces**
    $r = -a/b; \quad s = 1/\sqrt{1+r^2}; \quad c = sr$
  si no
    $r = -b/a; \quad c = 1/\sqrt{1+r^2}; \quad s = cr$
  fin si
fin si

Algoritmo 5 Ortogonalización de Givens

Entrada: Dada $A \in \mathbb{R}^{m \times n}$.

Salida: El algoritmo sobrescribe A con $Q^T A = R$, donde R es triangular superior y Q es ortogonal.

para $j = 1 : n$ **hacer**
  para $i = m : -1 : j + 1$ **hacer**
    $[c, s] =$ **givens**$(A(i-1, j), A(i, j))$
    $A(i-1 : i, j : n) = \begin{bmatrix} c & s \\ -s & c \end{bmatrix}^T A(i-1 : i, j : n)$
  fin para
fin para

Este algoritmo requiere $3n^2(m - n/3)$ operaciones [11].

Algoritmo 6 PMCL y Ortogonalización de Householder (Givens)

Entrada: Dada la matriz $A \in \mathbb{R}^{m\times n}$, con $rango(A) = n$ y $b \in \mathbb{R}^m$.
Salida: Solución $\mathbf{x}$ del problema mín $\|A\mathbf{x} - \mathbf{b}\|_2$.

1: Calcular la factorización $Q^T A = R = \begin{bmatrix} R_1 \\ 0 \end{bmatrix}$, donde $Q \in \mathbb{R}^{m\times m}$ es ortogonal y $R \in \mathbb{R}^{m\times n}$ triangular superior. Si hacemos

$$Q^T\mathbf{b} = \begin{bmatrix} \mathbf{c} \\ \mathbf{d} \end{bmatrix},$$

con $\mathbf{c} \in \mathbb{R}^n$ y $\mathbf{d} \in \mathbb{R}^{m-n}$, entonces

$$\|A\mathbf{x} - \mathbf{b}\|_2^2 = \left\|Q^T A\mathbf{x} - Q^T\mathbf{b}\right\|_2^2 = \|R_1\mathbf{x} - \mathbf{c}\|_2^2 + \|\mathbf{d}\|_2^2$$

2: Resolver el sistema $R_1\mathbf{x} = \mathbf{c}$. El vector $\mathbf{x} = R_1^{-1}\mathbf{c}$ es la solución deseada (siendo el $rango(A) = rango(R_1) = n$) y $\mathbf{r}^2 = \|\mathbf{d}\|_2^2$

Método de Gram-Schmidt Clásico

Proceso de ortogonalización de Gram Schmidt: Sea $\{\mathbf{x}_1, ..., \mathbf{x}_n\}$ un conjunto de n vectores linealmente independientes. Entonces podemos construir un conjunto $\{\mathbf{v}_1, .., \mathbf{v}_n\}$ de vectores ortogonales con el siguiente proceso. Iniciamos haciendo $\mathbf{v}_1 = \mathbf{x}_1$, y en general

$$\mathbf{v}_k = \mathbf{x}_k - \sum_{i=1}^{k-1} \frac{\mathbf{x}_k \cdot \mathbf{v}_i}{\|\mathbf{v}_i\|^2}\mathbf{v}_i, \text{ para } k = 1, ..., n. \tag{2.10}$$

Si además, los vectores resultantes son unitarios, el proceso se denomina *proceso de ortonormalización.*

Teorema 2.4.3. *Si A es una matriz de $m \times n$ con columnas linealmente independientes, entonces A puede factorizarce como $A = QR$ donde Q es una matriz $m \times n$ cuyas columnas forman una base ortonormal para el espacio generado por las columnas de A, y $R \in \mathbb{R}^{n\times n}$ es una matriz triangular superior.*

Demostración. Sean $\mathbf{a}_1, ..., \mathbf{a}_n$ las columnas linealmente independientes de A, las cuales forman una base para el espacio columna de A. Mediante el proceso

de Gram-Schmidt, podemos obtener una base ortonormal $\{\mathbf{q}_1, ..., \mathbf{q}_n\}$ para el espacio generado por las columnas de A.

Los coeficientes necesarios para realizar dicha ortonormalización son almacenados en la matriz R, de manera que

$$r_{11} = \|a_1\|, \quad q_1 = a_1/r_{11},$$

y para $k = 2, ..., n$,

$$\mathbf{q}_k = \widehat{\mathbf{q}}_k/r_{kk}, \quad \text{con} \quad \widehat{\mathbf{q}}_k = \mathbf{a}_k - \sum_{i=1}^{k-1} r_{ik}\mathbf{q}_i, \tag{2.11}$$

donde

$$r_{ik} = \mathbf{q}_i^T \mathbf{a}_k \text{ para } i = 1, ..., k-1, \quad r_{kk} = \|\widehat{\mathbf{q}}_k\|.$$

Por lo tanto, $\mathbf{a}_k = \sum_{i=1}^{k} r_{ik}\mathbf{q}_i$, $k = 1, ..., n$. □

El siguiente algoritmo construye la factorización QR encontrada.

Algoritmo 7 Gram Schmidt Clásico

Entrada: Dada $A \in \mathbb{R}^{m\times n}$ con $rango(A) = n$.
Salida: El algoritmo calcula $A = QR$, donde $Q \in \mathbb{R}^{m\times n}$ tiene columnas ortonormales y $R \in \mathbb{R}^{n\times n}$ es triangular superior.

1: **para** $k = 1, ..., n$, **hacer**
2: $\quad \mathbf{q}_k = \mathbf{a}_k$;
3: $\quad$ **para** $i = 1, ..., k-1$ **hacer**
4: $\quad\quad r_{ik} = \mathbf{q}_i^T \mathbf{a}_k$,
5: $\quad\quad \mathbf{q}_k = \mathbf{q}_k - r_{ik}\mathbf{q}_i$,
6: $\quad$ **fin para**
7: $\quad r_{kk} = \|\mathbf{q}_k\|_2$,
8: $\quad \mathbf{q}_k = \mathbf{q}_k/r_{kk}$,
9: **fin para**

El método de Gram-Schmidt Clásio requiere $2mn^2$ operaciones [11].

Método de Gram Schmidt modificado

Desafortunadamente, la forma de obtener la matriz Q con el método de Gram Schmidt no es estable, en el sentido de que los $\mathbf{q}_k$ pierden ortogonalidad, ya que el numerador en el cálculo de las columnas $\mathbf{q}_k$ se puede anular usando precisión fija en la aritmética de punto flotante. No obstante, el algoritmo se puede modificar fácilmente para que se comporte mejor desde el punto de vista numérico. Por ello, se usa el *Método de Gram Schmidt modificado*, que sólo cambia la forma de calcular las columnas $\mathbf{q}_k$, y que es numéricamente estable y equivalente al clásico.

Algoritmo 8 Gram Schmidt Modificado

Entrada: Dada $A \in \mathbb{R}^{m\times n}$ con $rango(A) = n$.
Salida: El algoritmo calcula $A = QR$, donde $Q \in \mathbb{R}^{m\times n}$ tiene columnas ortonormales y $R \in \mathbb{R}^{n\times n}$ es triangular superior.

1: **para** $k = 1, ..., n$, **hacer**
2: $\quad$ *%Normalizar* $\mathbf{a}_k$;
3: $\quad r_{kk} = \|\mathbf{a}_k\|_2$,
4: $\quad \mathbf{q}_k = \mathbf{a}_k / r_{kk}$;
5: $\quad$ **para** $j = k + 1, ..., n$ **hacer**
6: $\quad\quad r_{kj} = \mathbf{q}_k^T \mathbf{a}_j$,
7: $\quad\quad \mathbf{a}_j = \mathbf{a}_j - r_{kj}\mathbf{q}_k$,
8: $\quad$ **fin para**
9: **fin para**

Este algoritmo requiere $2mn^2$ operaciones [11].

Algoritmo 9 PMCL y Algoritmo de Gram-Schmidt clásico (Modificado)

Entrada: Dada $A \in \mathbb{R}^{m\times n}$ con $rango(A) = n$ y $b \in \mathbb{R}^m$.
Salida: Solución $\mathbf{x}$ del problema mín $\|A\mathbf{x} - \mathbf{b}\|_2$

1: Calcular $A = QR$, donde $Q \in \mathbb{R}^{m\times n}$ con $Q^TQ = I_n$ y $R \in \mathbb{R}^{n\times n}$ es triangular superior.
2: Calcular $y = Q^Tb$
3: Resolver el sistema $Rx = y$ por sustitución hacia atrás. Entonces $\mathbf{x}$ es la solución del problema de mínimos cuadrados.

Comparación de los algoritmos que resuelven el PMCL

En la tabla (2.1), se comparan, en lo que respecta al número de operaciones involucradas en sus procesos, todos los métodos para resolver el problema de mínimos cuadrados lineales que hemos presentado en esta sección.

MÉTODO	OPERACIONES
Ecuaciones Normales	$mn^2 + \frac{n^3}{3}$
Transformaciones de Householder	$2mn^2 - \frac{2n^3}{3}$
Método de Gram Schmidt	$2mn^2$
Método de Gram Schmidt Modificado	$2mn^2$
Transformaciones de Givens	$3mn^2 - n^3$

Tabla 2.1: Número de operaciones necesarias para resolver el PMCL con rango completo.

- Como el PMCL consiste en encontrar un $\mathbf{x}$ tal que $\mathbf{b}$ esté cerca de $A\mathbf{x}$, es decir que minimice una norma residual $\mathbf{r} = \mathbf{b} - A\mathbf{x}$. La interpretación y naturaleza de la solución de este problema varía dependiendo de la norma que se use. La norma más comúnmente usada es la norma euclidiana (norma l_2). Entonces, la *solución* de mínimos cuadrados de $A\mathbf{x} = \mathbf{b}$ es el vector $\mathbf{x}$ que hace de $\|r\|_2 = \|\mathbf{b} - A\mathbf{x}\|_2$ un mínimo. Al contrario, minimizar la norma l_1 o l_∞ es equivalente a un problema de programación lineal no diferenciable, cuya solución necesita una técnica de iteración.

- Debido a las posibles dificultades numéricas del uso de las ecuaciones normales, los modernos métodos de solución del problema de mínimos cuadrados lineales se han desarrollado basándose en las transformaciones ortogonales, que preservan las distancias euclideas y no empeoran las condiciones de la matriz A.

- Si la matriz es de rango completo y está bien condicionada, el algoritmo basado en las ecuaciones normales sería el preferible para resolver el PMCL. Si la matriz no está muy bien condicionada, el basado en la

factorización QR con los algortimos de Gram-Schmidt produciría resultados más fiables aunque el costo es el doble que el del método de ecuaciones normales si m es mucho mayor que n y aproximadamente el mismo si $n = m$.

- La factorización QR obtenida con el algoritmo de Gram-Schmid clásico es numéricamente inestable: pequeños errores de redondeo pueden ocasionar que los vectores calculados no sean ortogonales. Una ligera modificación del algoritmo evita este problema. Aunque estos procedimientos son matemáticamente equivalentes, cuando se implementan en punto flotante, el método de Gram-Schmidt modificado resulta mucho más estable que el clásico [31].

- La factorización QR puede hacerse también por medio de transformaciones de Householder, de modo aún más estable que con Gram-Schmidt modificado. Lo primero que hay que observar es que mientras los métodos de ortonormalización de Gram-Schmidt producen una factorización QR reducida, el de Householder produce una factorización completa. En el primer caso, $\widehat{Q}$ es de tamaño $m \times n$ y $\widehat{R}$ es $n \times n$. En el método de Householder, Q es de tamaño $m \times m$ y R será de tamaño $m \times n$.

- En cuanto a las Transformaciones de Householder y Givens, la pregunta que nos surge es: ¿Por qué utilizar Givens y no Householder? La respuesta a esta pregunta se basa en considerar la estructura de la matriz A del problema: si ésta es densa, es decir, muchos de sus coeficientes son distintos de cero, el método de Householder es sin duda el más aconsejable; si, por el contrario, la estructura de A es dispersa, convendrá centrarse en hacer cero sólo aquellos elementos no nulos en las columnas correspondientes, por lo que, a priori, si hay pocos de éstos, el método de Givens es más ventajoso [7].

PMCL para el caso de rango deficiente

Cuando $A \in \mathbb{R}^{m \times n}$ tiene rango deficiente, la solución del problema de mínimos cuadrados lineales no es única, y para calcular una de ellas, hay que imponer algún requerimiento extra, como es que la solución tenga norma

mínima.

Además, la factorización ortogonal de la forma $A = QR$ presenta grandes dificultades prácticas siendo necesario el intercambio de columnas y encontrar el criterio que permita decidir cuándo la base ortonormal que forma las columnas de Q está calculada, ya que esto puede no detectarse cuando la deficiencia de rango no es muy grande. El siguiente teorema es una herramienta importante en este caso.

Teorema 2.4.4. *Dada la matriz $A \in \mathbb{R}^{m\text{x}n}$ de rango $r < n$, existe una permutación, representada por la matriz P, y una matriz ortogonal $Q \in \mathbb{R}^{m\text{x}m}$, tales que*

$$Q^T AP = \begin{bmatrix} R_{11} & R_{12} \\ 0 & 0 \end{bmatrix}, \tag{2.12}$$

donde $R_{11} \in \mathbb{R}^{r\times r}$ es triangular superior con todos los elementos de la diagonal principal positivos (obsérvese que no se requiere que $m \geq n$).

Demostración. Si el $rango(A) = r$, siempre es posible elegir una permutación P tal que

$$AP = [A_1, A_2],$$

donde las columnas de A_1 son linealmente independientes. Sea la descomposición ortogonal QR de A_1 tal que

$$Q^T A_1 = \begin{bmatrix} R_{11} \\ 0 \end{bmatrix}.$$

Entonces

$$Q^T AP = [A_1, A_2] = \begin{bmatrix} R_{11} & R_{12} \\ 0 & R_{22} \end{bmatrix}$$

Como $rango(Q^T AP) = rango(A) = r$, la matriz $R_{22} = 0$, de lo contrario $Q^T AP$ tendría más de r filas linealmente independientes, lo cual es imposible. □

2.4.3. QR con Pivoteo de Columna ($A \in \mathbb{R}^{m\times n}$, $m > n$ ó $m < n$)

Supongamos que tenemos $A \in \mathbb{R}^{m\times n}$ que tiene rango $r < n$, con $m \geq n$. Si le aplicamos la descomposición QR con pivoteo (Teorema 2.4.4), obtendremos

una factorización $AP = QR$, donde Q es una matriz ortogonal, P una matriz de permutación y R es una matriz triangular de la forma

$$R = \begin{bmatrix} R_{11} & R_{12} \\ 0 & 0 \end{bmatrix}, \tag{2.13}$$

donde R_{11} es una matriz triangular superior de orden r.

Usando esta descomposición, el PMCL puede ser resuelto. En efecto, para cualquier $\mathbf{x} \in \mathbb{R}^n$, tenemos

$$\begin{aligned} \|A\mathbf{x} - \mathbf{b}\|_2^2 &= \left\| \left(Q^T A P\right) \left(P^T \mathbf{x}\right) - \left(Q^T \mathbf{b}\right) \right\|_2^2 \\ &= \|R_{11}\mathbf{y} - (\mathbf{c} - R_{12}\mathbf{z})\|_2^2 + \|\mathbf{d}\|_2^2 \end{aligned}$$

donde $P^T\mathbf{x} = \begin{bmatrix} \mathbf{y} \\ \mathbf{z} \end{bmatrix}$ y $Q^T\mathbf{b} = \begin{bmatrix} \mathbf{c} \\ \mathbf{d} \end{bmatrix}$. Entonces si $\mathbf{x}$ minimiza, debemos tener

$$\mathbf{x} = P \begin{bmatrix} R_{11}^{-1}\left(\mathbf{c} - R_{12}\mathbf{z}\right) \\ \mathbf{z} \end{bmatrix}. \tag{2.14}$$

Si $\mathbf{z}$ es cero en esta expresión, entonces tendremos la *solución básica*

$$\mathbf{x}_B = P \begin{bmatrix} R_{11}^{-1}\mathbf{c} \\ 0 \end{bmatrix} = P \begin{bmatrix} R_{11}^{-1} & 0 \\ 0 & 0 \end{bmatrix} Q^T\mathbf{b}.$$

Si $\mathbf{x}$ tiene la forma de la ecuación (2.14), entonces $\mathbf{x}$ minimiza $\|A\mathbf{x} - \mathbf{b}\|_2^2$, pero, entre todos los minimizadores, nos interesa encontrar aquella $\mathbf{x}$ que tenga norma mínima; para esto, observemos que si tenemos la factorización $AP = QR$, donde R tiene la forma de (2.13), entonces

$$Q^T A P = R.$$

Podemos utilizar transformaciones de Householder para eliminar el factor R_{12} de la matriz R, y obtenemos

$$Q^T A P \Pi = \begin{bmatrix} \overline{R}_{11} & 0 \\ 0 & 0 \end{bmatrix} = T.$$

De esta forma, la matriz A nos queda como $A = QT\Pi^T P^T$. Por lo tanto

$$\begin{aligned} \|A\mathbf{x} - \mathbf{b}\|_2^2 &= \left\|QT\Pi^T P^T\mathbf{x} - \mathbf{b}\right\|_2^2 = \left\|Q\left(T\Pi^T P^T\mathbf{x} - Q^T\mathbf{b}\right)\right\|_2^2 \\ &= \left\|T\Pi^T P^T\mathbf{x} - Q^T\mathbf{b}\right\|_2^2 \end{aligned}$$

y si consideramos $\mathbf{y} = \Pi^T P^T \mathbf{x}$, entonces $\mathbf{x} = P\Pi\mathbf{y}$, tenemos que

$$\begin{aligned}\|A\mathbf{x} - \mathbf{b}\|_2^2 &= \left\| \begin{bmatrix} \overline{R}_{11} & 0 \\ 0 & 0 \end{bmatrix} \mathbf{y} - Q^T\mathbf{b} \right\|_2^2 = \left\| \begin{bmatrix} \overline{R}_{11} & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} \mathbf{w} \\ \mathbf{s} \end{bmatrix} - \begin{bmatrix} \mathbf{c} \\ \mathbf{d} \end{bmatrix} \right\|_2^2 \\ &= \left\| \overline{R}_{11}\mathbf{w} - \mathbf{c} \right\|_2^2 + \|\mathbf{d}\|_2^2 .\end{aligned}$$

Cláramente, el primer sumando de la expresión anterior se anula, si resolvemos el sistema $\overline{R}_{11}\mathbf{w} = \mathbf{c}$ por sustitución hacia atrás, $\mathbf{w} = \overline{R}_{11}^{-1}\mathbf{c}$. Además, $\mathbf{y}$ tendrá norma mínima cuando tenga la forma

$$\mathbf{y} = \begin{bmatrix} \mathbf{w} \\ 0 \end{bmatrix} = \begin{bmatrix} \overline{R}_{11}^{-1}\mathbf{c} \\ 0 \end{bmatrix} = \begin{bmatrix} \overline{R}_{11}^{-1} & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} \mathbf{c} \\ \mathbf{d} \end{bmatrix} = \begin{bmatrix} \overline{R}_{11}^{-1} & 0 \\ 0 & 0 \end{bmatrix} Q^T\mathbf{b}.$$

De esta manera, podemos encontrar la solución al problema de mínimos cuadrados de norma mínima mediante la expresión

$$\mathbf{x}_{LS} = P\Pi\mathbf{y} = P\Pi \begin{bmatrix} \overline{R}_{11}^{-1} & 0 \\ 0 & 0 \end{bmatrix} Q^T\mathbf{b}. \tag{2.15}$$

La solución básica no es la solución de norma 2 mínima, a menos que la submatriz R_{12} sea cero, dado que

$$\|\mathbf{x}_{LS}\|_2 = \min_{\mathbf{z}\in\mathbb{R}^{n-r}} \left\| \mathbf{x}_B - \Pi \begin{bmatrix} \overline{R}_{11}^{-1}R_{12} \\ -I_{n-r} \end{bmatrix} \mathbf{z} \right\|_2 .$$

Algoritmo 10 Factorización QR con pivoteo

Entrada: Dada la matriz $A \in \mathbb{R}^{m\times n}$ con $rango(A) = r < n$ y $b \in \mathbb{R}^m$.

Salida: El algoritmo calcula la factorización $AP = QR$ definida por (2.13). El elemento a_{ij} se sobreescribe por r_{ij} $(i \leq j)$. La permutación $P = [e_{c1}, ..., e_{cn}]$ se determina de acuerdo a la escogencia de la máxima norma de columna en cada paso.

1: $c_j := j \quad (j = 1, 2, ..., n)$,
2: $r_j := \sum_{i=1}^{m} a_{ij}^2 \quad (j = 1, 2, ..., n)$,
3: **para** $k = 1, ..., n$, **hacer**
4: determinar p con $(k \leq p \leq n)$ tal que $r_p = \underset{k\leq j\leq n}{\text{máx}}\, r_j$.
5: **si** $r_p = 0$ **entonces**
6: **stop** ;
7: **si no**
8: intercambiar c_k y c_p, r_k y r_p, y a_{ik} y a_{ip}, para $i = 1, ..., m$.
9: determinar una matriz de Householder $\widehat{Q}_k$ tal que

$$\widehat{Q}_k \begin{bmatrix} a_{kk} \\ \vdots \\ \vdots \\ a_{mk} \end{bmatrix} = \begin{bmatrix} * \\ 0 \\ \vdots \\ 0 \end{bmatrix}.$$

10: $A := diag\left(I_{k-1}, \widehat{Q}_k\right) A$
11: $r_j := r_j - a_{kj}^2 \quad (j = k + 1, ..., n)$
12: **fin si**
13: **fin para**

Este algoritmo requiere $4mnr - 2r^3(m + n) + 4r^3/3$ operaciones, donde $r = rango(A)$ [11].

2.4.4. La Descomposición en Valores Singulares (DVS)

En esta sección, discutimos algunos de los detalles asociados con el uso de la DVS para resolver el PMCL de rango deficiente. En el siguiente capítulo, mostramos como encontrar dicha descomposición.

El siguiente resultado establece la existencia de la DVS para cualquier matriz rectangular.

Teorema 2.4.5. *[11] Sea $A \in \mathbb{R}^{m\times n}$ con rango$(A) = r \leq n$, entonces existen matrices ortogonales $U \in \mathbb{R}^{m\times m}$ y $V \in \mathbb{R}^{n\times n}$ y una matriz $\Sigma \in \mathbb{R}^{m\times n}$, tales que*

$$A = U\Sigma V^T, \quad \Sigma = \begin{bmatrix} \Sigma_1 & 0 \\ 0 & 0 \end{bmatrix}, \quad \Sigma_1 = \begin{bmatrix} \sigma_1 & 0 & \cdots & 0 \\ 0 & \sigma_2 & \cdots & 0 \\ & & \ddots & \\ 0 & 0 & \cdots & \sigma_r \end{bmatrix} \in \mathbb{R}^{r\times r}.$$

Los elementos no nulos en la diagonal principal del bloque Σ_1 satisfacen $\sigma_1 \geq \sigma_2 \geq ... \geq \sigma_r > 0$ y se denominan valores singulares de A.

Observemos que si U y V se escriben como

$$U = [\mathbf{u}_1, ..., \mathbf{u}_r] \quad y \quad V = [\mathbf{v}_1, ..., \mathbf{v}_r],$$

la DVS de A se puede reescribir de la forma

$$A = U\Sigma V^T = \sum_{i=1}^{r} \sigma_i \mathbf{u}_i \mathbf{v}_i^T \tag{2.16}$$

Así, se tiene que la DVS de A induce a una descomposición de A como la suma de $r =$ rango(A) matrices de rango 1 (estas son las matrices $\sigma_i \mathbf{u}_i \mathbf{v}_i$).

La DVS de A es de interés, ya que permite definir la solución al PMCL en función de las matrices U, V y Σ mencionadas en el Teorema 2.4.5, como lo resume el siguiente teorema.

Teorema 2.4.6. *Consideremos el problema general de mínimos cuadrados lineales:*

$$\min_{\mathbf{x}\in\mathbf{X}} \|\mathbf{x}\|_2, \quad \mathbf{X} = \{\mathbf{x} \in \mathbb{R}^n : \|A\mathbf{x} - \mathbf{b}\|_2 \quad \textit{es mínima}\}, \tag{2.17}$$

donde $A \in \mathbb{R}^{m\times n}$ y rango$(A) = r \leq \min(m, n)$. Este problema siempre tiene una solución única $\mathbf{x}$, la cual se puede escribir en términos de las matrices resultantes de la DVS de A como:

$$\mathbf{x} = V \begin{bmatrix} \Sigma_1^{-1} & 0 \\ 0 & 0 \end{bmatrix} U^T \mathbf{b}.$$

Demostración. En vista de que V y U son matrices ortogonales, se tiene que $VV^T = I_n$. Así

$$\mathbf{b} - A\mathbf{x} = \mathbf{b} - AVV^T\mathbf{x}.$$

Por otro lado

$$\|\mathbf{b} - A\mathbf{x}\|_2 = \left\|U^T(\mathbf{b} - A\mathbf{x})\right\|_2 = \left\|U^T(\mathbf{b} - AVV^T\mathbf{x})\right\|_2$$

Ahora, consideramos

$$\mathbf{z} = V^T\mathbf{x} = \begin{bmatrix} \mathbf{z}_1 \\ \mathbf{z}_2 \end{bmatrix}, \quad \mathbf{c} = U^T\mathbf{b} = \begin{bmatrix} \mathbf{c}_1 \\ \mathbf{c}_2 \end{bmatrix} = \begin{bmatrix} \left[U^T\mathbf{b}\right]_r \\ \left[U^T\mathbf{b}\right]_{m-r} \end{bmatrix},$$

donde $\mathbf{z}_1$, $\mathbf{c}_1 \in \mathbb{R}^r$. Con lo anterior, podemos rescribir la norma del residual $\mathbf{b} - A\mathbf{x}$ como

$$\begin{aligned} \|\mathbf{b} - A\mathbf{x}\|_2 &= \left\|U^T(\mathbf{b} - AVV^T\mathbf{x})\right\|_2 \\ &= \left\|U^T\mathbf{b} - (U^TAV)V^T\mathbf{x}\right\|_2 \\ &= \left\| \begin{bmatrix} \mathbf{c}_1 \\ \mathbf{c}_2 \end{bmatrix} - \begin{bmatrix} \Sigma_1 & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} \mathbf{z}_1 \\ \mathbf{z}_2 \end{bmatrix} \right\|_2 \\ &= \left\| \begin{bmatrix} \mathbf{c}_1 - \Sigma_1\mathbf{z}_1 \\ \mathbf{c}_2 \end{bmatrix} \right\|_2 \end{aligned}$$

Podemos observar que para $\mathbf{c}_1$ y $\mathbf{c}_2$ fijos, el residual se minimiza cuando $\mathbf{c}_1 - \Sigma_1\mathbf{z}_1 = 0$, esto es, $\mathbf{z}_1 = \Sigma_1^{-1}\mathbf{c}_1$, mientras que $\mathbf{z}_2$ puede ser cualquier valor. En particular, si escogemos $\mathbf{z}_2 = 0$, también se minimiza $\|\mathbf{z}\|_2$, lo cual implica que se minimiza $\|\mathbf{x}\|_2$; por lo tanto, el valor de $\mathbf{x}$ que resuelve el problema (2.17) es

$$\begin{aligned} \mathbf{x} = V\mathbf{z} &= V \begin{bmatrix} \mathbf{z}_1 \\ \mathbf{z}_2 \end{bmatrix} \\ &= V \begin{bmatrix} \Sigma_1^{-1}\mathbf{c}_1 \\ 0 \end{bmatrix} \\ &= V \begin{bmatrix} \Sigma_1^{-1}\left[U^T\mathbf{b}\right]_r + 0\left[U^T\mathbf{b}\right]_{m-r} \\ 0\left[U^T\mathbf{b}\right]_r + 0\left[U^T\mathbf{b}\right]_{m-r} \end{bmatrix} \\ &= V \begin{bmatrix} \Sigma_1^{-1} & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} \left[U^T\mathbf{b}\right]_r \\ \left[U^T\mathbf{b}\right]_{m-r} \end{bmatrix} \\ &= V \begin{bmatrix} \Sigma_1^{-1} & 0 \\ 0 & 0 \end{bmatrix} U^T\mathbf{b}. \end{aligned}$$

□

La construcción de la solución del problema de mínimos cuadrados generalizados en la demostración del teorema 2.4.6 define la matriz

$$C = V \begin{bmatrix} \Sigma_1^{-1} & 0 \\ 0 & 0 \end{bmatrix} U^T, \tag{2.18}$$

la cual resulta ser A^+, tal como se deduce del siguiente resultado.

Teorema 2.4.7. *[2] La matriz*

$$C = V \begin{bmatrix} \Sigma_1^{-1} & 0 \\ 0 & 0 \end{bmatrix} U^T$$

satisface las condiciones de Penrose y por consiguiente, $C = A^+$.

Con este último resultado, y a partir del Teorema 2.4.6, se deriva que la solución del PMCL se puede expresar, en términos de A^+, como

$$\mathbf{x} = A^+\mathbf{b}.$$

Para el caso de $rango(A) = n$ $(m \geq n)$, $A^+ = \left(A^TA\right)^{-1}A^T$. Si el $rango(A) = m$ $(m \leq n)$, entonces $A^+ = A^T\left(AA^T\right)^{-1}$. Si $m = n = rango\,(A)$, entonces $A^+ = A^{-1}$.

Observemos que el cálculo de la DVS provee, desde el punto de vista numérico, un método directo para la resolución del PMCL. Sin embargo, el cálculo de la DVS tiene un altísimo costo computacional, lo cual hace que el enfoque de la DVS sea poco atractivo para la resolución del PMCL, en el caso general. No obstante, para el caso de matrices densas de rango deficiente, la DVS es la primera elección para la resolución del PMCL.

Capítulo 3

Métodos Directos para Calcular la Matriz Pseudoinversa

3.1. Introducción

En los últimos años, una literatura considerable en relación con el cálculo de pseudoinversas ha acompañado el renacimiento del interés por ésta teoría.

En este capítulo, describimos enfoques distintos para el cálculo de ésta matriz. El primer método calcula la pseudoinversa a partir de la descomposición en valores singulares; el segundo está basado en la Factorización de Cholesky Generalizada; el tercero, en el proceso de Ortogonalización de Gramm-Schmidt; el cuarto, en el método de la Escalonada Reducida; el quinto se basa en las ideas de la Proyección del Gradiente, y el último es un procedimiento derivado del Teorema de Cayley-Hamilton. Finalmente, realizamos un análisis numérico comparativo de los algoritmos que calculan la matriz pseudoinversa.

3.2. Método basado en la DVS

En esta sección, estudiaremos la obtención de la matriz psudoinversa a partir de la descomposición en valores singulares de una matriz rectangular con el algoritmo de Golub-Reinsh. Este algoritmo se divide en dos fases. La primera es un proceso finito que consiste en la bidiagonalización de la matriz mediante transformaciones de Householder. La segunda fase es un proceso iterativo que diagonaliza la matriz bidiagonal obtenida. Este método es álta-

mente eficiente para la calcular la DVS. Ello se debe a su rapidez y su buen comportamiento con la mayoría de tipos de matrices.

Veamos ahora paso a paso, cómo obtener la matriz de valores singulares de una matriz dada.

- Reducción de la matriz a forma bidiagonal

Una matriz $B \in \mathbb{R}^{m\times n}$ con $m \geq n$ es bidiagonal si $b_{i,j} = 0$ si $i > j$ o $i < j-1$. Es decir, una matriz bidiagonal B tiene la forma siguiente

$$\left[\begin{array}{ccccc} * & * & & & \\ & * & * & & 0 \\ & & * & \ddots & \\ & 0 & & \ddots & * \\ & & & & * \\ \hline & & 0 & & \end{array}\right], \tag{3.1}$$

Aquí y en lo que sigue, $*$ denota posibles elementos no nulos de la matriz.

Teorema 3.2.1. *[7] Sea la matriz $A \in \mathbb{R}^{m\times n}$, $m \geq n$. Existen matrices ortogonales $\widehat{U} \in \mathbb{R}^{m\times m}$ y $\widehat{V} \in \mathbb{R}^{n\times n}$ ambas producto de un número finito de matrices de Householder, y una matriz bidiagonal $B \in \mathbb{R}^{m\times n}$ tales que*

$$A = \widehat{U}B\widehat{V}^T.$$

La demostración del Teorema 3.2.1 es constructiva. El primer paso es crear ceros en la primera columna y fila de A. Sea $\widehat{U}_1 \in \mathbb{R}^{m\times m}$ una matriz de Householder tal que

$$\widehat{U}_1 \begin{pmatrix} a_{11} \\ a_{21} \\ \vdots \\ a_{m1} \end{pmatrix} = \begin{pmatrix} \widehat{a}_{11} \\ 0 \\ \vdots \\ 0 \end{pmatrix}.$$

Entonces la primera columna de $\widehat{U}_1 A$ está formada por ceros, salvo la entrada $(1,1)$. Ahora, tomamos $(\widehat{a}_{11}, \widehat{a}_{12}, ..., \widehat{a}_{1n})$, la primera fila de $\widehat{U}_1 A$, y sea $\widehat{V}_1 \in \mathbb{R}^{n\times n}$ una matriz de la forma

$$\widehat{V}_1 = \begin{pmatrix} 1 & 0 & \cdots & 0 \\ 0 & & & \\ \vdots & & \overline{V}_1 & \\ 0 & & & \end{pmatrix}.$$

donde $\overline{V}_1$ es una matriz de Householder de $\mathbb{R}^{(n-1)\times(n-1)}$, tal que

$$(\widehat{a}_{12}, \widehat{a}_{13}, ..., \widehat{a}_{1n})\overline{V}_1 = (*, 0, ..., 0).$$

Por ser la primera columna de $\widehat{V}_1$, igual a e_1, la primera columna de $\widehat{U}_1 A$ no se ve modificada al multiplicar por $\widehat{V}_1$ por la derecha. Por tanto, la primera fila de $\widehat{U}_1 A \widehat{V}_1$ está formada por ceros salvo las dos primeras entradas, teniendo así

$$\widehat{U}_1 A \widehat{V}_1 = \begin{pmatrix} * & * & \cdots & 0 \\ 0 & & & \\ \vdots & & \widehat{A} & \\ 0 & & & \end{pmatrix}.$$

El segundo paso del algoritmo es análogo al primero pero actuando sobre la submatriz $\widehat{A}$. Las matrices de Householder usadas en el segundo paso no destruyen los ceros creados en el paso anterior porque $\widehat{U}_2$ se define a partir de una matriz de householder de tamaño $(m-1)\times(m-1)$ ampliada con la primera fila y columna de la identidad y $\widehat{V}_2$ se define a partir de una matriz de Householder $(n-2)\times(n-2)$ ampliada con las dos primera filas y columnas de la identidad. Tras los dos primeros pasos, se tiene

$$\widehat{U}_2 \widehat{U}_1 A \widehat{V}_1 \widehat{V}_2 = \begin{pmatrix} * & * & 0 & 0 & \cdots & 0 \\ 0 & * & * & 0 & \cdots & 0 \\ 0 & 0 & & & & \\ \vdots & \vdots & & & \widehat{\widehat{A}} & \\ 0 & 0 & & & & \end{pmatrix}.$$

El tercer paso actúa sobre la submatriz $\widehat{\widehat{A}} \in \mathbb{R}^{(m-2)\times(n-2)}$ y así sucesivamente. Si $m > n$ hay que realizar un total de n pasos y, si $m = n$, solo se necesitan $n-1$. Tras completar el proceso, se obtiene una matriz de la forma

$$\widehat{U}_n \cdots \widehat{U}_2 \widehat{U}_1 A \widehat{V}_1 \widehat{V}_2 \cdots \widehat{V}_{n-2} = \left[\begin{array}{ccccc} * & * & & & \\ & * & * & & 0 \\ & & * & \ddots & \\ & 0 & & \ddots & * \\ & & & & * \\ \hline & & 0 & & \end{array}\right] = B, \qquad (3.2)$$

teniendo en cuenta que en los dos últimos pasos solo se aplican matrices de Householder por la izquierda para hacer ceros en las dos últimas columnas de la matriz y que además si $m = n$, $\widehat{U}_n = I_m$.

Por tanto, tomando

$$\widehat{U} = \widehat{U}_1 \widehat{U}_2 \cdots \widehat{U}_n \quad \text{y} \quad \widehat{V} = \widehat{V}_1 \widehat{V}_2 \cdots \widehat{V}_{n-2}$$

y teniendo en cuenta que las matrices de Householder son matrices simétricas, se tiene $\widehat{U}^T A \widehat{V} = B$, o equivalentemente,

$$A = \widehat{U} B \widehat{V}^T.$$

En algunas aplicaciones, como en el problema de mínimos cuadrados lineales, $A \in \mathbb{R}^{m \times n}$ con m mucho mayor que n. En este caso, es más eficiente realizar la reducción a forma bidiagonal en dos pasos.

En el primer paso, se realiza una factorización QR de A

$$A = QR = \left[\begin{array}{cc} Q_{mn} & Q_{m,m-n} \end{array}\right] \left[\begin{array}{c} \widehat{R} \\ 0 \end{array}\right],$$

donde $Q_{mn} \in \mathbb{R}^{m \times n}$, $Q_{m,m-n} \in \mathbb{R}^{m \times (m-n)}$ y $\widehat{R} \in \mathbb{R}^{n \times n}$ es triangular superior. Esto involucra multiplicaciones por matrices de Householder solo por el lado izquierdo.

En el segundo paso, $\widehat{R}$ es reducida a forma bidiagonal $\widehat{R} = \widehat{\widehat{U}} \widehat{B} \widehat{\widehat{V}}^T$, donde todas las matrices son cuadradas de dimensión $n \times n$, teniendo así,

$$A = \left[\begin{array}{cc} Q_{mn} & Q_{m,m-n} \end{array}\right] \left[\begin{array}{cc} \widehat{\widehat{U}} & 0 \\ 0 & I_{m-n} \end{array}\right] \left[\begin{array}{c} \widehat{B} \\ 0 \end{array}\right] \widehat{\widehat{V}}^T.$$

Tomando

$$\begin{aligned}\widehat{U} &= \begin{bmatrix} Q_{mn} & Q_{m,m-n} \end{bmatrix} \begin{bmatrix} \widehat{\widehat{U}} & 0 \\ 0 & I_{m-n} \end{bmatrix} \\ &= \begin{bmatrix} Q_{mn}\widehat{\widehat{U}} & Q_{m,m-n} \end{bmatrix} = \begin{bmatrix} \widehat{U}_{mn} & \widehat{U}_{m,m-n} \end{bmatrix} \in \mathbb{R}^{m\times m},\end{aligned}$$

donde $\widehat{U}_{mn} \in \mathbb{R}^{m\times n}$ está formada por las n primeras columnas de $\widehat{U}$ y $\widehat{U}_{m,m-n} \in \mathbb{R}^{m\times(m-n)}$ está formada por las columnas restantes de $\widehat{U}$,

$$B = \begin{bmatrix} \widehat{B} \\ 0 \end{bmatrix} \in \mathbb{R}^{m\times n}$$

y

$$\widehat{V} = \widehat{\widehat{V}} \in \mathbb{R}^{n\times n}$$

se tiene $A = \widehat{U}B\widehat{V}^T$. No obstante, también es cierto que $A = \widehat{U}_{mn}\widehat{B}\widehat{V}^T$.

La ventaja de este procedimiento, en el caso de matrices con $m >> n$, es que las matrices de Householder por la derecha son aplicadas a la matriz pequeña $\widehat{R}$ en lugar de a la matriz A, por lo que el costo operativo es menor. La desventaja es que las matrices de Householder destruyen la forma triangular de $\widehat{R}$ y la mayor parte de las multiplicaciones por la izquierda deben ser repetidas, pero en la matriz pequeña $\widehat{R}$, ver [32] para un análisis más detallado.

Algoritmo 11 Bidiagonalización de Householder

Entrada: Dada $A \in \mathbb{R}^{m\times n}$ $(m \geq n)$.

Salida: El siguiente algoritmo sobreescribe A como $\widehat{U}^T A\widehat{V} = B$, donde B es bidiagonal superior y $\widehat{U} = \widehat{U}_1 \cdots \widehat{U}_n$ y $\widehat{V} = \widehat{V}_1 \cdots \widehat{V}_{n-2}$ son ortogonales.

1: **para** $j = 1, ..., n$ **hacer**

2: Calcular una matriz de Householder $\widehat{U}_j \in \mathbb{R}^{(m-j+1)\times(m-j+1)}$ tal que

$$\widehat{U}_j A(j:m, j) = (\|A(j:m, j)\|_2, 0, \cdots, 0)^T$$

3: Aplicar la matriz de Householder $\widehat{U}_j$ a la matriz

$$A(j:m, j:n) \leftarrow \widehat{U}_j A(j:m, j:n)$$

4: **si** $j \leq n-2$ **entonces**

5: Calcular una matriz de Householder $\widehat{V}_j \in \mathbb{R}^{(n-j)\times(n-j)}$, tal que

$$\widehat{V}_j A(j:j+j:n)^T = (\left\|A(j, j+1:n)^T\right\|_2, 0, \cdots, 0)^T$$

6: Aplicar la matriz de Householder $\widehat{V}_j$ a la matriz

$$A(j:m, j+1:n) \leftarrow A(j:m, j+1:n)\widehat{V}_j$$

7: **fin si**

8: **fin para**

Este algoritmo requerirá $O(4mn^2 - \frac{4}{3}n^3)$ operaciones [11].

- El algoritmo QR implícito para matrices bidiagonales

Una vez hallada la matriz bidiagonal cuadrada $\widehat{B}$, el problema de calcular la descomposición en valores singulares de A se reduce a calcular los valores singulares de la matriz $\widehat{B}$. Si $\widehat{B} = \widetilde{U}\widehat{\Sigma}\widetilde{V}^T$ con $\widetilde{U}, \widetilde{V} \in \mathbb{R}^{n\times n}$ ortogonales y $\widehat{\Sigma} \in \mathbb{R}^{n\times n}$ diagonal, es la descomposición en valores singulares de $\widehat{B}$, entonces

$$
\begin{aligned}
A &= \widehat{U}_{mn}\widehat{B}\widehat{V}^T = \widehat{U}_{mn}\widetilde{U}\widehat{\Sigma}\widetilde{V}^T\widehat{V}^T \\
&= \begin{bmatrix} \widehat{U}_{mn} & \widehat{U}_{m,m-n} \end{bmatrix} \begin{bmatrix} \widetilde{U} & 0 \\ 0 & I_{m-n} \end{bmatrix} \begin{bmatrix} \widehat{\Sigma} \\ 0 \end{bmatrix} \widetilde{V}^T\widehat{V}^T \\
&= \begin{bmatrix} \widehat{U}_{mn}\widehat{U} & \widehat{U}_{m,m-n} \end{bmatrix} \begin{bmatrix} \widehat{\Sigma} \\ 0 \end{bmatrix} \left(\widehat{V}\widetilde{V}\right)^T,
\end{aligned}
$$

que es la descomposición en valores singulares de A, tal como se definió en el Teorema 2.4.5, aunque desde el punto de vista práctico es suficiente considerar

$$A = (\widehat{U}_{mn}\widetilde{U})\widehat{\Sigma}(\widehat{V}\widetilde{V})^T.$$

Por conveniencia en la notación, a partir de ahora en lugar de $\widehat{B}$ utilizaremos $B \in \mathbb{R}^{n\times n}$ para referirnos a la matriz bidiagonal cuadrada

$$
B = \begin{pmatrix}
\beta_1 & \gamma_1 & & & \\
 & \beta_2 & \gamma_2 & 0 & \\
 & & \ddots & \ddots & \\
 & 0 & & \beta_{n-1} & \gamma_{n-1} \\
 & & & & \beta_n
\end{pmatrix}.
$$

Diremos que B es una matriz propiamente bidiagonal si $\beta_i \neq 0$ y $\gamma_i \neq 0$ para todo i.

Si B no es una matriz propiamente bidiagonal, se puede reducir el problema de encontrar la descomposición en valores singulares de B a dos subproblemas de dimensión menor. En [10], se pueden encontrar los detalles.

Asumimos entonces, sin pérdida de generalidad, que B es propiamente bidiagonal, y pasamos a describir el algoritmo QR implícito para encontrar la descomposición en valores singulares de B.

Si $B \in \mathbb{R}^{n\times n}$ es una matriz propiamente bidiagonal, entonces BB^T y B^TB son matrices propiamente tridiagonales, y se pueden calcular sus autovalores mediante la iteración QR con desplazamiento. El algoritmo que vamos a desarrollar es equivalente al algoritmo QR, aplicado tanto a BB^T como a B^TB, pero sin la construcción explícita de las matrices producto.

Comenzamos el primer paso del algoritmo QR implícito eligiendo el desplazamiento adecuado. La submatriz inferior derecha de dimensión 2×2 de BB^T es

$$\begin{pmatrix} \beta_{n-1}^2 + \gamma_{n-1}^2 & \beta_n \gamma_{n-1} \\ \beta_n \gamma_{n-1} & \beta_n^2 \end{pmatrix}. \tag{3.3}$$

Calculamos los autovalores de esta submatriz y tomamos como desplazamiento σ el autovalor de (3.3) más cercano a β_n^2, (desplazamiento de Wilkinson para BB^T). Podríamos escoger σ también a partir de B^TB, pero la forma de BB^T es algo más simple.

Una iteración del algoritmo QR con desplazamiento σ aplicado a B^TB comienza hallando la factorización QR

$$B^TB - \sigma I = QR. \tag{3.4}$$

Para realizar una iteración implícita, necesitamos la primera columna de Q. Por ser la matriz R triangular superior, la primera columna de Q es proporcional a la primera columna de $B^TB - \sigma I$ que viene dada por

$$\begin{pmatrix} \beta_1^2 - \sigma \\ \gamma_1 \beta_1 \\ 0 \\ \vdots \\ 0 \end{pmatrix}. \tag{3.5}$$

Tomamos entonces V_{12}, una rotación de Givens en el plano $(1, 2)$, cuya primera columna sea proporcional a (3.5). Multiplicando B por V_{12} por la derecha, modifica solo las dos primeras columnas de B y se crea una entrada no nula en la posición $(2, 1)$.

Ahora buscamos una rotación U_{12}^T, en las filas $(1, 2)$ tal que $U_{12}^T B V_{12}$ tenga un cero en la posición $(2, 1)$. Esta operación actúa en las filas 1 y 2 y crea un nuevo valor no nulo en la posición $(1, 3)$. Tomamos ahora la rotación V_{23} que actúa sobre las columnas 2 y 3 de modo que $U_{12}^T B V_{12} V_{23}$ tenga un cero en la posición $(1, 3)$, pero aparece un valor no nulo en la posición $(3, 2)$.

Continuando de este modo, aplicando una rotación $U_{i,i+1}^T$ por la izquierda que anula el elemento $(i + 1, i)$ y genera un nuevo elemento no nulo en la

posición $(i, i+2)$ seguida de una matriz $V_{i+1,i+2}$ por la derecha que anula el elemento $(i, i+2)$ y genera un elemento no nulo en la posición $(i+2, i+1)$ para $i = 2, ..., n-2$, se consigue que tras aplicar una última rotación $U_{n-1,n}^T$ por la izquierda se anule el elemento $(n, n-1)$ y se obtenga una matriz bidiagonal

$$\widehat{B} = U_{n-1,n}^T \cdots U_{23}^T U_{12}^T B V_{12} V_{23} \cdots V_{n-1,n}. \tag{3.6}$$

Tomando $U = U_{12}U_{23}\cdots U_{n-1,n}$ y $V = V_{12}V_{23}\cdots V_{n-1,n}$ podemos escribir (3.6) como

$$\widehat{B} = U^T B V.$$

Con esto finaliza una iteración del algoritmo QR implícito. Además, tenemos $\widehat{B}\widehat{B}^T = U^T B B^T U$ y $\widehat{B}^T\widehat{B} = V^T B^T B V$, por lo que $\widehat{B}\widehat{B}^T$ y $\widehat{B}^T\widehat{B}$ son esencialmente las mismas matrices que habríamos obtenido si hubiésemos dado una iteración del algoritmo QR con desplazamiento σ partiendo de las matrices BB^T y B^TB para aproximar sus autovalores.

En esta afirmación juega un papel crucial el hecho de que la matriz V y la matriz Q de la ecuación (3.4) tengan la primera columna igual salvo posiblemente el signo. Un análisis detallado puede verse en [33].

El algoritmo resultante que muestra los pasos de cada una de las etapas que acabamos de describir se denomina Algoritmo de Golub- Kahan.

Algoritmo 12 Golub-Kahan: etapa k de la DVS

Entrada: Dada una matriz bidiagonal $B \in \mathbb{R}^{m\times n}$, $m \geq n$ sin ningún elemento nulo ni en la diagonal principal ni en la sobrediagonal inmediata a esa diagonal principal.

Salida: Este algoritmo calcula la matriz bidiagonal $\widehat{B}$ que reemplaza a B, talque $\widehat{B} = U^T BV$, donde U y V son matrices ortogonales, siendo V esencialmente la que se obtendrá al aplicar el algoritmo QR con desplazamiento implícito a $T = B^T B$ para calcular sus autovalores.

1: Determinar el autovalor σ de la submatriz 2×2 de $T = B^T B$ que forman $t_{n-1\,n-1}$, $t_{n-1\,n}$,$t_{n\,n-1}$ y t_{nn} más próximo en valor a t_{nn}. Hacer

$$y = t_{11} - \sigma$$
$$z = t_{12}.$$

2: Para $k = 1, ..., n-1$:

a) Determinar los parámetros de $G(k, k+1)$, $c = \cos\theta$ y $s = \sin\theta$, tales que

$$\begin{bmatrix} y & z \end{bmatrix} \begin{bmatrix} c & s \\ -s & c \end{bmatrix} = \begin{bmatrix} \times & 0 \end{bmatrix}.$$

b) Hacer

$$B = BG(k, k+1)$$
$$y = b_{kk}$$
$$z = b_{k+1,k}.$$

c) Determinar los parámetros de $G(k, k+1)$, $c = \cos\theta$ y $s = \sin\theta$, tales que

$$\begin{bmatrix} c & s \\ -s & c \end{bmatrix}^T \begin{bmatrix} y \\ z \end{bmatrix} = \begin{bmatrix} \times \\ 0 \end{bmatrix}.$$

d) Hacer $B = G(k, k+1)^T B$.

e) Si $k < n-1$, hacer $y = b_{k,k+1}$ y $z = b_{k,k+2}$

Si tomamos como matriz B, la matriz $\widehat{B}$ y repetimos la iteración QR implíci-

ta, las matrices BB^T y B^TB tenderán a una forma diagonal y las entradas de la diagonal principal convergerán a los autovalores. La utilización del desplazamiento de Wilkinson hace que las entradas $(n, n-1)$ y (n, n) de BB^T y B^TB converjan la primera a 0, y la segunda a un autovalor. Por supuesto no se trabaja con BB^T ni con B^TB sino con B. La rápida convergencia de BB^T y B^TB hacia una matriz diagonal se traduce en la convergencia de γ_{n-1} a 0 y de β_n a un valor singular de B.

Una vez que γ_{n-1} sea menor que una tolerancia fijada, puede considerarse como 0 y reducir el problema a uno de dimensión $(n-1)\times(n-1)$, ignorando la última fila y última columna de la matriz B. Repitiendo el proceso, se encuentran todos los valores singulares de B.

El esquema completo del algoritmo que hemos decrito para obtener numéricamente los valores singulares de una matriz A se denomina algoritmo de Golub-Reinsch y se muestra a continuación.

Algoritmo 13 DVS por el Método de Golub-Reinsch

Entrada: Dada $A \in \mathbb{R}^{m\times n}$ con $m \geq n$ y ϵ, un pequeño múltiplo de la unidad de redondeo.

Salida: Este algoritmo reemplaza la matriz A por $U(D+E)V^T$, donde $U \in \mathbb{R}^{m\times n}$ y $V \in \mathbb{R}^{n\times n}$ son ortogonales y $D \in \mathbb{R}^{m\times n}$ es diagonal y E satisface $\|E\|_2 \approx \epsilon \|A\|_2$.

1: Bidiagonalizar la matriz A mediante transformaciones de Householder (Algoritmo 11). Hacer

$$B \leftarrow (U_1 \cdots U_n)^T A (V_1 \cdots V_{n-2}).$$

2: Realizar las siguientes operaciones:

a) Hacer $a_{i,i+1} = 0$, si $|a_{i,i+1}| \leq \epsilon(|a_{i,i}| + |a_{i+1,i+1}|)$ para todo $i = 1, ..., n-1$.

b) Determinar el mayor q y el menor p tales que si

$$A = \begin{bmatrix} B_{11} & 0 & 0 \\ 0 & B_{22} & 0 \\ 0 & 0 & B_{33} \end{bmatrix} \begin{matrix} p \\ n \quad -p- \quad q, \\ q \end{matrix}$$

B_{33} es diagonal y B_{22} tiene todos sus elementos de la sobrediagonal próxima a la diagonal principal distintos de cero.

c) Si $q = n$, parar; el procedimiento ha concluido.

d) Si cualquier elemento en la diagonal de B_{22} es cero, anular el elemento en la sobrediagonal de la misma fila i y comenzar de nuevo en a).

e) Aplicar el algoritmo 12 a B_{22}. Hacer

$$B = diag(I_p, U, I_{q+m-n})^T B diag(I_p, V, I_q).$$

Comenzar de nuevo en a).

Finalmente, el proceso para obtener la matriz pseudoinversa a partir de la descomposición en valores singulares de una matriz rectangular de rango deficiente se presenta en el siguiente algoritmo.

Algoritmo 14 Pseudoinversa por descomposición en valores singulares (DVS)

Entrada: Dada $A \in \mathbb{R}^{m\times n}$, $rango(A) = r$.
Salida: $A^+ \in \mathbb{R}^{n\times m}$.

1: Calcular, con el algoritmo 13, la descomposición en valores singulares de A tal que $A = U\Sigma V^T$, donde $U \in \mathbb{R}^{m\times m}$, y $V \in \mathbb{R}^{n\times n}$ son ortogonales y

$$\Sigma = \begin{bmatrix} \Sigma_1 & 0 \\ 0 & 0 \end{bmatrix}, \quad \Sigma_1 = \begin{bmatrix} \sigma_1 & 0 & \cdots & 0 \\ 0 & \sigma_2 & \cdots & 0 \\ & & \ddots & \\ 0 & 0 & \cdots & \sigma_r \end{bmatrix} \in \mathbb{R}^{r\times r}.$$

2: Calcular $A^+ = V\Sigma^+U^T$ donde

$$\Sigma^+ = \begin{bmatrix} \Sigma_1^{-1} & 0 \\ 0 & 0 \end{bmatrix}$$

3.3. Factorización de Cholesky Generalizada

Pierre Courrieu[1] propuso el método Fast Computation of Moore-Penrose Inverse Matrices [5] para reducir el tiempo de cálculo de la pseudoinversa basado en una generalización de la factorización de Cholesky [4].

Sea $A \in \mathbb{R}^{m\times n}$ con $rango(A) = r \leq n$. Courrieu consideró que si A^TA es una matriz $n \times n$ simétrica semidefinida positiva, existiría una matriz triangular superior no singular R con $n - r$ filas de ceros y $R^TR = A^TA$.

La matriz R puede encontrarse usando el resultado fundamental para la factorización de cholesky generalizada.

Teorema 3.3.1. *[6] Sea $A^TA = G$ una matriz simétrica semidefinida positiva de orden $n \times n$. Entonces existe una matriz triangular superior R tal que $R^TR = A^TA = G$, $r_{ii} \geq 0, \quad 1 \leq i \leq n$, y si para un índice i se tiene*

[1]Pierre Courrieu es un investigador del Centro Nacional de Investigaciones Científicas (CNRS) de la Universidad de Provence (Francia). Actualmente trabaja con Psicólogos y Neurocientíficos [5].

que $r_{ii} = 0$*, entonces* $r_{ij} = 0$*,* $1 \leq j \leq n$*. Además, la matriz* R *con estas propiedades es única.*

El algoritmo para calcular el factor de Cholesky generalizado R definido en el Teorema 3.3.1, es una simple variante del usual algoritmo de la factorización de Cholesky con la misma complejidad computacional. Sin embargo, la generalización tiene la ventaja de proporcionar un factor adecuado en todos los casos, incluso si la matriz $A^T A$ es singular [6].

Algoritmo 15 Factor Generalizado de Cholesky [6]

Entrada: Dada la matriz $A \in \mathbb{R}^{m \times n}$ con $rango(A) = r \leq n$ y $G = A^T A \in \mathbb{R}^{n \times n}$.

Salida: El algoritmo proporciona una matriz triangular superior R con r filas no nulas y $n - r$ filas de ceros.

1: $r_{ij} \leftarrow 0, \quad 1 \leq i, j \leq n$ {inicialización de R}
2: $r_{11} = \sqrt{g_{11}}$
3: **para** $j \leftarrow 2$ to n **hacer**
4: **para** $i \leftarrow 1$ to j **hacer**
5: **si** $i = j$ **entonces**
6: $r_{ii} \leftarrow \sqrt{g_{ii} - \sum_{k=1}^{i-1} r_{ki}^2}$
7: **si no**, **si** $r_{ii} > 0$ **entonces**
8: $r_{ij} \leftarrow \left(g_{ij} - \sum_{k=1}^{i-1} r_{ki} r_{kj}\right) / r_{ii}$
9: **si no**
10: $r_{ij} = 0$ {como resultado de la inicialización}
11: **fin si**
12: **fin para**
13: **fin para**

La complejidad del algoritmo es $O\left(n^3\right)$ [6].

El resultado fundamental para el cálculo de la matriz pseudoinversa por el método de Courrieu está dado por el siguiente teorema.

Teorema 3.3.2. *[6] Sea la matriz* $A \in \mathbb{R}^{m \times n}$ *con* $m \geq n$ *y* $G = A^T A$*. Al calcular la factorización generalizada de Cholesky* $G = R^T R$*, usando el Algoritmo 15 y eliminando todas las filas de ceros de* R*, se obtiene la matriz*

$L^T \in \mathbb{R}^{r\times n}$ de rango completo r, con $r < n$, tal que $LL^T = G$, entonces:

$$A^+ = L(L^TL)^{-1}\left(L^TL\right)^{-1}L^TA^T.$$

Demostración. Consideremos una relación general en cuanto a la pseudoinversa del producto de matrices EF dada por la Ecuación (A.1) del apéndice

$$(EF)^+ = F^T\left(E^TEFF^T\right)^+E^T. \tag{3.7}$$

Si hacemos que $E = A$ y $F = I$, de (3.7) obtenemos

$$(A)^+ = \left(A^TA\right)^+A^T. \tag{3.8}$$

Ahora, si $E = L$ y $F = L^T$ en (3.7), obtenemos

$$A^+ = (LL^T)^+A^T = L\left(L^TLL^TL\right)^+L^TA^T$$

donde L^TL es invertible porque L^T es de rango completo y, por el Teorema A.0.1 del apéndice, finalmente podemos concluir que

$$A^+ = L\left(L^TL\right)^{-1}\left(L^TL\right)^{-1}L^TA^T. \tag{3.9}$$

□

Si A es una matriz $m \times n$, con $m < n$, basta utilizar la relación $A^+ = ((A^T)^+)^T$. Notemos que (3.9) proporciona una sencilla fórmula para la pseudoinversa de cualquier matriz simétrica semidefinida positiva, y si L^T es de rango completo $r = n$, entonces $A^+ = A^{-1}$.

Algoritmo 16 Pseudoinversa mediante la Factorización de Cholesky Generalizada

Entrada: $A \in \mathbb{R}^{m\times n}, rango(A) = r$
Salida: $A^+ \in \mathbb{R}^{n\times m}$

1: Calcular el Factor Generalizado de Cholesky R de la matriz simétrica semidefinida positiva $G = A^TA$, utilizando el Algoritmo 15 tal que $R^TR = A^TA$.
2: Remover la fila de ceros de la matriz R para obtener la matriz L^T de rango r.
3: Calcular $(A)^+ = L\left(L^TL\right)^{-1}\left(L^TL\right)^{-1}L^TA^T$.

La función *geninv* en el apédice propuesta por Courrie [5], proporciona todos los detalles de implementación del proceso anterior, en el código de MATLAB.

3.4. Método de Ortogonalización de Gram Schmidt (OGS)

El método de OGS propuesto por Rust, Burrus y Schneeberger [28] es considerado una extensión del método de ortogonalización convencional para calcular la pseudoinversa de Moore-Penrose.

Sea $A \in \mathbb{R}^{m\times n}$ una matriz de rango $r \leq \text{mín}(m, n)$. Siempre es posible reordenar las columnas de A para que las primeras r sean linealmente independientes y las columnas restantes sean combinaciones lineales de las primeras r. Esto es lo mismo que decir que para alguna matriz de permutación P:

$$AP = [R\ S]\,, \tag{3.10}$$

donde $R \in \mathbb{R}^{m\times r}$ es una matriz de rango r y las columnas de $S \in \mathbb{R}^{m\times(n-r)}$ son combinaciones lineales de las columnas de R (Teorema A.0.2 del apéndice), esto es

$$S = RU, \text{ para algún } U. \tag{3.11}$$

Teorema 3.4.1. *[28] La matriz S tiene una factorización única de la forma*

$$S = RU$$

y la matriz $U \in \mathbb{R}^{r\times(n-r)}$ está dada por

$$U = R^{+}S.$$

Demostración. (Existencia de la factorización). Supongamos que

$$S = [\mathbf{s}_1, \mathbf{s}_2, ..., \mathbf{s}_{n-r}]\,.$$

Como cada columna de S es una combinación lineal de las columnas de R, $\mathbf{s}_i = R\mathbf{u}_i$ para algún $\mathbf{u}_i$, $i = 1, ..., n-r$. Por lo tanto,

$$\begin{aligned} S &= [R\mathbf{u}_1, R\mathbf{u}_2, ..., R\mathbf{u}_{n-r}] \\ &= R\,[\mathbf{u}_1, \mathbf{u}_2, ..., \mathbf{u}_{n-r}]\,; \end{aligned}$$

es decir, $S = RU$, donde $U = [\mathbf{u}_1, \mathbf{u}_2, ..., \mathbf{u}_{n-r}]$.

(*Unicidad de la factorización*). El vector $\mathbf{u}_i$ es único para cada $\mathbf{s}_i$, $i = 1, ..., n-r$, porque las columnas de R forman un conjunto linealmente independiente por lo tanto, U es única.

□

Teorema 3.4.2. *[1] Si P es una matriz de permutación (probablemente un producto de matrices elementales de permutación) y A^+ es la pseudoinversa de A, entonces*

$$[AP]^+ = P^T A^+.$$

Demostración. Necesitamos solamente verificar que la matriz $P^T A^+$ satisface las condiciones de Moore-Penrose. Teniendo en cuenta que $PP^T = P^T P = I$, tenemos que

(a) $(AP)\left(P^T A^+\right)(AP) = AP$

(b) $P^T A^+ (AP) P^T A^+ = P^T A^+$

(c) $\left[(AP)\left(P^T A^+\right)\right]^T = (AA^+)^T = AA^+ = (AP)\left(P^T A^+\right)$

(d) $$\begin{aligned}\left[\left(P^T A^+\right)(AP)\right]^T &= \left[P^T (A^+ A) P\right]^T = P^T (A^+ A)^T P \\ &= P^T (A^+ A) P \\ &= \left(P^T A^+\right)(AP).\end{aligned}$$ □

Como P es una matriz ortogonal, por (3.10) y (3.11),

$$AP = [R\ RU] = R[I\ U]$$

y, por el teorema anterior,

$$A^+ = \left([R\ RU]\, P^T\right)^+ = P(R[I\ U])^+. \tag{3.12}$$

Ahora, dirigimos nuestra atención a la pseudoinversa de un producto. Si X y Y son matrices no singulares, $(XY)^{-1} = Y^{-1}X^{-1}$, pero generalmente no es cierto que $(XY)^+ = Y^+X^+$.

Greville [12], encontró muchos casos particulares en los que la propiedad

$$(XY)^+ = Y^+X^+ \tag{3.13}$$

se cumple. Por ejemplo, esto es cierto si

(a) $X^T X = I$ ó

(b) $YY^T = I$ ó

(c) $Y = X^T$ ó

(d) $Y = X^+$ ó

(e) Si X tiene todas las columnas linealmente independientes (rango columna completo) y Y todas las filas linealmente independientes (rango fila completo).

Teniendo en cuenta que, en la ecuación (3.12), el rango de $[I\ U] \in \mathbb{R}^{r\times n}$ es r, al igual que el rango de $[I\ U]\,[I\ U]^T = I + UU^T$ (Teorema A.0.3), y que las filas de $[I\ U]$ son linealmente independientes, tenemos que $[I\ U]^+ = [I\ U]^T \left(I + UU^T\right)^{-1}$, así por la propiedad (3.13-e)

$$\begin{aligned}(R[I\ U])^+ &= [I\ U]^+ R^+ \\ &= [I\ U]^T \left(I + UU^T\right)^{-1} R^+\end{aligned} \tag{3.14}$$

y por tanto, usando (3.12) y (3.14), la pseudoinversa de A se calcula como

$$A^+ = P\,[I\ U]^T \left(I + UU^T\right)^{-1} R^+. \tag{3.15}$$

Esta ecuación es el punto de partida para calcular la pseudoinversa con procedimientos basados en la Ortogonalización de Gram-Schmidt (OGS); es decir, la OGS es usada para calcular P, R^+, U, y $\left(I + UU^T\right)^{-1}$.

(a) *Evaluación de P*

Si denotamos las columnas de A por $\mathbf{a}_1, \mathbf{a}_2, ..., \mathbf{a}_n$, y realizamos una OGS, sin normalizar, tenemos:

$$\begin{aligned}\mathbf{c}_1^* &= \mathbf{a}_1 \\ \mathbf{c}_j^* &= \mathbf{a}_j - \sum_{i\in M_j} \frac{\mathbf{a}_j^T \mathbf{c}_i^*}{\|\mathbf{c}_i^*\|^2} \mathbf{c}_i^*\end{aligned}$$

donde $M_j = \{i : i \leq j-1 \quad \text{y} \quad \mathbf{c}_i^* \neq 0\}$.

Si los vectores $\mathbf{c}_1^*, \mathbf{c}_2^*, ..., \mathbf{c}_n^*$ son permutados, empezando por los vectores no nulos (de los cuales habrá r), la matriz de permutación aplicada a los vectores $\mathbf{a}_1, \mathbf{a}_2, ..., \mathbf{a}_n$, los organiza de tal manera que los primeros r vectores sean linealmente independientes, mientras que los últimos $n-r$ sean combinaciones lineales de los primeros r, ya que $\mathbf{c}_j^* = 0$, si y solo si, $\mathbf{a}_j$ es una combinación lineal de las anteriores columnas de A. Por lo tanto, si P es cualquier matriz para la cual

$$[\mathbf{c}_1^*, \mathbf{c}_2^*, ..., \mathbf{c}_n^*]\, P = [\mathbf{c}_1, \mathbf{c}_2, ..., \mathbf{c}_n]\,, \tag{3.16}$$

donde

$$\|\mathbf{c}_j\| = \begin{cases} > 0 & \text{si } j = 1, 2, .., r \\ = 0 & \text{si } j = r+1, ..., n \end{cases},$$

entonces $AP = [\mathbf{a}_1, \mathbf{a}_2, ..., \mathbf{a}_n]\, P = [R\ S]$ donde $R \in \mathbb{R}^{m \times r}$ de rango r y las columnas de S son combinaciones lineales de las columnas de R.

(b) *Cálculo de* R^+

Los vectores (no nulos), $\mathbf{c}_1, \mathbf{c}_2, ..., \mathbf{c}_r$ definidos anteriormente, representan una OGS de las columans de R. Si hacemos

$$Q = \left[\frac{\mathbf{c}_1}{\|\mathbf{c}_1\|}, \frac{\mathbf{c}_2}{\|\mathbf{c}_2\|}, ..., \frac{\mathbf{c}_r}{\|\mathbf{c}_r\|} \right], \tag{3.17}$$

entonces el espacio columna de Q es igual al espacio columna de la matriz R, es decir, $C_Q = C_R$ y por el Teorema (A.0.4 del apéndice), existe una matriz B de orden $r \times r$ tal que

$$RB = Q.$$

De hecho, como R tiene rango r, $B = \left(R^T R\right)^{-1} R^T Q$, y como las columnas de Q son ortonormales ($Q^T Q = I$), y B es no singular $\left(Q^T RB = I\right)$, asi que

$$R = QB^{-1}.$$

Por la propiedad (3.13-a), obtenemos

$$R^+ = BQ^+ = B\left(Q^T Q\right)^{-1} Q^T = BQ^T.$$

(c) *Cálculo de* B y U

Denotamos las columnas de R por $\mathbf{r}_1, \mathbf{r}_2, ..., \mathbf{r}_r$ y las columnas de S por $\mathbf{s}_1, \mathbf{s}_2, ..., \mathbf{s}_{n-r}$. Los vectores $[\mathbf{c}_1, \mathbf{c}_2, ..., \mathbf{c}_r, \mathbf{c}_{r+1}, ..., \mathbf{c}_n]$ definidos en (3.16) representan una OGS no normalizada de $[\mathbf{r}_1, \mathbf{r}_2, ..., \mathbf{r}_r, \mathbf{s}_1, ..., \mathbf{s}_{n-r}]$. En efecto

$$\begin{aligned} \mathbf{c}_1 &= \mathbf{r}_1 \\ \mathbf{c}_j &= \mathbf{r}_j - \sum_{i=1}^{j-1} \frac{\mathbf{r}_j^T \mathbf{c}_i}{\|\mathbf{c}_i\|^2} \mathbf{c}_i \qquad j = 2, ..., r \end{aligned} \tag{3.18}$$

y

$$0 = \mathbf{c}_{r+j} = \mathbf{s}_j - \sum_{i=1}^{r} \frac{\mathbf{s}_j^T \mathbf{c}_i}{\|\mathbf{c}_i\|^2} \mathbf{c}_i \qquad j = 1, ..., n-r \tag{3.19}$$

De (3.18), deducimos (por inducción sobre j) que

$$\mathbf{c}_j = \sum_{i=1}^{j} \gamma_{ij} \mathbf{r}_i \qquad j = 1, ..., r, \tag{3.20}$$

donde

$$\gamma_{ij} = \begin{cases} 0 & i > j \\ 1 & i = j \\ -\sum_{\alpha=i}^{j-1} \frac{\mathbf{r}_j^T \mathbf{c}_\alpha}{\|\mathbf{c}_\alpha\|^2} \gamma_{i\alpha} & i < j. \end{cases} \tag{3.21}$$

Por otro lado, (3.19) muestra que

$$\mathbf{s}_j = \sum_{i=1}^{r} \omega_{ij} \mathbf{r}_i, \tag{3.22}$$

donde ω_{ij} se obtiene por la sustitución de (3.20) en (3.19)

$$\begin{aligned} \mathbf{s}_j &= \sum_{\alpha=1}^{r} \frac{\mathbf{s}_j^T \mathbf{c}_\alpha}{\|\mathbf{c}_\alpha\|^2} \left(\sum_{i=1}^{\alpha} \gamma_{i\alpha} \mathbf{r}_i \right) \\ &= \sum_{i=1}^{r} \left(\sum_{\alpha=i}^{r} \frac{\mathbf{s}_j^T \mathbf{c}_\alpha}{\|\mathbf{c}_\alpha\|^2} \gamma_{i\alpha} \right) \mathbf{r}_i. \end{aligned}$$

$$\omega_{ij} = \sum_{\alpha=i}^{\mathbf{r}} \frac{\mathbf{s}_j^T \mathbf{c}_\alpha}{\|\mathbf{c}_\alpha\|^2} \gamma_{i\alpha} \qquad i = 1, 2, ..., r; \quad j = 1, 2, ..., n-r. \tag{3.23}$$

De (3.20) y (3.17), vemos que

$$Q = \left[\frac{\mathbf{c}_1}{\|\mathbf{c}_1\|}, \frac{\mathbf{c}_2}{\|\mathbf{c}_2\|}, ..., \frac{\mathbf{c}_r}{\|\mathbf{c}_r\|} \right] = RB,$$

donde B es una matriz $r \times r$ cuya (i, j)-ésima entrada es

$$\beta_{ij} = \frac{\gamma_{ij}}{\|c_j\|}, \tag{3.24}$$

mientras que a partir de (3.22), concluimos que

$$S = RU,$$

donde U es la matriz de dimensión $r \times (n-r)$ cuya (i, j)-ésima entrada es ω_{ij}.

Notemos que (3.6) define γ_{ij} en términos de $\gamma_{ij-1}, \gamma_{ij-2}, ..., \gamma_{ii}$. Los cálculos los llevamos a cabo convenientemente en el siguiente orden

$$\gamma_{11}; \quad \gamma_{22}, \gamma_{12}; \quad \gamma_{33}, \gamma_{23}, \gamma_{13}; \quad \gamma_{44}, \gamma_{34}, \gamma_{24}, \gamma_{14}; \quad \text{etc.}$$

(d) *Evaluación de* $(I + UU^T)^{-1}$

Esta inversión se logra a través de una OGS adicional.

Teorema 3.4.3. *[1] Si U es una matriz $r \times k$ ($k = (n-r)$) y se lleva a cabo una OGS en las columnas de*

$$\begin{bmatrix} U \\ I \end{bmatrix},$$

la matriz resultante de vectores ortonormales es

$$V = \begin{bmatrix} V_1 \\ V_2 \end{bmatrix},$$

donde

$$V_2 V_2^T = \left(I + U^T U\right)^{-1}$$

y

$$I - V_1 V_1^T = \left(I + UU^T\right)^{-1}.$$

Comentario 3.4.1. *El rango de* $\begin{bmatrix} U \\ I \end{bmatrix}$ *es* k *ya que* $\begin{bmatrix} U \\ I \end{bmatrix}$ *tiene el mismo rango de* $\begin{bmatrix} U \\ I \end{bmatrix}^T \begin{bmatrix} U \\ I \end{bmatrix} = I + U^T U \in \mathbb{R}^{k \times k}$ *que es no singular (Teorema A.0.5). Además, cuando se realiza la OGS sobre las columnas de* $\begin{bmatrix} U \\ I \end{bmatrix}$*, se generan solamente vectores no nulos.*

Demostración. Sea $H = \begin{bmatrix} U \\ I \end{bmatrix}$ con rango k ($H^+H = I$, Teorema A.0.2) y $C_V = C_H$, tal que la ecuación

$$HZ = V$$

tiene una solución $Z = H^+V$ por el Torema (A.0.4).

Como las columnas de V son ortonormales, $V^TV = I$, por lo tanto $V^+ = \left(V^TV\right)^+ V^T = V^T$. Puesto que $C_V = C_H$,

$$HH^+ = VV^+ = VV^T.$$

Además

$$\begin{aligned} ZZ^T &= H^+VZ^T \\ &= H^+(VV^T)(H^+)^T \\ &= (H^TH)^+ \quad \text{por el Teorema (A.0.3), } rango(H) = rango(H^TH) = r, \\ &= (H^TH)^{-1} \quad \text{donde } H^TH \text{ es cuadrada y no singular.} \end{aligned} \tag{3.25}$$

Por lo tanto:

$$HZ = \begin{bmatrix} U \\ I \end{bmatrix} Z = \begin{bmatrix} V_1 \\ V_2 \end{bmatrix},$$

de donde tenemos

$$UZ = V_1$$

y

$$Z = V_2.$$

Además,

$$V_2V_2^T = ZZ^T = \left(H^TH\right)^{-1} = \left(I + U^TU\right)^{-1}.$$

La segunda parte del teorema se sigue de la identidad [28]

$$\left(I + UU^T\right)^{-1} = I - U(U^TU + I)^{-1}U^T$$

y del hecho que

$$V_1 = UZ = UV_2.$$

Por tanto:

$$\left(I + UU^T\right)^{-1} = I - U(V_2V_2^T)U^T = I - V_1V_1^T$$

□

Algoritmo 17 Pseudoinversa por el Método de Ortogonalización de Gram Schmidt (OGS)

Entrada: $A \in \mathbb{R}^{m\times n}$, $rango(A) = r$
Salida: $A^+ \in \mathbb{R}^{n\times m}$

1: Realizar una OGS no normalizada en las columnas de A. Llamar a este conjunto de vectores $[\mathbf{c}_1^*, \mathbf{c}_2^*, ..., \mathbf{c}_n^*]$.
2: Permutar los $\mathbf{c}_j^*$ de modo que $[\mathbf{c}_1, \mathbf{c}_2, ..., \mathbf{c}_n] = [\mathbf{c}_1^*, \mathbf{c}_2^*, ..., \mathbf{c}_n^*]\, P$ donde

$$\mathbf{c}_j \neq 0 \quad j = 1, 2, .., r$$

$$\mathbf{c}_j = 0 \quad j = r+1, ..., n.$$

3: Calcular γ_{ij} para $j = 1, ..., r$; $i = 1, ..., j$, de acuerdo con (3.6).
4: Calcular las matrices B y U. $B \in \mathbb{R}^{r\times r}$ es la matriz cuya (i,j)-ésima entrada es $\beta_{ij} = \frac{\gamma_{ij}}{\|\mathbf{c}_j\|}$ y $U \in \mathbb{R}^{r\times(n-r)}$ cuya (i,j)-ésima entrada es ω_{ij} dada por (3.23).
5: Realizar una OGS normalizada en las columnas de $\begin{bmatrix} U \\ I \end{bmatrix}$ obteniendo la matriz de vectores ortonormales $V = \begin{bmatrix} V_1 \\ V_2 \end{bmatrix}$ y calcular $M = \left(I + UU^T\right)^{-1} = I - V_1V_1^T$.
6: $Q = \left[\frac{\mathbf{c}_1}{\|\mathbf{c}_1\|}, ..., \frac{\mathbf{c}_r}{\|\mathbf{c}_r\|}\right]$.
7: Finalmente, $A^+ = P\begin{bmatrix} I & U \end{bmatrix}^T MBQ^T$.

3.5. Método de la Escalonada Reducida

A. Ben Israel, S.J. Wersan [16] y Ben Noble [20] describen métodos basados en la Eliminación de Gaus-Jordan para calcular la pseudoinversa de un matriz. Sin embargo, para reducir el costo computacional, proponemos el método de la escalonada reducida para hallar la pseudoinversa.

Si $A \in \mathbb{R}^{m\times n}$ es una matriz de rango r, existe una matriz no singular E y una matriz de permutación P, tal que

$$EA^TAP = \begin{bmatrix} I & L \\ 0 & 0 \end{bmatrix} = \begin{bmatrix} H \\ 0 \end{bmatrix}, \tag{3.26}$$

donde las matrices L y $H = [I \quad L]$ se determinan por E, A^TA y P [16].

En efecto, existen muchas E y P. Por ejemplo, si P diagonaliza a A^TA, de tal forma que los valores propios no nulos de A^TA aparezcan en la esquina superior izquierda

$$P^TA^TAP = \begin{bmatrix} D & 0 \\ 0 & 0 \end{bmatrix},$$

entonces

$$E = \begin{bmatrix} D^{-1} & 0 \\ 0 & I \end{bmatrix} P^T;$$

por tanto,

$$EA^TAP = \begin{bmatrix} D^{-1} & 0 \\ 0 & I \end{bmatrix} \left(P^TA^TAP\right) = \begin{bmatrix} I & 0 \\ 0 & 0 \end{bmatrix},$$

que es una versión más fuerte de (3.26). Sin embargo, el método descrito aqui requiere sólamente que E y P sean escogidos para sastisfacer la ecuación (3.26) y está basado en el siguiente teorema.

Teorema 3.5.1. *[1] Si E, no singular, y P, ortogonal, son elegidos para satisfacer la ecuación (3.26), entonces*

$$A^+ = P\left(EA^TAP\right)^+ EA^T.$$

Demostración. La ecuación

$$A^T AX = A^T$$

siempre tiene una solución (Teorema A.0.4); por lo tanto, la ecuación

$$A^T APY = A^T \tag{3.27}$$

siempre tiene una solución, ya que P es no singular, y al efectuar operaciones elementales de filas, tenemos que

$$EA^T APY = EA^T \tag{3.28}$$

siempre tiene una solución.

Entre todas las soluciones para (3.27), la única matriz que minimiza la $traza(Y^T Y)$ (Teorema A.0.6) es

$$\widehat{Y} = \left[EA^T AP\right]^+ EA^T. \tag{3.29}$$

El conjunto de matrices Y que satisfacen la ecuación (3.28) es el mismo que el conjunto de matrices que satisfacen (3.27), ya que E es no singular, por lo tanto, $\widehat{Y}$ también minimiza

$$traza[Y^T Y] = traza\left[(PY)^T (PY)\right],$$

ya que P es ortogonal.

Por el Teorema A.0.6 y la ecuación (3.27), tenemos que

$$P\widehat{Y} = \left[A^T A\right]^+ A^T = A^+. \tag{3.30}$$

La demostración del teorema finaliza cuando (3.30) y (3.29) son combinados. □

Corolario 3.5.1. *Si E es no singular, P es ortogonal y $EA^T AP = \begin{bmatrix} H \\ 0 \end{bmatrix}$, entonces:*

$$A^+ = P\left[H^+ \; 0\right]\left(EA^T\right).$$

A continuación mostramos cómo calcular EA^T, H^+ y P.

(a) *Cálculo de H, EA^T y P*

Paso 1. Escribimos la matriz aumentada $\begin{bmatrix} A^TA & A^T \end{bmatrix}$ y efectuamos operaciones elementales en las filas de esta matriz hasta conseguir la forma escalonada reducida.

$$\begin{bmatrix} A^TA & A^T \end{bmatrix} \xrightarrow{\text{Operaciones de fila}} \begin{bmatrix} EA^TA & EA^T \end{bmatrix}$$

Paso 2. Permutamos las primeras n columnas de la matriz escalonada reducida obteniendo, al final de este paso, una matriz que la podemos describir por bloques asi

$$\begin{bmatrix} EA^TAP & EA^T \end{bmatrix} = \begin{bmatrix} I & L & \\ 0 & 0 & EA^T \end{bmatrix}.$$

Las operaciones elementales de fila son reversibles, por lo que E es no singular, como se quería. La matriz de permutación P es ortogonal y reordena las columnas de EA^TA, para obtener una matriz escalonada reducida. La matriz EA^T ocupa el bloque del lado derecho de la matriz aumentada después del Paso 1. La matriz $H \in \mathbb{R}^{r\times n}$ está dada por $[I\ L]$ después del paso 2.

(b) *Cálculo de H^+*

$$\begin{aligned} H^+ = [I\ L]^+ &= [I\ L]^T \left([I\ L]\,[I\ L]^T\right)^{-1} \\ &= \begin{bmatrix} I \\ L^T \end{bmatrix} \left(I + LL^T\right)^{-1}. \end{aligned}$$

Evaluamos $\left(I + LL^T\right)^{-1}$ utilizando el Teorema (3.4.3).

Algoritmo 18 Pseudoinversa por el Método de la Escalonada Reducida

Entrada: $A \in \mathbb{R}^{m\times n}$, $rango(A) = r$
Salida: $A^{+} \in \mathbb{R}^{n\times m}$

1: Calcular la matriz aumentada $\begin{bmatrix} A^TA & A^T \end{bmatrix}$ y realizar operaciones de fila hasta obtener la matriz escalonada $\begin{bmatrix} EA^TA & EA^T \end{bmatrix}$.
2: Permutar las primeras n columnas de modo que

$$EA^TAP = \begin{bmatrix} I & L \\ 0 & 0 \end{bmatrix} = \begin{bmatrix} H \\ 0 \end{bmatrix}.$$

3: Calcular $H^{+} = [I \quad L]^{+} = \begin{bmatrix} \left(I + LL^T\right)^{-1} \\ L^T\left(I + LL^T\right)^{-1} \end{bmatrix}$.
4: Finalmente, $A^{+} = P\left(EA^TAP\right)^{+}EA^T = P\left[H^{+} \quad 0\right]EA^T$.

3.6. Método basado en una variante del método de la proyección del gradiente

L. Duane Pyle [25], propuso calcular la matriz pseudoinversa usando el método de proyección del gradiente. El procedimiento básico requiere la aplicación del proceso de ortogonalización de Gram-Schmidt. En este sentido, el método que presentamos a continuación está basado en las ideas del método de proyección del gradiente de Pyle.

Consideremos el sistema de ecuaciones lineales

$$A\mathbf{x} = b, \quad A \in \mathbb{R}^{m\times n},\ \mathbf{b} \in \mathbb{R}^{m}\ m > n. \tag{3.31}$$

Como este sistema es sobredeterminado, no se puede garantizar la existencia de una solución $\mathbf{x}$ para una matriz A y un vector $\mathbf{b}$ arbitrarios. Por lo tanto, es deseable hallar un vector $\mathbf{x}$, tal que $A\mathbf{x}$ sea la mejor aproximación a $\mathbf{b}$.

Denotamos las columnas de A^T por $\mathbf{a}_1, \mathbf{a}_2, ..., \mathbf{a}_m$; por A_k, a la matriz $k \times n$ obtenida mediante la eliminación de las últimas $m - k$ filas de A y definimos por $\mathbf{b}_k$ al vector $k \times 1$ resultante de la eliminación de los últimos $m - k$ componentes de $\mathbf{b}$. Por lo tanto el conjunto de ecuaciones simultáneas

$$A_k\mathbf{x}_k = \mathbf{b}_k \quad k = 1, 2, ..., m, \tag{3.32}$$

es un subconjunto de aquellas representadas por la ecuación (3.31).

Además,

$$\widehat{\mathbf{x}}_k = A_k^+ \mathbf{b}_k$$

es solución para la ecuación (3.32) que se encuentra en $C_{A_k^T} \equiv Gen(\mathbf{a}_1, \mathbf{a}_2, ..., \mathbf{a}_k)$, el subespacio generado por los vectores $(\mathbf{a}_1, \mathbf{a}_2, ..., \mathbf{a}_k)$.

El procedimiento que se presenta a continuación desarrolla una simple recursión que relaciona $\widehat{\mathbf{x}}_{k+1}$ con $\widehat{\mathbf{x}}_k$. Puesto que $\widehat{\mathbf{x}} = A^+\mathbf{b}$ es igual a $\widehat{\mathbf{x}}_m$, la recursión puede realizarce m veces y la solución deseada para la ecuación (3.31) resulta siempre que $\mathbf{b}$ se encuentre en C_A.

En general, por definición, $\widehat{\mathbf{x}}_1$ es el único vector en $Gen(\mathbf{a}_1)$ que satisface

$$\mathbf{a}_1^T \mathbf{x}_1 = \beta_1,$$

donde β_j es la j-ésima componente de $\mathbf{b}$ $(j = 1, ..., m)$ y

$$\widehat{\mathbf{x}}_1 = \left(\mathbf{a}_1^T\right)^+ \beta_1. \tag{3.33}$$

Si $\widehat{\mathbf{x}}_{k-1}$ se conoce, construimos $\widehat{\mathbf{x}}_k$ de la siguiente manera.

Sea $(\mathbf{h}_1, \mathbf{h}_2, ..., \mathbf{h}_m)$ el conjunto de vectores no normalizado que resulta de una ortogonalización de Gramm-Schmidt de $(\mathbf{a}_1, .., \mathbf{a}_m)$:

$$\begin{aligned} \mathbf{h}_1 &= \mathbf{a}_1 \\ \mathbf{h}_k &= \mathbf{a}_k - \sum_{j \in S_k} \left(\mathbf{a}_k^T \mathbf{h}_j\right) \mathbf{h}_j / \left\|\mathbf{h}_j\right\|^2, \end{aligned} \tag{3.34}$$

donde $S_k = \{j : j \leq k-1 \quad \text{y} \quad \|\mathbf{h}_j\| \neq 0\} \quad k = 1, .., m.$

Como $\mathbf{h}_k \perp Gen\,(\mathbf{a}_1, ..., \mathbf{a}_{k-1})$[1], y dado que $\widehat{\mathbf{x}}_{k-1}$ satisface

$$A_{k-1}\widehat{\mathbf{x}}_{k-1} = \mathbf{b}_{k-1}, \tag{3.35}$$

se sigue que

$$\mathbf{a}_j^T \left(\widehat{\mathbf{x}}_{k-1} + \alpha_k \mathbf{h}_k\right) = \beta_j \quad j = 1, 2, ..., k-1$$

para cualquier escalar α_k.

En particular, si

$$\widehat{\alpha}_k = \begin{cases} 0 & \text{si} \quad \mathbf{h}_k = 0 \\ \left(\beta_k - \mathbf{a}_k^T \widehat{\mathbf{x}}_{k-1}\right) / \left(h_k^T \mathbf{a}_k\right) & \text{de otra manera} \end{cases}$$

entonces, $\widehat{\mathbf{y}}_k = \widehat{\mathbf{x}}_{k-1} + \widehat{\alpha}_k \mathbf{h}_k$ satisface

$$\mathbf{a}_j^T \mathbf{y}_k = \beta_j \quad j \in S_{k+1}. \tag{3.36}$$

y

$$\widehat{\mathbf{y}}_k \in Gen(\mathbf{h}_1, ..., \mathbf{h}_k).$$

Por construcción,

$$\begin{aligned} Gen(\mathbf{h}_j; j \in S_{k+1}) &= Gen(\mathbf{h}_1, ..., \mathbf{h}_k) \\ &= Gen(\mathbf{a}_1, ..., \mathbf{a}_k) = Gen(\mathbf{a}_j; j \in S_{k+1}), \end{aligned}$$

donde $\mathbf{h}_j = 0$, si y solo si, $\mathbf{a}_j$ es una combinación lineal de sus antecesores. Por lo tanto, $\widehat{\mathbf{y}}_k$ es el único vector en $Gen(\mathbf{a}_j; j \in S_{k+1})$ que satisface la ecuación (3.36). Por otra parte,

$$\widehat{\mathbf{x}}_k = A_k^+ \mathbf{b}_k \in Gen(\mathbf{a}_1, ..., \mathbf{a}_k) = Gen(\mathbf{a}_j; j \in S_{k+1})$$

satisface a

$$\mathbf{a}_j^T \mathbf{x}_k = \beta_j \quad j = 1, ..., k \tag{3.37}$$

y también a (3.36). Como $\widehat{\mathbf{y}}_k$ es el único vector en $Gen(\mathbf{a}_j; j \in S_{k+1})$ que satisface la ecuación (3.36), $\widehat{\mathbf{y}}_k = \widehat{\mathbf{x}}_k$.

En resúmen:

Teorema 3.6.1. *Si*

$$A^T = [\boldsymbol{a}_1, \boldsymbol{a}_2, \cdots, \boldsymbol{a}_m] \quad y \quad \boldsymbol{b} = \begin{bmatrix} \beta_1 \\ \beta_2 \\ \vdots \\ \beta_m \end{bmatrix} \in C_A,$$

entonces

$$A^{+}\boldsymbol{b} = \widehat{\boldsymbol{x}}_m,$$

donde

$$\begin{aligned}
&\widehat{\boldsymbol{x}}_0 = 0 \\
&\widehat{\boldsymbol{x}}_k = \widehat{\boldsymbol{x}}_{k-1} + \widehat{\alpha}_k \boldsymbol{h}_k \quad k = 1, 2, ..., m \\
&\widehat{\alpha}_k = \begin{cases} 0 & si \quad \boldsymbol{h}_k = 0 \\ \left(\beta_k - \boldsymbol{a}_k^T \widehat{\boldsymbol{x}}_{k-1}\right) / \left(\boldsymbol{h}_k^T \boldsymbol{a}_k\right) & de\ otra\ manera \end{cases}
\end{aligned}$$

y $\{\boldsymbol{h}_1, \boldsymbol{h}_2, ..., \boldsymbol{h}_m\}$ *son los vectores no normalizados resultantes de una OGS de* $\{\boldsymbol{a}_1, \boldsymbol{a}_2, ..., \boldsymbol{a}_m\}$.

En general, si $\mathbf{b} \notin C_A$ (o si no se ha decidido si $\mathbf{b} \in C_A$), podemos recurrir a la siguiente estrategia.

Sea $\mathbf{d} = A^T\mathbf{b}$ y $C = A^TA$. Entonces $\mathbf{d} \in C_{A^T} = C_C$ de manera que el Teorema (3.6.1) se puede utilizar para calcular $C^{+}\mathbf{d} = \left(A^TA\right)^{+} A^T\mathbf{b}$.

Si, en lugar de $A^{+}\mathbf{b}$, se desea calcular A^{+}, procedemos de la siguiente manera.

En primer lugar, realizamos una OGS en las columnas de A. Denotamos al conjunto normalizado de vectores no nulos por $\mathbf{d}_1, \mathbf{d}_2, \cdots, \mathbf{d}_r$. Estos vectores generan el mismo subespacio que generan las columnas de A, es decir C_A. Por lo tanto, por el teorema A.0.7 se deduce que

$$AA^{+} = \sum_{j=1}^{r} \mathbf{d}_j \mathbf{d}_j^T.$$

Denotamos las columnas de AA^{+} por $\mathbf{b}_1, \mathbf{b}_2, ..., \mathbf{b}_m$. Como $C_A = C_{AA^{+}} = Gen(\mathbf{b}_1, \mathbf{b}_2, ..., \mathbf{b}_m)$, se tiene que $\mathbf{b}_j \in C_A$ para cada j, de modo que el Teorema (3.6.1) se utiliza m veces para calcular $A^{+}\mathbf{b}_j$, $j = 1, ..., m$. Puesto que

$$\left[A^{+}\mathbf{b}_1\ A^{+}\mathbf{b}_2 \cdots A^{+}\mathbf{b}_m\right] = A^{+}\left[\mathbf{b}_1\ \mathbf{b}_2 \cdots\ \mathbf{b}_m\right] = A^{+}\left[AA^{+}\right] = A^{+},$$

el procedimiento genera el resultado deseado.

Algoritmo 19 Pseudoinversa usando una variante del método de la Proyección del Gradiente

Entrada: $A \in \mathbb{R}^{m\times n}$, $rango(A) = r$
Salida: $A^{+} \in \mathbb{R}^{n\times m}$

1: Si $\mathbf{b} \in C_A$, usar el Teorema (3.6.1) para construir $A^{+}\mathbf{b}$.
2: Si $\mathbf{b} \notin C_A$(o no sabemos si $\mathbf{b} \in C_A$), calcular $\mathbf{d} = A^T\mathbf{b}$, $C = A^TA$ y usar el Teorema (3.6.1) para calcular $C^{+}\mathbf{d}$ que coincide con $A^{+}\mathbf{b}$
3: Para calcular A^{+}, primero realizar una OGS en las columnas de A. Denotar los vectores ortonormales resultantes por $\mathbf{d}_1, \mathbf{d}_2, \cdots, \mathbf{d}_r$. Construir la matriz $\sum_{j=1}^{r} \mathbf{d}_j\mathbf{d}_j^T$ y denotar las columnas de esta matriz por $\mathbf{b}_1, \mathbf{b}_2, \cdots, \mathbf{b}_m$. Entonces, para cada j, $\mathbf{b}_j \in C_A$ y por el Teorema (3.6.1), construimos $\mathbf{v}_j = A^{+}\mathbf{b}_j, j = 1, ..., m$, de donde

$$A^{+} = [\mathbf{v}_1 \quad \mathbf{v}_2 \cdots \mathbf{v}_m]$$

3.7. Método basado en el teorema de Cayley-Hamilton

Henry P. Decell [8], A. Ben Israel y A. Charnes [14], utilizando el teorema de Cayley-Hamilton propusieron un algoritmo para calcular la pseudoinversa de una matriz.

El presente método está basado en dos teoremas. El primero utiliza la clásica relación Cayley-Hamilton para deducir una expresión para la inversa generalizada de una matriz en términos de su polinomio característico. El segundo teorema es un resultado de Faddeev y Fadeeva [9], que genera los coeficientes de dicho polinomio característico.

Teorema 3.7.1 (Caley-Hamilton)**.** *Sea*

$$f(\lambda) = (-1)^m \left(\lambda^m + a_1\lambda^{m-1} + ... + a_{m-1}\lambda + a_m\right)$$

el polinomio caracteristico de una matriz $B \in \mathbb{R}^{m\times m}$ y sean $I \in \mathbb{R}^{m\times m}$ y $Z \in \mathbb{R}^{m\times m}$, respectivamente, las matrices unidad y cero; entonces

$$B^m + a_1B^{m-1} + ... + a_{m-1}B + a_mI = Z.$$

A continuación, presentamos el resultado principal del Método basado en el teorema de Cayley-Hamilton, que da lugar al algoritmo para el cálculo de la matriz pseudoinversa.

Teorema 3.7.2. *[8] Sea $A \in \mathbb{R}^{m\times n}$ y*

$$f(\lambda) = (-1)^m \left(\lambda^m + a_1\lambda^{m-1} + ... + a_{m-1}\lambda + a_m\right),$$

con $a_0 = 1$, el polinomio caracteristico de AA^T. Si $k \neq 0$ es el mayor entero tal que $a_k \neq 0$, entonces la matriz pseudoinversa de A esta dada por

$$A^+ = -a_k^{-1}A^T\left[(AA^T)^{k-1} + a_1(AA^T)^{k-2} + \cdots + a_{k-1}I\right].$$

Si $k = 0$ es el mayor entero tal que $a_k \neq 0$, entonces $A^+ = 0$.

Demostración. De acuerdo con el teorema de Cayley-Hamilton, AA^T satisface su ecuación caracteristica

$$(AA^T)^m + a_1(AA^T)^{m-1} + ... + a_{m-1}(AA^T) + a_mI = Z.$$

Si $k \neq 0$ es el mayor entero tal que $a_k \neq 0$, tomamos $B = AA^T$. Si definimos $B^0 = I$, podemos escribir

$$B^{m-k}(B^k + a_1B^{k-1} + ... + a_{k-1}B + a_kI) = Z.$$

Esta ecuación garantiza una solución a la ecuación matricial $B^{m-k}X = Z$ y por tanto, por el Teorema A.0.8, todas las soluciones estarían dadas por

$$\begin{aligned} X &= (B^{m-k})^+Z + Y - (B^{m-k})^+B^{m-k}Y \\ &= Y - (B^{m-k})^+B^{m-k}Y, \end{aligned} \tag{3.38}$$

donde Y en una matriz arbitraria con las mismas dimensiones que X. En particular, existe Y_1 tal que

$$B^k + a_1B^{k-1} + ... + a_{k-1}B + a_kI = Y_1 - (B^{m-k})^+B^{m-k}Y_1.$$

Notemos que $B^TB = BB^T$, así que para cada entero p, $(B^p)^+ = (B^+)^p$ y $B^+B = BB^+$ (Teorema 2.3.1-5.). Esto, junto con el hecho de que B^+B es idempotente, implica que

$$(B^{m-k})^+B^{m-k} = (B^+)^{m-k}B^{m-k} = (B^+B)^{m-k} = B^+B.$$

Sin embargo, tenemos que $(AA^T)^+AA^T = AA^+$, así que $B^+B = AA^+$ y

$$B^k + a_1B^{k-1} + ... + a_{k-1}B + a_kI = Y_1 - AA^+Y_1.$$

Premultiplicando la última ecuación por A^+, junto con la propiedad $A^+AA^+ = A^+$

$$A^+B^k + a_1A^+B^{k-1} + ... + a_{k-1}A^+B + a_kA^+ = Z,$$

y como $A^+AA^T = A^+B = A^T$ y $a_k \neq 0$, obtenemos

$$\begin{aligned} A^+BB^{k-1} + a_1A^+BB^{k-2} + ... + a_{k-1}A^+B + a_kA^+ &= Z \\ A^TB^{k-1} + a_1A^TB^{k-2} + ... + a_{k-1}A^T + a_kA^+ &= Z, \end{aligned}$$

de donde

$$A^+ = -a_k^{-1}A^T\left[(AA^T)^{k-1} + a_1(AA^T)^{k-2} + \cdots + a_{k-1}I\right].$$

Si $k = 0$, tenemos que $(AA^T)^m = Z$; y por tanto $A = Z$ y $A^+ = A = Z$. □

Como una consecuencia del Teorema 3.7.2, la modificación de Faddeev del método de Leverrier, puede ser utilizada para describir el algortimo que calcule la matriz pseudoinversa de A. Si construimos la secuencia $A_0, A_1, \cdots, A_k$ bajo las hipótesis del teorema, tenemos

$$\begin{array}{lll} A_0 = Z, & -1 = q_0, & B_0 = I; \\ A_1 = AA^T, & traza\, A_1 = q_1, & B_1 = A_1 - q_1I; \\ A_2 = AA^TB_1, & \dfrac{traza\, A_2}{2} = q_2, & B_2 = A_2 - q_2I; \\ \vdots & \vdots & \vdots \\ A_{k-1} = AA^TB_{k-2}, & \dfrac{traza\, A_{k-1}}{k-1} = q_{k-1}, & B_{k-1} = A_{k-1} - q_{k-1}I; \\ A_k = AA^TB_{k-1}, & \dfrac{traza\, A_k}{k} = q_k, & B_k = A_k - q_kI; \end{array}$$

D.K. Faddeev demostró que $q_i = -a_i,\ \ i = 1, ..., k$. Por el teorema principal tenemos que $A^+ = Z$ ó

$$\begin{aligned} A^+ &= -a_k^{-1}A^T\left[(AA^T)^{k-1} + a_1(AA^T)^{k-2} + \cdots + a_{k-1}I\right] \\ &= -a_k^{-1}A^TB_{k-1}. \end{aligned}$$

Algoritmo 20 Pseudoinversa por el Método basado en el teorema de Cayley-Hamilton

Entrada: $A \in \mathbb{R}^{m\times n}$,
Salida: $A^{+} \in \mathbb{R}^{n\times m}$

1: Calcular los coeficientes a_k del polinomio caracteristico de la matriz AA^T, con $k = 0, ..., m$ por el método de Faddeev.

- Conjunto de valores iniciales: $B_0 = I$, $q_0 = -1$
- Calcular $A_k = AA^T B_{k-1}$
- Calcular $q_k = \text{traza}(A_k)/k$
- Calcular $B_k = A_k - q_k I$
- Tomar $a_k = -q_k$

2: Sea k el máximo índice tal que $a_k \neq 0$. Por tanto la pseudoinversa se obtiene como

$$A^{+} = \begin{cases} -a_k^{-1} A^T \left[(AA^T)^{k-1} + a_1 (AA^T)^{k-2} + \cdots + a_{k-1} I \right] & \text{si} \quad k > 0 \\ 0 & \text{si} \quad k = 0 \end{cases}$$

3.8. Prueba computacional para el cálculo de la matriz pseudoinversa

En esta sección, se presenta el estudio experimental comparativo de los programas que calculan la matriz pseudoinversa, los cuales se pueden revisar detenidamente en el Apéndice B

- *pinv*: algoritmo basado en la DVS propuesto en el paquete de MATLAB.
- *geninv*: algoritmo para el cálculo de la pseudoinversa, basado en una Generalización de la Factorización de Cholesky de Pierre Courrie [5].
- *OGSpseudo*: algoritmo basado en el método de Ortogonalización de Gram Schmidt OGS para el cálculo de la pseudoinversa.

- *ERpseudo*: algoritmo basado en el método de la Escalonada Reducida.
- *PGpseudo*: algoritmo basado en las ideas del método de la Proyección del Gradiente.
- *CHpseudo*: algoritmo basado en el Teorema de Cayley-Hamilton.

Todos los algoritmos fueron implementados en MATLAB (7.14.0.739), en un computador personal Intel(R) Core(TM) i5-3317U CPU@ 1.70GHz . En primer lugar, presentamos el tiempo promedio (en segundos) de cinco ejecuciones para el cálculo de la pseudoinversa de matrices aleatorias de rango deficiente con los algoritmos mencionados. Para las matrices de prueba recurrimos a *Pierre Courrieu* que en su artículo [5] ofrece una serie de indicaciones que utilizamos para las simulaciones.

Las matrices de prueba utilizadas para calcular el tiempo de ejecución de los algoritmos tienen las siguientas caracteristicas: tamaño $m \times n$, donde $n = 2^k$, con $k = 4, .., 7$, y $m = 2n$, rango deficiente, con rango $r = 7n/8$. Los coeficientes de la matriz son números aleatorios en el intervalo [-1,1].

$m \times n$	pinv	geninv	OGSpseudo	ERpseudo	PGpseudo	CHpseudo
32×16	$2{,}4263e-04$	$6{,}6841e-04$	0,0264	0,0361	0,0229	$9{,}1768e-04$
64×32	0,0045	0,0010	0,1827	0,1196	0,0928	0,0028
128×64	0,0061	0,0020	1,2317	0,4192	0,3600	0,0157
256×128	0,0093	0,0047	9,7367	1,4683	1,4782	0,2086

Tabla 3.1: Comparación del tiempo de ejecución en el cálculo de la matriz Pseudoinversa de matrices aleatorias de rango deficiente.

Revisando los tiempos de cálculo (en segundos) que se presentan en la Tabla 6.1, podemos ver que los tiempos de ejecución de la función *geninv* resultan ser mucho menores en la mayoría de las pruebas (excepto para $m \times n = 32 \times 16$), siendo éste el método más rápido, seguido de *pinv* y el algoritmo basado en el teorema de Cayley-Hamilton. En cambio, el tiempo de las ejecuciones con el método basado en la OGS, es mucho mayor con respecto a los demás métodos, seguido de los algoritmos basados en la escalonada reducida y la variante del método de proyección del gradiente.

En segundo lugar, para analizar la precisión de los algoritmos, utilizamos cuatro matrices mal condicionadas y de rango deficiente denominadas: A, B, C y D, con las mismas características de las matrices de la prueba anterior. En las pruebas (Tabla 3.2), la exactitud de los algoritmos se examinó de acuerdo a los errores presentes en las cuatro propiedades que caracterizan a la inversa Moore - Penrose.

$$\begin{aligned} E1 &= \left\| AA^{+}A - A \right\|_2 \\ E2 &= \left\| A^{+}AA^{+} - A^{+} \right\|_2 \\ E3 &= \left\| (AA^{+})^{T} - AA^{+} \right\|_2 \\ E4 &= \left\| (A^{+}A)^{T} - A^{+}A \right\|_2 \end{aligned} \tag{3.39}$$

A partir de esto, observamos que la función *pinv*, demostró ser la más precisa al calcular la pseudoinversa de las matrices aleatorias de rango deficiente, produciendo una aproximación fiable para matrices mal condicionadas ya que los errores asociados, **E1**, **E2**, y **E4** son pequeños con respecto a los presentes en los otros métodos. Con la función *geninv*, concluimos que es un algorítmico rápido para calcular la pseudoinversa pero menos precisa que *pinv*. Sin embargo, el error asociado **E3** es mucho menor en la función *geninv* obteniendo una mejor aproximación de la pseudoinversa con respecto a los demás métodos.

Por otro lado, en los experimentos numéricos encontramos que los algoritmos basados en la escalonada reducida (ERpseudo), OGS (OGSpseudo) y la variante del método de la proyección del gradiente (PGpseudo), no son tan precisos como *pinv* pero la matriz obtenida por estos métodos es razonablemente precisa y tan fiable como la aproximación obtenida con *geninv*.

En cuanto al algoritmo basado en el teorema de Cayley-Hamilton (CHpseudo), para el caso de matrices de orden pequeño, proporcionó resultados más precisos al calcular la pseudoinversa, lo que no ocurrió con matrices de orden elevado como se observa en los grandes errores presentes en las cuatro propiedades que caracterizan a la pseudoinversa. Una de las principales desventajas de la aplicación del teorema de Cayley, en el cálculo de la pseudoinversa, es que requiere la construcción en forma recursiva de los coeficientes del polinomio característico (método de Leverrier modificado), lo cual no resultó práctico desde una perspectiva computacional ya que no es numéricamente estable y es muy costoso en cuanto a aritmética de punto

flotante.

Algoritmos	**Valores de la norma**			
	E1	**E2**	**E3**	**E4**
Matriz=$A \in \mathbb{R}^{32\times16}$, $k_2(A) = 1{,}1430e+016$, $rango(A) = 14$				
pinv	$1{,}5337e-14$	$7{,}4820e-15$	$8{,}5387e-15$	$6{,}2550e-15$
geninv	$1{,}6022e-12$	$6{,}3949e-12$	$1{,}7822e-15$	$4{,}0383e-12$
OGSpseudo	$9{,}0086e-012$	$6{,}5417e-12$	$1{,}0455e-11$	$1{,}3752e-12$
ERpseudo	$5{,}5243e-11$	$3{,}7588e-11$	$6{,}8974e-11$	$4{,}8911e-13$
PGpseudo	$8{,}1187e-13$	$1{,}8609e-12$	$3{,}2050e-13$	$1{,}7507e-12$
CHpseudo	$7{,}24e-02$	$4{,}5e-03$	$3{,}9e-03$	$5{,}7e-03$
$B \in \mathbb{R}^{64\times32}$, $k_2(B) = 1{,}1955e+016$, $rango(B) = 28$				
pinv	$7{,}1358e-15$	$9{,}9473e-15$	$9{,}6290e-15$	$8{,}7005e-015$
geninv	$5{,}5896e-013$	$7{,}2950e-12$	$1{,}4574e-15$	$2{,}1061e-12$
5.5896e-013 **OGSpseudo**	$1{,}6121e-13$	$6{,}1898e-13$	$4{,}3140e-13$	$4{,}8531e-013$
ERpseudo	$2{,}2411e-13$	$3{,}4072e-13$	$5{,}7976e-13$	$6{,}0295e-014$
PGpseudo	$3{,}4695e-13$	$6{,}9703e-013$	$7{,}7329e-13$	$5{,}2300e-013$
CHpseudo	$2{,}7552e+13$	$1{,}0061e+25$	$2{,}2690e+12$	$3{,}2033e+12$
$C \in \mathbb{R}^{128\times64}$, $k_2(C) = 8{,}6521e+015$, $rango(C) = 56$				
pinv	$2{,}2086e-14$	$1{,}2598e-14$	$2{,}1142e-14$	$1{,}8637e-14$
geninv	$5{,}7207e-13$	$1{,}3983e-12$	$2{,}7135e-15$	$2{,}0328e-12$
OGSpseudo	$2{,}9178e-12$	$7{,}1538e-013$	$2{,}4029e-12$	$9{,}3059e-13$
ERpseudo	$1{,}9429e-12$	$4{,}1987e-13$	$1{,}5147e-12$	$2{,}2924e-013$
PGpseudo	$4{,}7166e-12$	$1{,}9282e-11$	$1{,}3754e-11$	$4{,}0067e-12$
CHpseudo	$1{,}6212e+95$	$5{,}8836e+187$	$1{,}4279e+94$	$2{,}1285e+94$
$D \in \mathbb{R}^{256\times128}$, $k_2(D) = 7{,}7595e+015$, $rango(D) = 112$				
pinv	$2{,}6255e-14$	$1{,}5085e-14$	$2{,}2652e-014$	$2{,}0178e-014$
geninv	$5{,}5509e-12$	$1{,}08841e-11$	$3{,}3550e-15$	$1{,}1093e-11$
OGSpseudo	$5{,}6438e-12$	$1{,}1539e-12$	$4{,}3376e-12$	$1{,}5059e-12$
ERpseudo	$6{,}7503e-12$	$1{,}3155e-12$	$5{,}3020e-12$	$4{,}0483e-013$
PGpseudo	$3{,}7456e-11$	$1{,}0716e-10$	$1{,}0236e-10$	$1{,}7641e-11$
CHpseudo	$5{,}6694e+220$	NaN	$2{,}7741e+219$	$4{,}2371e+219$

Tabla 3.2: Precisión de los algoritmos que calculan la matriz pseudoinversa.

Capítulo 4

Redes Neuronales Artificiales (RNA)

4.1. Introducción

En este capítulo, realizamos una introducción a la teoría de las redes neuronales artificiales, describiendo los elementos que forman una red neuronal y los mecanismos de aprendizaje, y en general estudiaremos la red denominada Asociador lineal, indicando su estructura y algoritmos de aprendizaje.

4.2. Historia de las RNA

En 1936, Alan Turing fue el primero en estudiar el cerebro como una forma de ver el mundo de la computación. Sin embargo, como es descrito en [3], los primeros teóricos que concibieron los fundamentos de la computación neuronal fueron Warren McCulloch, un neurofisiólogo, y Walter Pitts, un matemático, quienes en 1943, lanzaron una teoría acerca de la forma de trabajar de las neuronas.

En 1949, Donald Hebb escribió un importante libro: La organización del comportamiento, en el que se establece una conexión entre psicología y fisiología. Fue el primero en explicar los procesos del aprendizaje (que es el elemento básico de la inteligencia humana) desde un punto de vista psicológico, desarrollando una regla de cómo el aprendizaje ocurría; éste es el fundamento de la mayoría de las funciones de aprendizaje que pueden hallarse en una red

neuronal [10]. Los trabajos de Hebb formaron las bases de la Teoría de las Redes Neuronales.

En 1950, Karl Lashley, en sus series de ensayos, encontró que la información no era almacenada en forma centralizada en el cerebro sino que era distribuida [3].

En 1957, Frank Rosenblatt comenzó el desarrollo del Perceptrón, la red neuronal más antigua, la cual era capaz de generalizar; es decir, después de haber aprendido una serie de patrones podía reconocer otros similares, aunque no se le hubiesen presentado anteriormente. Sin embargo, tenía una serie de limitaciones; en general, era incapaz de clasificar clases no separables linealmente [10].

En 1960, Bernard Widrow y Marcial Hoff desarrollaron el modelo Adaline, la primera red neuronal aplicada a un problema real, la cual se ha utilizado comercialmente durante varias décadas para eliminar ecos en las líneas telefónicas [10].

En 1969, surgieron críticas que frenaron, hasta 1982, el crecimiento que estaban experimentando las investigaciones sobre redes neuronales artificiales. En 1974, Paul Werbos desarrolló la idea básica del algoritmo de aprendizaje de propagación hacia atrás, cuyo significado quedó definitivamente aclarado en 1985 [3].

En 1986, David Rumelhart y Geoffrey Hinton redescubrieron el algoritmo de aprendizaje de propagación hacia atrás. A partir de 1986, el panorama fue alentador con respecto a las investigaciones y el desarrollo de las redes neuronales [3].

En la actualidad, las RNA son una teoría que aún esta en proceso de desarrollo, su verdadera potencialidad aún no se ha alcanzado; aunque se han desarrollado potentes algoritmos de aprendizaje de gran valor práctico, las representaciones y procedimientos de que se sirve el cerebro son aún desconocidas.

4.3. Neurona biológica

Se estima que en nuestro cerebro cohabitan unas cien mil millones de neuronas y cada neurona tiene aproximadamente diez mil conexiones con otras neuronas. A través de estas interconexiones, una neurona recoge señales procedentes de otras neuronas, luego procesa estas señales y transmite otra señal hacia otras neuronas, de esta manera la información se transmite de unas neuronas a otras. Las neuronas están cubiertas por una membrana que separa el interior de la célula del fluido que la rodea (fluido extracelular); esta membrana actúa de tal forma que se mantenga una diferencia de potencial entre el interior de la célula y el fluido extracelular [3].

El término neurona es usado para denominar a la célula nerviosa y todas sus prolongaciones; al contrario de otras células del organismo, éstas no se dividen ni se reproducen. El tamaño y forma de las neuronas es variable, pero todas tienen los mismos componentes; en la Figura 4.1, se muestran los principales componentes de una célula neuronal; éstos son: el soma o cuerpo de la célula, el axón y las dendritas; las conexiones entre neuronas son materializadas por medio de la sinapsis.

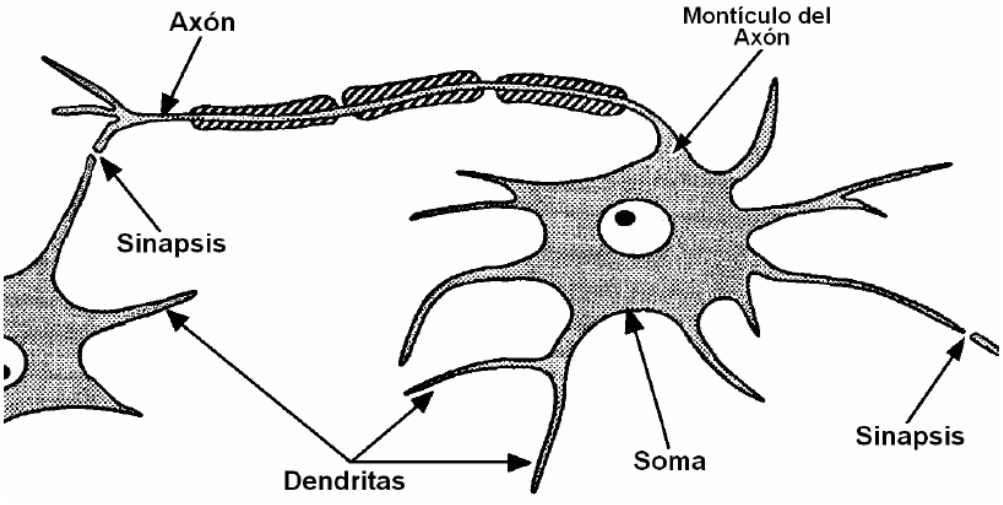


Figura 4.1: Principales componentes de una neurona [19]

4.3.1. El soma

Es la porción central o cuerpo celular que contiene el núcleo y tiene unida una o más estructuras denominadas dendritas y axones, éstas son extensiones de la célula y están implicadas en la recepción y envío de las señales; en el soma, se realiza la suma de todas las señales provenientes de otras neuronas.

4.3.2. El axón

El axón es una fibra larga y delgada que lleva la señal desde su montículo hasta otras neuronas; generalmente, existe un gran número de ramificaciones en el extremo del axón, para permitir que las señales de una neurona sean transmitidas a muchas otras. En estos sistemas biológicos, las señales consisten en diferencias de potencial que se transmiten sin atenuación.

4.3.3. Las dendritas

Las dendritas son extensiones de la célula que se encargan de la recepción de señales provenientes de otras neuronas. Son un grupo de fibras nerviosas muy ramificadas y de forma irregular que están conectadas directamente al soma; cada neurona posee un gran número de dendritas que permiten recibir información de muchas otras neuronas. Las dendritas transmiten un potencial eléctrico hasta el soma, este potencial es de magnitud variable y puede ser excitador o inhibidor, según contribuya o no a la generación de un potencial de acción en el axón.

4.3.4. Las sinápsis

La unión existente entre dos neuronas se denomina unión sináptica o sinápsis; está unión consiste en un punto de contacto entre el axón de una neurona y la dendrita de otra neurona. En la unión sináptica, se produce una conversión entre una señal eléctrica a una química. La señal eléctrica consiste en una diferencia de potencial entre el axón y el fluido extracelular; está diferencia de potencial, llamado potencial de acción, hace que la membrana presináptica, que se encuentra en el extremo del axón, libere unas sustancias llamadas neurotransmisores.

Los neurotransmisores se difunden a través de la unión sináptica y se unen a la membrana postsináptica que se encuentra en el extremo de la dendrita. El efecto de los neurotransmisores sobre la membrana postsináptica da lugar a un potencial eléctrico entre la dendrita y el fluido extracelular; este potencial eléctrico es variable y puede ser de naturaleza excitadora o inhibidora, según los neurotransmisores que se liberen; este potencial es transmitido por la dendrita hasta el soma, donde se combinará con otros potenciales. La Figura 4.2 muestra la unión sináptica cuando los neurotransmisores son liberados.

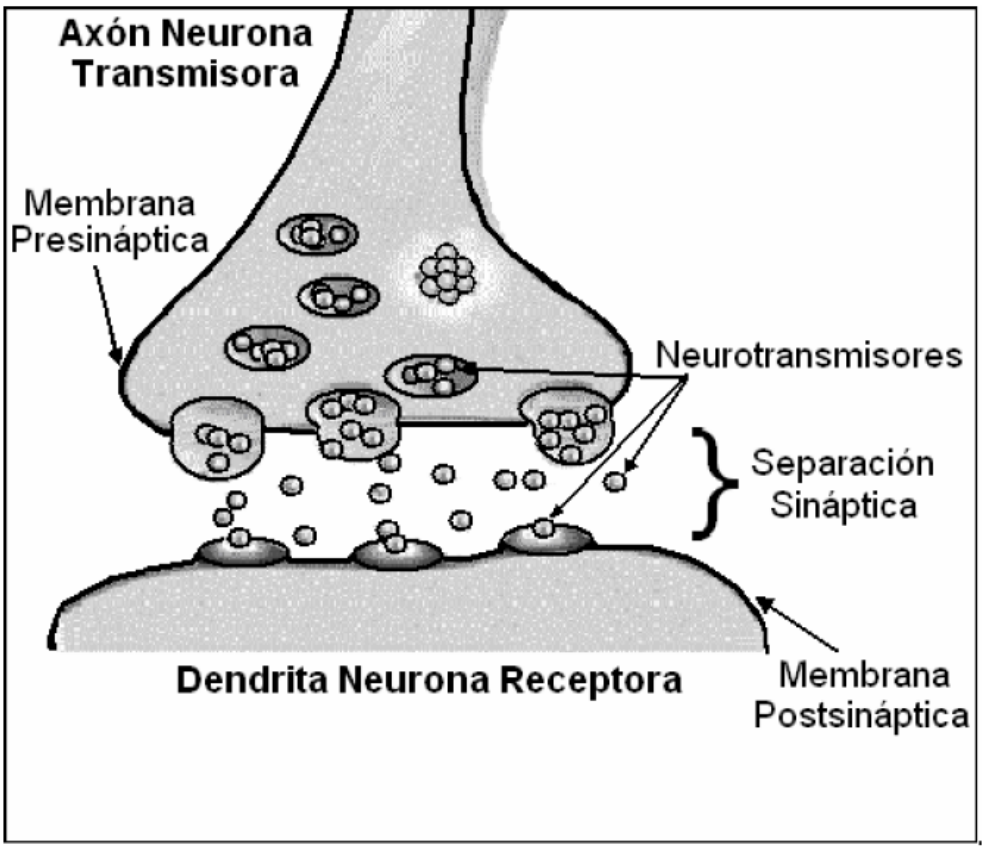


Figura 4.2: Unión Sináptica [19]

4.3.5. Teoría básica del aprendizaje

Los sistemas biológicos no nacen preprogramados con todo el conocimiento y las capacidades que llegarán a tener eventualmente. Un proceso de aprendizaje, que tiene lugar a lo largo de un período de tiempo, modifica de alguna forma la red neuronal para incluir la nueva información. La teoría básica del aprendizaje procede de un libro escrito en 1949 por Donald O. Hebb [13], la cual proporciona una idea de cómo es guardada la información en el cerebro, ésta dice así [3]:

Cuando un axón de una célula A está suficientemente próximo para excitar a una célula B o toma parte en su disparo de forma persistente, tiene lugar algún proceso de crecimiento o algún cambio metabólico en una de las células, o en las dos, de tal modo que la eficiencia de A, como una de las células que desencadena el disparo de B, se ve incrementada.

Dado que la conexión entre neuronas se hace a través de la sinapsis, es razonable suponer que cualquier cambio que puedan tener lugar durante el aprendizaje deberán producirse en ellas. La teoría de Hebb suponía un aumento en el área de la unión sináptica; teorías mas recientes afirman que el responsable es un incremento en la velocidad con que se liberan los neurotransmisores; en todo caso, durante el aprendizaje, se producen cambios en la sinápsis.

En el cerebro, el conocimiento se encuentra almacenado en las conexiones ponderadas entre las neuronas (sinápsis); en el proceso de aprendizaje, estas conexiones ponderadas se ajustan.

4.4. Elementos de las redes neuronales artificiales

En la emulación de un sistema neuronal biológico por medio de un sistema artificial, se puede establecer una estructura jerárquica similar a la existente en el cerebro [18]. El elemento esencial será la neurona artificial, la cual se organiza en capas. Varias capas constituyen una red neuronal. Finalmente, una red neuronal, junto con las interfases de entrada y salida, componen el sistema global del proceso.

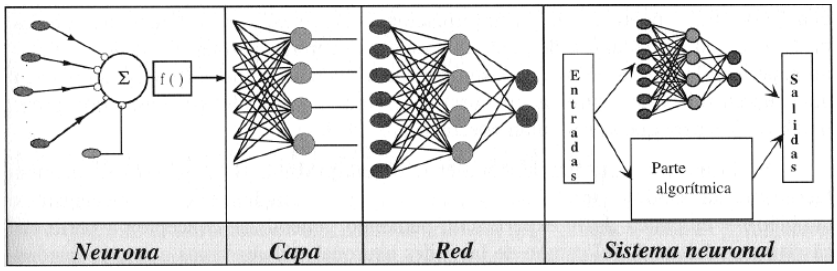


Figura 4.3: Sistema global del proceso de una red neuronal artificial [19]

Las RNA son nuevos modelos de procesamiento de información, inspirados por la forma en que el cerebro procesa información. Por ejemplo, son capaces de aprender de la experiencia, de generalizar de casos anteriores a nuevos casos, de abstraer características esenciales a partir de entradas que representan información imprecisa, etc. Esto hace que ofrezcan numerosas ventajas, entre ellas se incluyen:

- *Aprendizaje Adaptativo:* son capaces de aprender con base en un entrenamiento o experiencia inicial.
- *Generalización*: una vez entrenada, a la red se le pueden presentar datos distintos a los usados durante el aprendizaje. La respuesta obtenida dependerá del parecido de los datos con los ejemplos de entrenamiento.
- *Abstracción o tolerancia al ruido:* son capaces de extraer o abstraer las características esenciales de las entradas aprendidas, de esta manera pueden procesar correctamente datos incompletos o distorsionados.
- *Procesamiento paralelo*: la información es procesada por las neuronas artificiales en forma paralela.
- *Memoria distribuida*: el conocimiento acumulado por la red se halla distribuido en numerosas conexiones.
- *Tolerancia a fallos*: una red neuronal es capaz de seguir funcionando adecuadamente a pesar de sufrir lesiones como la destrucción de neuronas o conexiones, ya que la información se halla distribuida por toda la red.

4.4.1. Estructura

Formalmente, y desde el punto de vista del grupo PDP (*Parallel Distributed Processing Research Group*, de la Universidad de California en San Diego), de D.E. Rumelhart y J.L. McClelland [27], [26], un sistema neuronal o conexionista está compuesto por los siguientes elementos:

- Un conjunto de procesadores elementales o neuronas artificiales
- Un patron de conectividad o arquitectura
- Una dinámica de activaciones
- Una regla o dinámica de aprendizaje
- Un entorno donde opera.

Modelo general de neurona artificial

En el marco establecido por el grupo PDP, se denomina *procesador elemental o neurona* a un dispositivo simple de cálculo que, a partir de un vector de entrada procedente del exterior o de otras neuronas, proporciona una única respuesta o salida [19]. Los elementos que constituyen la i-ésima neurona artificial son los siguientes (ver Figura 4.5):

- *Conjunto de entradas* $\mathbf{x}_j(t)$.
- *Pesos sinápticos* w_{ij}, que representan la intensidad de interacción entre cada neurona presináptica j y la neurona postsináptica i.
- *Regla de propagación* $\sigma(w_{ij}, x_j(t))$, que proporciona el valor del potencial potsináptico $h_i(t) = \sigma(w_{ij}, x_j(t))$ de la neurona i en función de sus pesos presinápticos y entradas. La función más habitual es de tipo lineal y se basa en la suma ponderada de las entradas con los pesos sinápticos:

  $$h_i = \sum_j w_{ij} x_j.$$

  Formalmente puede interpretarse como el producto escalar de los vectores de entrada y pesos

  $$h_i(t) = \sum_j w_{ij} x_j = \mathbf{w}_i^T . \mathbf{x}.$$

- *Función de activación o función de transferencia* $f_i(a_i(t-1), h_i(t))$, proporciona el estado de activación actual $a_i(t) = f_i(a_i(t-1), h_i(t))$ de la neurona i, en función de su estado anterior $a_i(t-1)$ y de su potencial postsináptico actual. Sin embargo, muchos modelos de RNA consideran que el estado actual de una neurona no depende de su estado anterior, $a_i(t) = f_i(h_i(t))$.

  Las funciones de activación más empleadas en las RNA se muestran en la Figura 4.4. Para abreviar, en ella designamos con x al potencial postsináptico y, con y, el estado de activación.

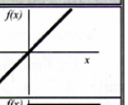
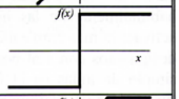
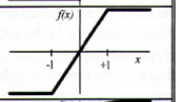
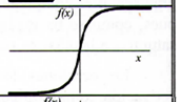
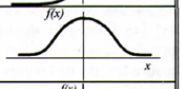
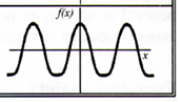
	Función	Rango	Gráfica
Identidad	$y = x$	$[-\infty, +\infty]$	
Escalón	$y = \text{signo}(x)$ $y = H(x)$	{-1, +1} {0, +1}	
Lineal a tramos	$y = \begin{cases} -1, & si\ x < -l \\ x, & si\ +l \le x \le -l \\ +1, & si\ x > +l \end{cases}$	[-1, +1]	
Sigmoidea	$y = \frac{1}{1+e^{-x}}$ $y = \text{tgh}(x)$	[0, +1] [-1, +1]	
Gaussiana	$y = A.e^{-Bx^2}$	[0,+1]	
Sinusoidal	$y = A.\text{sen}(\omega x + \varphi)$	[-1,+1]	

Figura 4.4: Ejemplos de funciones de activación [19]

- *Función de salida* $F_i(a_i(t))$, proporciona la salida $y_i(t) = F_i(a_i(t))$ de la neurona i en función de su estado de activación actual $(a_i(t))$.

De este modo, la operación de la neurona i puede expresarse como:

$$y_i(t) = F_i\left(f_i\left[(a_i(t-1), h_i(t))\right]\right) \tag{4.1}$$

Este modelo de neurona formal, como se observa en la Figura (4.5), se inspira en la operación de la biológica, en el sentido de integrar una serie de entradas y proporcionar cierta respuesta, que se propaga por el axón.

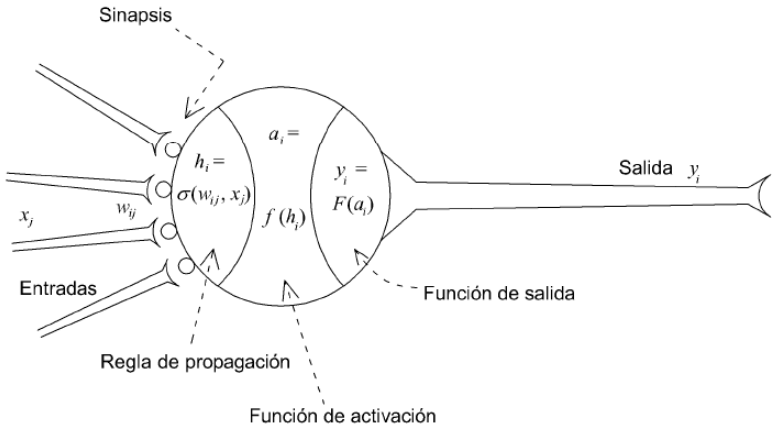


Figura 4.5: Modelo genérico de neurona artificial [27]

Las variables de entrada y salida son binarias (digitales) ó continuas (analógicas), dependiendo del modelo y su aplicación.

Modelo estándar de neurona artificial

El modelo estándard de neurona artifcial constituye un caso particular de neurona artificial, donde la regla de propagación es la suma ponderada y función de salida es la identidad. De esta forma, la i-ésima neurona artificial estándard consiste en [19]:

- *Un conjunto de entradas* $\mathbf{x}_j(t)$ y unos pesos sinápticos w_{ij}.
- *Una regla de propagación* $h_i(t) = \sigma(w_{ij}, x_j(t)); h_i(t) = \sum w_{ij}x_j$.
- *Función de activación* $y_i = f_i(h_i(t))$ que representa simultáneamente la salida de la neurona y su estado de activación.

Con frecuencia, se añade al conjunto de pesos de la neurona un parámetro adicional θ_i, que denominaremos umbral, el cual se resta del potencial postsináptico. Es decir,

$$h_i(t) = \sum_j w_{ij}x_j - \theta_i. \tag{4.2}$$

Si los índices comienzan en 0, y denotamos por w_{i0}=θ_i y x_0=-1, podemos expresar la regla de propagación (que representa simultáneamente la función de salida) como:

$$y_i(t) = f_i(\sum_{\text{J}=0}^{n} w_{ij}x_j). \tag{4.3}$$

A continuación, se puede ver (Figura 4.6) el modelo de neurona artificial estándar descrito previamente.

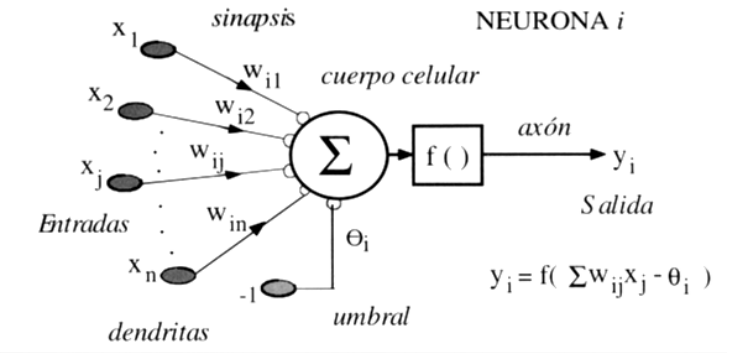


Figura 4.6: Modelo de neurona artificial estándar [19]

4.4.2. Arquitectura

Se denomina arquitectura a la topología, estructura o patrón de conexiones de una red neuronal. En una RNA, las neuronas se agrupan en unidades estructurales denominadas "capas". El conjunto de una o más capas constituye la red neuronal. Se distinguen tres tipos de capas: de entrada, de salida y ocultas. Una "capa de entrada" o sensorial, está compuesta por neuronas que reciben datos o señales procedentes del entorno. Una "capa de salida" o "efectora", se compone de neuronas que proporcionan la respuesta de la red neuronal. Una "capa oculta" es aquella que no tiene conexión directa con el entorno, es decir, no se conecta directamente ni a órganos sensores ni a efectores. Este tipo de capa proporciona grados de libertad a la red neuronal, gracias a los cuales puede encontrar representaciones internas correspondientes a determinados rasgos del entorno, proporcionando una mayor riqueza computacional.

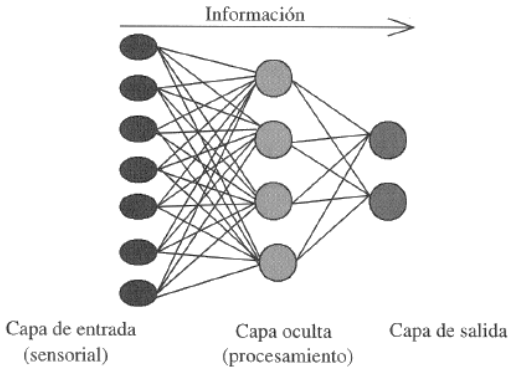


Figura 4.7: Arquitectura unidireccional de tres capas (entrada, oculta y salida) [19]

Atendiendo a distintos conceptos, pueden establecerse diferentes tipos de arquitecturas neuronales. Así, en relación a su estructura en capas, podemos hablar de redes monocapa y de redes multicapa. *Las redes monocapa* son aquéllas compuestas por una única capa de neuronas. *Las redes multicapa* son aquellas cuyas neuronas se organizan en varias capas.

Asi mismo, atendiendo al flujo de datos en la red neuronal, podemos hablar de redes unidireccionales (feedforward) y redes recurrentes (feedback). En las *redes unidireccionales*, la información circula en un único sentido, desde las neuronas de entrada hacia las de salida. En las *redes recurrentes o realimentadas*, la información puede circular entre las capas en cualquier sentido, incluido el de salida-entrada [19].

4.4.3. Mecanismos de aprendizaje

El procedimiento utilizado para realizar el proceso de aprendizaje es llamado algoritmo de aprendizaje y su función es determinar los pesos sinápticos de la red de forma ordenada para alcanzar un objetivo definido.

Todo conocimiento de una red neuronal se encuentra distribuido en las conexiones y pesos sinápticos de las neuronas. Una red neuronal aprende a través de un proceso de ajuste de sus pesos sinápticos. El aprendizaje es el proceso por el cual una red neuronal modifica sus pesos sinápticos en respuesta a una entrada, para proporcionar la salida adecuada.

Se define como algoritmo de aprendizaje al procedimiento numérico de ajuste de los pesos. Existe gran variedad de algoritmos de aprendizaje teniendo cada uno sus propias ventajas. Los algoritmos de aprendizaje difieren entre sí en el tipo de red para el cual están hechos y la forma como se formulan los cambios en los pesos sinápticos. El aprendizaje en una red neuronal puede ser supervisado, no supervisado o híbrido.

- *Aprendizaje supervisado.* Se le proporciona a la RNA una serie de ejemplos consistentes en unos patrones de entrada, junto con las salidas que debería dar la red. El proceso de entrenamiento consiste en el ajuste de los pesos sinapticos para que la salida de la red sea lo más parecida posible a la salida desada. Es por ello que se hace uso de alguna función que nos de cuenta del error o el grado de acierto que alcanza la red, lo cual permite modificar los pesos sinápticos entre las neuronas, con el fin de acercar las salidas reales a las deseadas.

- *Aprendizaje no supervisado.* Se presenta a la red un conjunto de patrones de entrada, pero no hay información disponible sobre la salida esperada. En este caso, el proceso de entrenamiento deberá ajustar sus pesos de acuerdo a las características, similitudes o categorías que se pueden establecer entre los datos que se presentan en su entrada.

- *Aprendizaje híbrido.* Es una mezcla de los anteriores. Unas capas de la red tienen un aprendizaje supervisado y otras capas, un aprendizaje de tipo no supervisado.

En la actualidad, se están realizando numerosas investigaciones relacionadas con muchos tipos de redes neuronales artificiales y cada vez se crean nuevas arquitecturas, paradigmas y algoritmos de aprendizaje o mejoramiento de los ya existentes.

4.5. El Asociador Lineal

El Asociador Lineal es una RNA unidireccional y hace parte de los modelos de redes neuronales estándar. Tiene su origen en los trabajos pioneros publicados por Anderson y Kohonen. James A. Anderson y Teuvo Kohonen obtuvieron resultados similares a pesar de que trabajaron independientemente .

Esta red neuronal artificial, mediante una transformación lineal, asocia un conjunto de patrones de entrada a otros de salida. Consta de dos capas: una de entradas denotada por $\mathbf{x}$ y una de salida denotada por $\mathbf{y}$. Asi mismo, denotaremos por $W=\{w_{ij}\}$ a la matriz de pesos sinápticos donde cada fila de W contiene los pesos presinápticos que llegan a una neurona i.

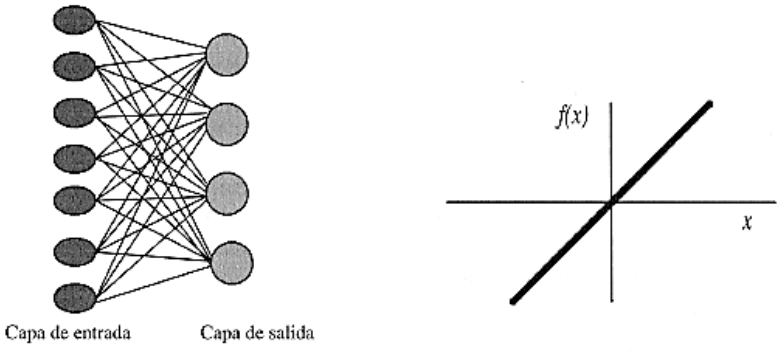


Figura 4.8: Arquitectura (izquierda) y función de activación (derecha) del Asociador Lineal [19]

Dentro del marco de neurona estándar, el Asociador Lineal calcula el potencial postsináptico por medio de la convencional suma ponderada, cantidad a la que aplica posteriormente una función de activación de tipo identidad.

La operación efectuada por el Asociador Lineal es:

$$\mathbf{y} = W\mathbf{x} \tag{4.4}$$

o bien

$$y_i = \sum_{\text{J}=1}^{n} w_{ij}x_j, \quad \text{con} \quad i = 1, ..., m. \tag{4.5}$$

El Asociador Lineal debe aprender a asociar p pares entrada-salida

$$D = \{(\mathbf{x}^\mu, \mathbf{y}^\mu), 1 \leq \mu \leq p\}, \tag{4.6}$$

ajustando la matriz de pesos W, de tal manera que ante entradas similares a $\mathbf{x}^\mu$ responda con salidas similares a $\mathbf{y}^\mu$. Para ello, se hace uso de una regla de aprendizaje, que a partir de los patrones de entrada con sus respectivas salidad deseadas proporcione el conjunto óptimo de pesos W.

4.5.1. La regla de Hebb

Se trata de uno de los modelos clásicos de aprendizaje en redes neuronales. Donald Hebb, en 1949 [13], postuló un mecanismo de aprendizaje para una neurona biológica, cuya idea básica consiste en que *cuando un axón presináptico causa la activación de cierta neurona, la eficacia de la sinapsis que las relaciona se refuerza.*

Si bien este tipo de aprendizaje es simple y local, su importancia radica en que fue pionero tanto en neurociencias como en neurocomputación, de ahí que otros algoritmos más complejos lo tomen como punto de partida.

Se denomina *aprendizaje Hebbiano* a un aprendizaje que involucra una modificación en los pesos, Δw_{ij}, proporcional al producto de una entrada x_j por la salida y_i de la neurona. Es decir,

$$\Delta w_{ij} = \epsilon y_i x_j, \tag{4.7}$$

siendo ϵ un parámetro denominado *ritmo de aprendizaje*, que suele ser una cantidad entre 0 y 1. Esta expresión es la representación matemática del modelo descrito por Hebb.

En concreto, al considerar el Asociador Lineal, la regla de actualización de pesos sinápticos se expresa como

$$w_{ij}^{new} = w_{ij}^{old} + \Delta w_{ij} \tag{4.8}$$

con Δw_{ij} dado por (4.7).

Las RNA se adaptan fácilmente al entorno principalmente modificando sus sinápsis, y aprenden de la experiencia, pudiendo generalizar conceptos a partir de casos particulares. En el caso del Asociador Lineal, la regla de Hebb formalmente puede resumirse en la siguiente dinámica de adaptación:

En el tiempo 0, todos los pesos en la configuración son cero, es decir, $w_{ij}^{(0)} = 0 \quad (i = 1, ..., m, \quad j = 1, ..., n)$. En el tiempo de adaptación discreto $t = 1, ..., p$, el μ-ésimo patrón de entrenamiento es presentado a la red y los pesos son actualizados de acuerdo con la regla de Hebb

$$w_{ij}^{t} = w_{ij}^{t-1} + y_i^{\mu} x_j^{\mu} \qquad i = 1, ..., m \quad j = 1, ..., n, \tag{4.9}$$

donde y_i^μ es el i-ésimo elemento de la salida esperada y^μ cuando la entrada es x^μ (ϵ es seleccionado como 1 por simplicidad).

Dado que la fase de adaptación termina, en este caso, después de los p pasos, cuando todos los patrones de entrenamiento han sido presentados, la configuración resultante se puede expresar, como una suma finita

$$w_{ij} = \sum_{\mu=1}^{p} y_i^\mu x_j^\mu \qquad i = 1, ..., m \quad j = 1, ..., n. \tag{4.10}$$

La dinámica de adaptación (4.9) del Asociador Lineal también puede ser escrita en la notación matricial:

$$W^{(0)} = 0,$$

$$W^{(\mu)} = W^{(\mu-1)} + \mathbf{y}^\mu \mathbf{x}^{\mu T}, \quad \mu = 1, ..., p.$$

La configuración resultante, de nuevo puede describirse como un producto de matrices (compárese con (4.10)):

$$W = W^{(p)} = \sum_{\mu=1}^{p} \mathbf{y}^\mu \mathbf{x}^{\mu T} = Y X^T. \tag{4.11}$$

donde las columnas de la matriz $X \in \mathbb{R}^{n \times p}$ y las de la matriz $Y \in \mathbb{R}^{m \times p}$ son los vectores de entrada de $\mathbf{x}^\mu$ y los de salida $\mathbf{y}^\mu$, respectivamente; es decir,

$$X = \begin{pmatrix} x_{11} & \cdots & x_{1p} \\ \vdots & \ddots & \vdots \\ x_{n1} & \cdots & x_{np} \end{pmatrix}, \quad Y = \begin{pmatrix} y_{11} & \cdots & y_{1p} \\ \vdots & \ddots & \vdots \\ y_{m1} & \cdots & x_{mp} \end{pmatrix}. \tag{4.12}$$

Especialmente, la dinámica de adaptación para el caso de la memoria autoasociativa donde $Y = X$ puede reescribirse:

$$W = X X^T. \tag{4.13}$$

Ahora vamos a suponer que los vectores de entrada $\{\mathbf{x}^1, ..., \mathbf{x}^p\}$ son ortonormales (necesariamente $p \leq n$). Esto significa que estos vectores son mutuamente ortogonales; es decir, $\mathbf{x}^{rT}\mathbf{x}^s = 0$ para $r \neq s$ ($1 \leq r, s \leq p$), y, al mismo tiempo, de norma 1; es decir, $\mathbf{x}^{rT}\mathbf{x}^r = 1$ ($r = 1, ..., p$).

Bajo esta condición, el Asociador Lineal posee la denominada *propiedad de reproducción*; es decir, la red responde con la salida deseada $\mathbf{y}^\mu$ ante la entrada $\mathbf{x}^\mu$ $(1 \leq \mu \leq p)$, cumpliendose

$$\begin{aligned} W\mathbf{x}^\mu &= \left(\mathbf{y}^1\mathbf{x}^{1T} + \cdots + \mathbf{y}^p\mathbf{x}^{pT}\right)\mathbf{x}^\mu \\ &= \mathbf{y}^1\left(\mathbf{x}^{1T}\mathbf{x}^\mu\right) + \cdots + \mathbf{y}^p\left(\mathbf{x}^{pT}\mathbf{x}^\mu\right) \\ &= \mathbf{y}^\mu \end{aligned} \tag{4.14}$$

y por tanto, la regla de Hebb ha conseguido en este caso que la red aprenda a realizar las asociaciones deseadas.

Eliminando la condición de ortogonalidad, se tiene (manteniendo el requisito de vectores de norma 1):

$$\begin{aligned} W\mathbf{x}^\mu &= \mathbf{y}^1\left(\mathbf{x}^{1T}\mathbf{x}^\mu\right) + \mathbf{y}^2\left(\mathbf{x}^{2T}\mathbf{x}^\mu\right) + \cdots + \mathbf{y}^p\left(\mathbf{x}^{pT}\mathbf{x}^\mu\right) \\ &= \mathbf{y}^\mu + \sum_{\nu\neq\mu}\mathbf{y}^\nu\left(\mathbf{x}^{\nu T}\mathbf{x}^\mu\right) \\ &= \mathbf{y}^\mu + \delta. \end{aligned} \tag{4.15}$$

La expresión anterior se denomina **expansión señal-ruido** y proporciona la salida deseada más un término adicional.

4.5.2. Regla de la pseudoinversa

Recordemos que la tarea del Asociador Lineal es producir la salida $\mathbf{y}^\mu$ para una entrada $\mathbf{x}^\mu$

$$W\mathbf{x}^\mu = \mathbf{y}^\mu \quad \mu = 1 \cdots p.$$

Si no es posible escoger una matriz de pesos tal que esas ecuaciones sean satisfechas en forma exacta, entonces interesa encontrar una solución aproximada. En este sentido, los algoritmos de aprendizaje para el Asociador Lineal habitualmente se deducen a partir de una función que mida el error en la salida de la red. Una de estas funciones es el error cuadrático medio de las salidas de la red $(W\mathbf{x}^\mu)$ respecto de las salidas deseadas $(\mathbf{y}^\mu)$:

$$E\left(W\right) = \frac{1}{p}\sum_{\mu=1}^{p}\left|\mathbf{y}^\mu - W\mathbf{x}^\mu\right|^2 = \frac{1}{p}\sum_{\mu=1}^{p}\sum_{\imath=1}^{n}\left(y_i^\mu - W_{i.}^T x^\mu\right)^2 \tag{4.16}$$

De esta manera, el problema de aprendizaje del Asociador Lineal se transforma en el de obtener un conjunto de pesos que minimicen la expresión (4.16).

Si definimos la norma de Frobenius $\|M\|^2$ de una matriz $M \in \mathbb{R}^{p \times q}$ como

$$\|M\|^2 = \|(m_{ij})\|^2 = \sum_{i=1}^{p} \sum_{j=1}^{q} m_{ij}^2$$

la ecuación (4.16) se puede expresar como

$$E(W) = \frac{1}{p} \|Y - WX\|^2 . \tag{4.17}$$

Con esta nomenclatura, una regla de aprendizaje basada en la utilización de la matriz pseudoinversa de Moore-Penrose [17] puede escribirse como

$$W = YX^+ \tag{4.18}$$

donde X^+ denota la pseudoinversa de X.

Esta elección para W minimiza el error cuadrático medio (Ecuaciones (4.16) y (4.17)); es decir, que es óptima respecto de este error [19].

Capítulo 5

Algoritmos de Aprendizaje del Asociador Lineal

En este capítulo, nos centraremos en el análisis de los métodos específicos que resuelven el problema de mínimos cuadrados lineales que se presenta en los algoritmos de aprendizaje del Asociador Lineal, a fin de obtener la matriz de pesos sinápticos, especialmente cuando son problemas lineales de rango deficiente y se desea obtener la solución de norma mínima.

Compararemos los algoritmos que usan pseudoinversa con aquellos que no la usan de acuerdo a los apartados anteriores, con base en una serie de simulaciones desde el punto de vista de la precisión y la complejidad del algoritmo. Para esta labor, utilizaremos MATLAB y analizaremos los resultados obtenidos.

5.1. Algoritmos de aprendizaje

El PMCL presente en los algoritmos de aprendizaje del Asociador Lineal, con el fin de encontrar la matriz de pesos W óptima, se plantea en los siguientes términos:

Dadas las matrices $X \in \mathbb{R}^{n\times p}$, con $n \geq p$ *y rango* $r \leq \min(n,p)$, y $Y \in \mathbb{R}^{m\times p}$, *encontrar la matriz* $W \in \mathbb{R}^{m\times n}$ *que minimice* $\|\mathbf{Y} - \mathbf{WX}\|_2$.

5.1.1. Sin uso de la matriz pseudoinversa

PMCL en el Asociador Lineal de rango completo.

- *Ecuaciones Normales*

El problema de encontrar la solución de mínimos cuadrados del sistema $WX = Y$, equivale a resolver el sistema lineal de la forma

$$WXX^T = YX^T. \tag{5.1}$$

Algoritmo 21 PMCL en el Asociador Lineal con Ecuaciones normales

Entrada: Dada $X \in \mathbb{R}^{n\times p}$ con $n \geq p$, $rango(X) = p$ y $Y \in \mathbb{R}^{m\times p}$.
Salida: Solución W del problema mín $\|Y - WX\|_2$.

1: Formar las matrices $C = XX^T$ y $D = YX^T$.
2: Calcular la factorización de Cholesky de $C = GG^T$, con G triangular inferior.
3: Resolver $ZG^T = D$ por sustitución hacia adelante.
4: Resolver $WG = Z$ por sustitución hacia atráz.

- *Householder(Givens)*

Sabiendo cómo se resuelve $A\underline{X} = B$ (Algoritmo 6), la solución del PMCL del sistema $WX = Y$ es relativamente fácil. Para ello, tomemos la transpuesta de la ecuación a resolver: $X^TW^T = Y^T$. Ahora basta realizar el cambio $\underline{X} = W^T$, $A = X^T$ y $B = Y^T$, y solucionar este nuevo sistema. Teniendo en cuenta que para la matriz $A \in \mathbb{R}^{p\times n}$, con $n \geq p$, el procedimiento implica algunos cambios.

En primer lugar, calculamos la factorización QR por el método de Householder (Givens) de la matriz A^T (en lugar de A). De este proceso, obtenemos que $Q^T A^T = R = \begin{bmatrix} R_1 \\ 0 \end{bmatrix}$, donde Q es una matriz ortogonal $n \times n$ y R_1 una triangular superior $p \times p$. La matriz original A será $A = \begin{bmatrix} R_1^T & 0^T \end{bmatrix} Q^T$.

Sustituyendo A en la ecuación $A\underline{X} = B$ obtenemos

$$\begin{bmatrix} R_1^T & 0^T \end{bmatrix} Q^T \underline{X} = B.$$

A continuación, particionamos la matriz Q tal que $Q = [Q_1\ Q_2]$ y si hacemos el cambio de variable $Z = \begin{bmatrix} Q_1^T \\ Q_2^T \end{bmatrix} \underline{X}$, tenemos

$$\begin{bmatrix} R_1^T & 0^T \end{bmatrix} Z = B.$$

Como $Z^T Z = (Q^T \underline{X})^T (Q^T \underline{X}) = \underline{X}^T Q^T Q \underline{X} = \underline{X}^T \underline{X}$, las normas euclídeas de $\underline{X}$ y Z serán iguales. Por consiguiente, particionando la matriz $Z \in \mathbb{R}^{n \times m}$ (para simplificar el cálculo de la matriz Z) en $\begin{bmatrix} Z_1 \\ Z_0 \end{bmatrix}$, donde $Z_1 \in \mathbb{R}^{p \times m}$ y $Z_0 \in \mathbb{R}^{(n-p) \times m}$ es una matriz nula, entonces, la solución de $\begin{bmatrix} R_1^T & 0^T \end{bmatrix} Z = B$ saldrá de resolver

$$R_1^T Z_1 = B.$$

La matriz solución $\underline{X}$ que buscamos resulta de deshacer el cambio de variable introducido, $\underline{X} = [Q_1\ Q_2] \begin{bmatrix} Z_1 \\ 0 \end{bmatrix}$. Una vez resuelta $\underline{X}$, tomamos $W = \underline{X}^T$, donde $W = Z_1^T Q_1^T$.

Algoritmo 22 PMCL en el Asociador Lineal con Householder(Givens)

Entrada: Dada $A = X^T \in \mathbb{R}^{p\times n}$ con $n \geq p$, $rango(A) = p$ y $B = Y^T \in \mathbb{R}^{p\times m}$.

Salida: $\underline{X} \in \mathbb{R}^{n\times m}$, solución del problema mín $\|A\underline{X} - B\|_2$.
$W = \underline{X}^T$, solución del problema mín $\|Y - WX\|_2$.

1: Calcular la factorización $Q^T A^T = R = \begin{bmatrix} R_1 \\ 0 \end{bmatrix}$, donde $Q \in \mathbb{R}^{n\times n}$ es ortogonal y $R_1 \in \mathbb{R}^{p\times p}$ triangular superior.
2: Sustituir la matriz $A = \begin{bmatrix} R_1^T & 0^T \end{bmatrix} Q^T$ en la ecuación $A\underline{X} = B$.
3: Particionar la matriz Q tal que $Q = [Q_1\ Q_2]$ y hacer el cambio de variable $Z = \begin{bmatrix} Q_1^T \\ Q_2^T \end{bmatrix} \underline{X}$.
4: Particionar la matriz Z en $\begin{bmatrix} Z_1 \\ Z_0 \end{bmatrix}$, donde $Z_1 \in \mathbb{R}^{p\times m}$ y $Z_0 \in \mathbb{R}^{(n-p)\times m}$ es una matriz nula.
5: La solución de $\begin{bmatrix} R_1^T & 0^T \end{bmatrix} Z = B$ saldrá de resolver $R_1^T Z_1 = B$.
6: Deshacer el cambio de variable introducido $\underline{X} = [Q_1\ Q_2] \begin{bmatrix} Z_1 \\ 0 \end{bmatrix}$.
7: Como $W = \underline{X}^T$, $W = Z_1^T Q_1^T$.

- *Gram Schmidt Clásico(Modificado)*

De igual manera, si hacemos el cambio $\underline{X} = W^T$, $A = X^T$ y $B = Y^T$, el procedimiento para la solución del problema mín $\|A\underline{X} - B\|_2$ es el siguiente.

Algoritmo 23 PMCL en el Asociador Lineal con Gram Schmidt Clásico(Modificado)

Entrada: Dada $A \in \mathbb{R}^{p\times n}$ con $n \geq p$ y $rango(A) = p$ y $B \in \mathbb{R}^{p\times m}$.

Salida: $\underline{X} \in \mathbb{R}^{n\times m}$, solución del problema mín $\|A\underline{X} - B\|_2$.
$W = \underline{X}^T$, solución del problema mín $\|Y - WX\|_2$.

1: Calcular la factorización QR de A^T por el algoritmo de Gram Schmidt clásico (Modificado), donde $Q \in \mathbb{R}^{n\times p}$ y $R \in \mathbb{R}^{p\times p}$ es triangular superior. La matriz original será $A = R^T Q^T$.
2: Sustituir en la ecuación $A\underline{X} = B$ obteniendo $R^T Q^T \underline{X} = B$ y resolver $R^T G = B$ por sustitución hacia atrás.
3: Calcular $\underline{X} = QG$. Entonces $\underline{X}$ es la solución al problema de mínimos cuadrados. Como $W = \underline{X}^T$, $W = G^T Q^T$

PMCL en el Asociador Lineal de rango deficiente

- *factorización QR con pivoteo*

Utilizando el cambio $\underline{X} = W^T$, $A = X^T$ y $B = Y^T$, la solución del problema mín $\|A\underline{X} - B\|_2$ se calcula de igual manera que en la subsection 2.4.3.

Algoritmo 24 PMCL en el Asociador Lineal y QR con Pivoteo de Columna

Entrada: Dada $A \in \mathbb{R}^{p\times n}$ con $n \geq p$ y $rango(A) = r$ y $B \in \mathbb{R}^{p\times m}$.

Salida: $\underline{X} \in \mathbb{R}^{n\times m}$, solución del problema mín $\|A\underline{X} - B\|_2$.
$W = \underline{X}^T$, solución del problema mín $\|Y - WX\|_2$.

1: Calcular la factorización QR con Pivoteo de columna de la matriz A (Algoritmo 8), $Q^T AP = R = \begin{bmatrix} R_{11} & R_{12} \\ 0 & 0 \end{bmatrix}$.

2: Eliminar el factor R_{12} de la matriz R con transformaciones de Householder, obteniendo

$$Q^T AP\Pi = \begin{bmatrix} \overline{R}_{11} & 0 \\ 0 & 0 \end{bmatrix} = T$$

3: Sustituir en la ecuacion $A\underline{X} = B$ obteniendo, $QT\Pi^T P^T \underline{X} = B$. Por lo tanto

$$\|A\underline{X} - B\|_2^2 = \left\|Q(T\Pi^T P^T X - Q^T B)\right\|_2^2 = \left\|T\Pi^T P^T X - Q^T B\right\|_2^2.$$

4: Considerar $Z = \Pi^T P^T \underline{X}$ donde $\underline{X} = P\Pi Z$ y $Q^T B = \begin{bmatrix} C \\ D \end{bmatrix}$, tal que

$$\|A\underline{X} - B\|_2^2 = \left\|\begin{bmatrix} \overline{R}_{11} & 0 \\ 0 & 0 \end{bmatrix} Z - Q^T B\right\|_2^2 = \left\|\overline{R}_{11}W - C\right\|_2^2 + \|D\|_2^2$$

y resolver $\overline{R}_{11}W = C$ donde $W = \overline{R}_{11}^{-1}C$. Además, Z tendrá norma mínima cuando tenga la forma

$$Z = \begin{bmatrix} W \\ 0 \end{bmatrix} = \begin{bmatrix} \overline{R}_{11}^{-1}C \\ 0 \end{bmatrix} = \begin{bmatrix} \overline{R}_{11}^{-1} & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} C \\ D \end{bmatrix} = \begin{bmatrix} \overline{R}_{11}^{-1} & 0 \\ 0 & 0 \end{bmatrix} Q^T B.$$

5: La solución que se busca es

$$\underline{X} = P\Pi Z = P\Pi \begin{bmatrix} \overline{R}_{11}^{-1} & 0 \\ 0 & 0 \end{bmatrix} Q^T B.$$

6: Finalmente, la solución del problema mín $\|Y - WX\|_2$ es $W = \underline{X}^T = Z^T\Pi^T P^T$

5.1.2. Usando la matriz pseudoinversa

Dadas las matrices $X \in \mathbb{R}^{n\times p}$, con $n \geq p$ y $Y \in \mathbb{R}^{m\times p}$, la solución $W \in \mathbb{R}^{m\times n}$ que minimiza $\|\mathbf{Y} - \mathbf{WX}\|_2$ está dada por $W = YX^+$. Veamos el cálculo de la matirz $X^+ \in \mathbb{R}^{p\times n}$.

PMCL en el Asociador Lineal de rango completo

Si X es de rango completo, la solución del sistema se determina como sigue.

Algoritmo 25 PMCL en el Asociador Lineal de rango completo

Entrada: Dada $X \in \mathbb{R}^{n\times p}$ con $rango(X) = p$ y $Y \in \mathbb{R}^{m\times p}$.
Salida: Solución W del problema mín $\|Y - WX\|_2$.

1: Si el $rango(X) = p$ con $n > p$, entonces $X^+ = \left(X^T X\right)^{-1} X^T$.
2: Si el $rango(X) = n$ con $n < p$, entonces $X^+ = X^T \left(XX^T\right)^{-1}$
3: Si $n = p = rango\,(X)$, entonces $X^+ = X^{-1}$.
4: Calcular $W = YX^+$.

- Para el caso 1, resolver el sistema $(X^T X)U = X^T$ de donde $W = YU$
- Para el caso 2, resolver el sistema $V(XX^T) = X^T$ de donde $W = YV$
- Para el caso 3, resolver el sistema $WX = Y$

PMCL en el Asociador Lineal de rango deficiente

- *Descomposición en Valores Singulares (DVS)*

Si el $rango(X) = r < p$, la solución del problema mín $\|\mathbf{Y} - \mathbf{WX}\|_2$ utilizando la descomposición en valores singulares se otiene de la siguiente manera.

Algoritmo 26 PMCL en el Asociador Lineal-Descomposición en valores singulares (DVS)

Entrada: Dada $X \in \mathbb{R}^{n\times p}$ con $n \geq p$, $rango(X) = r$ y $Y \in \mathbb{R}^{m\times p}$.
Salida: Solución W del problema mín $\|Y - WX\|_2$.

1: Calcular la descomposición en valores singulares de X tal que $X = U\Sigma V^T$, donde $U \in \mathbb{R}^{n\times n}$ y $V \in \mathbb{R}^{p\times p}$ son ortogonales y

$$\Sigma = \begin{bmatrix} \Sigma_1 & 0 \\ 0 & 0 \end{bmatrix}, \quad \Sigma_1 = \begin{bmatrix} \sigma_1 & 0 & \cdots & 0 \\ 0 & \sigma_2 & \cdots & 0 \\ & & \ddots & \\ 0 & 0 & \cdots & \sigma_r \end{bmatrix} \in \mathbb{R}^{r\times r}.$$

2: Calcular $X^+ = V\Sigma^+U^T$ donde

$$\Sigma^+ = \begin{bmatrix} \Sigma_1^{-1} & 0 \\ 0 & 0 \end{bmatrix}.$$

3: Para calcular la solución al PMCL en el Asociador Lineal dada por $W = YV\Sigma^+U^T$

- Particionar las matrices $U \in \mathbb{R}^{n\times n}$ y $V \in \mathbb{R}^{p\times p}$ por columnas, donde $U = [U_1 \quad U_2]$ y $V = [V_1 \quad V_2]$.
- Calcular los productos

$$\Sigma^+U^T = \begin{bmatrix} \Sigma_1^{-1} & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} U_1^T \\ U_2^T \end{bmatrix} = \begin{bmatrix} \Sigma_1^{-1}U_1^T \\ 0 \end{bmatrix} \quad \text{y}$$

$$V(\Sigma^+U^T) = [V_1 \quad V_2] \begin{bmatrix} \Sigma_1^{-1}U_1^T \\ 0 \end{bmatrix} = V_1\Sigma_1^{-1}U_1^T$$

- Finalmente, la expresión para W es

$$W = \left(Y \left(V_1 \left(\Sigma_1^{-1}U_1^T\right)\right)\right).$$

- *Factorización de Cholesky Generalizada*

La solución de mínimos cuadrados del sistema $WX = Y$, por éste método y utilizando el Algoritmo 16, es la siguiente.

Algoritmo 27 PMCL en el Asociador Lineal-Factorización de Cholesky Generalizada

Entrada: Dada $X \in \mathbb{R}^{n\times p}$ con $n \geq p$ y $rango(X) = r$ y $Y \in \mathbb{R}^{m\times p}$.
Salida: Solución W del problema mín $\|Y - WX\|_2$.

1: Calcular el Factor Generalizado de Cholesky $R \in \mathbb{R}^{p\times p}$ de la matriz simétrica semidefinida positiva $G = X^T X$, utilizando el Algoritmo 15 tal que $R^T R = X^T X$.
2: Remover las $p - r$ filas de ceros de la matriz R para obtener la matriz $L^T \in \mathbb{R}^{r\times p}$ de rango r, con $r < p$.
3: Calcular $X^+ = L\left(L^T L\right)^{-1}\left(L^T L\right)^{-1} L^T X^T$.
4: Para calcular la solución al PMCL en el Asociador Lineal dada por

$$W = YL\left(L^T L\right)^{-1}\left(L^T L\right)^{-1} L^T X^T$$

- Resolver primero el sistema

$$\left(L^T L\right) Z = L^T$$

- Finalmente, la expresión para W es

$$W = \left(\left(YZ^T\right)\left(ZX^T\right)\right).$$

- *Método de Ortogonalización de Gram Schmidt (OGS)*

Por el Método de Ortogonalización de Gram Schmidt (OGS), y de acuerdo al Algoritmo 17, la pseudoinversa se calcula como

$$X^+ = P\left[I \quad U\right]^T\left(I + UU^T\right)^{-1} BQ^T.$$

La solución del PMCL en el Asociador Lineal se determina como sigue.

Algoritmo 28 PMCL en el Asociador Lineal-Método de Ortogonalización de Gram Schmidt (OGS)

Entrada: Dada $X \in \mathbb{R}^{n\times p}$ con $n \geq p$ y $rango(X) = r$ y $Y \in \mathbb{R}^{m\times p}$.
Salida: W, solución del problema mín $\|Y - WX\|_2$.

1: Realizar una OGS no normalizada en las columnas de X. LLamar a este conjunto de vectores $\left[\mathbf{c}_1^*, \mathbf{c}_2^*, ..., \mathbf{c}_p^*\right]$.
2: Permutar los $\mathbf{c}_j^*$ de modo que $[\mathbf{c}_1, \mathbf{c}_2, ..., \mathbf{c}_p] = \left[\mathbf{c}_1^*, \mathbf{c}_2^*, ..., \mathbf{c}_p^*\right] P$, donde

$$\mathbf{c}_j \neq 0 \quad j = 1, 2, .., r$$

$$\mathbf{c}_j = 0 \quad j = r+1, ..., p.$$

3: Calcular γ_{ij} para $j = 1, ..., r$; $i = 1, ..., j$, de acuerdo con (3.6).
4: Calcular las matrices B y U. $B \in \mathbb{R}^{r\times r}$ es la matriz cuya (i, j)-ésima entrada es $\beta_{ij} = \frac{\gamma_{ij}}{\|\mathbf{c}_j\|}$ y $U \in \mathbb{R}^{r\times(p-r)}$ cuya (i, j)-ésima entrada es ω_{ij} dada por (3.23).
5: Realizar una OGS normalizada en las columnas de $\begin{bmatrix} U \\ I \end{bmatrix}$ obteniendo la matriz de vectores ortonormales $V = \begin{bmatrix} V_1 \\ V_2 \end{bmatrix}$ y calcular $M = \left(I + UU^T\right)^{-1} = I - V_1 V_1^T$.
6: $Q = \left[\frac{\mathbf{c}_1}{\|\mathbf{c}_1\|}, ..., \frac{\mathbf{c}_r}{\|\mathbf{c}_r\|}\right]$.
7: Finalmente, $X^+ = P\left[I \quad U\right]^T MBQ^T$. Entonces la solución al PMCL en el Asociador Lineal está dada por

$$W = \left(\left(Y\left(P\left(\left[I\ U\right]^T (MB)\right)\right)\right) Q^T\right).$$

- *Método de la Escalonada Reducida*

El cálculo de la matriz pseudoinversa se obtiene de la siguiente manera de acuerdo al Algoritmo 18

$$X^+ = P\left[H^+\ 0\right]EX^T.$$

El procedimiento para encontrar la solución del PMCL es el siguiente.

Algoritmo 29 PMCL en el Asociador Lineal-Método de la Escalonada Reducida

Entrada: Dada $X \in \mathbb{R}^{n\times p}$ con $n \geq p$ y $rango(X) = r$ y $Y \in \mathbb{R}^{m\times p}$.
Salida: W, solución del problema mín $\|Y - WX\|_2$.

1: Calcular la matriz aumentada $\left[X^TX \quad X^T\right]$ y realizar operaciones de fila hasta obtener la matriz escalonada $\left[EX^TX \quad EX^T\right]$ donde $EX^T = C$.
2: Permutar las primeras p columnas de modo que

$$EX^TXP = \begin{bmatrix} I & L \\ 0 & 0 \end{bmatrix} = \begin{bmatrix} H \\ 0 \end{bmatrix}.$$

3: Calcular $H^+ = [I \quad L]^+ = \begin{bmatrix} \left(I + LL^T\right)^{-1} \\ L^T\left(I + LL^T\right)^{-1} \end{bmatrix}$.
4: Finalmente, $X^+ = P\left[EX^TXP\right]^+C = P\left[H^+\ 0\right]C$.
5: Para calcular la solución al PMCL en el Asociador Lineal

- Particionar la matriz $C \in \mathbb{R}^{p\times n}$ por columnas, donde $C = \begin{bmatrix} C_1 \\ 0 \end{bmatrix}$ con $C_1 \in \mathbb{R}^{r\times n}$.
- Calcular el producto $\left(P\left[H^+\ 0\right]\right)\begin{bmatrix} C_1 \\ 0 \end{bmatrix}$
- Finalmente, la expresión para W es

$$W = \left(Y\left(\left(PH^+\right)C_1\right)\right)$$

- *Metodo basado en una variante del método de la Proyección del Gradiente*

La solución de mínimos cuadrados del sistema WX = Y de acuerdo al algoritmo 19, es la siguiente.

$$X^{+} = \left[X^{+}\mathbf{y}_1, X^{+}\mathbf{y}_2, \cdots, X^{+}\mathbf{y}_n\right].$$

La solución del PMCL se determina de la siguiente manera.

Algoritmo 30 PMCL en el Asociador Lineal-Método basado en una variante del método de Proyección del Gradiente

Entrada: Dada $X \in \mathbb{R}^{n\times p}$ con $n \geq p$ y $rango(X) = r$ y $Y \in \mathbb{R}^{m\times p}$.
Salida: W, solución del problema mín $\|Y - WX\|_2$.

1: Realizar una OGS en las columnas de X. Denotar los vectores ortonormales resultantes por $\mathbf{d}_1, \mathbf{d}_2, \cdots, \mathbf{d}_r$. Construir la matriz $\sum_{j=1}^{r} \mathbf{d}_j\mathbf{d}_j^T$ y denotar las columnas de esta matriz por $\mathbf{y}_1, \mathbf{y}_2, \cdots, \mathbf{y}_n$. Entonces, para cada j, $\mathbf{y}_j \in C_X$ y por el Teorema (3.6.1), construimos $\mathbf{v}_j = X^{+}\mathbf{y}_j, j = 1, ..., n$, de donde

$$X^{+} = [\mathbf{v}_1, \mathbf{v}_2, \cdots, \mathbf{v}_n].$$

Entonces la solución al PMCL en el Asociador Lineal está dada por

$$W = YX^{+}.$$

- *Método basado en el teorema de Cayley-Hamilton*

La solución al PMCL en el Asociador Lineal de acuerdo al Algoritmo 20 está dada como sigue.

Algoritmo 31 PMCl en el Asociador Lineal-Método basado en el teorema de Cayley-Hamilton

Entrada: Dada $X \in \mathbb{R}^{n\times p}$ con $n \geq p$ y $rango(X) = r$ y $Y \in \mathbb{R}^{m\times p}$.
Salida: W, solución del problema mín $\|Y - WX\|_2$.

1: Calcular los coeficientes x_k del polinomio caracteristico de la matriz XX^T, con $k = 0, ..., n$ por el método de Faddeev.
   - Conjunto de valores iniciales: $B_0 = I$, $q_0 = -1$
   - Calcular $X_k = XX^T B_{k-1}$
   - Calcular $q_k = \text{traza}(X_k)/k$
   - Calcular $B_k = X_k - q_k I$
   - Tomar $x_k = -q_k$

2: Sea k el máximo índice tal que $x_k \neq 0$. Por tanto, la pseudoinversa se obtiene como

$$X^+ = \begin{cases} -\frac{1}{x_k} X^T \left[(XX^T)^{k-1} + x_1 (XX^T)^{k-2} + \cdots + x_{k-1} I \right] & \text{si} \quad k > 0 \\ 0 & \text{si} \quad k = 0 \end{cases}$$

3: Para calcular la solución al PMCL en el Asociador Lineal
   - Sean $Z = XX^T \in \mathbb{R}^{n\times n}$ y $T = 0 \in \mathbb{R}^{n\times n}$

4: **para** $i = 0 : k - 2$ **hacer**
5: **para** $j : 1 : n$ **hacer**
6: $T(j, j) = T(j, j) + \mathbf{x}_i$
7: **fin para**
8: $T = TZ$
9: **fin para**
10: $T = T + x_{k-1} I_{n\times n}$
11: Finalmente, la expresión para W es

$$W = \left(-\frac{1}{x_k} \right) \left(Y \left(X^T T \right) \right).$$

5.2. Estudio experimental de los algoritmos

En esta sección, llevamos a cabo el análisis experimental comparativo de los algoritmos implementados que resuelven el problema de mínimos cuadrados lineales que se presenta en los algoritmos de aprendizaje del Asociador Lineal para el caso de rango deficiente.

Presentamos el tiempo promedio (en segundos) de cinco ejecuciones en MATLAB sobre matrices diferentes de cada uno de los algoritmos. Los resultados reflejan el tiempo necesario para el cálculo de la matriz de pesos W por cada uno de los algoritmos, para el caso de rango deficiente.

Los resultados obtenidos en la tabla 5.1 muestran que el método más rápido es *geninv* y el más lento es OGS. Cabe tener en cuenta que el método QR con pivoteo ocupó el primer lugar para las matrices X_1 y X_2 pero para matrices de orden superior como X_3 y X_4 y más grandes, *geninv* supera a QR con pivoteo.

Con respecto al método basado en las ideas de la proyección del gradiente (PGpseudo), el tiempo de cálculo es menor que el de la escalonada reducida (ERpseudo) ocupando los puestos cuarto y quinto, respectivamente.

De éste análisis, podemos concluir que el algoritmo *geninv* es una herramienta robusta y eficiente para el cálculo de la inversa de Moore-Penrose de una matriz grande y de rango deficiente, y por ende, para la solución del PMCL en el Asociador Lineal.

En cuanto a los algoritmos, PGpseudo y ERpseudo, podemos concluir que el basado en las ideas de la proyección del gradiente es más rápido que el de la escalonada reducida.

Algoritmos	Tiempo (segundos)
$X_1 \in \mathbb{R}^{32\times16}$, $rango(X_1) = 14$, $Y \in \mathbb{R}^{64\times16}$	
QRpivoteo	$6{,}5199e-04$
pinv	$8{,}2272e-04$
geninv	0,0022
OGSpseudo	0,0285
ERpseudo	0,0337
PGpseudo	0,0253
$X_2 \in \mathbb{R}^{64\times32}$, $rango(X_2) = 28$, $Y \in \mathbb{R}^{128\times32}$	
QRpivoteo	0,0034
pinv	0,0026
geninv	0,0012
OGSpseudo	0,1785
ERpseudo	0,1211
PGpseudo	0,1009
$X_3 \in \mathbb{R}^{128\times64}$, $rango(X_3) = 56$, $Y \in \mathbb{R}^{256\times64}$	
QRpivoteo	0,0057
pinv	0,0029
geninv	0,0020
OGSpseudo	1,2329
ERpseudo	0,4213
PGpseudo	0,3829
$X_4 \in \mathbb{R}^{256\times128}$, $rango(X_4) = 112$, $Y \in \mathbb{R}^{512\times128}$	
QRpivoteo	0,0106
pinv	0,0105
geninv	0,0098
OGSpseudo	9,8746
ERpseudo	1,4757
PGpseudo	1,4928

Tabla 5.1: Tiempo de ejecución de los Algortimos que resuelven el PMCL en el Asociador Lineal con rango deficiente

5.3. Ejemplo: Reconocimiento de Caracteres

El objetivo de esta sección es presentar un ejemplo ilustrativo de una de las aplicaciones del Asociador Lineal como es el reconocimiento de caracteres (las letras del alfabeto (A-Z)) y demostrar la importancia de utilizar la matriz pseudoinversa para el rendimiento de la red frente a patrones con ruido, en comparación con la regla de Hebb (Sección 4.5.1).

En este caso particular, trabajamos con el Asociador Lineal como una memoria Autoasociativa (el mismo patrón es usado para la entrada y la salida de la red neuronal) donde los patrones de entrada y salida sólo toman los valores 1 ó -1 (reemplazamos la función identidad por una función escalón).

Supongamos que disponemos de un conjunto de imágenes formado por 26 patrones correspondientes a las letras del alfabeto desde A a Z, donde cada imágen se representa por una matriz de tamaño 7×5.

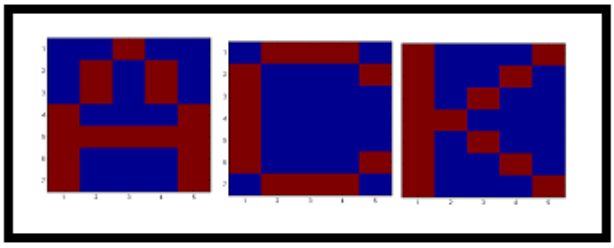


Figura 5.1: Algunos elementos del Conjunto de entrenamiento (Letras A, C, K)

El problema se divide en 5 secciones:

1. La generación de los vectores alfabéticos deseados.
2. El cálculo de la matriz de pesos **W** con la regla de la pseudoinversa/Hebb respectivamente.
3. Pruebas con patrones sin ruido
4. Pruebas con patrones ruidosos

5. Comparación de los resultados.

El código en MATLAB genera todas las secciones de la prueba.

Una vez que la red ha aprendido el conjunto de entrenamiento, queremos saber que tan bien funciona. Su desempeño lo probaremos con varios patrones de prueba, esto es, qué es lo que se aprende de hecho, nuevos patrones del mismo tipo que aprendió, para ver si puede generalizar, o con otras entradas para ver si puede discriminar. El pocentaje de ruido (No. de pixeles errados/-total de pixeles) que contendrán los patrones a recuperar serán del 0 %, 2 %, 4 %, 6 %, 8 %, 10 %, 20 %, 40 %, 80 %, y 100 %. La apariencia de los patrones afectados por estos porcentajes de ruido se ilustra a continuación.

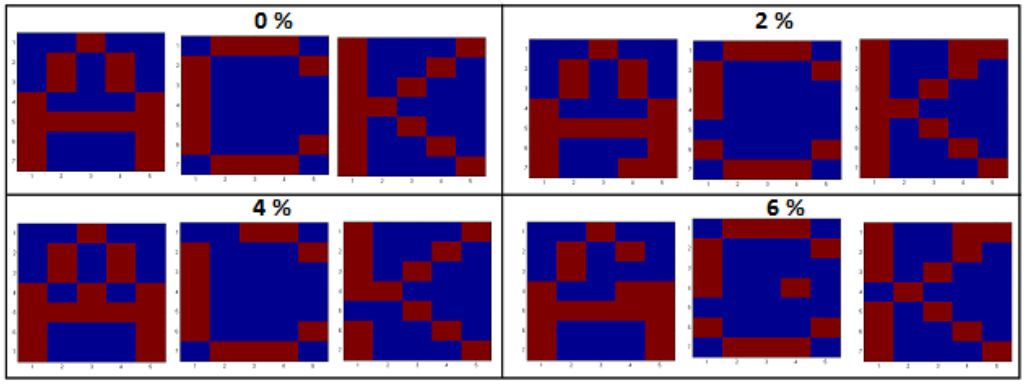


Figura 5.2: Patrones con nivel de ruido (0 %, 2 %, 4 %, 6 %)

En un primer experimento, entrenamos el Asociador lineal usando como conjunto de entrenamiento las imágenes correspondientes a las 26 letras del alfabeto haciendo uso de la regla de la pseudoinversa y posteriormente le presentamos a la red los prototipos ruidosos (Figura 5.2). Los resultados de la recuperación de algunos los patrones (A,C,K) se presenta a continuación.

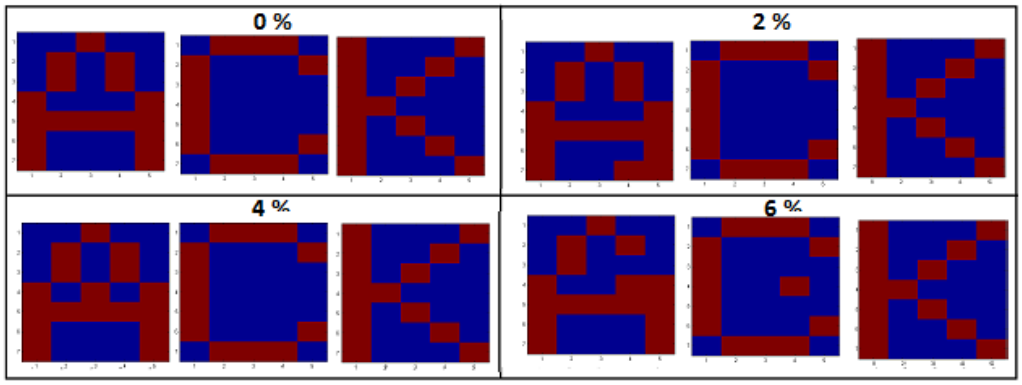


Figura 5.3: Apariencia de las letras A, C y K recuperadas por el Asociador Lineal entrenado con la Pseudoinversa (errores: 0 %, 2 %, 4 %, 6 %)

De los resultados, obtuvimos que con un nivel de ruido de hasta el 6 %, el Asociador Lineal entrenado con la regla de la Pseudoinversa recuperó algunos patrones del conjunto de entrenamiento de manera perfecta. Con un porcentaje de ruido superior la red tuvo problemas para la recuperación. En la Tabla 5.2, presentamos los resultados correspondientes al número de letras recuperadas al 100 % por la red.

[%] de ruido	0	2	4	6	8	10	20	40	80	100
Número de Letras Recuperadas al 100 %	26	5	5	1	0	0	0	0	0	0

Tabla 5.2: Recuperación de patrones para un cierto nivel de ruido (Entrenamiento: Regla de Pseudoinversa)

En la siguinte Tabla (5.3), presentamos los resultado referentes a los porcentajes de precisión que tuvo la red entrenada con la regla de la pseudoinversa, en la recuperación de las 26 letras para un nivel de ruido comprendido entre el 0 % y el 100 % .

[%] de ruido / Letras	0	2	4	6	8	10	20	40	80	100
A	100	97	97	94	91	88	74	48	17	0
B	100	97	97	94	91	88	68	57	17	0
C	100	100	100	97	91	91	71	42	14	0
D	100	97	97	94	94	91	62	48	22	0
E	100	100	97	94	94	88	74	48	22	0
F	100	97	97	94	91	88	74	45	22	0
G	100	97	97	94	91	88	74	48	179	0
H	100	97	97	97	94	94	68	51	17	0
I	100	97	97	94	91	88	68	45	17	0
J	100	97	100	97	94	94	74	51	20	0
K	100	100	100	100	91	88	71	54	17	0
L	100	97	97	97	91.	91	68	42	17	0
M	100	97	97	97	94	85	65	51	20	0
N	100	97	97	94	91	88	65	51	17	0
O	100	97	97	94	94	91	65	45	11	0
P	100	97	97	97	91	91	77	42	17	0
Q	100	97	97	94	91	88	71	57	20	0
R	100	97	97	94	94	88	65	57	17	0
S	100	97	97	94	94	91	65	42	200	0
T	100	97	97	97	91	91	68	54	14	0
U	100	100	97	97	91	85	71	40	20	0
V	100	97	100	94	91	85	71	48	20	0
W	100	97	97	97	91	88	68	48	17	0
X	100	97	100	94	91	88	68	48	17	0
Y	100	97	97	94	91	88	68	54	17	0
Z	100	100	97	97	94	88	62	48	17	0

Tabla 5.3: Porcentaje de recuperación de patrones para los niveles de ruido de las 26 letras (Entrenamiento: Regla de Pseudoinversa)

En un segundo experimento, entrenamos el Asociador Lineal haciendo uso de la regla de Hebb y luego le presentamos los mismos prototipos ruidosos usados en el primer experimento (Figura 5.2). En la Tabla 5.4, presentamos los resultados correspondientes al número de letras recuperadas al 100 % por la red y finalmente, en la Tabla 5.5, observamos la recuperación de un máximo de 4 patrones con un nivel error de hasta el 6 %. Para niveles superiores de ruido, ninguno de los patrones recuperados alcanzó el 100 %.

[%] de ruido	0	2	4	6	8	10	20	40	80	100
Número de Letras Recuperadas al 100 %	4	3	4	2	0	0	0	0	0	0

Tabla 5.4: Recuperación de patrones para un cierto nivel de ruido (Entrenamiento: Regla de Hebb)

[%] de ruido / Letras	0	2	4	6	8	10	20	40	80	100
A	88	82	82	82	74	91	71	60	25	11
B	94	94	94	97	91	91	80	28	8	5
C	97	91	97	91	94	97	94	54	8	2
D	100	100	100	97	97	97	100	31	2	0
E	88	88	91	88	94	91	77	25	5	11
F	97	91	91	97	97	97	82	60	20	2
G	94	94	91	97	94	91	94	48	2	5
H	94	94	94	97	90	94	91	34	8	5
I	100	100	100	100	97	97	85	57	8	0
J	91	91	94	94	97	85	74	42	17	8
K	85	94	91	85	77	74	82	22	20	14
L	85	85	88	80	85	88	82	57	8	14
M	91	88	88	88	91	88	91	25	8	8
N	88	88	88	91	85	88	85	48	14	11
O	97	97	97	97	97	97	97	65	2	2
P	94	94	97	100	88	97	94	40	11	2
Q	82	80	82	82	80	85	80	74	20	17
R	94	94	94	94	94	88	40	37	8	5
S	94	85	91	94	85	94	91	57	5	5
T	97	97	97	97	85	95	68	51	28	2
U	97	97	97	97	85	88	85	54	11	2
V	88	82	85	85	85	82	77	48	20	116
W	94	94	94	94	94	94	77	65	5	5
X	100	100	100	85	88	94	62	48	8	2
Y	100	91	100	88	88	85	77	42	25	0
Z	85	85	88	88	88	94	57	42	28	14

Tabla 5.5: Porcentaje de recuperación de patrones para los niveles de ruido de las 26 letras (Entrenamiento: Regla de Hebb)

Podemos concluir que en el problema, la regla de la pseudoinversa proporcionó mejores resultados que la de Hebb. Como vimos, la regla de la pseudoinversa aseguró el almacenamiento perfecto de los patrones, o en todo caso, su óptimo almacenamiento en el sentido del error cuadrático medio.

Capítulo 6

Red de Hopfield y Funciones de Base Radial

En este capítulo, presentamos una breve descripción de las RNA de Hopfield y Funciones de Base Radial, y analizaremos en especial el comportamiento de la matriz pseudoinversa en el proceso de aprendizaje de cada una de ellas.

6.1. Red de Hopfield

Una de las mayores contribuciones al área de las redes neuronales artificiales fue realizada por John Hopfield, quién formuló en 1982 un novedoso modelo de redes neuronales que se denominó Red de Hopfield.

Debido a la arquitectura y al funcionamiento, la red de Hopfield se puede incluir dentro de las redes de neuronas recurrentes, pues cada neurona está conectada con todas las demás y además, se produce un procesamiento temporal de los patrones. Lo que diferencia a esta red de las demás redes recurrentes es que actúa a la manera de una memoria asociativa; es decir, al igual que la mente humana, relaciona acontecimientos con sensaciones; estas redes asociarán patrones con variaciones de los mismos.

La *arquitectura de la red de Hopfield* consiste de una única capa de neuronas, donde cada una se conecta a todas las demás salvo a ella misma. Se trata de neuronas con un sencillo elemento de umbral, con entradas binarias $x_j(t)$ que pueden ser 0,1 o bien -1 o 1 y salidas que en prinipio llamaremos $y_i(t)$,

también binarias. La matriz de conexiones de la red de Hopfield es una matriz $W = (w_{ij})$, donde w_{ij} representa el peso de la conexión de la neurona i a la neurona j. Dicha matriz posee las siguientes particularidades:

- Es una matriz simétrica, es decir, $w_{ij} = w_{ji}$. Esto implica que el peso de la conexión de la neurona i a la neurona j es igual al peso de la conexión de la neurona j a la neurona i.

- Los elementos de la diagonal de la matriz son iguales a cero, es decir los $w_{ii} = 0$, debido a que en la red de Hopfield no existen conexiones de una neurona con ella misma.

La neurona i calcula el potencial postsináptico h_i, como la suma de las entradas ponderada con los pesos sinápticos, menos un cierto umbral de disparo

$$h_i(t) = \sum_j w_{ij} x_j(t) - \theta_i.$$

Finalmente, al potencial local se le aplica una función de tipo escalón $f(.)$ para obtener la salida digital de la neurona

$$y_i(t) = f(h_i(t)) = f\left[\sum_j w_{ij} x_j(t) - \theta_i\right].$$

Cuando las neuronas de la red de Hopfield hacen uso de valores $\{-1, 1\}$, se denominan de tipo *Ising*, siendo la función de activación que emplea $f(x) = sign(x)$. En este caso, el estado $+1$ indica una neurona activada, y el -1 desactivada.

Al comienzo (t=0), cada neurona recibe las entradas provenientes del exterior $x_j(0)$, calculando las salidas asociadas $y_i(0)$. Debido a las realimentaciones, estas salidas se convierten en las nuevas entradas de la red $x_j(1)$, proporcionando ésta una nueva salida $y_i(1)$. En general, ante las entradas $x_i(t)$, la red proporciona unas salidas $y_i(t)$, que se convierten en las nuevas entradas $x_i(t+1)$, de modo que la operación de la red de Hopfield se puede expresar como

$$x_i(t+1) = f\left[\sum_j w_{ij} x_j(t) - \theta_i\right], \tag{6.1}$$

regla que rige su **dinámica**. Habitualmente, diremos que el estado $\mathbf{x}$ de la red de Hopfield en t viene determinado por el estado de activación de todas y cada una de sus neuronas

$$\mathbf{x}(t) = (x_1(t) \ \ldots \ x_i(t) \ \ldots \ x_n(t))^T. \tag{6.2}$$

Dinámicas de la red: Dada la regla que rige la actualización del estado de una neurona individual (ecuación 6.1), podemos plantearnos a continuación los diferentes esquemas de actualización o dinámicas de las neuronas de la red:

- *Dinámica asíncrona.* En un instante t, solamente una neurona de la red, seleccionada aleatoriamente o siguiendo un orden preestablecido, actualiza su estado.
- *Dinámica síncrona o modo paralelo.* En un instante t, varias neuronas actualizan su estado a la vez. Si lo hacen todas, se tiene el modo completamente paralelo. No obstante, cuando hablemos de dinámica síncrona o paralela, nos referiremos a la actualización de todas a la vez por ser el caso más común.

La red de Hopfield como memoria asociativa-Estados estables. Una red de Hopfield, en cada iteración t, pasa de un estado $\mathbf{x}(t)$ a otro estado $\mathbf{x}(t+1)$. El proceso finalizará cuando se alcance un *estado estable* o punto fijo de la red $\mathbf{x}^*$, es decir, un estado que cumpla la siguiente condición a partir de un cierto t

$$\mathbf{x}(t+1) = \mathbf{x}(t) \equiv \mathbf{x}^*$$

pues ello supondrá que la salida de la red ya no cambia; es decir, que la red se ha estabilizado. En ese momento, podemos suponer que la red neuronal ha cambiado de procesar el patrón original procedente del exterior $\mathbf{x}(0)$, siendo $\mathbf{x}^*$ la respuesta final que proporciona.

Función energia de la red: Muchas de las investigaciones acerca de la estabilidad de las redes se basan en el establecimiento de una función, denominada *función energia* E de la red, para representar los posibles estados (puntos de equilibrio) de la red. Esta función tiene como propiedad ser monotónicamente decreciente, lo que asegura que la evolución en el tiempo de la red represente una trayectoria en el espacio de estados que busca un mínimo

de la función E.

La función energía de una red Hopfield discreta [19], con neuronas tipo umbral de salida $\{0, 1\}$ y dinámica asíncrona, por la cual en cada iteración la neurona a la que le corresponde actualizar su estado según (6.1), es elegida aleatoriamente, tiene la siguiente forma

$$E = -\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} w_{ij}x_i x_j + \sum_{i=1}^{n}\theta_i.x_i \tag{6.3}$$

Realizando un estudio rigurso [19], se llega a la concusión de que para que el funcional (6.3) sea verdaderamente función energía (es decir, que a partir de una dinámia asíncrona por la acción de (6.1) sea no creciente), bastan las siguientes condiciones

$$w_{ij} = w_{ji} \text{ y } w_{ii} \geq 0.$$

6.1.1. Aprendizaje

Un estado estable de la red es un mínimo local de la función energía, por lo tanto, los recuerdos o memorias que la red almacena son mínimos locales de la red, por lo que en el modelo de Hopfield, el aprendizaje consistirá en determinar los valores que deben tomar los pesos de la red para almacenar (o memorizar) como estados estables un conjunto de recuerdos o memorias dado.

La primera regla de aprendizaje propuesta para esta red fue la regla de Hebb, que el propio Hopfield sugirió en su trabajo original y que se estudia a propfundidad en [19]. No obstante, esta regla presenta los siguientes problemas: no asegura que los patrones de entrenamiento sean mínimos de la función energía, es decir, no se garantiza el almacenamiento exacto de dichos patrones; las memorias almacenadas pueden tener asociadas cuencas de atracción pequeñas, presenta una limitada capacidad de almacentamiento, y presupone que los patrones son no correlacionados y aleatorios. En este sentido, es interesante disponer de reglas de aprendizaje más eficientes que mejoren algunos de estos problemas.

Regla de la pseudoinversa.

Ésta regla, propuesta por Personnaz y otros [24], mejora el rendimiento de la red de Hopfield. Este algoritmo hace uso de la matriz pseudoinversa y es

válida para patrones aleatorios y no aleatorios, ortogonales y no ortogonales y asegura el almacenamiento de hasta un número de patrones igual al número de neuronas de la red. Pasemos a introducir esta regla de aprendizaje. Para ello, sea W la matriz de pesos sinápticos de la red de Hopfield, si $\mathbf{x}^\mu$ es uno de los p vectores a memorizar, se debe cumplir

$$signo\,[W.\mathbf{x}^\mu] = \mathbf{x}^\mu,$$

si trabajamos con neuronas $\{-1, +1\}$. Podemos sustituir la condición genérica anterior por otra particular, más restricitva

$$W.\mathbf{x}^\mu = \mathbf{x}^\mu,\ \mu = 1, ..., p.$$

Si colocamos los p patrones de aprendizaje $\mathbf{x}^\mu$ como columnas de cierta matriz Σ, ésta tendrá por dimensiones $n \times p$, y la condición anterior se puede reescribir como

$$W\Sigma = \Sigma. \tag{6.4}$$

El objetivo del aprendizaje será encontrar la matriz W que cumple esta condición. La solución de (6.4)

$$W = \Sigma\Sigma^+ \tag{6.5}$$

siendo Σ^+ la matriz pseudoinversa de Σ.

Con el algoritmo de aprendizaje basado en la pseudoinversa, obtenemos la matriz de pesos de la siguiente manera: construimos la matriz Σ con los patrones como columnas, la matriz de pesos W de la red de Hopfield se obtiene directamente a partir de ella aplicando (6.5); para calcular la pseudoinversa recurriremos al método más rápido y eficiente, *pinv*, resultante del análisis del Capítulo 4.

Finalmente, recordemos que si el conjuto de vectores de aprendizaje es linealmente independiente, la regla de la pseudoinversa da la matriz W que cumple la condición (6.4), en este caso, almacena la totalidad de los patrones de aprendizaje como memorias. Si los vectores son linealmente dependientes, la solución que proporciona es óptima en el sentido del error cuadrático medio.

6.1.2. Ejemplo: Reconocimiento de Dígitos

Un ejemplo ilustrativo de la operación de la red de Hopfield se basa en el reconocimiento de caracteres. En este trabajo, planteamos la recuperación de dígitos ruidosos utilizando una red de Hopfield discreta en la que se escogen como posibles estados de las neuronas los valores -1 o 1. Para ello, codificamos inicialmente un conjunto de 6 patrones correspondientes a los dígitos 0,1,2,3,4,6 (Figura 6.1), que serán los que la red aprenderá. Cada imágen se representa por una matriz de tamaño 12×10. Después iremos presentándole los mismos patrones pero con ruido añadido para observar como la red es capaz de eliminarlo.

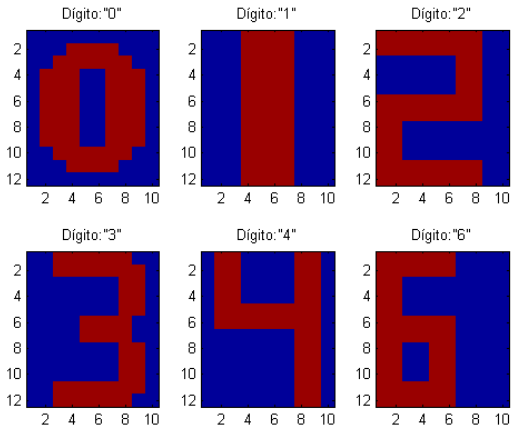


Figura 6.1: Conjunto de entrenamiento

En una primera experiencia, entrenamos la red de Hopfield discreta usando como conjunto de entrenamiento las imágenes correspondientes a los seis dígitos anteriores haciendo uso de la **regla de Hebb**. A continuación, introducimos un cierto nivel de ruido en los patrones originales (invirtiendo aleatoriamente un determinado número de píxeles) y presentamos los nuevos prototipos ruidosos a la red . El pocentaje de ruido (No. de pixeles errados/total de pixeles) que contendrán los patrones a recuperar serán del 0 %, 10 %, 20 %, 30 %, 40 %, 50 %, 60 %, 70 %, 80 %, 90 %, y 100 %. La apariencia de los patrones afectados por algunos de estos porcentajes de ruido se ilustra a continuación.

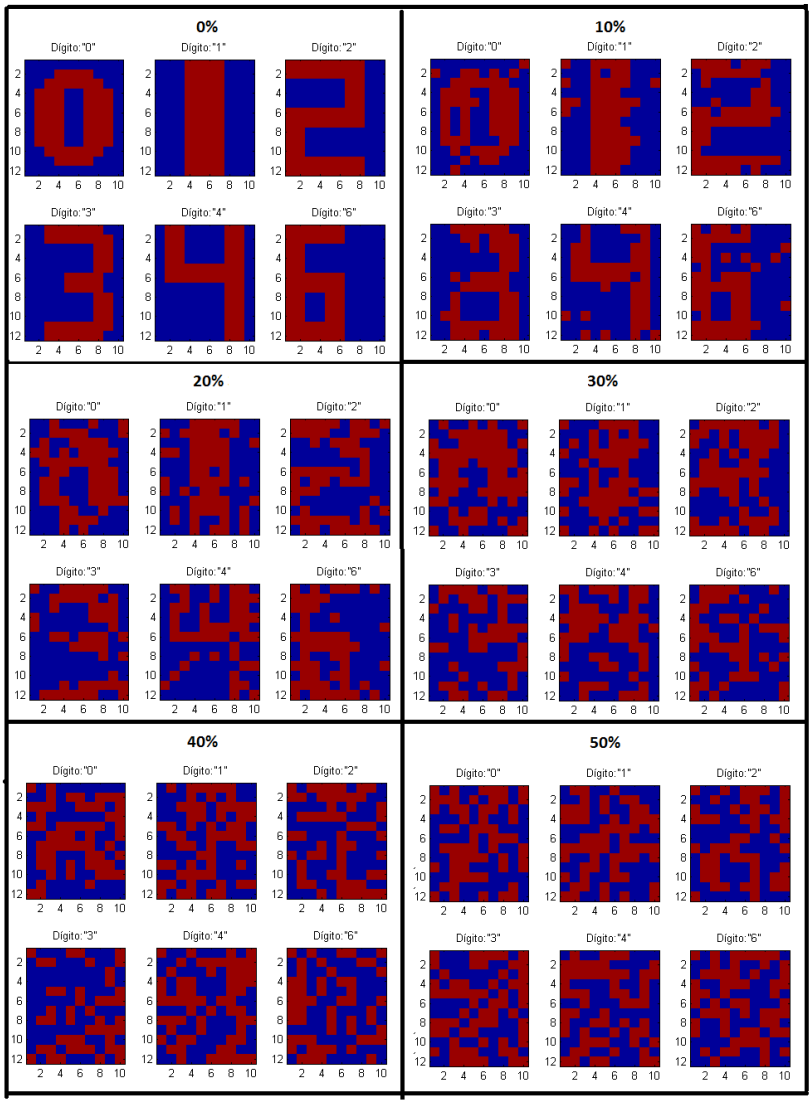


Figura 6.2: Patrones con nivel de ruido (0 %, 10 %, 20 %, 30 %, 40 %, 50 %)

Los resultados de la recuperación de los patrones por la red de Hopfield entrenada con la regla de Hebb los podemos observar en la Figura 6.3.

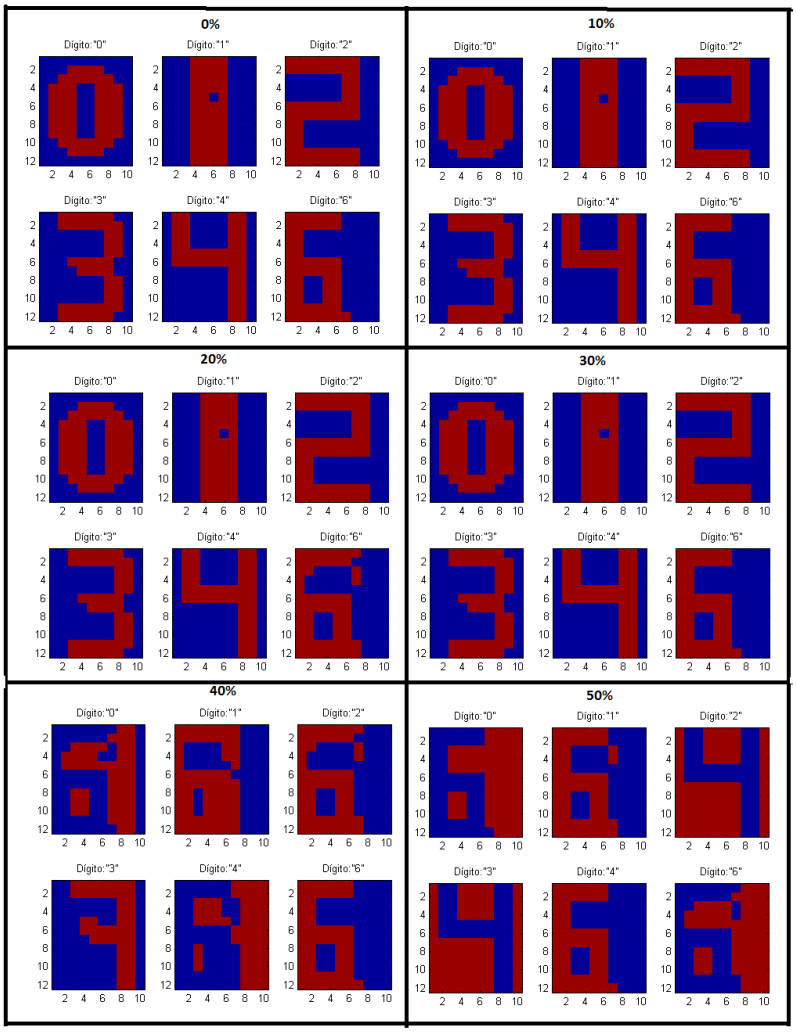


Figura 6.3: Apariencia de los patrones recuperados por la red de Hopfield entrenada con Hebb (Errores en la entrada del 0 %, 10 %, 20 %, 30 %, 40 %, 50 %)

Estos resultados ilustran que, con un nivel de ruido de hasta un 30 %, la red de Hopfield entrenada con Hebb recuperó hasta 3 patrones del conjunto de entrenamiento de manera perfecta. Con un nivel de ruido superior, la red tiene problemas para la recuperación, pues observamos que no es capaz de recuperar perfectamente ninguno de los seis patrones. En la tabla 6.1, presentamos los resultados referentes a la cantidad de dígitos recuperados al 100 %

por la red.

[%] de ruido	0	10	20	30	40	50	60	70	80	90	100
Número de Dígitos Recuperados al 100 %	3	2	3	3	0	0	0	0	0	0	0

Tabla 6.1: Recuperación de patrones para un cierto nivel de ruido (Entrenamiento: Regla de Hebb)

A continuación, presentamos, en la Tabla 6.2, los resultados referentes a los procentajes de precisión que tuvo la red etrenada con la regla de Hebb, en la recuperación de los seis dígitos y, en la Figura 6.4, los porcentajes de éxito en la recuperación de los patrones para un nivel de ruido comprendido entre el 0 % y el 30 %.

[%] de ruido / Dígitos	0	10	20	30	40	50	60	70	80	90	100
0	100	96	100	100	74	63	26	0	0	0	0
1	99	99	99	99	71	58	56	1	1	1	1
2	100	100	100	100	81	53	58	0	0	0	0
3	99	99	99	99	86	33	1	1	1	1	1
4	100	100	100	100	71	38	29	0	0	0	0
6	99	99	94	99	99	6	6	1	1	1	1

Tabla 6.2: Porcentaje de recuperación de patrones para los niveles de ruido de los seis dígitos(Entrenamiento: Regla de Hebb)

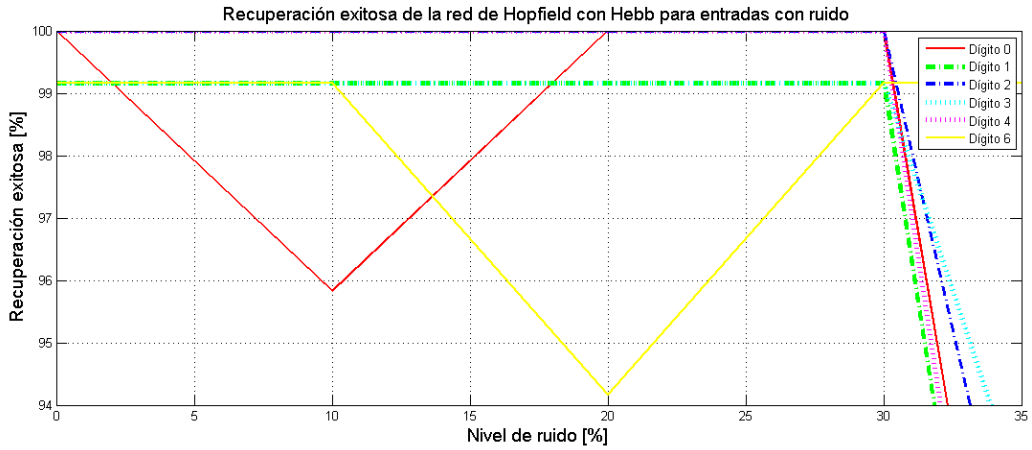


Figura 6.4: Porcentaje de éxito en la recuperación de patrones afectados por ruido del 0 %, 10 %, 20 % y 30 % (Entrenamiento: Regla de Hebb)

En una segunda experiencia, entrenamos la red haciendo uso de la Pseudo-inversa y luego le presentamos los mismos prototipos ruidosos usados en la primera experiencia (Figura 6.2). Como observamos en la Figura 6.5, con un nivel de ruido de hasta un 30 %, los seis patrones se recuperaron perfectamente. Cuando el nivel de ruido es del 40 %, la red recuperó sólo 2 dígitos. Para niveles superiores de ruido, ninguno de los patrones recuperados alcanzó el 100 %, aunque su imágen puede recordar la de alguno de ellos (Figura 6.5).

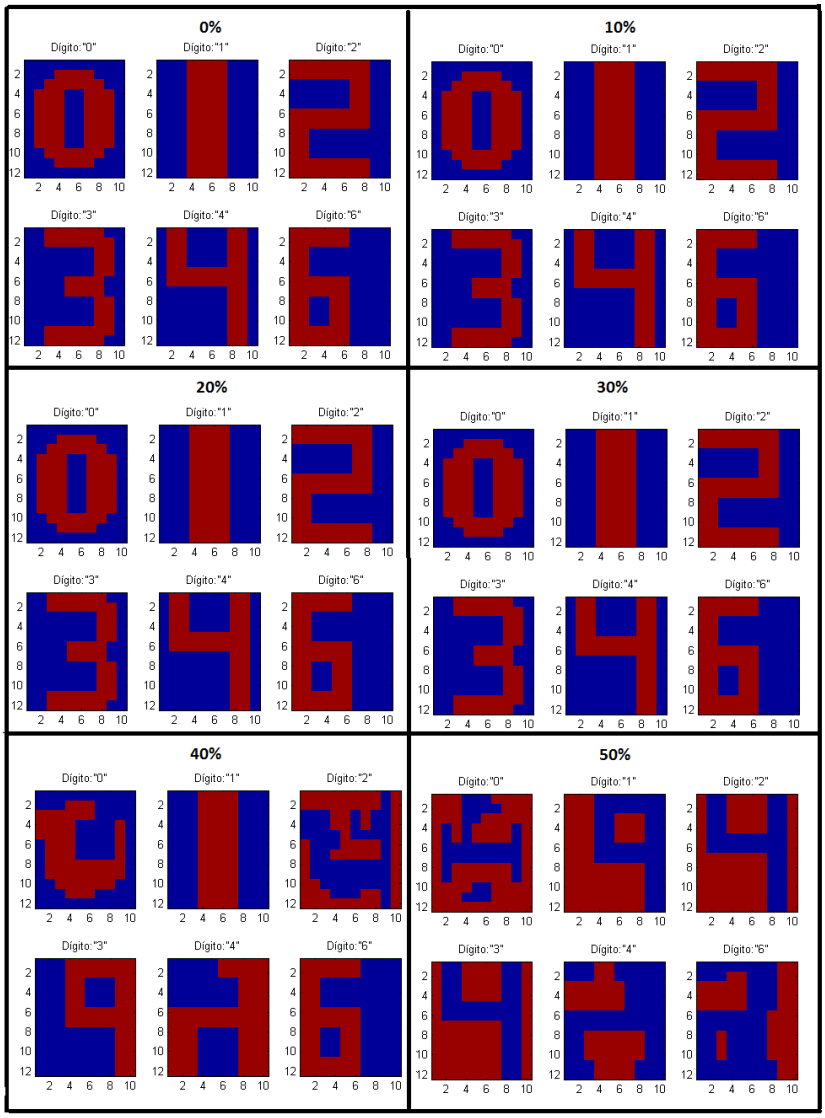


Figura 6.5: Apariencia de los patrones recuperados por la red de Hopfield entrenada usando la Pseudoinversa (Errores en la entrada del 0 %, 10 %, 20 %, 30 %, 40 %, 50 %)

En la Tabla 6.3, damos a conocer los resultados de la red de Hopfield entrenada con la Pseudoinversa, referentes a los patrones recuperados.

[%] de ruido	0	10	20	30	40	50	60	70	80	90	100
Número de Dígitos Recuperados al 100 %	6	6	6	2	0	0	0	0	0	0	0

Tabla 6.3: Recuperación de patrones para un cierto nivel de ruido

Finalmente, presentamos la Tabla 6.4 con los resultados de los porcentajes de exactitud que obtuvo la red entrenada con la regla de la pseudoinversa en la recuperación de los seis dígitos y la Figura 6.4, con los porcentajes de éxito en la recuperación de los patrones para un nivel de ruido comprendido entre el 0 % y el 30 %.

[%] de ruido / Dígitos	0	10	20	30	40	50	60	70	80	90	100
0	100	100	100	100 83	25	17,5	0	0	0	0	
1	100	100	100	100	100	45	35	0	0	0	0
2	100	100	100	100	74	53	36	0	0	0	0
3	100	100	100	100	63	33	30	0	0	0	0
4	100	100	100	100	55	43	36	0	0	0	0
6	100	100	100	100	100	27	12	0	0	0	0

Tabla 6.4: Porcentaje de la recuperación de patrones para un cierto nivel de ruido de los seis dígitos

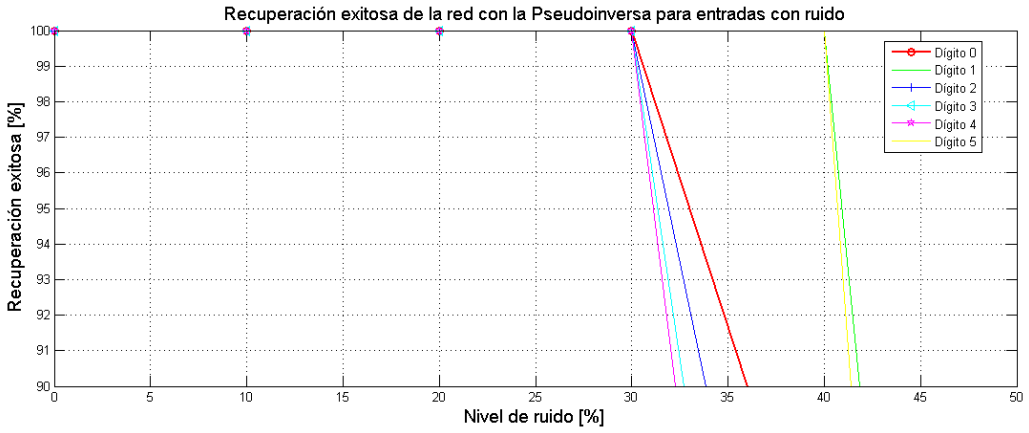


Figura 6.6: Porcentaje de éxito en la recuperación de patrones afectados por ruido del 0 %, 10 %, 20 % y 30 % (Entrenamiento: Pseudoinversa)

6.2. Funciones de Base Radial (RBF)

Las redes neuronales de Funciones de Base Radial son redes multicapa con conexiones hacia adelante formadas por tres capas de neuronas: la capa de entrada, una única capa oculta y la capa de salida. La capa de entrada la compone un conjunto de neuronas que reciben las señales del exterior, transmitiéndolas a la siguiente capa sin realizar ningún tipo de modificación sobre ellas. La capa oculta recibe las señales de la capa de entrada y realiza una transformación local y no lineal sobre dichas señales. Este carácter local (en el sentido de que cada neurona oculta de la red se activa en una región diferente del espacio de patrones de entrada) viene dado por el uso de las llamadas funciones de base radial, generalmente la función gausiana, como funciones de activación. Las neuronas de la capa de salida son lineales y esencialmente calculan la suma ponderada de las salidas que proporcionan la capa oculta.

Activación de las neuronas de la red de Función de base radial

Dada una red de neuronas de base radial, denominaremos x_i a las entradas de la red, y_j serán las salidas de la capa oculta, y z_k las salidas de la capa final (y globales de la red). La red se caracteriza porque las conexiones de la capa de entrada a la capa oculta no llevan asociado ningún peso, mientras que, y como es habitual en el contexto de las redes neuronales, las conexiones de la capa oculta a la capa de salida sí llevan asociado un número real o peso de la conexión. En lo referente a los umbrales de las neuronas, en las redes de base radial únicamente las neuronas de salida poseen un umbral, que también se suele tratar como una conexión más de la neurona cuya entrada es constante e igual a 1 o -1.

De manera general, entre la capa de entrada y la capa oculta existe un procedimiento que calcula la distancia euclídea r_j entre el vector de entrada x_i y su centroide c_{ji} (punto en el cual se centra la campana gaussiana de la función de activación):

$$r_j^2 = \|\mathbf{x} - \mathbf{c}_j\|^2 = \sum_i = (x_i - c_{ji})^2 .$$

La salida de la neurona y_j se calcula a partir de una función de activación

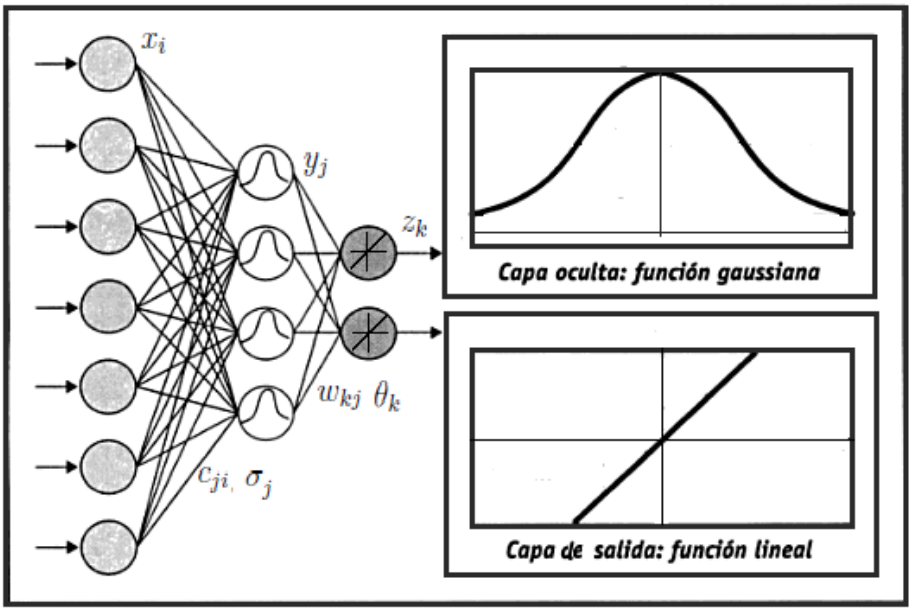


Figura 6.7: Arquitectura de la RBF

denominada *función radial* $\phi(r)$ que puede adoptar diferentes formas y expresiones, entre ellas

- Función gaussiana:
$$\phi(r) = e^{-r^2/2\sigma^2}$$
- Función inversa cuadrática:
$$\phi(r) = 1/\left(1 + r^2\right)$$
- Función inversa multicuadrática:
$$\phi(r) = 1/\sqrt{1 + r^2}.$$

El témino *función de base radial* procede precisamente de la simetría radial de estas funciones (el nodo da una salida idéntica para aquellos patrones que están a la misma distancia del centroide). En el contexto de las redes de neuronas de base radial, la más utilizada es la función gaussiana. El parámetro de normalización σ (desviación o factor de escala) mide la anchura de la

gaussiana, y equivaldrá al radio de influencia de la neurona en el espacio de las entradas; a mayor σ, la región que la nurona *domina* en torno al centroide es más amplia.

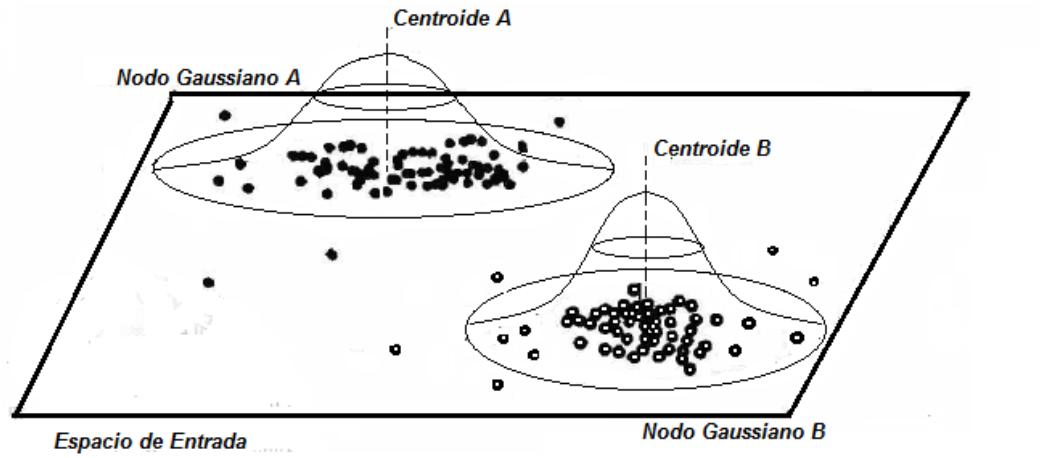


Figura 6.8: Respuesta localizada de las neuronas ocultas en la RBF

Con base en lo anterior, la salida de la neurona oculta j está dada por

$$y_j = e^{-r_j^2/2\sigma_j^2} = e^{-\sum_i (x_i - c_{ji})^2/2\sigma_j^2}. \tag{6.6}$$

Las salidas de las neuronas ocultas son a su vez las entradas de las neuronas de salida, las cuales calculan su respuesta z_k de la forma

$$z_k = \sum_j w_{kj} y_j + \theta_k = \sum_j w_{kj} \phi(r_j) + \theta_k, \tag{6.7}$$

siendo w_{kj} el peso que conecta la neurona oculta j con la de salida k, y θ_k el parámetro adicional de la neurona k (umbral).

6.2.1. Aprendizaje

El proceso de aprendizaje implica la determinación de todos los parámetros que intervienen en la red de base radial. Éstos son: los centroides y las desviaciones de las neuronas ocultas, y los pesos de la capa oculta a la capa de

salida, así como los umbrales de las neuronas de salida.

Método de aprendizaje híbrido.

Teniendo en cuenta que las capas de neuronas en una red de base radial realizan tareas diferentes, es razonable separar el proceso de estimación de los parámetros de la capa oculta y los de la capa de salida mediante la utilización de diferentes técnicas. Por tanto, uno de los mecanismos más usados para el aprendizaje de las redes de base radial es el llamado método híbrido, que combina dos fases:

- Fase no supervisada: determinación de los centroides y amplitudes de las neuronas de la capa oculta.
- Fase supervisada: determinación de pesos y umbrales de la capa de salida.

Fase no supervisada: Puesto que las neuronas ocultas de las redes de base radial se caracterizan porque representan zonas diferentes del espacio de patrones de entrada, los centros y las desviaciones de las funciones de base radial deben ser determinados con este objetivo; es decir, con el objetivo de clasificar el espacio de entrada en diferentes clases. El representante de cada clase será el centro de la función de base radial y la desviación vendrá dada por la amplitud de cada clase.

- **Determinación de los centroides c_{ji}: Algoritmo de k-medias**

El algoritmo de k-medias, es un algoritmo de clasificación no supervisado mediante el cual el espacio de patrones de entrada se divide en k grupos o clases. El número de clases es el número de neuronas ocultas (nodos gaussianos) en la red de base radial. Así, en este procedimiento hay que proponer en primer lugar un número k de grupos, es decir, de neuronas, luego

1. Elegimos los valores de los k centroides $\mathbf{c}_j$ de partida. Suelen tomarse como centroides los primeros k patrones de aprendizaje (la elección concreta no es relevante para el resultado final).
2. En cada iteración t, repartimos los patrones de aprendizaje $\mathbf{x}$ entre las k neuronas. Cada patrón se asigna a la neurona de cuyo centroide dista menos.

3. Calculamos los nuevos centroides de cada neurona como promedio de los patrones de aprendizaje asignados en el paso (2). Asi, definiendo N_j al número de patrones que han correspondido a la neurona j en el reparto, se tiene
$$c_{ji} = \frac{1}{N_j} \sum_{\mathbf{x} \in \text{neurona} j} \mathbf{x} \tag{6.8}$$

4. Si los valores de los centroides no han variado respecto de la iteración anterior, el algoritmo ya ha convergido. Si no es así, se repite el paso (2).

- **Determinación de las desviaciones σ_j de cada neurona**

Obtenidos los centroides, procedemos al cálculo de las desviaciones de cada neurona j, para lo cual hacemos uso de criterios heurísticos. Un procedimiento consiste en seleccionar los centroides de las N neuronas que están más próximos al del nodo j, y a continuación, calculamos el promedio de las distancias cuadráticas entre ellos; es decir
$$\sigma_j^2 = \frac{1}{N} \sum_{l=1}^{N} \|\mathbf{c}_l - \mathbf{c}_j\|^2 = \frac{1}{N} \sum_{l=1}^{N} \sum_{k} (c_{lk} - c_{ik})^2 . \tag{6.9}$$

El caso extremo $N = 1$, que consiste en tener en cuenta únicamente el noda más cercano, obviamente resulta el más rápido de calcular y en general proporciona buenos resultados en muchos casos.

Otro procedimiento consiste en el cálculo del promedio de la distancia de diversos patrones representativos al centroide, respecto al número de patrones N_j tomados para el cálculo de las desviaciones del nodo j,
$$\sigma_j^2 = \frac{1}{N} \sum_{x \in \text{nodo} j}^{p} \|\mathbf{x} - \mathbf{c}_j\|^2 .$$

Calculando las desviaciones de una forma u otra, finalizamos el entrenamiento de las neurona de la capa oculta.

Fase Supervisada: En esta fase, se calculan los pesos y umbrales de las neuronas de salida de la red. En este caso, el objetivo es minimizar las diferencias entre las salidas de la red y las salidas deseadas. Por tanto, el proceso

de aprendizaje está guiado por la minimización de una función error computada en la salida de la red

$$E = \frac{1}{2}\sum_{k}(t_k - z_k)^2,$$

donde t_k es la salida deseada y z_k la que produce la red.

Método de la Pseudoinversa

Debido a que la salida de la red (Ecuación 6.7) depende linealmente de los pesos y umbrales, el *método de la pseudoinversa* cálcula dichos parámetros proporcionando una solución directa al problema de optimización. Dicha solución viene dada por la siguiente expresión

$$W = G^{+}S, \tag{6.10}$$

donde W es la matriz de pesos y umbrales de la red de base radial, G es una matriz que contiene las salidas de las neuronas ocultas para los patrones de entrada y S es la matriz de salidas deseadas para la red.

6.2.2. Ejemplo: Clasificación de Patrones

El problema de clasificación consiste en establecer una correspodencia entre un conjunto de elementos dados y un conjunto de clases; es decir, se desea clasificar patrones de determinados objetos en una de las categorías preestablecidas; fundamentalmente, es un problema de agrupamiento.

Para cada patron, se extrae un número de características, las que sirven de entrada a un clasificador; éste entrega como resultado a que clase pertenece. Los patrones a clasificar pueden ser de cualquier naturaleza, letras, imágenes, sonidos, etc. Y tienen que ser codificados en un vector. La codificación de los patrones juega un papel muy importante en el problema de clasificación hasta el punto que una codificación no apropiada puede producir resultados no satisfactorios.

El patrón se describe mediante un vector de características. El proceso de clasificación consiste en construir un mapa de relaciones entre el espacio de características y el conjunto de las clases, de tal forma que se pueda

distinguir a qué clase corresponde cualquier patrón de entrada representado por el vector de características.

Aproximaciones al problema de clasificación

Las redes de neuronas de base radial son utilizadas para la resolución de problemas de aproximación, predicción o clasificación. Utilizan la distancia a un punto (centro) para determinar la activación de las neuronas. Debido a que cada neurona tiene una función de activación en particular; es decir, es posible que todas tengan diferente centro, cada una tiene una región de activación o región receptiva, la cual tiene un radio de acción que permite agrupar teniendo en cuenta la distancia mínima al centro de acción.

El código en MATLAB que presentamos en el Apéndice C, nos resuelve el problema de la función lógica XOR usando una red RBF. El propósito es ilustrar cómo este tipo de red puede solucionar problemas no linealmente separables apoyados en el uso de la matriz pseudoinversa en la segunda etapa de su proceso de aprendizaje.

En el siguiente ejemplo, definimos dos grupos de datos de clasificación no lineal (A,B) en un espacio de 2 dimensiones como se observa en la figura 6.9, donde para las entradas del grupo A(+), se desea una salida 1 y para las entradas del grupo B(*) se desea una salida -1.

La Neural Network Toolbox es un paquete de MATLAB que contiene una serie de funciones para crear y trabajar con redes de neuronas artificiales. Entre ellas, están *newrbe* y *newbr*, para crear funciones de base radial. En los dos, se utiliza el método de la matriz pseudoinversa para calcular los pesos de la capa de salida. Para este ejemplo, utilizamos la función *newrbe* (ver Apéndice C) puesto que el conjunto de patrones de entrenamiento es pequeño. Si tuvieramos aplicaciones cuyo conjunto de patrones de entrenamiento es grande, se recomienda utilizar el algoritmo de la función *newbr*.

Trabajando con la función *net=newrbe(p,t,spread)* para crear la RBF, donde p es la matriz de entrada a la red; es decir, en p se encuentran los patrones de entrenamiento de la red; $\mathbf{t}$ es la matriz de salida deseada, y *spread* es el parámetro de difusión de las funciones de base radial, por defecto 1. El valor de *spread* determina el ancho del área del espacio de entrada al que cada

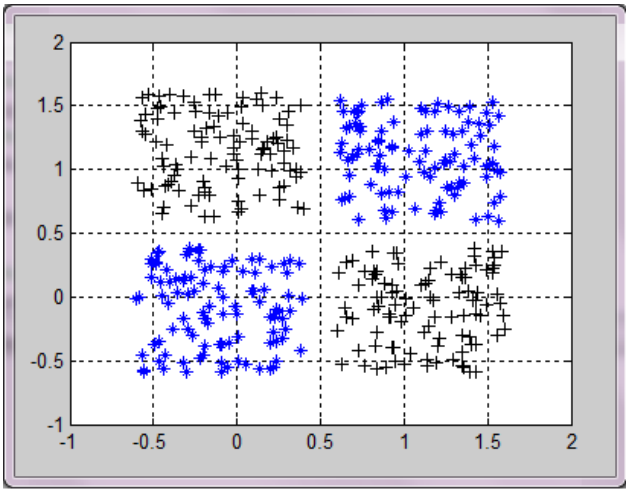


Figura 6.9: Datos de entrada de la RBF

neurona responde.

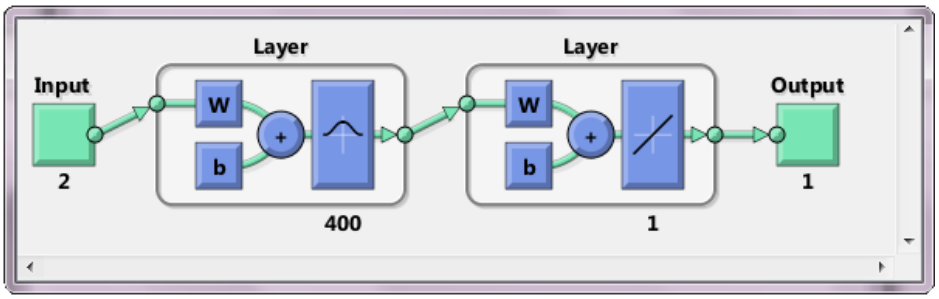
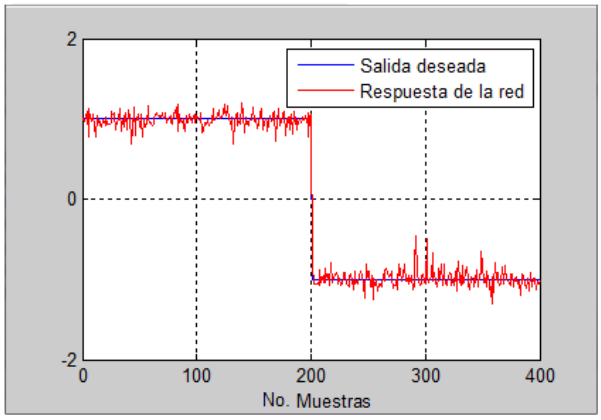


Figura 6.10: Red neuronal: Funcion de base radial

Una vez creada la red, el siguiente paso es realizar el entrenamiento con los patrones de entrada y las salidas deseadas, para luego evaluar el desempeño de la red calculando el porcentaje(%) de las clasificaciones correctas. La gráfica de las salidas deseadas y la respuesta de la red se muestra a continuación.

```
Dispersión          = 1.00
Numero de neuronas  = 400
Clases correctas    = 100.00 %
```



Tras la fase de entrenamiento, la red está lista para ser usada; es decir, la red es capaz de producir una salida adecuada a un conjunto de datos de entrada. La función Y = sim(NET,P) es la encargada de pasar un conjunto de datos de entrada a la red y de obtener su salida. Donde, NET representa una red entrenada, P es el conjunto de datos de entrada, Y es la salida de la red.

Finalmente, graficamos los resultados de la clasificación. Las salidas de los puntos de entrada del grupo A identificados en el plano por el símbolo +, se asocian a la superficie amarilla que representa la salida 1; y las salidas de los puntos del grupo B, identificados por $*$ se asocian a la superficie en color rojo que representa la salida -1.

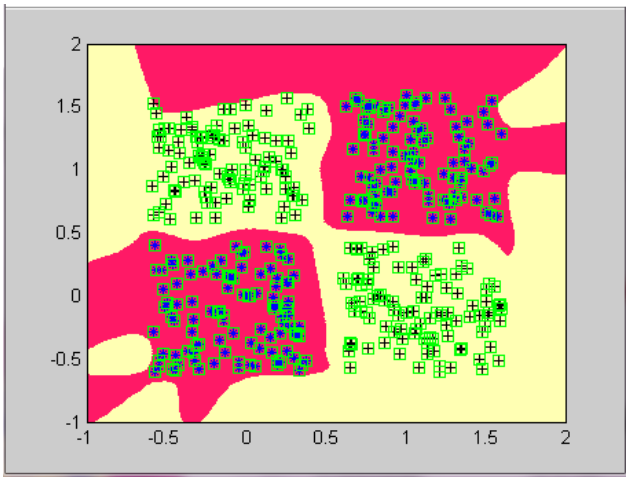


Figura 6.11: Regiones de clasificación de la RBF

En este ejemplo de clasificación, la RBF generó un número determinado de sectores que agrupan a los datos que poseen características similares en el espacio de entrada, llevando a cabo el proceso de clasificación.

Capítulo 7

Conclusiones y Trabajo futuro

De los estudios y análisis desarrollados en el presente trabajo, se concluye lo siguiente.

7.1. Conclusiones

- Los sistemas de ecuaciones $A\mathbf{x} = \mathbf{b}$ con $A \in \mathbb{R}^{m\times n}$, $m > n$ y $\mathbf{b} \in \mathbb{R}^m$ y $rango(A|\mathbf{b}) \neq rango(A) \leq n$, no tienen solución; sin embargo, se les puede encontrar una pseudosolución siguiendo el criterio de encontrar $\mathbf{x} \in \mathbb{R}^n$ que minimice la norma $\|A\mathbf{x} - \mathbf{b}\|_2$, o escoger, entre las pseudosoluciones existentes, aquella cuya norma sea mínima.

  En el caso de que una matriz A no sea de rango completo, las transformaciones ortogonales requieren modificaciones adecuadas para encontrar una solución. De otro lado, la matriz pseudoinversa permite la solución de norma mínima-2, independientemente del rango y de las dimensiones de la matriz.

  En este documento, estudiamos seis maneras distintas de calcular la pseudoinversa de Moore-Penrose. El primer método calcula la pseudoinversa a partir de la descomposición en valores singulares; el segundo está basado en la Factorización de Cholesky Generalizada; el tercero, en el proceso de Ortogonalización de Gramm-Schmidt; el cuarto, en el método de la Escalonada Reducida; el quinto se basa en las ideas de la Proyección del Gradiente, y el último es un procedimiento derivado del

Teorema de Cayley-Hamilton.

Posteriormente, las cuatro propiedades que caracterizan a la pseudoinversa fueron utilizadas para estudiar la exactitud de los algoritmos. El análisis nos muestra que ninguno de los cinco primeros algoritmos presentó errores en las matrices mayor que 2×10^{-10}, por lo tanto obtuvimos una aproximación fiable de la pseudoinversa. Sin embargo, el último algoritmo, basado en el teorema de Cayley-Hamilton, arrojó resultados confiables al calcular la pseudoinversa con matrices de orden pequeño, pero para el caso de matrices de orden elevado este algoritmo no fue capaz de producir un resultado numérico confiable de la matriz pseudoinversa debido a la construcción en forma recursiva de los coeficientes del polinomio característico, lo cual resulta no ser práctico para propósitos computacionales ya que no es numéricamente estable.

La función *pinv*, demostró ser la más precisa en la mayor parte de las pruebas, produciendo una aproximación fiable para matrices mal condicionadas con errores muy pequeños con respecto a los presentes en los otros métodos. La función *geninv* a pesar de ser un algorítmo rápido para calcular la pseudoinversa, el error de cálculo que presenta fue mayor en comparación con *pinv*, aunque la matriz obtenida es razonablemente precisa y tan fiable como la aproximación obtenida con los algoritmos basados en la escalonada reducida, OGS, y en las ideas de la proyección del gradiente que ocuparon el tercer, cuarto y quinto puesto en cuanto a precisión. En cuanto al algoritmo basado en el teorema de Cayley Hamilton, el resultado que proporciona no es confiable debido a la acumulación de errores en el cálculo de la pseudoinversa.

- El principal aporte del trabajo es el estudio y comparación de algoritmos de aprendizaje para la RNA conocida con el nombre de Asociador Lineal. En este sentido, hemos propuesto algoritmos para resolver el problema de mínimos cuadrados lineales presente en los algoritmos de aprendizaje de esta red para el caso de matrices con rango deficiente. Comparamos aquellos algoritmos que usan la matriz pseudoinversa (algoritmos: pinv(DVS en MATLAB), 27, 28, 29, 30, 31) con los que no la usan (algoritmo 24).

Con base a los análisis llevados a cabo en la sección 5.2, apreciamos en los resultados que *geninv*, *pinv*, y QR con pivoteo (algoritmo 24) son más rápidos computacionalmente que los algoritmos, 30 (PGpseudo, basado en las ideas de la proyección del gradiente), 29 (ERpseudo, basado en la escalonada reducida) y 28 (OGSpseudo, basado en el proceso de ortogonalización de Gramm-Schmidt).

- Analizamos el comportamiento de la matriz pseudoinversa en el proceso de aprendizaje de tres tipos de redes neuronales artificiales: Asociador Lineal, Hopfield y Funciones de Base Radial estudiadas en este trabajo.

  Como aplicación del Asociador Lineal, propusimos un ejemplo basado en el reconocimiento de las letras del alfabeto (A-Z). En este problema, la regla de la pseudoinversa aseguró el almacenamiento perfecto de los patrones si éstos son linealmente independientes, o en todo caso, su óptimo almacenamiento en el sentido del error cuadrático medio; y demostramos con esta implementación, la importancia de utilizar la matriz pseudoinversa para el rendimiento de la red frente a patrones con ruido, en comparación con la regla de Hebb.

  En el caso de la red de Hopfield, presentamos un ejemplo ilustrativo de la operación de ésta red basado en el reconocimiento de dígitos. En ésta experiencia, encontramos que ante la presencia de patrones de entrenamiento ruidosos o altamente distorsionados, el número de aciertos fue del 30 % para todos los dígitos. Además, el nivel de ruido tolerado por la red de Hopfield puede ser relativamente alto si se hace uso de un algoritmo de aprendizaje como el de la pseudoinversa, pues puede introducirse un exceso de ruido, y no por ello la red neuronal deja de converger al estado original correcto.

  Finalmente, propusimos un ejemplo de cómo la RBF resuelve el problema de la función lógica XOR e ilustramos cómo éste tipo de red puede solucionar problemas que no son linealmente separables haciendo uso en la fase supervisada del proceso de aprendizaje, del método de la

pseudoinversa.

7.2. Trabajo Futuro

- Las técnicas de Regularización son importantes en la solución del Problema de mínimos cuadrados lineales. El enfoque de ver este problema como un problema inverso lo pensamos sugerir como un trabajo de otra tesis o como un trabajo que abordaríamos posteriormente.

- El estudio, diseño, implementación y experimentación de distintas formas de multiplicar una cadena de matrices, es un problema que surge en los algoritmos de aprendizaje del Asociador Lineal donde principalmente se busca reducir el tiempo de cómputo de la matriz de pesos que resuelve el problema de mínimos cuadrados lineales. Como trabajo futuro se podría estudiar la eficiencia computacional de los algoritmos con uno de los métodos más eficientes, conocido como el Método de Programación dinámica.

- La capacidad de aprendizaje y almacenamiento, la eficiencia en la respuesta o recuperación de patrones, la rapidez y la inmunidad al ruido, son tópicos de interés en el campo de las redes neuronales artificiales, que pueden ser atacados de manera directa desde las matématicas aplicadas a fin de proponer variaciones y generalizaciones con ventajas claras sobre los modelos conocidos, propicias para su aplicación en problemas reales.

Apéndice A

Propiedades de la Pseudonversa

En ésta sección enunciaremos los teoremas que han sido empleados a lo largo del documento, sin embargo, no proporcionaremos una demostración de estos resultados pues dicha demostración puede encontrarse en la fuente especificada en cada teoorema.

Definición A.0.1. *Dada una matriz $A \in \mathbb{R}^{m\times n}$ con $rango(A) = r$, una factorización de la forma $A = FG$, donde $F \in \mathbb{R}^{m\times r}$, $G \in \mathbb{R}^{r\times n}$ y $rango(F) = rango(G) = r$, se conoce como descomposición de rango completo de la matriz A.*

Aparentemente (en 1959), C. C. MacDuffee fue el primero en observar que la factorización de rango completo de una matriz conduce a una fórmula explícita de su inversa de Moore-Penrose [15].

Teorema A.0.1 (MacDuffee [15])**.** *Si $A \in \mathbb{R}^{m\times n}$, con $rango(A) = r$, $r > 0$, tiene una factorización de rango completo $A = FG$, entonces*

$$A^{+} = G^{T}\left(F^{T}AG^{T}\right)^{-1}F^{T}. \tag{A.1}$$

Teorema A.0.2. *[28] Sea $R \in \mathbb{R}^{m\times r}$. Si $m \geq r$ y el $rango(R) = r$, entonces $R^{+}R = I$.*

Teorema A.0.3. *[1] Para toda matriz A*

$$rango(A) = rango(A^{T}A) = rango(A^{T}) = rango(AA^{T}).$$

Teorema A.0.4. *[1] Si A y B son matrices con el mismo número de filas, la ecuación $BX = A$ tiene solución, si y solo si, $C_A \subseteq C_B$. En este caso, cualquier matriz de la forma:*

$$X = B^{+}A + \left[I - B^{+}B\right] Y$$

(donde Y tiene el mismo número de filas y de columnas que X) es una solución. La solución es única, si y sólo si, $B^{+}B = I$ (la nulidad de B, es cero).

Teorema A.0.5. *[1] Si H es cualquier matriz y δ es distinto de cero, entonces $H^T H + \delta^2 I$ es no singular.*

Demostración. Si $\left(H^T H + \delta^2 I\right) x = 0$, entonces,

$$0 = x^T \left(H^T H + \delta^2 I\right) x = \|Hx\|^2 + \delta^2 \|x\|^2$$

que sólo puede ocurrir si $x = 0$. □

Teorema A.0.6. *[1] Si $C_B \subseteq C_A$, entonces, entre las matrices X que satisfacen $AX = B$, la que minimiza la traza de $X^T X$ es $\widehat{X} = A^{+}B$. Si $AZ = B$, entonces*

$$traza(Z^T Z) > traza(\widehat{X}^T \widehat{X}), \quad si\ Z \neq \widehat{X}.$$

Teorema A.0.7. *[25] Sea A una matriz de tamaño $m \times n$ y sean $\boldsymbol{d}_1, \boldsymbol{d}_2, \cdots, \boldsymbol{d}_r$, donde $r \leq n$, un conjunto ortonormal de vectores tal que*

$$C_A = Gen\left(\boldsymbol{d}_1, \cdots, \boldsymbol{d}_r\right)$$

entonces

$$AA^{+} = \sum_{j=1}^{r} \boldsymbol{d}_j \boldsymbol{d}_j^T.$$

Teorema A.0.8. *[8] Una condición necesaria y suficiente para que la ecuación matricial $PXQ = C$ tenga una solución es que $PP^{+}CQ^{+}Q = C$, en tal caso la solución general es*

$$X = P^{+}CQ^{+} + Y - P^{+}PYQQ^{+},$$

donde Y es una matriz del mismo tamaño de X.

Apéndice B

Códigos en MATLAB

A continuación, se muestran los códigos en MATLAB que realizan de las distintas rutinas generadas en este trabajo

B.1. PMCL para el caso de rango completo

B.1.1. Transformaciones de Householder

```
function [x Q R]=Hous(A,b)
[m n] = size(A);
x = zeros(n,1); Q = eye(m);
for j=1:n
    % Householder de a(j:m,j)
    w = Housv(A(j:m,j));
    A(j:m,j:n) = A(j:m,j:n)-
    ...2*w*(w'*A(j:m,j:n));
    b(j:m) = b(j:m)-2*w*...
    (w'*b(j:m));
    Qk = eye(m);
    Qk(j:m,j:m) = eye(m+1-j)
    ...-2*(w*w');
    Q = Qk*Q;
end
for i = n:-1:1  % Rx=b
    x(i) = (b(i)-x(i+1:n)'*...
    A(i,i+1:n)')/A(i,i);
end
% Matrices R y Q
R = triu(A);
Q = Q';
function w = Housv(x)
% Transformación de
    Householder del vector x.
m = max(abs(x)); w = x/m;
sw = 1;
if w(1)<0, sw = -1;
end
w(1) = w(1)+sw*norm(w);
w = w/norm(w); w = w(:);
```

B.1.2. Transformaciones de Givens

```
function x = Givens(A,b)
[m,n] = size(A);
x = zeros(n,1);
for i=1:n %Factorización de A
    for k = i+1:m
        if 1+abs(A(k,i))== 1,
            continue, end
          if abs(A(k,i))>= ...
          abs(A(i,i))
            t = A(i,i)/A(k,i);
            s = 1/sqrt(1+t*t);
            c = s*t;
            else
            t = A(k,i)/A(i,i);
            c = 1/sqrt(1+t*t);
            s = c*t;
            end
    A(i,i) =c*A(i,i)+s*A(k,i);
    q(i+1:n) = c*A(i,i+1:n)+s*
    ...A(k,i+1:n);
    A(k,i+1:n) = -s*A(i,i+1:n)
    ...+c*A(k,i+1:n);
    A(i,i+1:n) = q(i+1:n);
    %Transformar b
    q1 = c*b(i)+s*b(k);
    b(k) = -s*b(i)+c*b(k);
    b(i) = q1;
    end
end
%Sustitución hacia atrás
for i = n:-1:1
x(i) = (b(i)-A(i,i+1:n)*...
x(i+1:n))/A(i,i);
end
```

B.1.3. Método de Gram-Schmidt clásico

```
function [x Q R] =
Grmschclas(A,b)
[m,n] = size(A);
x = zeros(n,1);
Q = zeros(m,n);
R = triu(zeros(n,n));
for j = 1:n
    Q(:,j) = A(:,j);
    for i = 1:j-1
    R(i,j) = Q(:,i)'*A(:,j);
    Q(:,j) = Q(:,j)-R(i,j)
    ...*Q(:,i);
    end
    R(j,j) = norm(Q(:,j));
    Q(:,j) = Q(:,j)/R(j,j);
end
for i = n:-1:1  %Rx=b
x(i) = (Q(:,i)'*b-R(i,i+1:n)
...*x(i+1:n))/R(i,i);
end
```

B.1.4. Método de Gram Schmidt modificado

```
function [x Q R] =
Grmschmodif(A,b)
[m,n] = size(A);
x =zeros(n,1); Q = zeros(m,n);
R = triu(zeros(n,n));
for j = 1:n
    Q(:,j) = A(:,j);
    for i = 1:j-1
      R(i,j) = Q(:,i)'*Q(:,j);
      Q(:,j) = Q(:,j)-R(i,j)
      ...*Q(:,i);
    end
    R(j,j) = norm(Q(:,j));
    Q(:,j) = Q(:,j)/R(j,j);
end
for i = n:-1:1  %Rx=b
```

```
x(i)=(Q(:,i)'*b-R(i,i+1:n)
...*x(i+1:n))/R(i,i);
end
r = b; for j=1:n,
r=r-(Q(:,j)'*r)*Q(:,j);
end
```

B.2. PMCL para el caso de rango deficiente: Método QR con Pivoteo

Si A es una matriz $m \times n$ entonces $[QRP] = qr(A)$ calcula la matriz Q ortogonal, la matriz de permutaión P y la matriz triangular superior R tal que $Q^T AP = R$.

```
function [Q R P xLS] =
QRpivote(A,b)
[m,n] = size(A);
[Q,R,P] = qr(A,0);
r = sum(any(abs(R) >1e-5,2));
cb = size(b,2);
if r == n; Qt = Q'*b;
    xLS(P,1: cb) = R(1:n,:)\Qt;
else
% Cuando A es de rango
% deficiente.
    [Pr,S] = qr(R(1:r,:)',0);
    Qt = Q(:,1:r)'*b;
    xLS(P,1: cb) = Pr*(S'\ Qt)
        ;
end
```

B.3. PMCL en el Asociador Lineal usando la pseudoinversa

B.3.1. Descomposición en valores singulares-Función *pinv* de MATLAB

```
function W=pmclAL_pinv(X,Y)
[V1,S1P,U1T]=DVS_pinv(X);
W=(Y*(V1*(S1P*U1T)))

% función pinv
function [V1,S1P,U1T,X]=
    DVS_pinv(A);
if isempty(A)
X = zeros(size(A'),class(A));
  return
end
[m,n] = size(A);
if n > m
   X = pinv(A',varargin{:})';
else
[U,S,V] = svd(A,0);
if m > 1, s = diag(S);
elseif m == 1, s = S(1);
else s = 0;
end
if nargin == 2
tol = varargin{1};
else
tol = max(m,n) * eps(max(s));
```

```
end
r = sum(s > tol);
if (r == 0)
X = zeros(size(A'),class(A));
else
s = diag(ones(r,1)./s(1:r));
X = V(:,1:r)*s*U(:,1:r)';
end
V1=V(:,1:r);
U1T=U(:,1:r)';
S1P=s;
end
```

B.3.2. Factorización de Cholesky Generalizada

Código en MATLAB de la función geninv[5]:

```
function W=pmclAL_geninv(G,B);
[m,n]=size(G);transpose=false;
if m<n
transpose=true;
A=G*G';
n=m;
else
A=G'*G;
end
[L,Z]= geninv(G);
if transpose
W=((B*(G'*Z'))*Z);
else
W=(B*Z')*(Z*G');
end

%función geninv
function [L,Z,Y]= geninv(G);
% Trasponer si m < n
[m,n]=size(G);
transpose=false;
if m<n
transpose=true;
A=G*G';
n=m;
else
A=G'*G;
end
% Factorización de Cholesky
% de rango completo de A
dA=diag(A);
tol= min(dA(dA>0))*1e-9;
L=zeros(size(A));
r=0;
for k=1:n
r=r+1;
L(k:n,r)=A(k:n,k) -...
L(k:n,1:(r-1))*L(k,1:(r-1))';
% Nota: for r=1, el vector
% restado es cero
if L(k,r)>tol
L(k,r)=sqrt(L(k,r));
if k<n
L((k+1):n,r)=L((k+1):n,r)/...
L(k,r);
end
else
r=r-1;
end
end
L=L(:,1:r);
Z=(L'*L)\L';
if transpose
Y=(G'*Z')*Z;
      % Y=G'*Z'*Z;
else
Y=Z'*(Z*G');
      % Y =Z'*Z*G';
end
```

B.3.3. Método de Ortogonalización de Gram Schmidt (OGS)

```
function
...W=pmclAL_OGSpseudo(X,Y)
[ImUUUUt,B,Q,UU,P,UUFilas] =
... OGSpseudo(X);
W=((Y*(P*([(eye(UUFilas))UU]'
...*(ImUUUUt*B))))*Q');

% función OGSpseudo
function [ImUUUUt,B,Q,UU,P,
... UUFilas,Xm] = OGSpseudo(X);
[m,n]=size(X);
r=rank(X);
if r==n,
   Xm=(X'*X)\X';
else
u = OGSsinnormalizar(X,m,n);
conp1=0;
conp2=0;
P1=zeros(n,n);
P2=zeros(n,n);
for j=1:n
cc=find(u(:,j)~=0);
isn=find(isnan(u(:,j))==1);
if(isempty(cc)||~isempty(isn))
        u(:,j)=zeros(m,1);
        conp1=conp1+1;
        P1(j,conp1)=1;
      else
        conp2=conp2+1;
        P2(j,conp2)=1;
      end
end
P=[P2(:,1:conp2),
...P1(:,1:conp1)];
un=u*P;
Xp=X*P;
R=Xp(:,1:r);
S=Xp(:,r+1:end);
for j=1:r
      c(:,j)=zeros(m,1);
      for i=1:j
           ri=R(:,i);
           sumagama=0;
           if(i<j)
                for alfa=i:j-1,
sumagama=sumagama+dot
...(R(:,j),un(:,alfa))/dot(un
...(:,alfa),un(:,alfa))*gama
...(i,alfa);
                end
           gama(i,j)=-sumagama;
           else if(i==j)
                     gama(i,j)=1;
                else
                     gama(i,j)=0;
                end
           end
   c(:,j)=c(:,j)+gama(i,j)*ri;
    end
end
for j=1:r
      for i=1:j
      normc=norm(c(:,j));
      B(i,j)=gama(i,j)/normc;
      end
end
for j=1:n-r
      s(:,j)=zeros(m,1);
      for i=1:r
           ri=R(:,i);
           sumaomega=0;
           for alfa=i:r,
 sumaomega=sumaomega+dot
 ...(S(:,j),un(:,alfa))/dot
 ...(un(:,alfa),un(:,alfa))
 ...*gama(i,alfa);
           end
 omega(i,j)=sumaomega;
 s(:,j)=s(:,j)+omega(i,j)*ri;
 end
end
UU=omega;
for j=1:r
      for i= 1:j
```

```
85        unn=un(:,j);
86        normunn=norm(unn);
87        end
88  Q(:,j)=unn/norm (unn);
89  end
90  [UUFilas,UUCols]=size(UU);
91  UUident=[UU; eye
92  ...(UUFilas,UUCols)];
93  [filuuz,coluuz]=size(UUident);
94  v = OGSsinnormalizar
95  ...(UUident,filuuz,coluuz);
96  for j=1:coluuz
97        for i=1:filuuz
98              vnor=v(:,j);
99              normvnor=norm(vnor);
100       end
101 VV(:,j)=vnor/normvnor;
102 end
103 V1V1=VV(1:UUFilas,1:coluuz);
104 ImUUUUt=eye(UUFilas)-V1V1
105 ...*V1V1';
106 Xm=((P*([(eye(UUFilas)) UU]'
107 ...*(ImUUUUt*B)))*Q');
108 end
109
110 function
111 ...u = OGSsinnormalizar(A,m,n)
112 u = zeros(m,n);
113 u(:,1)=A(:,1);
114 for k=2:n
115  sum=zeros(m,1);
116       for j=1:k-1
117 sum=sum+((dot(A(:,k),u(:,j)))
          ...
118 /(dot(u(:,j),u(:,j))))*u(:,j);
119       end
120       vv=A(:,k)-sum;
121       fvv=find(abs(vv)>1e-10);
122       if(isempty(fvv));
123             fvv=zeros(m,1);
124       else
125             fvv=vv;
126       end
127    u(:,k) = fvv;
128 end
```

B.3.4. Método basado en la escalonada reducida

```
1  function
2  ...W=pmclAL_ERpseudo(X,Y);
3  [Hp,Ex1,P,r]=ERpseudo(X);
4  W=(Y*((P*Hp)*Ex1));
5
6  %función ERpseudo
7  function [Hp,Ex1,P,r,Xm]=
       ERpseudo(X);
8  [fil,col]=size(X);
9  r=rank(X);
10 if r==col,
11    Xm=(X'*X)\X';
12 else
13 XtX=X'*X;
14 XtXXt=[XtX X'];
15 EXtXEXt=rref(XtXXt);
16 EXtX=EXtXEXt(1:col,1:col);
17 conf=1;
18 P=zeros(col,col);
19 for i=1:1:col,
20       for j=i:1:col,
21       if(EXtX(i,j)==1)
22             P(j,i)=1;
23             break
24       else
25             P(j,r+conf)=1;
26             conf=conf+1;
27       end
28       end
29 end
30 P=P(1:end,1:size(EXtX,1));
31 EXtXP=EXtX*P;
32 L=EXtXP(1:r,r+1:col);
33 [LFilas,LCols]=size(L);
34 Lident=[L;eye(LFilas,LCols)];
35 [filuuz,coluuz]=size(Lident);
```

```
v = OGSsinnormalizar(Lident,
...filuuz,coluuz);
for j=1:coluuz
    for i=1:filuuz
        vnor=v(:,j);
        normvnor=norm(vnor);
    end
VV(:,j)=vnor/normvnor;
end
V1V1=VV(1:LFilas,1:coluuz);
IpLtInv=eye(LFilas)
...-V1V1*V1V1';
Hp=[IpLtInv; L'*IpLtInv];
EXt=EXtXEXt(:,col+1:end);
Ex1=EXt(1:r,:);
Xm=(P*(Hp*Ex1));
end

function
...u = OGSsinnormalizar(A,m,n)
u = zeros(m,n);
u(:,1)=A(:,1);
for k=2:n
 sum=zeros(m,1);
    for j=1:k-1
sum=sum+((dot(A(:,k),u(:,j)))
.../(dot(u(:,j),u(:,j))))*u(:,
    j);
    end
    vv=A(:,k)-sum;
    fvv=find(abs(vv)>1e-10);
    if(isempty(fvv));
        fvv=zeros(m,1);
    else
        fvv=vv;
    end
  u(:,k) = fvv;
end
```

B.3.5. Método basado en el método de proyección del gradiente

```
function
...W=pmclAL_PGpseudo(X,Y);
Xm=PGpseudo(X);
W=Y*Xm;

%función PGpseudo
function Xm= PGpseudo(X);
[m,n]=size(X);
d = OGSsinnormalizar(X,m,n);
con=n;
XXp=0;
for j=1:n
cc=find(d(:,j)~=0);
isn=find(isnan(d(:,j))==1);
if(isempty(cc)||~isempty(isn))
    d(:,j)=zeros(m,1);
    con=j;
    break
    else
    normd=norm(d(:,j));
    dnorm(:,j)=d(:,j)/normd;
XXp=XXp+dnorm(:,j)
...*dnorm(:,j)';
    end
end
Xt=X';
h = OGSsinnormalizar(Xt,n,m);
%vamos a calcular A+=[A+b1,
%...,A+bm]
    for col_XXp=1:m,
    x(:,1)=zeros(n,1);
    b=XXp(:,col_XXp);
            for k=2:m+1,
cc=find(abs(h(:,k-1))>1e-12);
            if(isempty(cc))
                alfa(k-1)=0;
            else
alfa(k-1)=(b(k-1)-Xt(:,k-1)'
...*x(:,k-1))/(h(:,k-1)'*Xt
...(:,k-1));
            end
x(:,k)=x(:,k-1)+alfa
```

```
... (k-1)*h(:,k-1);
            end
    xfinal(:,col_XXp)=x(:,k);
     end
Xm=xfinal;

function
...u = OGSsinnormalizar(A,m,n)
u = zeros(m,n);
u(:,1)=A(:,1);
for k=2:n
 sum=zeros(m,1);
      for j=1:k-1
sum=sum+((dot(A(:,k),u(:,j)))
      .../(dot(u(:,j),u(:,j))))
      ...*u(:,j);
     end
     vv=A(:,k)-sum;
     fvv=find(abs(vv)>1e-10);
     if(isempty(fvv));
          fvv=zeros(m,1);
     else
          fvv=vv;
     end
   u(:,k) = fvv;

end
```

B.3.6. Método basado en el teorema de Cayley-Hamilton

```
function
...W=pmclAL_CHpseudo(X,Y);
[T,a_indice]=CHpseudo(X);
W=(-(1/(a_indice)))*(Y*(X'*T))

%función CHpseudo
function
...[T,a_indice,Xm]=CHpseudo(X)
[m,n]=size(X);
if rank(X)==n,
   Xm=(X'*X)\X';
else
XXt=X*X';
[fil,col]=size(XXt);
pol=poly(XXt);
for i=col+1:-1:1
if(pol(i)>1e-5 ||
...pol(i)<-1e-5)
          indice=i-1;
          break
     end
end
 a_indice=pol(indice+1);
polT=pol(1:indice);
Z=XXt;
T=zeros(fil);
for i=1:indice-1,
     for j=1:fil,
       T(j,j)=T(j,j)+polT(i);
     end
     T=T*Z;
end
T=T+polT(indice)*eye(fil);
Xm=-(1/(a_indice))*(X'*T);
end
```

B.4. Ejemplos de redes neuronales

B.4.1. Asociador Lineal y Reconocimiento de Caracteres

```
%ASOCIADOR LINEAL
%(Memoria Autoasociativa)
delete diary.txt
diary diary.txt
close all
clear all
display('Desea calcular
...la matriz de pesos W con
... la pseudoinversa?')
reply=input('s=si
... n=no : ','s')
if isempty(reply)
    reply = 's';
end
%%%%%%%%%%%%%%%%%%%%
%1) Generando los vectores
%alfabéticos deseados
%%%%%%%%%%%%%%%%%%%%
%matriz de 35x26,7x5
%caracteres alfabeticos
load alpha.mat
%matriz de comprobación
%m=35 n=26
[m,n] = size(alpha)
%A es la primera columna
%de alpha
%A=alpha(:,1);
%A a Z son las columnas de
    alpha
%Graficar A
%A es el primer vector
    columna
A=alpha(:,1);
plot_vector7_5(A)
title('7x5 gráfica de "A"')
%%%%%%%%%%%%%%%%%%%%
%2) Calculando la matriz de
%pesos W con la pseudoinversa
%%%%%%%%%%%%%%%%%%%%
%Calcula la pseudoinversa
%de alpha (pinv-SVD)
pseudoin_alpha=pinv(alpha);
%pseudoin_alpha=geninv(alpha)
%pseudoin_alpha=
%...OGSpseudo(alpha)
%pseudoin_alpha=
%..eliminacion(alpha)
%pseudoin_alpha=
%proyecciongrad(alpha)
%pseudoin_alpha=Cayley(alpha)
[m,n] = size
...(pseudoin_alpha)
if (strcmp(reply,'s'))
%Calcular la matriz de pesos
%W usando alpha*pseudoinversa
W=alpha*pseudoin_alpha;
PlotRealMatrix(W)
title('Matiz de pesos
... con la Pseudoinversa')
else
%%%%%%%%%%%%%%%%%%%%
%5) Comparando los resultados
%sin usar la matriz
%...pseudoinversa
%(Regla de Hebb)
%Esta es una prueba para ver
%lo que sucede sin la pseud.
%%%%%%%%%%%%%%%%%%%%
    W=alpha*alpha';
    PlotRealMatrix(W)
title('Matriz de pesos sin
... la Pseudoinversa')
end
%%%%%%%%%%%%%%%%%%%%
%3)Probando la memoria
%autoasociativa sin ruido
%%%%%%%%%%%%%%%%%%%%
error_save=[]
```

```
77 % Grafica sin ruido
78 for i=1:26
79 a=hardlims(W*alpha(:,i));
80 t=alpha(:,i);
81 % calcula el número de errores
82 error=sum(abs(t-a))
83 error_save=[error error_save];
84 % todas las letras
85 % Gráfica deseada
86 plot_vector7_5(t)
87 title('Deseada')
88 % Comparando con la salida
89 % simulada
90 plot_vector7_5(a)
91 title('simulada')
92 end
93 figure
94 plot(error_save)
95 title('error en la salida con
96 ... patrones sin ruido')
97 xlabel('número de caracter
98 ... alfabético')
99 ylabel('errores en los
100 ... caracteres')
101 %%%%%%%%%%%%%%%%%%%%
102 % 4) Probando la memoria
103 % autoasociativa con ruido
104 %%%%%%%%%%%%%%%%%%%%
105 error_save=[]
106 noise_save=[]
107 noise_range = .8:1;
108 for noiselevel = noise_range
109 % gráfica con ruido
110 P = alpha + randn(35,26)*
    noiselevel;
111 for i=1:26
112 % entradas con ruido
113 in=P(:,i);
114 test=W*t;
115 plot_vector7_5(test)
116 title(['with noise
117 ... without limiting '
118 ... num2str(noiselevel)])
119 a=hardlims(W*in);
120 plot_vector7_5(a)
121 title(['with noise with
122 ... limiting' num2str
123 ...(noiselevel)])
124 t=alpha(:,i);
125 % Cálculo del no. de errores
126 error=sum(abs(t-a))
127 error_save=[error error_save];
128 noise_save=[noiselevel
129 ... noise_save];
130 end
131 end
132 figure
133 plot(noise_save,error_save,
134 ... 'rx')
135 title('error en la salida vs
136 ... nivel de ruido')
137 xlabel('nivel de error')
138 ylabel('número de errores
139 ... en la salida')
140 diary
141 edit diary.txt
142
143
144
145 %%%%%%%%%%%%%%%%%%%%
146
147 function plot_vector7_5(vector
    )
148 %convertir un vector de tamaño
149 %35x1 a una matriz de tamaño 7
    x5
150 n=1;
151 for i=1:7
152 out(i,:)=vector(n:n+4)';
153 n=n+5;
154 end
155 % Graficar un vector con la
156 % función imagesc
157 figure
158 % Encuentra el valor mínimo
159 % de la matriz
160 min_value=min(min(out));
161 % Encuentra el valor máximo
162 % de la matriz
163 max_value=max(max(out));
```

```
% Establecer limites mínimo
% y máximo con colorbar
clims=[min_value max_value];
imagesc(out,clims)
colormap('jet')
colorbar
H = gca;
set(H,'YDir','reverse')


%%%%%%%%%%%%%%%%%%
```

```
function...
PlotRealMatrix(matrix)
figure
min_value=min(min(matrix));
max_value=max(max(matrix));
clims=[min_value max_value];
imagesc(matrix,clims)
colormap('jet')
colorbar
H = gca;
set(H,'YDir','reverse')
```

B.4.2. Red de Hopfield y Reconocimiento de Dígitos

```
% Hopfield---Aprendizaje de
% Hebb vs Regla de la
% Pseudoinversa
clear all
close all
clc
load digits.mat
load noise_digits.mat
for m=1:size(noise_digits,
...1)/12,
% Creación la matriz de
% entrada
X=zeros(120,8);
for i=1:8
    x_temp=digits(:,:,i);
    X(:,i)=x_temp(:);

end
%%%%%%%%%%%%%%%%%%
% Cálculo de la matriz de
% pesos W con la regla de Hebb
% con modificacion
% haciendo ceros los elementos
% de la diagonal para evitar
% auto-retroalimentación
% (auto-feedback)

%W=X*(X') - 8*eye(120,120);
% Cálculo la matriz de pesos
%W con la Pseudoinversa
W=X*pinv(X);
W=W-diag(diag(W));

% Procesamiento a través de la
% red neuronal para los
    dígitos
% distorsionados
Y=zeros(120,6);
% Bucle para cada dígito
% (posición: 1 a 6,
% y son 0,1,2,3,4,6)
s_new=noise_digits((m-1)
...*12+1:m*12,:,i);
s_new=s_new(:);
s_old=zeros(length(s_new),1);
while (sum(abs(s_new-s_old))
...~=0)
s_old=s_new;

% actualización de cada
% elemento de "s" en orden
% aleatorio
for j=perm(1:length(s_new))
s_new(j)=sign(W(j,:)*s_new);
end
end
Y(:,i)=s_new;
end
```

```
% Graficando dígitos
% con ruido
figure;
colormap([0 0 0.6;0.6 0 0])
subplot(2,3,1),
...imagesc(noise_digits
...((m-1)*12+1:m*12,:,1));
title('Dígito:"0" ');
subplot(2,3,2),
...imagesc(noise_digits
...((m-1)*12+1:m*12,:,2));
title('Dígito:"1" ');
subplot(2,3,3),
...imagesc(noise_digits
...((m-1)*12+1:m*12,:,3));
title('Dígito:"2" ');
subplot(2,3,4),
...imagesc(noise_digits
...((m-1)*12+1:m*12,:,4));
title('Dígito:"3" ');
subplot(2,3,5),
...imagesc(noise_digits
...((m-1)*12+1:m*12,:,5));
title('Dígito:"4" ');
subplot(2,3,6),
...imagesc(noise_digits
...((m-1)*12+1:m*12,:,6));
title('Dígito:"6" ');
% Graficando dígitos
    reconstruidos;
figure;
colormap([0 0 0.6;0.6 0 0])
subplot(2,3,1),imagesc(reshape
...(Y(:,1),12,10));
title('Dígito:"0" ');
subplot(2,3,2),imagesc(reshape
...(Y(:,2),12,10));
title('Dígito:"1" ');
subplot(2,3,3),imagesc(reshape
...(Y(:,3),12,10));
title('Dígito:"2" ');
subplot(2,3,4),imagesc(reshape
...(Y(:,4),12,10));
title('Dígito:"3" ');
subplot(2,3,5),imagesc(reshape
...(Y(:,5),12,10));
title('Dígito:"4" ');
subplot(2,3,6),imagesc(reshape
...(Y(:,6),12,10));
title('Dígito:"6" ');
%%%%%%%%%%%%%%%%%%
dig=[1,2,3,4,5,6];
for i=1:6,
    y=reshape(Y(:,i),12,10);
    yr=y-digits(:,:,dig(i));
    dif=find(yr~=0);
    res1(i)=length(dif)/1.2;
end
res(:,m)=res1;
end
res_exito=100-res
save('exito2','res',...
'res_exito') %para
    pseudoinversa
% save('exito1','res',..
% 'res_exito') %para hebb

%%%%%%%%%%%%%%%%%%%%%%%%

clear all
close all
clc
load digits
levels=[0,10,20,30,40,50,60,
...70,80,90,100];
for k=1:length(levels),
noise_level=levels(k)/100
pix_change=round(120*
    noise_level)
r=randperm(120);
rpix=r(1:pix_change);
for i=1:8,
r=randperm(120);
rpix=r(1:pix_change);
dig=reshape(digits(:,:,i)
    ,1,[]);
for j=1:pix_change,
if(dig(rpix(j))==1)
dig(rpix(j))=-1;
else
```

```
dig(rpix(j))=1;
end
end
dignoise=reshape(dig(:),12,10);
noise_digits((k-1)*12+1:k*12,:,i)
...=dignoise;
end
end

%%%%%%%%%%%%%%%%%%%

function [p]=perm(a)
len=length(a);
p=[];
while (length(p)~=len)
num = round((len-1)*rand) + 1;
b=find(p==a(num));
if (length(b)==0)
p=[p,a(num)];
end
end
end

%%%%%%%%%%%%%%%%%%%

figure
plot([0,10,20,30,40,50,60,70,
...80,90,100],res_exito(1,:),'r-')
hold on
grid on
plot([0,10,20,30,40,50,60,70,
...80,90,100],
... res_exito(2,:),'g-')
plot([0,10,20,30,40,50,60,70,
...80,90,100],
... res_exito(3,:),'b-')
plot([0,10,20,30,40,50,60,70,
...80,90,100],
... res_exito(4,:),'c-')
plot([0,10,20,30,40,50,60,70,
...80,90,100],
... res_exito(5,:),'m-')
plot([0,10,20,30,40,50,60,70,
...80,90,100],
... res_exito(6,:),'y-')
```

B.4.3. RBF y Clasificación de patrones

```
%%%%%%%%%%%%%%%%%%%
% Crear los datos de entrada
%%%%%%%%%%%%%%%%%%%
close all, clear all, clc,
    format compact
% Número de muestras de
% cada cluster
K = 100;
% desplazamiento de los
    cluster
q = .6;
% definir 2 grupos de datos
% de entrada
A = [rand(1,K)-q rand(1,K)+q;
     rand(1,K)+q rand(1,K)-q];
B = [rand(1,K)+q rand(1,K)-q;
     rand(1,K)+q rand(1,K)-q];
% Graficar datos
plot(A(1,:),A(2,:),'k+',B(1,:)
    ,B(2,:),'b*')
grid on
hold on
%%%%%%%%%%%%%%%%%%%
% Definir codificación
% de salida
%%%%%%%%%%%%%%%%%%%
% codifición de (+1/-1) para
% el problema XOR de 2 clases
a = 1;
b = -1;
%%%%%%%%%%%%%%%%%%%
% Preparar entradas y salidas
% para el entrenamiento de la
% red
```

```
%%%%%%%%%%%%%%%%%
% definir las entradas
% (combinar las muestras de
    las
% cuatro clases)
P = [A B];
% definir las salidas deseadas
T = [repmat(a,1,length(A))
...repmat(b,1,length(B))];
%%%%%%%%%%%%%%%%%
% Crear un RBF
%%%%%%%%%%%%%%%%%
% Escoger una constante de
% difusión
spread = 1;
% crear una red neuronal de
% base radial
net = newrbe(P,T,spread);
% Observar la red
view(net)
%%%%%%%%%%%%%%%%%
% Evaluar el desempeño de la
% red
%%%%%%%%%%%%%%%%%
% simular una red con los
% datos de entrenamiento
Y = net(P);
% calcular [%] de las
% clasificaciones correctas
correct = 100 * length
...(find(T.*Y > 0))/length(T);
fprintf('\nDispersión ______=
...%.2f\n',spread)
fprintf('Numero de neuronas =
...%d\n',net.layers{1}.size)
fprintf('Clases correctas ___=
...%.2f %%\n',correct)
% Graficar la salida deseada
% y la respuesta de la red
figure;
plot(T')
hold on
grid on
plot(Y','r')
ylim([-2 2])
set(gca,'ytick',[-2 0 2])
legend('Salida deseada',
...'Respuesta de la red')
xlabel('No. Muestras')
%%%%%%%%%%%%%%%%%
% Graficar los reultados de
% la clasificación
%%%%%%%%%%%%%%%%%
% generar una cuadrícula
span    = -1:.025:2;
[P1,P2] = meshgrid(span,span);
pp      = [P1(:) P2(:)]';
% simular la red neuronal
% entrenada en una cuadrícula
aa      = sim(net,pp);
% graficar las regiones de
% clasificación basado en la
% activación MAX
figure(1)
ma = mesh(P1,P2,reshape
...(-aa,length(span),length
...(span))-5);
mb = mesh(P1,P2,reshape
...( aa,length(span),length
...(span))-5);
set(ma,'facecolor',[1 0.1
    0.4],
...'linestyle','none');
set(mb,'facecolor',[1 1.0 .7],
...'linestyle','none');
view(2)
%%%%%%%%%%%%%%%%%
% Graficar los centros
% de la RBF
%%%%%%%%%%%%%%%%%
plot(net.iw{1}(:,1),net.iw{1}
...(:,2),'gs')
```

Apéndice C

RBF en MATLAB: Función newrbe

MATLAB cuenta en su Toolbox de redes neuronales con la función *newrbe* para crear una Función de base radial. Las primeras líneas del código de la función son comentarios de la misma, en el que se hace una descripción y ejemplo de uso, esto es una ayuda que brinda MATLAB al usuario (todas la funciones propias de MATLAB tienen esta descripción y ejemplo de uso). Las líneas del código que siguen a la ayuda son de verificación y chequeo de errores, por ejemplo si se llama a la función newrbe con dos argumentos de entradas entonces el tercero (spread) es por defecto 1, también se verifica si los argumentos p y t tienen igual número de columnas. A continuación se muestra el resto del código numerado para facilitar el análisis.

```
1   [R,Q] = size(p);
2   [S,Q] = size(t);
3   net = network(1,2,[1;1],[1;0],[0  0;1  0],[0  1]);
4   net.inputs{1}.size = R;
5   net.layers{1}.size = Q;
6   net.inputWeights{1,1}.weightFcn = 'dist';
7   net.layers{1}.netInputFcn = 'netprod';
8   net.layers{1}.transferFcn = 'radbas';
9   net.layers{2}.size = S;
10  net.outputs{2}.exampleOutput = t;
11  [w1,b1,w2,b2] = designrbe(p,t,param.spread);
12  net.b{1} = b1;
13  net.iw{1,1} = w1;
14  net.b{2} = b2;
15  net.lw{2,1} = w2;
```

```
16  function [w1,b1,w2,b2] = designrbe(p,t,spread)
17  [r,q] = size(p);
18  [s2,q] = size(t);
19  w1 = p';
20  b1 = ones(q,1)*sqrt(-log(.5))/spread;
21  a1 = radbas(dist(w1,p).*(b1*ones(1,q)));
22  x = t/[a1; ones(1,q)];
23  w2 = x(:,1:q);
24  b2 = x(:,q+1);
```

En las líneas 1 y 2 se determinan las dimensiones de entrada y salida, `Q` es el número de patrones, aquí es claro por que el número de columnas de `p` y `t` debe ser igual. `R` es el número de nodos de entrada a la red y `S` el número de nodos de salida de la red. La línea 3 es para determinar la arquitectura de la red a utilizar, la función *network* la define según sus argumentos; en este caso así: $net = network(1, 2, [1; 1], [1; 0], [0 \quad 0; 1 \quad 0], [0 \quad 1])$. El primer argumento (1) indica que la red solo tendrá un nivel de entrada, el (2) determina que a continuación de la entrada hay dos capas de procesamiento, el arreglo [1;1] informa que las dos capas de procesamiento tendrán bias, $[1; 0]$ indica que debe haber conexión de la entrada hacia la primera capa, $[0 \quad 0; 1 \quad 0]$ es para conectar los pesos de la capa uno hacia la capa 2, y el último argumento $[0 \quad 1]$ define que la segunda capa será la capa de salida.

En los numerales 4,5 y 9 se establece el tamaño de la entrada, de la capa oculta y de la capa de salida de la red respectivamente. En 6 se indica que la forma de pensar las entradas es por la métrica de la distancia (distancia euclídea), luego 7 es para determinar que el valor neto se calcula con un producto componente a coponente y 8 proporciona la función de base radial para los nodos de la capa oculta.

En 10 se llama a la función *designrbe* para determinar los pesos y bias de la capa oculta y de salida; a continuación se asignan estos respectivos valores a la estructura de la red. De la línea 15 en adelante se tiene la implementación de la función *designrbe*. 16 y 17 son para determinar las dimensiones de entrada y salida. En 18 se asigna a `w1` la matriz `p` transpuesta, notemos que `w1` son los pesos que se conectan de la entrada a la capa oculta, es decir se está asignando cada fila de la matriz de entrada `p` como vector centro de cada neurona oculta. 19 es el cálculo del bias de la capa oculta, básicamente es un vector columna de tamaño igual al número de neuronas ocultas (igual al número de patrones). Si la dispersión (*spread*) es 1 entonces `b1` será un

vector de valor 0.8326 en todos sus elementos; evaluando este dato en la función de base radial ($exp(-n^2)$) se obtiene como valor 0.5, esto es la mitad del valor máximo que se puede alcanzar. En 20 se debe destacar varios puntos, obsérvese que `b1` es un vector columna de dimensiones `q`×1 y `ones(1,q)` que es un vector fila de unos de dimensión 1×`q`, luego el producto `b1*ones(1,q)` será de dimensión `q`×`q`. `w1` es de dimensiones `q`×`r`, `dist(w1,p)` devuelve una matriz de dimensiones `q`×`q` donde cada componente es la distancia euclidiana de cada vector fila de `w1` con cada vector columna de `p`. Luego se hace un producto componente a componente y por último se evalúa cada componente en la función de base radial ($exp(-n^2)$), por lo tanto a1 es también de dimensión `q`×`q`. En la línea que sigue (21) se calculan los pesos y bias de la capa de salida y por último se asignan en las variables correspondientes (`w2,b2`).

Observando con mas detalle la línea 21 del código en cuestión notamos que la matriz [`a1`; $ones(1, \texttt{q})$] es de dimensión `q1`×`q` donde `q1=q+1`, ésta es no cuadrada, y la expresión: `t`/[`a1`; $ones(1, \texttt{q})$] es equivalente a $\texttt{t} * inv([\texttt{a1}; ones(1, \texttt{q})])$, es decir A/B en MATLAB equivale a $A * inv(B)$ si A y B son matrices y además B es cuadrada e invertible, por lo tanto el determinante de B debe ser diferente de cero. Parece que en la línea 21 hay una inconsistencia, pero en realidad MATLAB lo que hace es encontrar la pseudoinversa de la matriz [`a1`; $ones(1, \texttt{q})$].

Volviendo a la línea 21 del código, y recordando que `a1` es de dimensión `q`×`q` la matriz [`a1`; $ones(1, \texttt{q})$] es de dimensión `q1`×`q`, se concluye entonces que MATLAB calcula la pseudoinversa por la izquierda pues el número de filas es siempre mayor que el de columnas, y es de dimensión `q`×`q1`.

Luego `x` tendrá dimensiones `S`×`q1`. En la línea 22, `w2=x(:,1:q)` asigna todas las filas de `x` desde la columna 1 hasta la columna `q` a `w2` y la última columna se asigna a `b2`.

Bibliografía

[1] A. Albert. *Regression and the Moore Penrose Pseudoinverse*. Academic Press, USA, 1972.

[2] A. Björck. *Numerical methods for least squares problem*. Society for Industrial and Applied Mathematics, Philadelphia, 1996.

[3] J.F. Castro. *Fundamentos para la Implementación de Red Neuronal Perceptrón Multicapa mediante Software*. PhD thesis, Escuela de Ingeniería Mecánica Eléctrica, Universidad de San Carlos de Guatemala, Noviembre 2006.

[4] P. Courrieu. Straight monotonic embedding of data sets in Euclidean spaces. *Neural Networks*, 15:1185–1196, 2002.

[5] P. Courrieu. Fast Computation of Moore-Penrose Inverse Matrices. *Neural Information Processing-Letters and Reviews*, 8(2):25–29, 2005.

[6] P. Courrieu. Fast solving of weighted pairing least-squares systems. *Journal of Computational and Applied Mathematics*, 231:39–48, 2009.

[7] J. L. de la Fuente. *Técnicas de Cálculo para Sistemas de Ecuaciones, Programación Lineal y Programación Entera*. Editorial Reverté, S.A., 1997.

[8] H. P. Decell. An Application of the Cayley Hamilton Theorem to Generalized Matrix Inversion. *SIAM Review*, 7:526–528, 1965.

[9] D.K. Faddeev and V.N. Faddeeva. *Computational Methods of Linear Algebra*. W.H. Freeman Co., San Francisco, 1963.

[10] L.O. Favela. *Introducción a las Redes Neuronales Artificiales*. PhD thesis, Departamento de Matemáticas, Universidad de Sonora, Agosto 2004. Disponible en http://lic.mat.uson.mx/tesis/121TesisFavela.PDF.

[11] G.H. Golub and Ch.F. Van Loan. *Matrix Computations, third edn.* The Johns Hopkins University Press, S.A., Baltimore, 1996.

[12] T.N.E. Greville. Note on the generalized inverse of a matrix product. *SIAM Review*, 8(4):518–521, 1966.

[13] D. Hebb. *The Organization of Behaviour*. Wiley, 1949.

[14] A.B. Israel and A. Charnes. Contribution to the theory of Generalized Inverses. *J. Soc. Indust. Appl. Math.*, 11(3):667–699, 1963.

[15] A.B. Israel and T.N.E. Greville. *Generalized Inverses: Theory and Applications*. Rutcor, 2001.

[16] A.B. Israel and S.J. Wersan. An Elimination method for Computing the Generalized Inverse for Arbitrary Complex Matrix. *J. Assoc. Comput. Match.*, 10:532–537, 1963.

[17] T. Kohonen. *Self Organization and Associative Memory*. Springer Verlag, 1989.

[18] P. Larranaga, I. Inza, and A. Moujahid. *Redes Neuronales*. Departamento de Ciencias de la Computación e Inteligencia Artificial, Universidad del País Vasco-Euskal Herriko Unibertsitatea. Disponible en http://www.sc.ehu.es/ccwbayes/docencia/mmcc/docs/t8neuronales.pdf.

[19] B. Martín and A. Sanz. *Redes Neuronales y Sistemas Borrosos*. Alfaomega, Universidad de Zaragoza, 2007.

[20] B. Noble. A Method for Computing the Generalized Inverse of a Matrix. *SIAM J. Numer. Anal.*, 3:582–584, 1966.

[21] R. Ospina, H. Parra, and H. Aguirre. Diseño de una Red Neuronal Artificial para Asistir la Automatización en un Taller de Mecanizado. *Scientia et Technica año XIII*, 37:181–186, 2007.

[22] M. Palacios. *Sistemas de ecuaciones lineales: métodos directos.* Departamento de Matemática Aplicada, Centro Politécnico Superior Universidad de Zargoza, 2001. Disponible en http://pcmap.unizar.es/m̃pala/C_N_lecci/CN_1II2_SELdir.pdf.

[23] R.A. Penrose. A generalized inverse for matrices. *Proceedings of the Cambridge Philosophical Society*, 51:406–413, 1955.

[24] L. Personnaz, I. Guyon I, and Dreyfus. Collective computational properties of neuronal networks: New learning mechanisms. *Physical Review A*, 34(5):4217–4228, 1986.

[25] L.D. Pyle. Generalized Inverse Computations Using the Gradient Projection Method. *Journal of the Association for Computing Machinery*, 11(4):422–428, 1964.

[26] D.E. Rumelhart and J.L.McClelland. *Parallel Distributed Processing. Vol 2: Psychological and biological models.* MIT Press, 1986.

[27] D.E. Rumelhart and J.L. McClelland. *Parallel Distributed Processing. Vol 1: Foundations.* MIT Press, 1986.

[28] B. Rust, W.R. Burrus, and C. Scheeberger. A Simple Algorithm for Computing the Generalized Inverse of a Matrix. *Communications of the ACM*, 9(5):381–387, 1966.

[29] P.B. Sánchez, I.D. García, and H.M. de la Vega. Sobre la factorización rrqr. *Cuadernos de investigación, Área I: Físco-Matemáticas e Ingeniería*, Universidad Autónoma de Coahuila, Coordinación General de Estudios de Postgrado e Investigación Peter Corke and James yan, 20:1-83, 2000.

[30] E.L. Silva. *Métodos de reducción de Redes Neuronales RBF usando la descomposición QLP para aplicaciones de ajuste de datos y clasificación.* Tesis, Departamento de Estadística e Investigación Operativa, Universidad de Granada, 2007. Disponible en http://www.acfr.usyd.edu.au/.

[31] A.E. Tomás. *Implementación paralela de métodos de Krylov con reinicio para problemas de valores propios y singulares.* Tesis, Universidad Politecnica de Valencia, 2009. Disponible en http://riunet.upv.es/bitstream/handle/10251/5082/tesisUPV3041.pdf.

[32] L.N. Trefethen and D. Bau. *Numerical Linear Algebra*. SIAM, Philadelphia,1997.

[33] D.S. Watkins. *Fundamentals of Matrix Computations*. John Wiley and Sons, Singapur,1991.