Aplicaciones de técnicas de inteligencia artificial basadas en aprendizaje profundo *(deep learning)* al análisis y mejora de la eficiencia de procesos industriales

Ana González Muñiz

TUTOR: Ignacio Díaz Blanco

Universidad de Oviedo

Departamento de Ingeniería Eléctrica, Electrónica,
de Computadores y Sistemas
UNIVERSIDAD DE OVIEDO

Un TFM presentado a la Universidad de Oviedo en conformidad con los requisitos del MÁSTER EN INGENIERÍA DE AUTOMATIZACIÓN E INFORMÁTICA INDUSTRIAL

Febrero 2018

Resumen

En la última década, las técnicas de aprendizaje profundo o *deep learning* se han convertido en una de las ramas más destacadas de la Inteligencia Artificial, aportando resultados muy superiores a los obtenidos hasta el momento con otras técnicas de aprendizaje automático. A día de hoy, estas técnicas ya han demostrado su éxito en variadas aplicaciones (como el reconocimiento facial y de voz, o el procesamiento natural del lenguaje) e importantes empresas tecnológicas (Google, Facebook, Microsoft, etc.) ya las han incorporado como herramientas de desarrollo de sus productos. Sin embargo, el uso del *deep learning* todavía no se ha implantado en el sector industrial, constituyendo así un interesante campo de investigación aún por explorar.

Es por ello que a lo largo del presente TFM se ha estudiado la aplicabilidad de las técnicas de *deep learning* en la industria, tratando de transferir las ideas que ya han demostrado buenos resultados en otros campos al ámbito de los procesos industriales y sistemas de ingeniería. Con este fin, se ha analizado el comportamiento de diferentes arquitecturas *deep learning* (redes *feedforward*, convolucionales, recurrentes y *deep autoencoders*) ante distintos casos de estudio (modelado del proceso industrial, predicción de su comportamiento, detección y diagnóstico de fallos, extracción de descriptores relevantes del proceso y reducción de la dimensión para analítica visual), utilizando para ello la combinación de librerías *open source* Keras-Tensorflow.

Tabla de contenidos

Índice de capítulos y secciones

Índice de tablas

Índice de figuras

1 Introducción

Surgida a mediados del siglo XX, la **Inteligencia Artificial** (IA) se perfilaba como una disciplina compleja y abstracta, típicamente ligada a grandes centros de investigación. Sin embargo, a lo largo de los últimos años se ha **popularizado** de forma masiva y sus aplicaciones se han extendido a la escena cotidiana. A día de hoy ya está presente, por ejemplo, en nuestros móviles, tabletas u ordenadores personales.

Este **auge de la IA** ha venido impulsado por dos factores fundamentales: la elevada **potencia de cálculo** de los computadores, que ha permitido afrontar el alto coste computacional de las técnicas basadas en IA; y la enorme **cantidad de información almacenada** digitalmente en la actualidad, que ha hecho surgir la demanda de este tipo de técnicas para el procesamiento y análisis automático de datos [1].

Mientras que las técnicas tradicionales de análisis están basadas en la ejecución de reglas preprogramadas, la IA ofrece sistemas capaces de autoprogramarse, es decir, capaces de aprender las reglas por sí solos a partir de los datos de trabajo. Y es en esta **capacidad de aprendizaje** donde reside el **éxito de la IA**.

En este contexto, se enmarca una de sus ramas más destacadas: el ***machine learning*** o aprendizaje automático (Figura 1.1), que propone un modelado analítico y automático de los datos. Para ello, se plantea el análisis como un proceso de aprendizaje, donde el programador proporciona una serie de

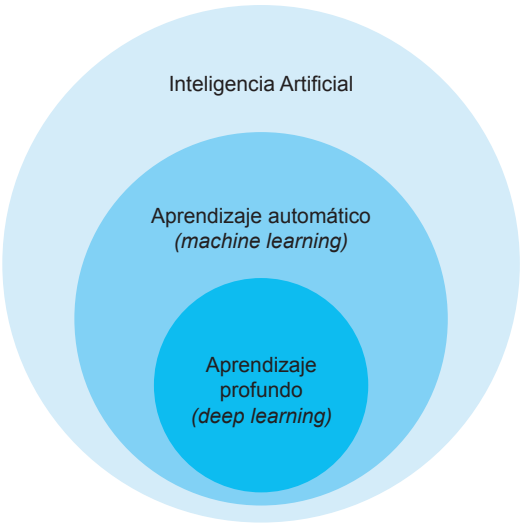

Figura 1.1. Jerarquía e interacción entre Inteligencia Artificial, *Machine Learning* y *Deep Learning*.

reglas de partida que el algoritmo de aprendizaje ha de ir adaptando y, también, creando otras nuevas, tratando así de mejorar la tasa de acierto del modelo generado.

A su vez, dentro del *machine learning*, destaca un subconjunto de técnicas conocidas con el nombre de ***deep learning*** o aprendizaje profundo (Figura 1.1). Este enfoque propone modelar abstracciones de alto nivel de los datos empleando para ello arquitecturas compuestas por un elevado número de capas de transformaciones que pueden ser tanto lineales como no lineales. Se trata de una idea inspirada en la arquitectura y funcionamiento del cerebro humano y, por ello, estas técnicas reciben también el nombre de **redes neuronales artificiales** (RNAs).

La gran aportación de las técnicas de *deep learning* se encuentra en la posibilidad de entrenar **arquitecturas funcionales profundas** (modelos complejos con gran cantidad de capas), donde las técnicas clásicas fallan debido a problemas de bajo gradiente en la adaptación de los pesos de las capas más internas. Además, estas técnicas son pioneras en su **escalabilidad** respecto a la cantidad de datos de entrada: cuantos más datos manejan, mejor es su comportamiento (Figura 1.2), lo cual permite explotar al máximo los beneficios de otros revolucionarios conceptos como son el Big Data o la Industria 4.0.

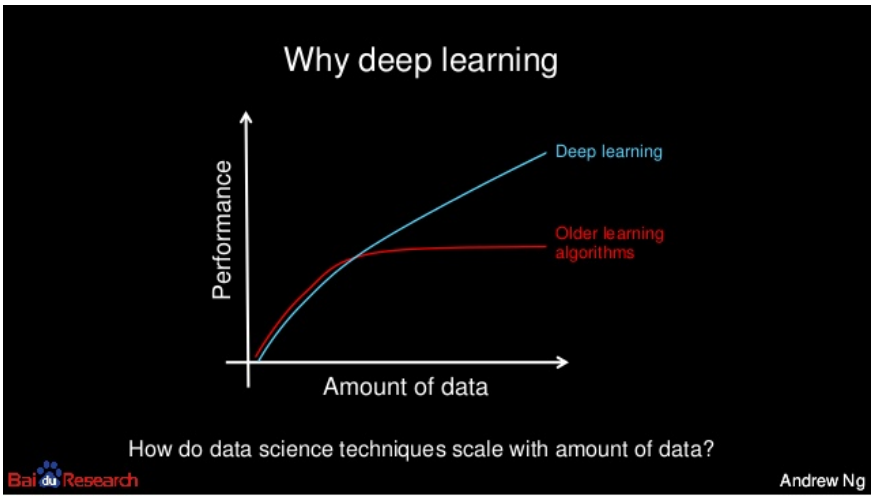

Figura 1.2. Escalabilidad del *deep learning*.
Diapositiva expuesta por Andrew Ng (Chief Scientist de Baidu) en la *ExtractConf talk 2015*
(Fuente: https://www.slideshare.net/ExtractConf/andrew-ng-chief-scientist-at-baidu)

Así es que en la última década, las técnicas de *deep learning* han **revolucionado el campo de la Inteligencia Artificial** [2], aportando resultados muy superiores a los obtenidos hasta el momento con otras técnicas de aprendizaje automático. Algunas **aplicaciones** en las que los algoritmos de *deep learning* se están usando con éxito son, por ejemplo, el lenguaje natural hablado y escrito, el reconocimiento de voz, la visión por computador, el reconocimiento de caras, la interpretación semántica o los traductores inteligentes [3].

En consecuencia, **el interés por el *deep learning* se ha disparado**. Aparece con frecuencia en revistas de prestigio como Science o Nature y la asistencia al NIPS (congreso

de referencia donde se publican los últimos avances en *deep learning*, https://nips.cc) no ha dejado de incrementarse en los últimos años (en 2013 acudieron unos 2000 asistentes y en 2017 se superaron los 8000).

Ante esta situación, no es de extrañar que **importantes empresas tecnológicas** como Google, Facebook o Microsoft, ya estén **apostando por el *deep learning*** como herramienta de trabajo [4][5][6]. Entre los casos de uso más populares nos encontramos con: el reconocimiento de voz de Siri, Google Now y Cortana; los recomendadores de Netflix, Amazon, Google, Facebook y Twitter; el etiquetado de imágenes de Google y Facebook; o también, la propuesta de coche autónomo de Google, Tesla y Uber.

En definitiva, el aprendizaje profundo ya ha demostrado su éxito en variadas aplicaciones y muchas de ellas son **extrapolables a problemas industriales** [3]. Existe, por ejemplo, una gran similitud entre el reconocimiento de voz y el análisis de vibraciones en motores, de igual manera que las técnicas de reconocimiento facial pueden relacionarse con el empleo de visión por computador para el diagnóstico de procesos industriales. Sin embargo, el uso del *deep learning* en el **ámbito industrial** está aún en ciernes, constituyendo así un interesante campo de estudio **todavía por explotar**.

1.1. Propósito de la investigación

Con la aparición del *deep learning*, nos encontramos ante una nueva tecnología que ya condiciona (y lo hará todavía más [7]) la economía y la sociedad. En la industria, la cantidad de datos que se generan actualmente se esta incrementando de forma exponencial y extraer información valiosa de ellos, como ya permite hacer el *deep learning* en otros campos, supone una **ventaja comparativa** que no debemos menospreciar.

Ante esta situación, se propone recurrir al *deep learning* para extraer, a partir de los datos, la máxima información útil de los procesos, mejorando así la comprensión de los mismos y persiguiendo, finalmente, el **beneficio del sector industrial**: optimización de los procesos, reducción de costes, mayor productividad y eficiencia.

Más allá de la motivación económica, este proyecto tiene también una motivación científica, pues se pretende **trasladar al ámbito industrial** los resultados, muy superiores respecto a los de enfoques precedentes, que el aprendizaje profundo ya ha tenido en otros campos. Y para ello, se propone recurrir a dos de las librerías más destacadas en la actualidad: **Tensorflow** (https://www.tensorflow.org), utilizada por los laboratorios de Google y liberada a la comunidad a finales de 2015; y **Keras** (https://keras.io), que aporta una capa de abstracción que simplifica aún más el diseño y entrenamiento de los modelos de *deep learning*. Estas librerías, *open source* y basadas en Python y, por tanto, integradas en el ecosistema de Python para análisis y visualización de datos, gozan de un amplísimo soporte, con un gran crecimiento en fiabilidad y potencial. Se trata de unas valiosas herramientas y, por ello, adquirir experiencia en su manejo es también propósito de este proyecto.

1.2. Antecedentes y trabajo previo

Aunque sus orígenes se remontan a los años 40, a menudo hablamos del aprendizaje profundo como si de una nueva disciplina se tratase. Esto se debe a la escasa popularidad de la que gozaba el *deep learning* antes de alcanzar su éxito actual y, también, al hecho de haber recibido diferentes nombres antes de ser denominado aprendizaje profundo. De hecho, este campo ha sido renombrado muchas veces, reflejando la influencia de diferentes perspectivas e investigadores. Comúnmente se distinguen tres etapas, como se muestra en la siguiente figura: ***cybernetics*** (1940-1960), ***connectionism*** (1980-1990) y ***deep learning*** (2006-actualidad).

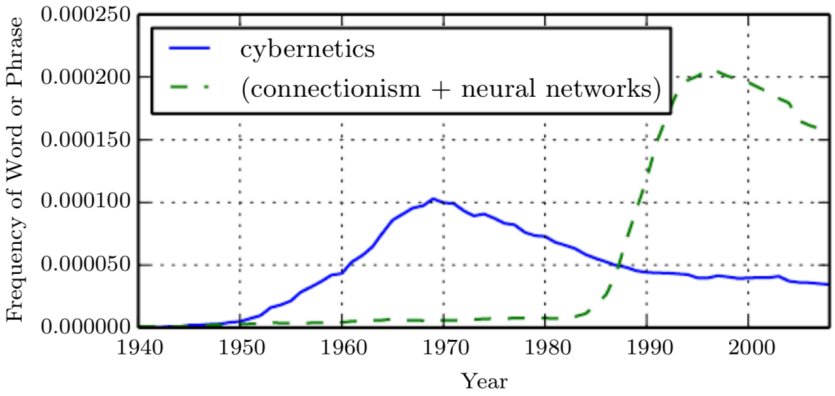

Figura 1.3. Evolución del aprendizaje profundo. Aquí se muestran dos de las tres etapas históricas, medidas por la frecuencia de las frases "*cybernetics*" y "*connectionism*" o "*neural networks*", según Google Books (la tercera etapa es demasiado reciente y por ello no aparece reflejada en la figura). (Fuente: http://www.deeplearningbook.org)

A continuación recorreremos la evolución del *deep learning* a lo largo de estas tres etapas, desde sus humildes comienzos hasta convertirse en una de las ramas más destacadas de la Inteligencia Artificial.

1.2.1. *Cybernetics* (1940-1960)

Hace cientos de años que se investiga el cerebro, pero es en esta época, al mismo tiempo que se desarrollaban las bases de la tecnología actual, cuando aparecen los primeros estudios desde la perspectiva de la computación. Dichos estudios proponen sistemas inspirados en el cerebro humano y, por ello, los algoritmos de aprendizaje profundo son habitualmente conocidos como redes neuronales artificiales (RNAs). No obstante, estos sistemas no pretenden ser modelos realistas del comportamiento biológico de nuestro cerebro, sino que aspiran a modelar la inteligencia humana por medio del **aprendizaje desde el ejemplo**.

Siguiendo esta idea, en **1943**, el neurólogo **Warren McCulloch** y el matemático **Walter Pitts** propusieron los primeros modelos matemáticos y eléctricos de redes

neuronales [8]. Describieron cómo funcionaban las neuronas y modelaron una red simple utilizando circuitos eléctricos. Se trataba de un modelo lineal, capaz de reconocer dos categorías y cuyos pesos debían ser ajustados por el operador humano.

Años más tarde, **Donald Hebb** propuso en **1949** la llamada "regla de aprendizaje de Hebb" [9], que define a grandes rasgos el proceso de aprendizaje neuronal y que todavía se utiliza en algunos modelos actuales. Por su parte, **Alan Turing** definió en **1950** el conocido "Test de Turing", cuyo objetivo consistía en determinar si una máquina era realmente inteligente. Sin duda, el concepto de inteligencia artificial estaba en el aire, hasta que en **1956** tuvo lugar la **conferencia de Darthmouth** (organizada por Martin Minsky y John McCarthy, con la ayuda de Claude Shannon y Nathan Rochester), donde se discutió sobre la capacidad de las máquinas para simular el aprendizaje humano. Se considera que es en este evento donde **nace la Inteligencia Artificial**, cuando **Minsky** convence a los asistentes para acuñar este término como nombre del nuevo campo.

Poco después, en **1958**, el neurobiólogo **Frank Rosenblatt** diseña el **Perceptrón** [10], que representa para muchos la **primera red neuronal artificial de la historia**. Se trata de una red de dos capas, una de entrada y otra de salida, capaz de discriminar entre dos clases linealmente separables. En este caso, la red ya es capaz de aprender por sí sola los pesos, a partir de ejemplos de entradas de cada categoría.

Tras el trabajo desarrollado por Frank Rosenblatt sobre el Perceptrón, **Bernard Widrow** y **Marcian Hoff** introdujeron en **1960** una importante variación del mismo, que dio lugar al modelo de red **ADALINE** (ADAptative LINear Element) [11], que utiliza una función de transferencia de tipo lineal en lugar de un limitador fuerte como el Perceptrón. Este nuevo modelo constituyó la **primera aplicación práctica de una red neuronal artificial**: la eliminación de ecos en las líneas telefónicas por medio de filtros adaptativos. También cabe mencionar que el algoritmo de entrenamiento utilizado para adaptar los pesos del ADALINE es un caso especial del algoritmo de descenso del gradiente estocástico, que sigue siendo el algoritmo dominante para el entrenamiento de los modelos actuales.

A pesar de estos brillantes inicios, en **1969**, los investigadores del MIT **Marvin Minsky** y **Seymour Papert** publican el libro “Perceptrons: An introduction to Computational Geometry” [12], que para muchos significó el fin de las RNAs. En él demostraron importantes limitaciones teóricas en el aprendizaje de los modelos utilizados hasta entonces, en particular de la red Perceptrón, exponiendo que se trataba de juguetes matemáticos sin aplicabilidad práctica real (destaca, por ejemplo, la incapacidad de estos modelos lineales para aprender una función tan simple como la XOR).

Esta fue la primera caída importante en la popularidad de las RNAs y también el fin de la neurociencia como guía predominante del aprendizaje profundo, aunque nunca dejaría de ser una fuente de inspiración para los investigadores de esta disciplina. Hoy en día, **el aprendizaje profundo se inspira en muchos campos**, además de en la neurociencia, lo hace también en fundamentos matemáticos aplicados como el álgebra lineal, la teoría de la probabilidad, la teoría de la información y la optimización numérica.

1.2.2. *Connectionism* (1980-1990)

En los años 80 el aprendizaje profundo entró en un nuevo ciclo: ***connectionism***, surgido en el contexto de la ciencia cognitiva. La idea central de este movimiento propone que un gran número de unidades computacionales simples pueden lograr un **comportamiento inteligente cuando se conectan en red**, lo cual es aplicable tanto a las neuronas de los sistemas nerviosos biológicos como a los modelos computacionales de las RNAs.

Varios conceptos clave surgidos en esta época siguen siendo centrales en el aprendizaje profundo a día de hoy. Destacan, por ejemplo, el concepto de **representación distribuida**, desarrollado por **Hinton** en **1986** [13], o el exitoso algoritmo de retropropagación (más conocido como ***back-propagation***) propuesto por **LeCun** en **1987** [14], que sigue siendo el enfoque dominante en el entrenamiento de redes profundas.

Otra gran aportación tuvo lugar en **1980**, cuando el investigador **Kunihiko Fukushima** introdujo el **neocognitrón** [15], una poderosa arquitectura para el procesamiento de imágenes inspirada en la estructura del sistema visual mamífero, la cual más tarde se convertiría en la base de las modernas **redes convolucionales** [16].

Poco después, en la **década de los 90**, se hicieron importantes avances en el **modelado de secuencias**: en 1991 y 1994, Hochreiter [17] y Bengio [18] respectivamente, identificaron las principales dificultades que conlleva este modelado; en **1997**, **Hochreiter** y **Schmidhuber** introdujeron las redes **LSTM** (*Long Short Term Memory*) [19], arquitecturas con memoria a largo y corto plazo, ampliamente utilizadas en la actualidad y capaces de hacer frente a dichas dificultades.

El **fin** de esta etapa llega a **mediados de los años 90**. Con los buenos resultados obtenidos hasta entonces, se establecieron nuevos y ambiciosos objetivos para el aprendizaje profundo, que al no cumplir con las expectativas, sufrió su segunda gran caída de popularidad y que duraría hasta 2006.

Durante este tiempo, las redes profundas siguieron aportando impresionantes resultados en algunas tareas, como la histórica **derrota del ajedrecista Gary Kaspàrov** ante **Deep Blue** de IBM [20]. Para mantener viva la investigación en este campo, el CIFAR (*Canadian Institute For Advanced Research*) puso en marcha la iniciativa **NCAP** (*Neural Computation and Adaptive Perception*). Este programa reunió a los grupos de investigación liderados por Geoffrey Hinton (Universidad de Toronto), Yoshua Bengio (Universidad de Montreal) y Yann LeCun (Universidad de Nueva York), tres referentes del aprendizaje profundo, junto con expertos en materias como la neurociencia o la visión por computador. Con el mismo propósito, surgió en 1988 la **IJCNN** (*International Joint Conference on Neuronal Networks*) y tres años más tarde la **ICANN** (*International Conference on Artificial Neural Networks*). Asimismo, desde 1987 se viene celebrando anualmente el **NIPS** (*Neural Information Processing Systems*), congreso de referencia del aprendizaje profundo.

Finalmente cabe mencionar que, al terminar esta etapa, se tenía la creencia de que las arquitecturas profundas eran demasiado difíciles de entrenar y por ello tenían un futuro

muy limitado. Sin embargo, **ahora sabemos que estas arquitecturas aportan resultados realmente satisfactorios**, pero eran demasiado costosas computacionalmente para el hardware de la época y hubo que esperar a la siguiente era para descubrirlo.

1.2.3. *Deep learning* (2006-actualidad)

La tercera era de las RNAs comenzó en **2006** de la mano de **Geoffrey Hinton**, que desarrolló una forma **más eficiente de entrenar** los modelos de *deep learning*, utilizando un aprendizaje por capas [21]: la primera capa aprende características primitivas, que son enviadas a la siguiente capa, la cual se entrena para reconocer características más complejas y así sucesivamente hasta entrenar todas las capas del modelo. Este avance permitió entrenar arquitecturas mucho **más profundas** que las utilizadas hasta el momento y, por ello, se popularizó el término *deep learning* o **aprendizaje profundo**. A partir de entonces, las RNAs empezaron a aportar resultados muy superiores a los obtenidos con otras técnicas de aprendizaje automático y se han ido convirtiendo en una de las ramas más destacadas de la IA.

Ante esta situación, las **grandes empresas tecnológicas** del momento están apostando por el *deep learning* para el desarrollo de sus aplicaciones. Por ejemplo: Google ha diseñado redes profundas capaces de reconocer voces en teléfonos Android e imágenes en Google Plus, Facebook utiliza *deep learning* para orientar sus anuncios e identificar rostros y objetos en imágenes y vídeos, Microsoft trabaja en proyectos de reconocimiento de voz y Baidu, el gran buscador chino, aplica *deep learning* para aplicaciones que van desde el reconocimiento facial a la ciberseguridad.

Entre las compañías mencionadas, una de las que más recursos ha invertido en investigar esta tecnología es **Google**, que comenzó su andadura en el sector en el año **2012**, cuando Jeff Dean (Google) y Andrew Ng (Universidad de Stanford) lideraron el proyecto **GoogleBrain**, bajo el cual se desarrolló una red profunda capaz de detectar patrones en vídeos e imágenes. Ese mismo año, el laboratorio de investigación **Google X**, utilizó GoogleBrain para analizar imágenes extraídas de vídeos de Youtube y detectar de forma automática aquellas que contenían gatos, objetivo que alcanzó con una elevada tasa de acierto [22].

Dos años después, Google compra **DeepMind**, una startup inglesa de *deep learning*, fundada en 2012 y dedicada al mundo de los videojuegos. Recientemente, en **2016**, el programa **AlphaGo** desarrollado por DeepMind fue capaz de vencer en el juego Go al jugador profesional Lee Sedol por 5 partidas a 1. Jugadores expertos en Go afirman que el algoritmo fue capaz de realizar movimientos “creativos” que no se habían visto hasta el momento. Y aunque este resultado fue realmente sorprendente, en **2017** apareció **AlphaGo Zero**, que no sólo supera al AlphaGo sino que además se trata de un sistema autodidacta (es entrenado mediante aprendizaje por refuerzo, no aprendizaje supervisado como ocurría en el AlphaGo, lo cual significa que la máquina aprende jugando contra sí misma, ni

utilizando datos históricos de otras partidas ni aprendiendo de las jugadas de otros oponentes) [23].

La otra gran compañía que ha apostado fuerte por el *deep learning* es **Facebook**. En 2013, mientras que Google contrataba a Geoffrey Hinton, Facebook hacía lo mismo con Yann LeCun, que se convirtió en el director del Laboratorio de IA de Facebook. Un año después, Facebook desarrollaba **DeepFace**, un algoritmo basado en redes profundas que es capaz de reconocer a personas con la misma precisión que un ser humano [24].

A día de hoy, nos encontramos todavía en **pleno auge** de esta tercera era del aprendizaje profundo. Aunque hay escépticos que no descartan un posible tercer invierno, esta vez los avances del *deep learning* están encontrando aplicabilidad en gran cantidad de sectores, hasta el punto de crear mercados propios, así como producir cambios significativos en la estrategia de grandes y pequeñas empresas.

En definitiva, ahora que los conjuntos de datos se han hecho lo suficientemente grandes y los computadores se han vuelto lo suficientemente rápidos, el *deep learning* no deja de crecer y su popularidad aumenta día a día. Adicionalmente, ante la previsión de que seguiremos disponiendo de más y más datos con los que alimentar nuestros modelos y a la vista de que la comunidad científica no deja de proponer nuevos enfoques y aplicaciones para el *deep learning*, todo parece indicar que **los próximos años serán frenéticos** para esta disciplina.

1.2.4. Trabajo previo

El uso del *deep learning* aún no ha llegado a la industria. No obstante, pueden trasladarse a este sector muchas de las ideas que ya han aportado buenos resultados en otros ámbitos. Destacan especialmente cuatro tipos de arquitecturas profundas.

En primer lugar, se encuentran las **redes prealimentadas** o *feedforward* [25]. Se trata de arquitecturas profundas y extensas (gran número de capas y de neuronas por capa), con muchas más capas internas que las empleadas clásicamente, lo que permite representar no linealidades complejas con un número menor de unidades y buenas propiedades de generalización [25,26]. Estas redes son utilizadas, por ejemplo, en aplicaciones de **reconocimiento facia**l [24], que trasladadas al ámbito industrial podrían ser útiles en el diagnóstico de procesos mediante técnicas de **visión por computador**. Otra opción sería utilizar estas arquitecturas en análisis de señales temporales, en lugar de imágenes, y distinguir, en lugar de rostros, los diferentes **estados de trabajo** de un proceso industrial.

También destacan las **redes convolucionales** (*Convolutional Neural Networks, CNN*) [27], que son redes profundas muy parecidas a las *feedforward*, con una característica adicional: incorporan una serie de capas de convolución en su arquitectura, compuestas por bancos de filtros cuyos coeficientes son ajustados para optimizar la función de coste. Estas arquitecturas han mostrado una extraordinaria capacidad para la detección de patrones en

señales, por lo que se utilizan con éxito en aplicaciones como las de **reconocimiento de voz** [28]. Dado el paralelismo entre el reconocimiento de voz y el **análisis de vibraciones**, estas redes tendrían aplicación en multitud de sistemas de ingeniería: permitirían desde el estudio de motores hasta el análisis de estructuras, determinando, por ejemplo, **el estado de trabajo** del sistema o **prediciendo su comportamiento** futuro. Adicionalmente, el análisis de los filtros generados por la red podría aportar **información adicional de los sistemas**, como las frecuencias de interés para el caso del análisis de vibraciones.

Otra arquitectura relevante es la de las **redes recurrentes** (*Recurrent Neural Networks, RNN*) [29], arquitecturas con bucles de realimentación que permiten que la información persista y cuyo máximo exponente son las redes LSTM [19], capaces de aprender dependencias a corto y largo plazo. Su naturaleza está íntimamente relacionada con las secuencias y listas, por lo que son ideales para el trabajo con este tipo de datos. En consecuencia, un ejemplo de aplicación es su capacidad para predecir el próximo carácter en una secuencia de texto [30], lo cual es asimilable a la **predicción de comportamiento** de un sistema de ingeniería.

Por último, cabe mencionar el uso de ***deep autoencoders***. Se trata de redes profundas capaces de reproducir a la salida la misma información que reciben a la entrada y son utilizadas generalmente como herramientas de **reducción de la dimensión**, pues permiten incluir un cuello de botella en la red de la dimensión deseada [31]. En el ámbito industrial, donde se maneja una gran cantidad de variables y, por tanto, los espacios de trabajo son n-dimensionales, los *deep autoencoders* permitirían generar **versiones comprimidas de los procesos**, que serían útiles, por ejemplo, para **visualizar la evolución** de los mismos en mapas de estados 2D.

A la vista de esta cuatro arquitecturas, no cabe duda de que las posibilidades de aplicar *deep learning* a los procesos industriales son muy amplias y si los resultados aportados en este ámbito son similares a los obtenidos en otros campos, esto podría suponer toda una **revolución en la industria**.

1.3. Formulación del problema

El problema central planteado en este TFM consiste en investigar **las posibilidades de aplicación de las técnicas de *deep learning* en el análisis y mejora de la eficiencia de procesos industriales**. Para abordarlo, se propone explorar el uso de distintos tipos de redes profundas sobre diferentes problemas industriales y valorar la calidad de los resultados obtenidos.

Este objetivo global se divide en otros más concretos, expuestos a continuación. Todos ellos tienen un punto de partida común: utilizar una arquitectura de red profunda para generar un modelo del proceso bajo estudio (a partir de variables, descriptores o índices del mismo). Y se diferencian en la funcionalidad que debe desempeñar la arquitectura en cada caso.

- **Objetivo 1: Clasificación**

El primer objetivo del proyecto consiste en proponer una arquitectura de red profunda capaz de aprender a distinguir los diferentes estados de trabajo de un proceso industrial.

En concreto, se manejarán datos de funcionamiento de un motor de inducción. La red propuesta ha de ser capaz de **determinar el estado en el que está trabajando el motor** (funcionamiento normal, fallo eléctrico, fallo mecánico, etc.) a partir de un conjunto de variables del proceso (típicamente, valores de aceleración y corriente, que son variables con una enorme riqueza de información acerca de la condición del proceso).

Para alcanzar este objetivo, se explorarán dos tipos de arquitecturas profundas: las redes *feedforward* [25] y las redes convolucionales [27].

- **Objetivo 2: Predicción**

El segundo objetivo del proyecto consiste en proponer una arquitectura de red profunda capaz de reproducir el comportamiento de un sistema en situaciones futuras o hipotéticas.

En concreto, se manejarán datos de consumos eléctricos. La red propuesta ha de ser capaz de **estimar la demanda eléctrica prevista para un instante de tiempo futuro** a partir de un conjunto de variables del sistema (demandas previas, día de la semana, día del año, etc.).

Para alcanzar este objetivo, se explorarán tres tipos de arquitecturas profundas: las redes *feedforward* [25], las redes convolucionales [27] y las redes LSTM [19].

- **Objetivo 3: Extracción de características relevantes**

El tercer objetivo del proyecto consiste en explorar una particularidad del *deep learning*: el r*epresentation learning* [32], que es la facultad de las estructuras neuronales profundas, con muchas capas ocultas, de aprender las fases de extracción de características. En ese sentido, se propone investigar en esquemas de redes profundas no sólo capaces de clasificar/predecir, sino también de extraer las características relevantes del proceso para llevar a cabo dicha clasificación/predicción de la forma más satisfactoria.

Para alcanzar este objetivo, se analizarán las **redes convolucionales** propuestas en los apartados previos, tratando de extraer información adicional sobre los procesos: análisis de los **filtros aprendidos por la red**, en busca de armónicos relevantes; análisis de los **pesos de las máscaras de convolución**, en busca de patrones horarios de consumo.

- **Objetivo 4: Visualización de datos del proceso**

El último objetivo del proyecto consiste en aplicar técnicas de **analítica visual** sobre datos de procesos industriales. Para ello, se explorarán dos vías: por un lado, se **visualizarán los parámetros internos** de las redes convolucionales propuestas en los apartados previos (bancos de filtros, máscaras de convolución), con el fin de generar representaciones interpretables que aporten intuición sobre el proceso; por el otro lado, se propondrá una arquitectura profunda del tipo ***deep autoencoder***, que permita proyectar el espacio de datos (se manejarán, de nuevo, datos de funcionamiento de un motor de inducción) sobre un espacio de baja dimensión (2D), para elaborar un "mapa" visual de los estados del proceso que permita su interpretación y visualización.

1.4. Estructura de este trabajo

Este documento está estructurado como sigue: en el Capítulo 1, se introduce el concepto de aprendizaje profundo, se habla de la importancia de trasladar esta técnica de análisis al ámbito industrial y se plantean los objetivos de este proyecto; en el Capítulo 2, se exponen los fundamentos teóricos sobre los que se asientan las arquitecturas profundas; en el Capítulo 3, se describen las arquitecturas propuestas para satisfacer los objetivos planteados en el Capítulo 1 y se exponen los resultados obtenidos; en el Capítulo 4, se reflexiona sobre el trabajo realizado y posibles líneas de trabajo futuro; finalmente, en el Capítulo 5, se exponen la planificación y presupuesto de este proyecto.

2
Métodos y técnicas

En sus inicios, la Inteligencia Artificial brillaba por resolver, y de manera eficiente, problemas intelectualmente **complicados para los seres humanos**, pero relativamente **sencillos para los computadores**. Se trataba de problemas fáciles de describir, cuya solución se recogía en un listado de reglas matemáticas.

Sin embargo, el **verdadero desafío de la IA** ha demostrado ser otro. Hoy en día, esta disciplina trata de resolver las tareas opuestas, aquellas que son fáciles de realizar para los humanos, que resolvemos **intuitivamente**, como reconocer palabras o caras en imágenes, y que no somos capaces de describir matemáticamente.

El ***deep learning*** es la rama de la IA que trata de abordar estos problemas más intuitivos. Y para ello, propone que los ordenadores aprendan de la experiencia, llegando así a comprender el mundo en términos de jerarquía de conceptos, donde cada concepto se define en relación a otros más simples. Si definimos un grafo con todos estos conceptos, el grafo será profundo, con muchas capas, y es por esta razón que hablamos de aprendizaje profundo.

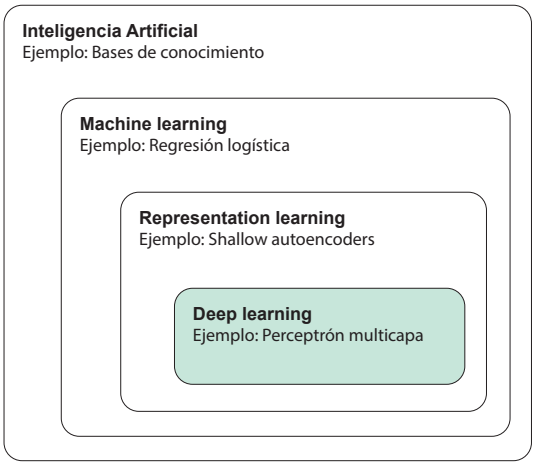

Figura 2.1. *Deep learning* en el contexto de la IA.

Este tipo de aprendizaje automático se caracteriza, además de por su **profundidad**, por su capacidad para **aprender las representaciones** de los datos. Es por ello que, como vemos en la Figura 2.1, se enmarca dentro del *representation learning*.

El rendimiento de los algoritmos de IA depende en gran medida de la representación de datos que reciben y, muchas veces, es difícil para el operador humano seleccionar qué características son las más relevantes. Una solución a este problema es utilizar el aprendizaje automático para describir no sólo el mapeo de la representación de salida, sino también la representación en si misma, lo que se conoce como ***representation learning***.

Combinando esta idea con la jerarquía de conceptos, el aprendizaje profundo aprende representaciones complejas en base a otras más sencillas. En la Figura 2.2 observamos cómo un sistema de este tipo puede representar el concepto de la imagen de una persona mediante la combinación de conceptos más simples, como esquinas y contornos, que a su vez se expresan en términos de bordes.

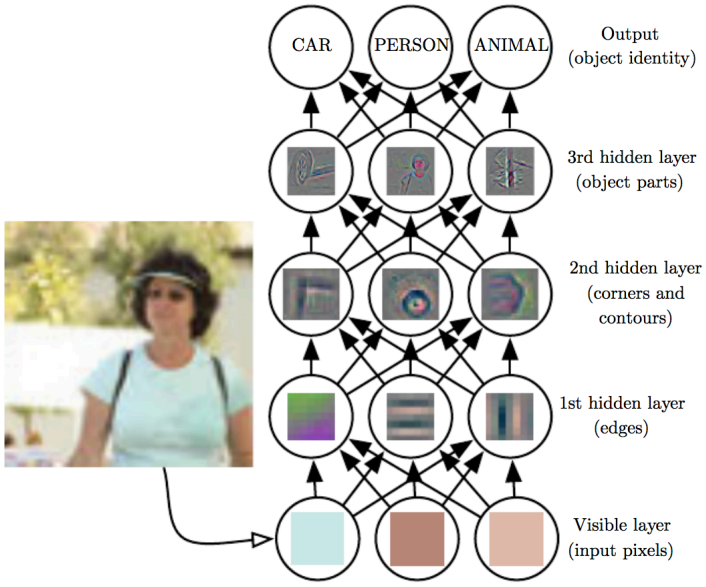

Figura 2.2. Ilustración de un modelo de d*eep learning*.
(Fuente: http://www.deeplearningbook.org)

De esta forma, los algoritmos de *deep learning* separan los factores de variación de los datos observados. Gracias a ello, son capaces de extraer características abstractas y de alto nivel de los datos crudos, con las que abordar eficientemente el problema objetivo.

Computacionalmente, las redes profundas son modelos complejos obtenidos por la interconexión de un gran número de unidades simples (neuronas artificiales). El poder de estas arquitecturas radica, de nuevo, en que muchas operaciones entre elementos simples permiten resolver problemas complejos.

A continuación, repasaremos los aspectos teóricos sobre los que se asientan estos modelos. Comenzaremos hablando de las **redes *feedforward***, que son la base del resto de arquitecturas que vamos a tratar: **redes convolucionales**, ***deep autoencoders*** y **redes recurrentes**.

2.1. Redes *feedforward*

Las redes *feedforward* [25], también conocidas como perceptrones multicapa (*Multilayer Perceptrons*, MLPs) son los **modelos de aprendizaje profundo por excelencia**. El objetivo de estas redes es aproximar una función f cualquiera, por ejemplo: para el caso de un clasificador donde $y = f(x;\theta)$, la red aprende la mejor aproximación del vector de parámetros θ que permite mapear las entradas x en una categoría y. Dado que en estas arquitecturas la información fluye en un solo sentido (de las entradas hacia la salida), han recibido siempre el nombre de redes *feedforward*. Cuando estas redes se extienden para incluir conexiones de realimentación, pasan a denominarse redes recurrentes (sección 2.4).

2.1.1. Estructura

Para establecer el mapeo entre entradas y salidas, las redes *feedforward* utilizan una arquitectura de red constituida por tres tipos de capas totalmente conectadas: capa de entrada, capas ocultas y capa de salida.

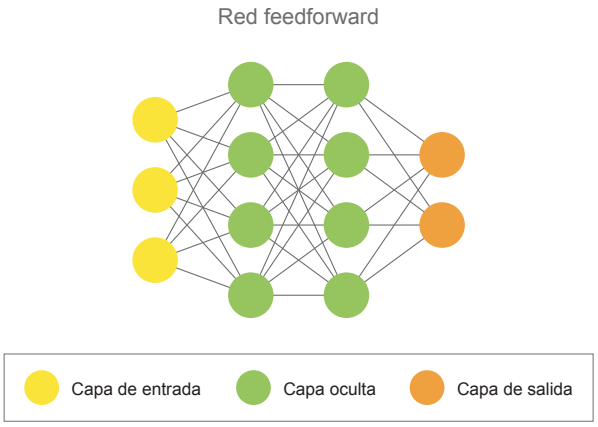

Figura 2.3. Esquema básico de una red *feedforward.*

- Capa de entrada: Se corresponde con el vector de datos de entrada. Si disponemos de un vector x de datos, cada elemento del vector $(x_1, x_2, \ldots, x_n)$ constituye una neurona de entrada.

- Capa de salida: Es la capa en la que se realiza la operación objetivo, por ejemplo, la clasificación. En ese caso, la capa de salida tiene tantas neuronas como clases presente el problema a tratar. Cada neurona tiene asociado un peso y un bias.

- Capas ocultas: Son las que contienen todos los cálculos intermedios de la red. Están formadas por neuronas ocultas, cada una de las cuales tiene un peso y un bias, al igual que las neuronas de la capa de salida.

Mientras que las capas de entrada y de salida son únicas, el número de capas ocultas es variable y no determinado. De hecho, la capacidad para manejar un **gran**

número de capas ocultas es lo que hace **profundo** a este tipo de aprendizaje y cabe mencionar aquí sus beneficios.

De acuerdo al teorema de aproximación universal [33,34], una red *feedforward* de una única capa oculta, con un número arbitrariamente grande de neuronas, se comporta como un aproximador universal. Pero esta capa oculta puede llegar a ser enorme, dificultando el aprendizaje y la generalización. En su lugar, el uso de **modelos profundos** permite reducir el número de neuronas necesarias para representar la función deseada, pudiendo completar el aprendizaje y respetando la capacidad de generalización de los modelos [25,26].

Esta idea se observa en la Figura 2.4, donde se muestra la habilidad de las redes profundas *feedforward* para remapear el espacio de entrada en busca de complejas simetrías en los datos.

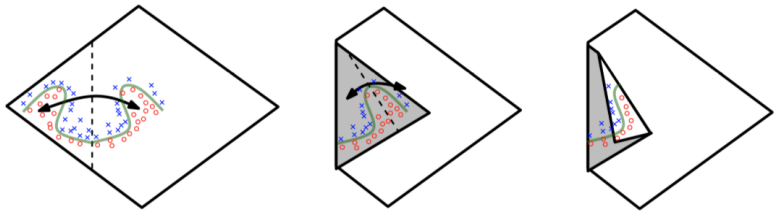

Figura 2.4. Plegado del espacio de datos 2D, resultado del proceso de aprendizaje de una arquitectura profunda. (Fuente: Imagen extraída de [26])

2.1.2. Función de activación

En los diagramas de la Figura 2.5 podemos apreciar las **operaciones** que tienen lugar **entre capas** de una arquitectura de red. En el diagrama de la izquierda, vemos que se trata de un ejemplo formado por una capa de entrada con dos neuronas (x_1, x_2), una capa oculta con otras dos neuronas (h_1, h_2) y una capa de salida con una neurona (y). A la derecha, en una versión simplificada de la misma red, observamos que la matriz W es la que define el mapeo de x a h, mientras que el vector w hace lo mismo de h a y.

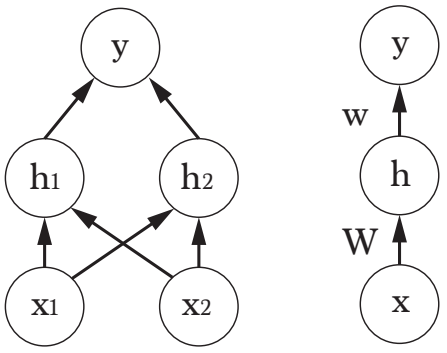

Figura 2.5. Relaciones entre capas de una red *feedforward.* Se muestra un mismo ejemplo de red de acuerdo a dos estilos diferentes.

Por tanto, la transformación que se aplica al vector de datos en cada capa oculta es la siguiente: $h(x) = s(Wx + b)$, donde b es el vector de bias, W la matriz de pesos y s **la función de activación**.

Esta función de activación es un elemento clave en las redes neuronales artificiales, pues es la que les otorga **flexibilidad**, proporcionándoles la capacidad de estimar complejas relaciones no lineales en los datos.

La elección de la función de activación es una decisión trascendente, dado que cada función se ajusta mejor a unos u otros problemas. Además, los pesos han de inicializarse de manera distinta según la función elegida. A este respecto, Glorot y Bengio han propuesto diferentes métodos de inicialización [35], basados en la longitud del vector de entrada, el número de neuronas de la capa oculta y el tipo de función de activación.

Aunque existe una gran variedad de funciones (identidad, binaria, lineal, sigmoidal, tangente hiperbólica, rectificadora, softplus, softmax, etc.), entre las más destacadas se encuentran:

Función sigmoidal: $$sigmoid(a) = \frac{1}{1 + exp^{-a}} \quad (2.1)$$

Función tangente hiperbólica: $$tanh(a) = \frac{exp^{a} - exp^{-a}}{exp^{a} + exp^{-a}} \quad (2.2)$$

No obstante, estas funciones tienen un elevado coste computacional y en los últimos años se han visto reemplazadas por la función rectificadora (*Rectified Linear Unit*, **ReLU**), introducida en el año 2000 por Hahnloser [36] con motivaciones biológicas y matemáticas. Esta función tiene un menor coste computacional, pues en lugar de hacer cálculos con exponenciales, utiliza una simple operación de máximo. Adicionalmente, su forma, lineal y no saturada, proporciona a las redes la capacidad para evitar los problemas de bajo gradiente [37] y permite acelerar la convergencia del entrenamiento. Por todo ello, la función ReLU se ha convertido en la más empleada para la activación de las **capas ocultas**.

Función ReLU: $$ReLU(a) = max(0,a) \quad (2.3)$$

En lo que respecta a la **capa de salida** destaca el uso de la función **identidad** en problemas de regresión y la función **softmax** en problemas de clasificación. Esta última es una generalización de la función sigmoidal, encargada de normalizar la salida a valores comprendidos entre [0,1]. Se utiliza especialmente en los problemas de clasificación multiclase, donde la sigmoidal no permite distinguir más de dos clases.

2.1.3. Mecanismo de aprendizaje

Como ya hemos mencionado, el éxito de las RNAs reside en que no necesitan ser programadas para resolver un problema, sino que estas arquitecturas son capaces de aprender por sí mismas la forma de resolverlo. Para ello, se recurre a un proceso de **aprendizaje supervisado**: se presenta a la red un conjunto de patrones de ejemplo (una serie de muestras de entrada con sus correspondientes salidas esperadas) y los pesos de

las neuronas se van modificando de manera proporcional al error que se produce entre la salida real y la salida esperada. El método de entrenamiento más utilizado para completar este proceso es el algoritmo de retropropagación o *backpropagation* [38].

- **Algoritmo de retropropagación o *backpropagation***

Este algoritmo es un método de aprendizaje supervisado de gradiente descendente, en el que se distinguen claramente dos fases: la **fase de propagación**, en la que se introduce un patrón de ejemplo a la red, que se va propagando desde la entrada hacia la salida; y la **fase de aprendizaje**, en la que los errores obtenidos a la salida de la red van propagándose hacia atrás, desde la salida hacia la entrada, con el objetivo de actualizar los pesos de las neuronas mediante el gradiente de la función de error.

Por tanto, el proceso de aprendizaje consiste en propagar hacia atrás el error entre la salida obtenida y la salida esperada, que se pretende vaya disminuyendo en el transcurso de las iteraciones. Para calcular dicho error, se recurre habitualmente a la función del error cuadrático medio, que se define como:

$$e(n) = \frac{1}{2} \sum_{i=1}^{n_s} (s_i(n) - y_i(n))^2 \tag{2.4}$$

Siendo: n cada una de las muestras de ejemplo; n_s el número de neuronas de la capa de salida; s_i e y_i la salida esperada y obtenida, respectivamente, en la neurona i de la capa de salida.

Una vez calculado el error en cada muestra, $e(n)$, el objetivo es minimizarlo. Para ello, aplicando el método del descenso del gradiente, cada peso de la red w se actualiza para cada patrón de entrada n, siendo α la **tasa de aprendizaje**, de acuerdo a la siguiente expresión:

$$w(n) = w(n-1) - \alpha \frac{\partial e(n)}{\partial w} \tag{2.5}$$

Dado que las neuronas de la red están agrupadas en niveles de capas ocultas, es posible aplicar de forma eficiente el método del descenso del gradiente, siguiendo para ello **la regla delta generalizada**.

- **Regla delta generalizada**

La regla delta generalizada no es más que una forma eficiente de aplicar el método de descenso del gradiente a los parámetros de la arquitectura de red. Su uso consiste en ajustar pesos y bias tratando de minimizar la suma del error cuadrático. Para ello, se modifican dichas variables en la **dirección contraria al gradiente del error**.

Entre las propiedades de la regla delta, destaca su capacidad de generalización. Por ello, las RNAs entrenadas con esta técnica dan respuestas más que razonables cuando el sistema recibe entradas que no ha visto antes.

- **Tasa de aprendizaje**

La tasa de aprendizaje α es el parámetro que determina la velocidad a la que cambian los pesos de la red y tiene rango [0,1]. Los valores más cercanos a cero hacen que los pesos cambien despacio, acercándose lentamente a la convergencia. Mientras, los valores cercanos a uno hacen que la red converja rápidamente, pudiendo oscilar demasiado una vez encontrado el peso óptimo final y terminar alejándose de él. Por esta razón, encontrar una **tasa de aprendizaje óptima** es uno de los factores **clave** para un buen entrenamiento.

Aquí es donde entra en juego un segundo concepto: el término **momento** η, que pondera la influencia del cambio de los pesos en la iteración anterior. Así, cuando los pesos han cambiado mucho sabemos que aún queda lejos la tasa de aprendizaje óptima y hay que avanzar en su búsqueda más rápidamente. En cambio, si los pesos no han cambiado apenas, sabremos que el valor óptimo de α está cerca y el avance debe ser más leve. Teniendo en cuenta esta mejora, la ecuación 2.5 quedaría de la siguiente manera:

$$w(n) = \eta \Delta w(n-1) - \alpha \frac{\partial e(n)}{\partial w} \tag{2.6}$$

Siendo $\Delta w(n-1) = w(n-1) - w(n-2)$ el incremento que sufrió el parámetro w en la anterior iteración.

En la Figura 2.6 se muestra cómo un buen equilibrio entre el momento y la tasa de aprendizaje (en la imagen β y α, respectivamente) permite alcanzar la solución óptima. En ocasiones, se recurre a algoritmos auxiliares, como Adam [39], capaces de realizar una optimización adaptativa de la tasa de aprendizaje.

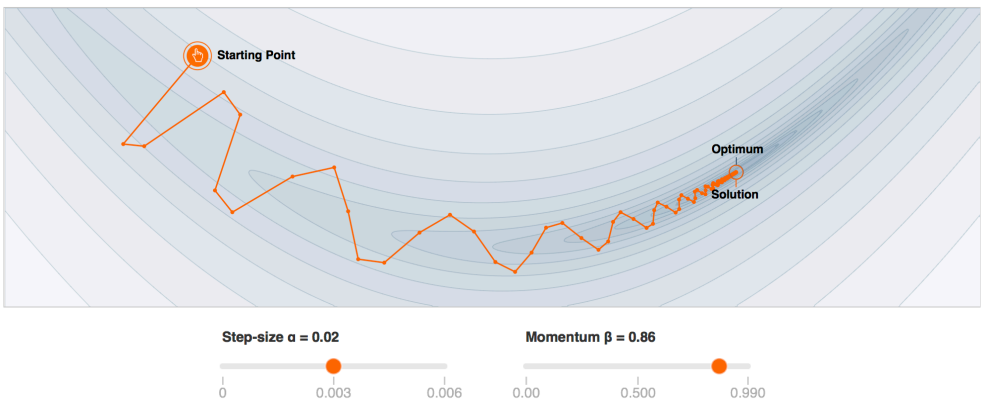

Figura 2.6. Momento y tasa de aprendizaje.
(Fuente: Goh, "Why Momentum Really Works", Distill, 2017. http://doi.org/10.23915/distill.00006)

- **Modo de entrenamiento**

Finalmente, es necesario mencionar que el aprendizaje se produce mediante la presentación sucesiva del conjunto de entrenamiento, donde cada presentación completa recibe el nombre de **época**. Así, el proceso de aprendizaje se repite época tras época, de acuerdo al algoritmo de *backpropagation*, hasta que los pesos se estabilizan y el rendimiento de la red converge a un valor aceptable.

La forma en que se actualizan los pesos da lugar a dos modos de entrenamiento:

- Modo secuencial: En este modo de entrenamiento la actualización de los pesos se produce tras la presentación de cada ejemplo de entrenamiento. Si el conjunto de entrenamiento consta de N ejemplos, el modo secuencial de entrenamiento tiene como resultado N correcciones de pesos durante cada época.

- Modo *batch*: En este modo la actualización de los pesos se produce una vez, tras la presentación de todo el conjunto de entrenamiento. Para cada época se calcula el error cuadrático medio producido por la red.

Si las muestras de entrenamiento se presentan a la red de manera aleatoria, el modo secuencial convierte la búsqueda de pesos en estocástica y disminuye la probabilidad de que el algoritmo de *backpropagation* quede atrapado en un mínimo local. Por su parte, el uso del modo de entrenamiento *batch* provee una estimación precisa del vector gradiente, garantizando así la convergencia hacia un mínimo local.

2.1.4. Sobreajuste

Una vez finalizado el proceso de aprendizaje, las redes profundas han de ser capaces de abordar la tarea para la que hayan sido diseñadas, a partir de lo aprendido mediante un conjunto de datos de entrenamiento. No obstante, para abordar esa tarea de forma satisfactoria, las redes deben tener también la habilidad de afrontar situaciones distintas a las presentes durante el entrenamiento. Es decir, han de ser capaces de **generalizar**.

El sobreajuste u *overfitting* es la consecuencia de sobreentrenar el aprendizaje de las redes profundas, de manera que estas pierdan su capacidad de generalización. Se trata de un **problema habitual** en estas arquitecturas y en la literatura se proponen distintos mecanismos de **regularización** para reducir su efecto.

Un mecanismo frecuente en el mundo del aprendizaje automático (el problema de sobreentrenamiento se da también en otros algoritmos de aprendizaje, no sólo en las RNAs) es el ***early stopping*** [40]. Este método propone aislar un subconjunto de muestras de entrenamiento y utilizarlo a modo de validación. Cuando el error sobre el conjunto de validación empeora, se detiene el entrenamiento y se conserva así la capacidad de generalización de la red. Sin embargo, el uso de este método en las arquitecturas profundas

no siempre aporta buenos resultados. Requiere un análisis detallado [41] y, por ello, es habitual recurrir a otras opciones.

Uno de los mecanismos más sencillos para reducir el sobreajuste consiste en **simplificar el aprendizaje**: disminuir el número de capas, el número de neuronas por capa, el número de épocas, etc. Otra opción es incluir una serie de restricciones en los pesos de las neuronas. En ese sentido, destacan: la técnica conocida como ***weight decay***, que consiste en incluir penalizaciones para los pesos grandes en función de sus valores al cuadrado (penalización L2) o absolutos (penalización L1); o la técnica ***max-norm***, que propone restringir a un valor máximo la norma del vector de pesos.

Sin embargo, a veces estos mecanismos no son suficientes para prevenir el sobreajuste y resulta necesario recurrir a otras técnicas más elaboradas. En este sentido, destaca la técnica de ***dropout*** [42], que propone transformar el entrenamiento de una red compleja en el entrenamiento de varias redes más simples, donde cada red simple es el resultado de ignorar parte de las neuronas de la red original.

Siguiendo esta idea, si se desactiva una neurona temporalmente, se deja fuera de la propia red y, por tanto, se pierden las conexiones entrantes y salientes asociadas. Esto conduce a un entrenamiento más uniforme y eficiente del modelo, ya que se simula el entrenamiento de varias subredes combinadas en una sola. Para llevar a cabo esta idea, se establece una probabilidad de retención de las neuronas y se genera una máscara aleatoria según esta probabilidad. La arquitectura de red final se obtiene multiplicando cada neurona por su probabilidad de retención correspondiente.

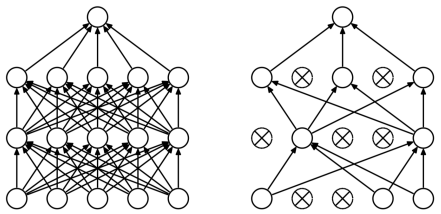

Figura 2.7. Ejemplo de *dropout*. A la izquierda se observa la red original y a la derecha esa misma red con *dropout*. (Fuente: Imagen extraída de [42])

Finalmente, otro mecanismo extendido es el de ***batch normalization*** [43], que optimiza y acelera el entrenamiento de las redes, mediante la reducción de la covarianza interna de los datos. Esta normalización es, principalmente, una técnica de optimización. Pero como efecto secundario, introduce ruido en las redes, colaborando así en la regularización. Cuando se tiene un conjunto de datos grande, la optimización se vuelve más importante que la regularización, por lo que el *batch normalization* adquiere especial protagonismo ante grandes volúmenes de datos.

En la práctica, la elección de la técnica óptima de regularización depende de cada problema. Además, no existen incompatibilidades entre las técnicas expuestas y las combinaciones entre ambas pueden proporcionar buenos resultados. Un ejemplo se muestra en [44], donde Goodfellow propone el uso conjunto de *dropout* y *batch normalization*. Otra combinación exitosa es la de *max-norm* y *dropout*, ante la cual se recomienda utilizar una tasa de aprendizaje y un momento elevados [42].

2.2. Redes convolucionales

Las redes convolucionales (*Convolutional Neural Networks, CNN*) [27] son **muy similares a las redes *feedforward***. Su principal diferencia radica en la inclusión de **capas convolucionales**, cuyas neuronas no están totalmente conectadas: cada neurona de una capa no recibe conexiones entrantes de todas las neuronas de la capa anterior, sino sólo de algunas, lo cual favorece que cada neurona se especialice únicamente en una región de la capa anterior, reduciendo drásticamente el número de operaciones a realizar. De esta forma, las redes convolucionales dividen y modelan la información en partes más pequeñas, para combinar después esta información en las capas más profundas de la red.

Por ejemplo, en el caso del tratamiento de una imagen: las primeras capas de la red buscarían contornos, las siguientes tratarían de detectar formas a partir de los contornos, las próximas prestarían atención a la posición de las formas, etc. Por último, en las capas finales del modelo, se combinarían todos los patrones descubiertos para conseguir una predicción final de la suma de todos ellos. Así es como las redes convolucionales consiguen modelar una gran cantidad de datos: dividiendo el problema en partes para conseguir predicciones más sencillas y precisas.

Aunque estas arquitecturas se emplean típicamente en problemas de tratamiento de imagen (donde han demostrado unos excelentes resultados [45]), también se emplean con éxito en otros contextos, como el análisis de series temporales para reconocimiento de voz [28].

2.2.1. Estructura

Para establecer el mapeo entre entradas y salidas, las redes convolucionales utilizan una arquitectura de red constituida por 5 tipos de capas: capa de entrada, capas convolucionales, capas de *pooling*, capas ocultas y capa de salida.

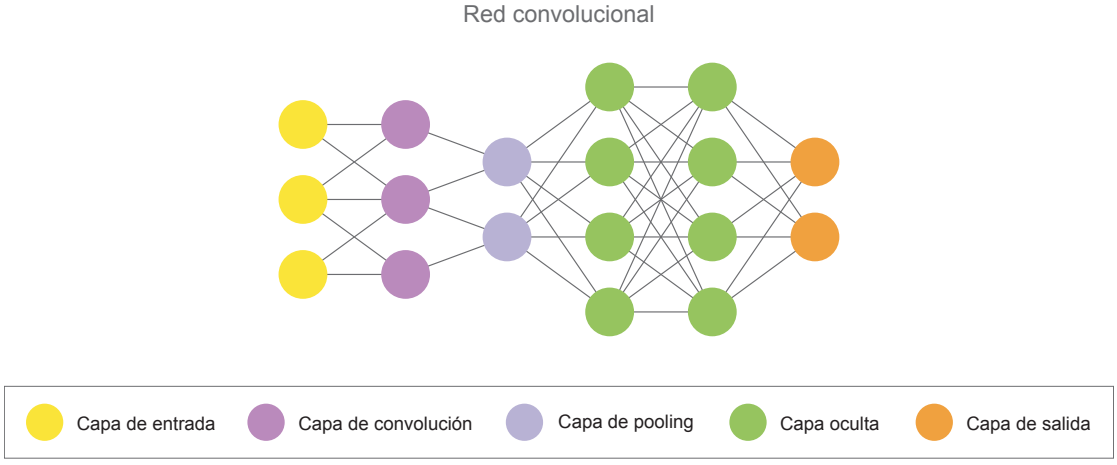

Figura 2.8. Esquema básico de una red convolucional.

Las capas de entrada y de salida se ajustan a las expuestas para el caso de las redes *feedforward* en la sección 2.1.1. Mientras, el resto de capas son propias de este tipo de arquitectura:

- Capas convolucionales: Son las capas que dan nombre a la red y se caracterizan por aplicar operaciones de convolución [46] en lugar de productos entre matrices. En este caso, las matrices de pesos son filtros de un tamaño prefijado que se convolucionan con los datos de trabajo, devolviendo una señal filtrada de menor tamaño que la original. Estas capas quedan determinadas por el número de filtros que incorporan y por el tamaño de los mismos.

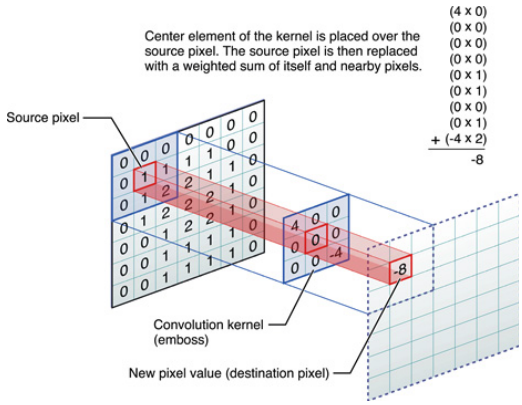

Figura 2.9. Ejemplo de convolución de un filtro con una imagen.
(Fuente: http://www.westworld.be/convolution-kernels/)

- Capas de *pooling* o submuestreo: Estas capas se sitúan a la salida de las capas convolucionales y se utilizan para reducir la dimensionalidad de los datos. Esto implica la reducción del coste computacional de las siguientes convoluciones (si las hubiese), del número de parámetros a determinar, proporciona un grado de invarianza ante traslaciones y ayuda también a controlar el sobreajuste de la arquitectura. La operación de *pooling* requiere determinar el tamaño de la región sobre la que se desea aplicar el submuestreo y el tipo de *pooling* a realizar. Típicamente, se recurre al *max-pooling* (se toma el valor máximo de la región) o al *average-pooling* (se toma el valor medio de la región).

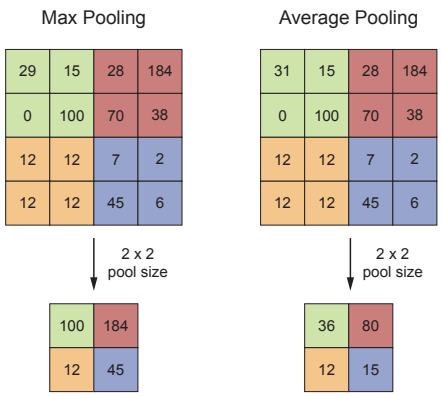

Figura 2.10. Ejemplos de submuestreo.

- Capas ocultas: Tienen las mismas características que en el caso de las redes *feedforward*. Se diferencian en que, en este caso, la primera capa oculta tiene una labor especial: es la que se conecta a la salida de las capas de convolución y *pooling*, encargándose de convertir la matriz de datos en un vector plano. Por ello, esta capa suele recibir el nombre de capa *flatten*. Cabe mencionar que las capas ocultas son las únicas totalmente conectadas en esta arquitectura.

Al igual que en las redes *feedforward,* las capas de entrada y salida son únicas, mientras que el número del resto de capas es variable. Esto da lugar a infinidad de posibilidades y la elección de la arquitectura definitiva dependerá del problema a tratar. Asimismo, deben elegirse otros parámetros, como el número de filtros de convolución y su tamaño, o el tipo y tamaño de las operaciones de submuestreo, los cuales tendrán una gran influencia en el comportamiento de la red.

2.3. *Deep autoencoders*

Los *deep autoencoders* [31] pueden ser vistos como un caso particular de las redes *feedforward*. Se trata de RNAs entrenadas con el objetivo de **reproducir a la salida** (r) **la misma información que reciben a la entrada** (x). Internamente, tienen una capa oculta (h) que describe la codificación empleada para representar la entrada. Ante esta situación, puede considerarse que la red está compuesta por dos partes: un ***encoder*** $h = f(x)$ que deconstruye la entrada y un ***decoder*** $r = g(h)$ que la reconstruye.

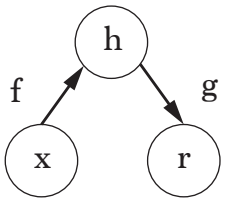

Figura 2.11. Estructura de un *autoencoder.*

Si el *autoencoder* se limita a aprender la función identidad, la arquitectura resulta de poca utilidad. Por ello, el interés de los *autoencoders* reside en imponer **restricciones** en la capa oculta, impidiendo que se produzca una copia de los datos de entrada. Como resultado, la codificación aprendida contendrá información útil acerca de los datos y si, además, esta codificación es de menor dimensionalidad que la entrada, el *autoencoder* se convertirá en una herramienta de **reducción de la dimensión**.

2.3.1. Estructura

La arquitectura de los *autoencoders* se caracteriza por presentar tantas neuronas en la **capa de salida** como en la **capa de entrada**, pues de otro modo sería imposible

reproducir a la salida los datos de entrada. En cuanto a la restricción impuesta al *autoencoder*, la más extendida consiste en incluir un **cuello de botella** en la red, es decir, una **capa oculta** de menor dimensión que las capas de entrada y salida, como se muestra en la Figura 2.12.

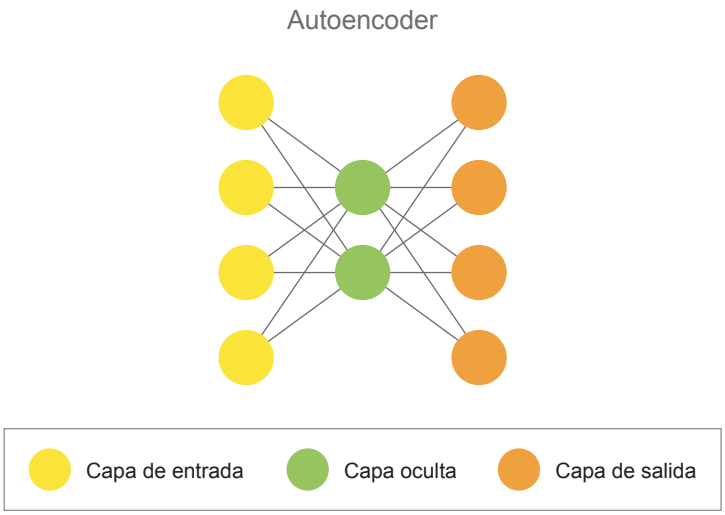

Figura 2.12. Esquema básico de un *autoencoder.*

De esta forma, la red se ve obligada a generar en su capa oculta una versión comprimida de la información, practicando una **reducción de la dimensión**. Esta reducción permite representar los datos de trabajo en espacios de baja dimensión (2D, 3D), facilitando la interpretación de los mismos y aportando un enfoque de analítica visual.

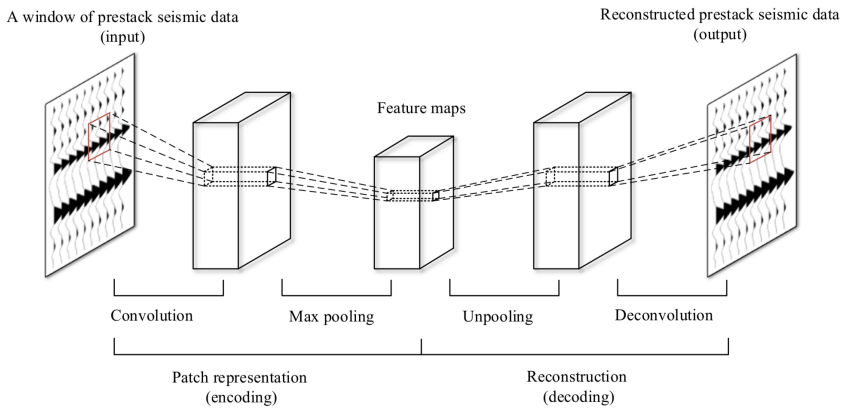

Figura 2.13. Arquitectura de un *autoencoder* convolucional empleado para el análisis de datos sísmicos.
(Fuente: Imagen extraída de [49])

Finalmente, cabe destacar que el número de capas ocultas es variable (al igual que en el resto de arquitecturas profundas expuestas) y que en ocasiones se intercalan con **capas de otra naturaleza**. En concreto, destaca el uso de capas de convolución, dando lugar a los llamados ***autoencoders* convolucionales** [47] (Figura 2.13). En este caso, la

arquitectura incorpora: operaciones de convolución y *pooling* en el encoder; y sus opuestas, deconvolución y *unpooling*, en el decoder (Figura 2.14).

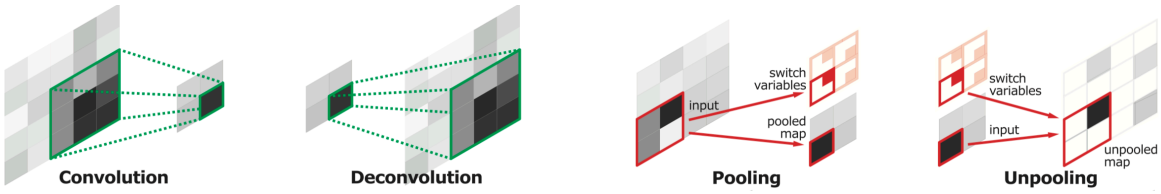

Figura 2.14. Ilustración de las operaciones de convolución/deconvolución y *pooling*/*unpooling*. Para esta última, se expone una operación de tipo *max-pooling* que almacena las posiciones de los máximos encontrados (*switch variables*).
(Fuente: Imagen extraída de [48])

2.4. Redes recurrentes

La forma de razonar de los seres humanos está íntimamente relacionada con la memoria. Por ejemplo, cuando leemos, entendemos cada palabra basándonos en el contexto que forman las palabras previas; no desperdiciamos las ideas anteriores, sino que estas tienen persistencia en nuestra memoria.

En este contexto, las **redes recurrentes** (*Recurrent Neural Networks, RNN*) [29] representan la arquitectura, entre las expuestas, que más se acerca a nuestra forma de razonar. Estas redes no tienen una estructura de capas, sino que habilitan conexiones arbitrarias entre todas las neuronas, lo que permite incorporar el concepto de temporalidad, dotando a las redes de **memoria**.

Las redes *feedforward* (así como las redes convolucionales y los *autoencoders*) no cuentan con esta capacidad de persistencia y esto las hace débiles a la hora de afrontar algunos problemas, pues podemos decir que empiezan a “pensar" desde cero. Gracias a la incorporación de **bucles de realimentación**, las redes recurrentes han solventado esta deficiencia de las redes tradicionales, convirtiéndose en uno de los grandes pilares del *deep learning*.

En la Figura 2.15 observamos un ejemplo básico de red recurrente, A, que recibe una entrada x y devuelve una salida h. El bucle de realimentación permite que la información pase de un ciclo al siguiente.

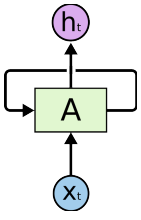

Figura 2.15. Esquema básico de una RNN
(Fuente: Imagen extraída de [52])

Aunque el concepto de bucle parezca algo extraño, las redes recurrentes no están tan alejadas de las redes *feedforward*. Una RNN puede ser creada a través de múltiples copias de la misma red, pasando su salida al nodo sucesor. Al desenrollar el bucle, la arquitectura quedaría como sigue:

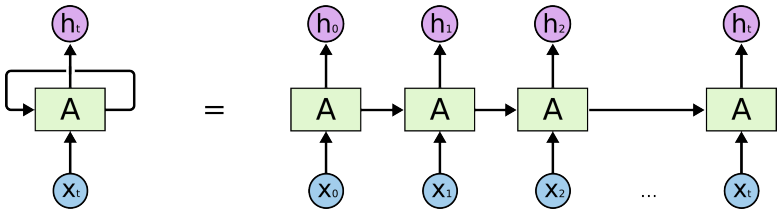

Figura 2.16. RNN desenrollada.
(Fuente: Imagen extraída de [52])

Esta naturaleza en forma de cadena explica que las redes recurrentes estén especialmente relacionadas con las secuencias y listas, por lo que son ideales para el trabajo con este tipo de datos. En los últimos años se han logrado grandes éxitos aplicando RNNs en una amplia variedad de problemas [29,50,51] como: reconocimiento de voz, modelado del lenguaje, traducción, etc.

Un factor clave en estos éxitos es el uso de las redes LSTM (*Long Short Term Memory*) [19], un tipo especial de RNNs que trabajan de forman más eficiente que las redes recurrentes estándar.

2.4.1. Redes LSTM

En ocasiones, es posible tomar decisiones en base al contexto reciente (Figura 2.17) pero otras veces es necesario acceder a información más alejada en el tiempo (Figura 2.18). Desafortunadamente, a medida que el espacio entre la información requerida y la tarea actual aumenta, las RNNs tardan cada vez más en aprender a conectar esa información, llegando a ser incapaces de completar el aprendizaje.

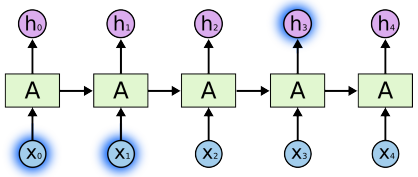

Figura 2.17. Dependencia a corto plazo.
(Fuente: Imagen extraída de [52])

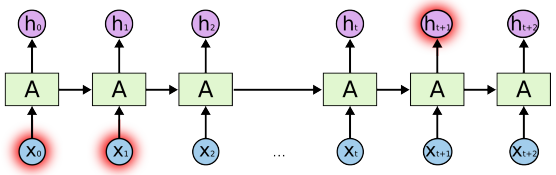

Figura 2.18. Dependencia a largo plazo.
(Fuente: Imagen extraída de [52])

El poder de las redes LSTM (*Long Short Term Memory*) [19] reside en que son capaces de hacer frente a esta dificultad, pues se trata de arquitecturas con **memoria a largo y corto plazo**. Para conseguirlo, utilizan unos módulos repetidores (A) más complejos que las RNNs estándar.

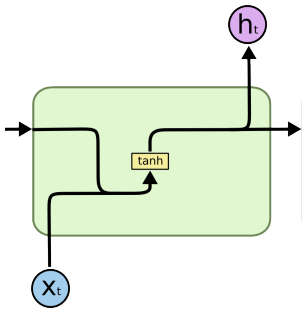

Figura 2.19. Módulo repetidor RNN estándar.
(Fuente: Imagen extraída de [52])

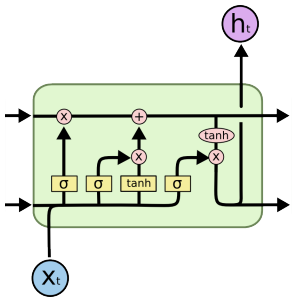

Figura 2.20. Módulo repetidor LSTM.
(Fuente: Imagen extraída de [52])

En las Figuras 2.19 y 2.20 se observa que el módulo repetidor de las RNNs estándar tiene una sola capa de red, mientras que el módulo de las LSTM tiene cuatro, que interactúan entre sí para proporcionar una memoria a más largo plazo [52].

2.4.2. Estructura

La arquitectura de las redes recurrentes consta de una **capa de entrada**, una **capa de salida** y un número indeterminado de capas intermedias, que pueden ser **capas recurrentes** (arquitectura RNN estándar) o **capas LSTM**, como se muestra en la Figura 2.21.

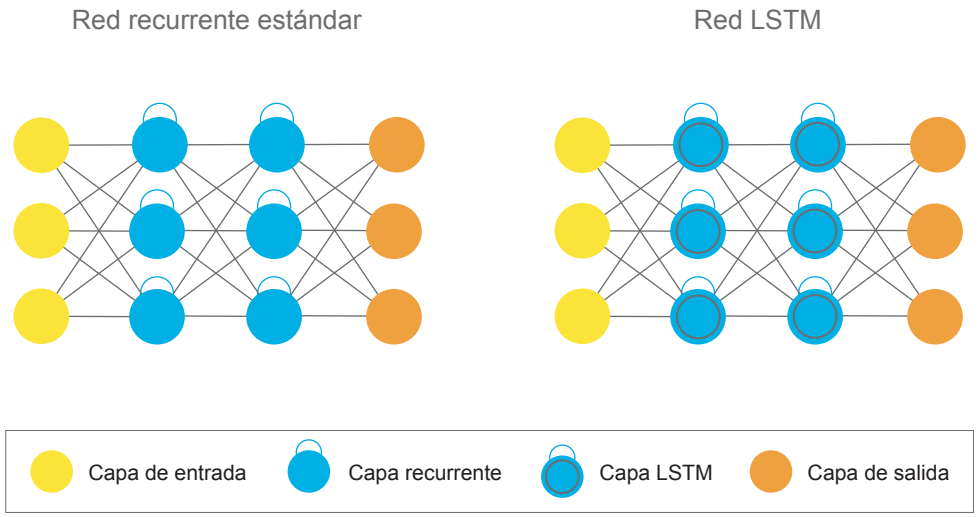

Figura 2.21. Esquema básico RNN y LSTM.

Cabe mencionar que, a menudo, se intercalan también otros tipos de capas en este tipo de arquitecturas (capas ocultas, capas convolucionales) [53].

2.5. Consideraciones de diseño

A la hora de diseñar una red profunda, el primer paso consiste en **seleccionar el tipo de arquitectura** que vamos a emplear. En este proyecto, se elegirá entre las cuatro arquitecturas expuestas: red *feedforward*, convolucional, recurrente y *deep autoencoder*.

A continuación, es necesario determinar el **número de capas** de la arquitectura, así como el **número de neuronas** de cada una (en el caso de capas de convolución o *pooling*, será preciso escoger también el nº de filtros, el tamaño de los mismos, el tamaño del submuestreo, etc.). Otros aspectos a tener en cuenta son los **parámetros del algoritmo de aprendizaje** (momento y tasa de aprendizaje), las **funciones de activación** de las capas o el mecanismo para evitar el **sobreajuste**.

En definitiva, son muchos los aspectos a considerar cuando trabajamos con arquitecturas profundas y el éxito del resultado dependerá del equilibrio que alcancemos entre todos ellos. En el próximo capítulo, propondremos una serie de arquitecturas (especificando estos aspectos de diseño) para resolver problemas de clasificación, predicción, extracción de características y visualización de datos en sistemas de ingeniería.

3
Trabajo realizado y resultados obtenidos

Una vez descritas las arquitecturas de red (Capítulo 2), se propone su uso para la resolución de problemas en el ámbito industrial. La implementación de estas redes profundas se ha desarrollado en lenguaje **Python** (https://www.python.org, versión 3.6.1) con la ayuda de las librerías gratuitas **Keras** (https://keras.io, versión 2.0.6) y **Tensorflow** (https://www.tensorflow.org, versión 1.2.1), ampliamente aceptadas por la comunidad y con un gran crecimiento en fiabilidad y potencial. El entorno de desarrollo empleado ha sido el **Notebook** de **Jupyter** (http://jupyter.org, versión 5.0.0).

En cuanto a los datos de trabajo, se han manejado dos escenas: análisis de **consumos energéticos** (datosConsumos.csv) y análisis de funcionamiento de un **motor de inducción** (dataicann.mat). El contenido de estos *datasets* se detalla en la primera sección del presente capítulo. A continuación, se describen las **arquitecturas de red generadas** y los **resultados obtenidos** con cada una de ellas. En relación con los objetivos planteados en el Capítulo 1, veremos que las arquitecturas expuestas tratan de satisfacer tareas de **clasificación**, **predicción**, **extracción de características** y **visualización de datos**. Con ello, se pretende explorar la viabilidad de las técnicas de *deep learning* en el análisis de procesos industriales.

3.1. Datos de trabajo

3.1.1. Motor de inducción (dataicann)

Este conjunto (proporcionado por el grupo de supervisión de procesos industriales de la Universidad de Oviedo, GSDPI: http://isa.uniovi.es/GSDPI) recoge datos de funcionamiento de un motor de inducción de 4kW, con rodamientos de tipo 6306-2Z/C3 y que gira a una velocidad de 1500rpm. Se trata de un motor sometido a nueve ensayos diferentes (Tabla 3.1), para cada uno de los cuales se han medido tres características de funcionamiento (Tabla 3.2) con una frecuencia de muestreo de 5000Hz. El *dataset* recibe el nombre de **dataicann** y se encuentra en formato MATLAB (.mat).

Ensayo	Descripción
1	Fallo mecánico (masa excéntrica en polea)
2	Fallo eléctrico y mecánico combinado
3	Funcionamiento normal
4	Fallo eléctrico (resistencia de 10 ohm en fase R)
5	Fallo eléctrico (resistencia de 15 ohm en fase R)
6	Fallo eléctrico (resistencia de 20 ohm en fase R)
7	Fallo eléctrico (resistencia de 5 ohm en fase R)
8	Fallo eléctrico gradual (resistencia aumenta y disminuye en fase R)
9	Fallo eléctrico (aumento gradual de resistencia en fase R)

Tabla 3.1. Ensayos del motor de inducción.
Se trata de nueve ensayos de 4 segundos de duración
(salvo el ensayo nº 9 que dura 8 segundos).

Variable	Descripción	Frecuencias relevantes
ax	Aceleración en dirección X (horizontal)	20-30Hz, 95-105Hz
ay	Aceleración en dirección Y (vertical)	20-30Hz, 95-105Hz
ir	Corriente en fase R	45-55Hz

Tabla 3.2. Variables presentes en el *dataset* dataicann.

3.1.2. Consumo energético (datosConsumos)

Este *dataset* recoge datos de demanda eléctrica de Gran Bretaña (tomados de http://www.gridwatch.templar.co.uk) durante un periodo de un año. Se trata de un conjunto de cuatro características (Tabla 3.3) tomadas con frecuencia horaria. El *dataset* recibe el nombre de **datosConsumos** y se encuentra en formato csv.

Variable	Descripción
C	Consumo en kW
W	Semana del año
D	Día de la semana
H	Hora del día

Tabla 3.3. Variables presentes en el *dataset* datosConsumos.

3.2. Clasificación

En esta sección se aborda la tarea de clasificación en el ámbito industrial. En concreto, se pretenden analizar datos de corriente y aceleración de un **motor de inducción** (dataicann) y, en base a ellos, **clasificar el estado de funcionamiento** del mismo (funcionamiento normal, fallo eléctrico, fallo mecánico, etc.). Para ello se proponen dos alternativas: una red *feedforward* y una red convolucional.

3.2.1. Clasificación con arquitectura *feedforward*

Esta arquitectura recibe como **entrada** un vector de cinco características, que serán los **valores eficaces** de aceleración y corriente calculados para las frecuencias indicadas en el *dataset* de trabajo (Tabla 3.2). Como **salida**, la arquitectura devuelve una representación vectorizada de la **clase de estado**, entre los nueve estados posibles (Tabla 3.1), a la que pertenece el vector de entrada.

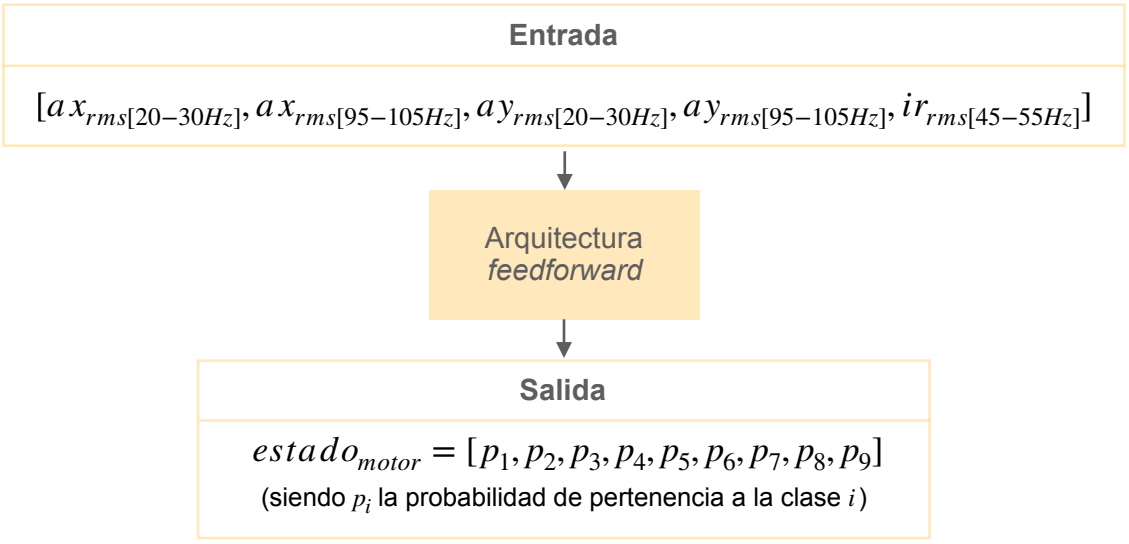

Figura 3.1. Esquema red *feedforward* para clasificación.

Para establecer la correspondencia entre el vector de entrada y el estado de salida, se propone la red *feedforward* expuesta en la Figura 3.2. Esta arquitectura consta una **capa de entrada** de 5 neuronas (tamaño del vector de entrada) y una **capa de salida** de 9 neuronas (número de posibles estados del motor). En cuanto a las **capas ocultas**, se ha optado por una combinación de baja carga computacional, con un pequeño número de capas y de neuronas por capa (5 capas ocultas con 16 neuronas cada una).

Para evitar el **sobreajuste** del modelo, se ha limitado la norma del vector de pesos, en las capas ocultas y de salida, a un valor máximo de 3 (valor típicamente empleado en la literatura [42]). También se aplica la técnica de *batch normalization* tras cada capa oculta.

Finalmente, se utiliza una función de **activación** tipo ReLU para las capas ocultas y una softmax para la capa de salida.

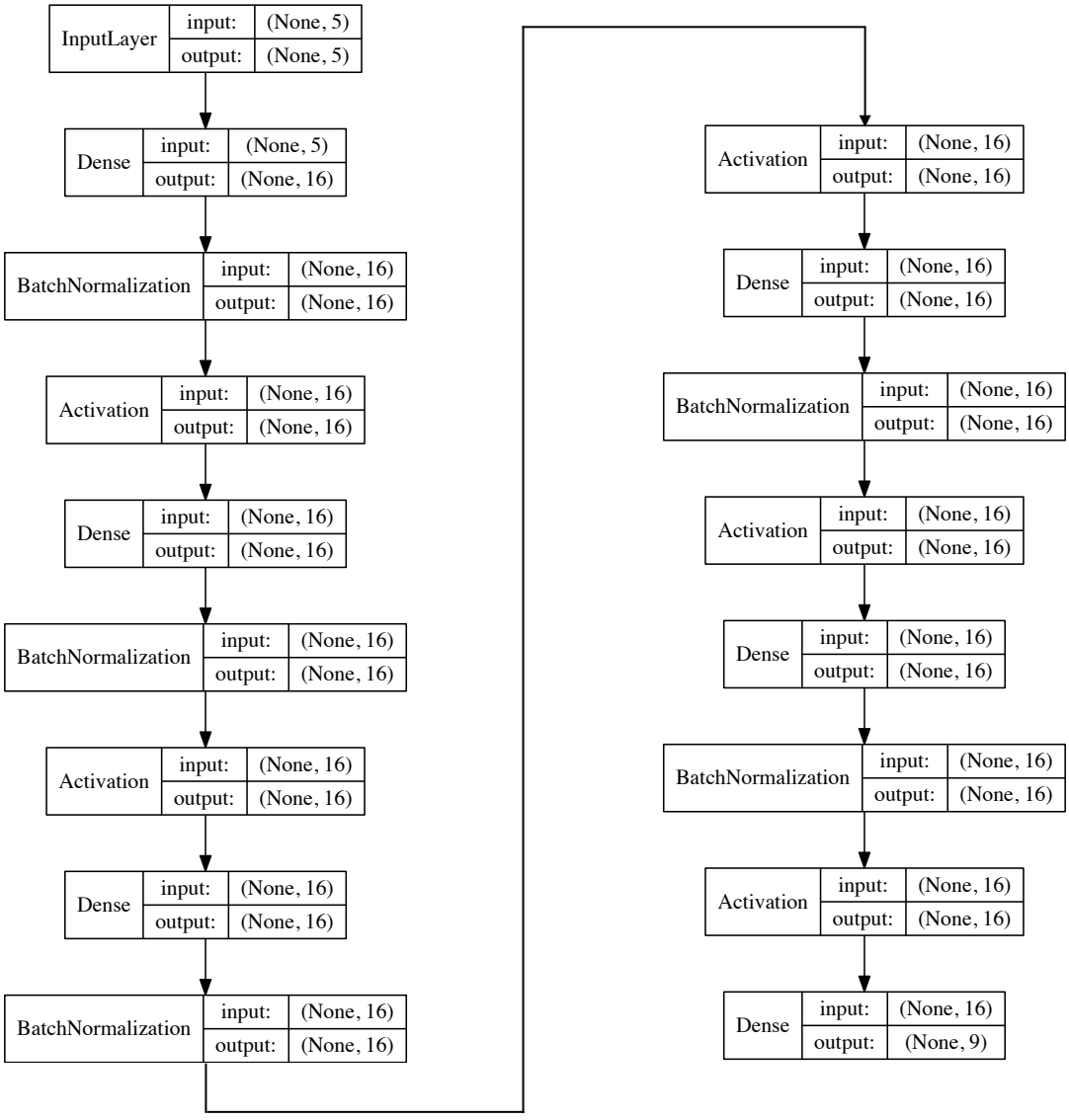

Figura 3.2. Arquitectura *feedforward* propuesta para la clasificación del estado del motor.

3.2.1.1. Resultados

La arquitectura expuesta en la sección anterior se ha puesto en práctica sobre el conjunto **dataicann**, calculando los valores eficaces para un tamaño de ventana de 400 muestras y sin solapamiento.

Para el **proceso de aprendizaje**, se han escogido valores típicos tanto para el momento ($\eta = 0.9$) como para la tasa de aprendizaje ($\alpha = 0.01$). El entrenamiento se ha completado en 200 épocas, con un tamaño del *batch* de 16 muestras y empleando para ello el 70% de los datos de trabajo (el 30% restante se ha utilizado a modo de test).

En la Figura 3.3 se muestra la evolución del comportamiento del modelo a lo largo de las épocas, evaluado sobre los datos de entrenamiento.

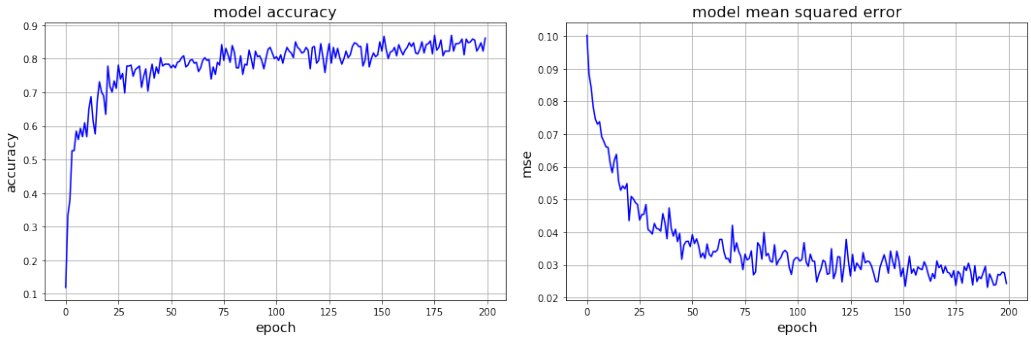

Figura 3.3. Evolución del modelo de clasificación *feedforward* a lo largo del entrenamiento. A la izquierda se muestra la precisión del modelo y a la derecha el error cuadrático medio.

Como resultado, este modelo es capaz de determinar el estado de funcionamiento del motor con una tasa de acierto del 86% (Tabla 3.4). Por tanto, podemos decir que la arquitectura *feedforward* propuesta es capaz de resolver satisfactoriamente el problema de clasificación, permitiendo abordar una tarea tan valiosa en el **sector industria**l como es la **detección de fallos**.

	Entrenamiento	Test
Precisión	0.861	0.865
Error cuadrático medio	0.024	0.025

Tabla 3.4. Resultados del modelo de clasificación *feedforward* para los dos conjuntos de datos (entrenamiento y test).

3.2.2. Clasificación con arquitectura convolucional

La arquitectura *feedforward* expuesta en la sección previa permite abordar la clasificación siendo conocidas las frecuencias de interés, es decir, las bandas de frecuencia

que contienen información relevante y para las cuales se calcula el vector de valores eficaces que actúa como entrada de la red.

Sin embargo, no siempre se dispone de información frecuencial acerca de los procesos, con lo que surge la necesidad de una arquitectura de red alternativa. En este contexto, se propone una arquitectura convolucional (Figura 3.4) capaz de encontrar por sí misma las frecuencias relevantes y, en base a ellas, clasificar los estados de trabajo del motor de inducción.

Esta red recibe **tres vectores entrada** con los valores históricos de las variables bajo estudio (vectores de N muestras, tomadas desde el instante actual k). Como **salida**, la arquitectura devuelve una representación vectorizada de la **clase de estado**, entre los nueve estados posibles (Tabla 3.1), a la que pertenece el vector de entrada.

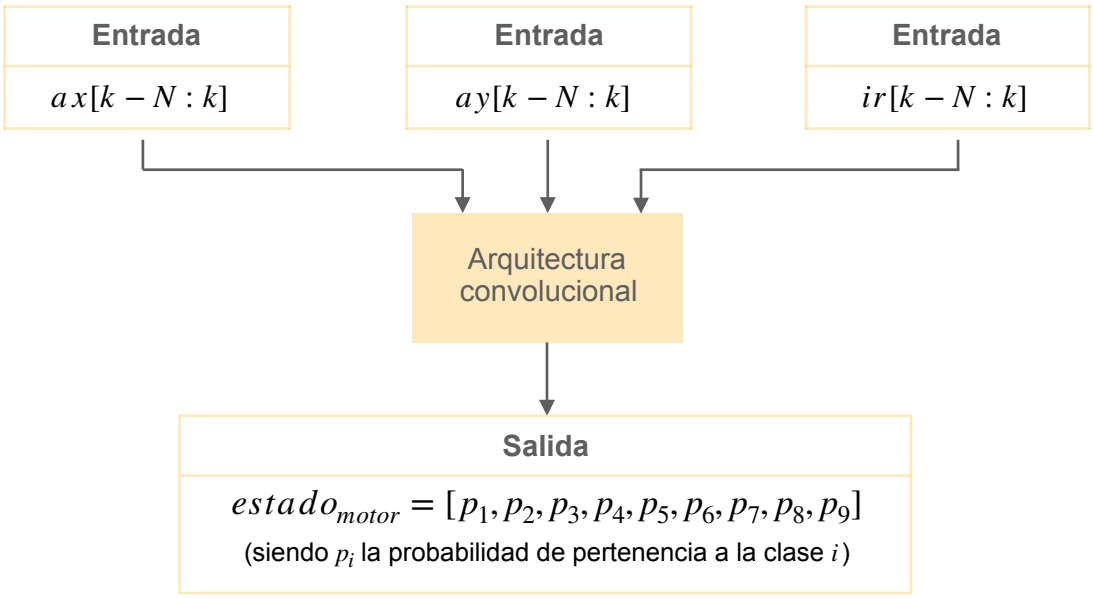

Figura 3.4. Esquema red convolucional para clasificación.

Para establecer la correspondencia entre los tres vectores de entrada y el estado de salida, se propone la arquitectura convolucional expuesta en la Figura 3.5, que consta de tres ramas de entrada, las cuales permiten filtrar de manera independiente las tres variables de trabajo. Cada rama consta de una **capa de entrada** de N neuronas (tamaño del vector de entrada), una **capa de convolución** (con un filtro unidimensional de tamaño 512 elementos), una **capa de *pooling*** (operación *max-pooling*, con una máscara de submuestreo de 8 elementos) y una **capa de *flatten*** para convertir el resultado de la rama en un vector plano. A continuación, se **concatenan** las tres ramas y se añaden tres **capas ocultas** de 12 neuronas cada una. Finalmente, el modelo de red termina con una **capa de salida** de 9 neuronas (número de posibles estados del motor).

Para evitar el **sobreajuste** del modelo, se han tomado varias medidas. En las capas de convolución, se establece una penalización de tipo L2 para los pesos elevados

(con una constante de penalización de 0.001), mientras que en las capas ocultas y de salida, se limita la norma del vector de pesos a un valor máximo de 3. Adicionalmente, se incluye una operación de *batch normalization* tras cada capa oculta.

Finalmente, se utiliza una función de **activación** tipo ReLU para las capas convolucionales y ocultas, y una softmax para la capa de salida.

Figura 3.5. Arquitectura convolucional propuesta para la clasificación del estado del motor.

3.2.2.1. Resultados

La arquitectura expuesta en la sección anterior se ha puesto en práctica sobre el conjunto **dataicann**, utilizando vectores de entrada de tamaño $N = 400$ muestras y sin solapamiento.

Para el **proceso de aprendizaje**, se han escogido valores típicos tanto para el momento $(\eta = 0.9)$ como para la tasa de aprendizaje $(\alpha = 0.01)$. El entrenamiento se ha completado en 100 épocas, con un tamaño del *batch* de 12 muestras y empleando para ello el 70% de los datos de trabajo (el 30% restante se ha utilizado a modo de test).

En la Figura 3.6 se muestra la evolución del comportamiento del modelo a lo largo de las épocas, evaluado sobre los datos de entrenamiento.

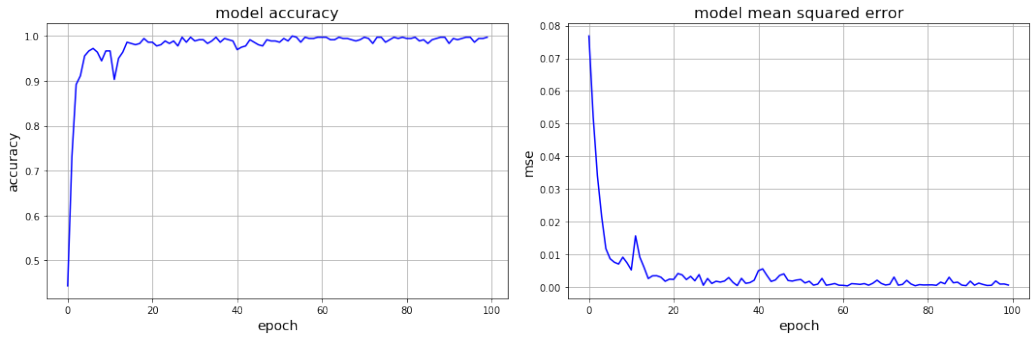

Figura 3.6. Evolución del modelo de clasificación convolucional a lo largo del entrenamiento. A la izquierda se muestra la precisión del modelo y a la derecha el error cuadrático medio.

Como resultado, este modelo es capaz de determinar el estado de funcionamiento del motor con **aproximadamente un 100% de precisión** (Tabla 3.5), destacando especialmente su buen comportamiento ante el conjunto de datos de test, donde se aprecian los beneficios de las medidas tomadas para evitar el sobreajuste. Podemos decir por tanto que la arquitectura convolucional propuesta, en comparación con la red *feedforward*, es capaz de resolver el problema de clasificación con mayor solvencia y en un contexto menos favorable.

	Entrenamiento	Test
Precisión	0.997	1.000
Error cuadrático medio	5.8×10^{-4}	2.2×10^{-8}

Tabla 3.5. Resultados del modelo de clasificación convolucional.

Estos buenos resultados se observan gráficamente en la Figura 3.7, donde se muestra la clasificación de estados del motor para todo el conjunto de datos (entrenamiento

y test). Aquí se exponen: a la izquierda, los estados reales del motor (en verde); a la derecha, los estados estimados por la arquitectura de red (en azul) y el error cometido en la estimación (en rojo).

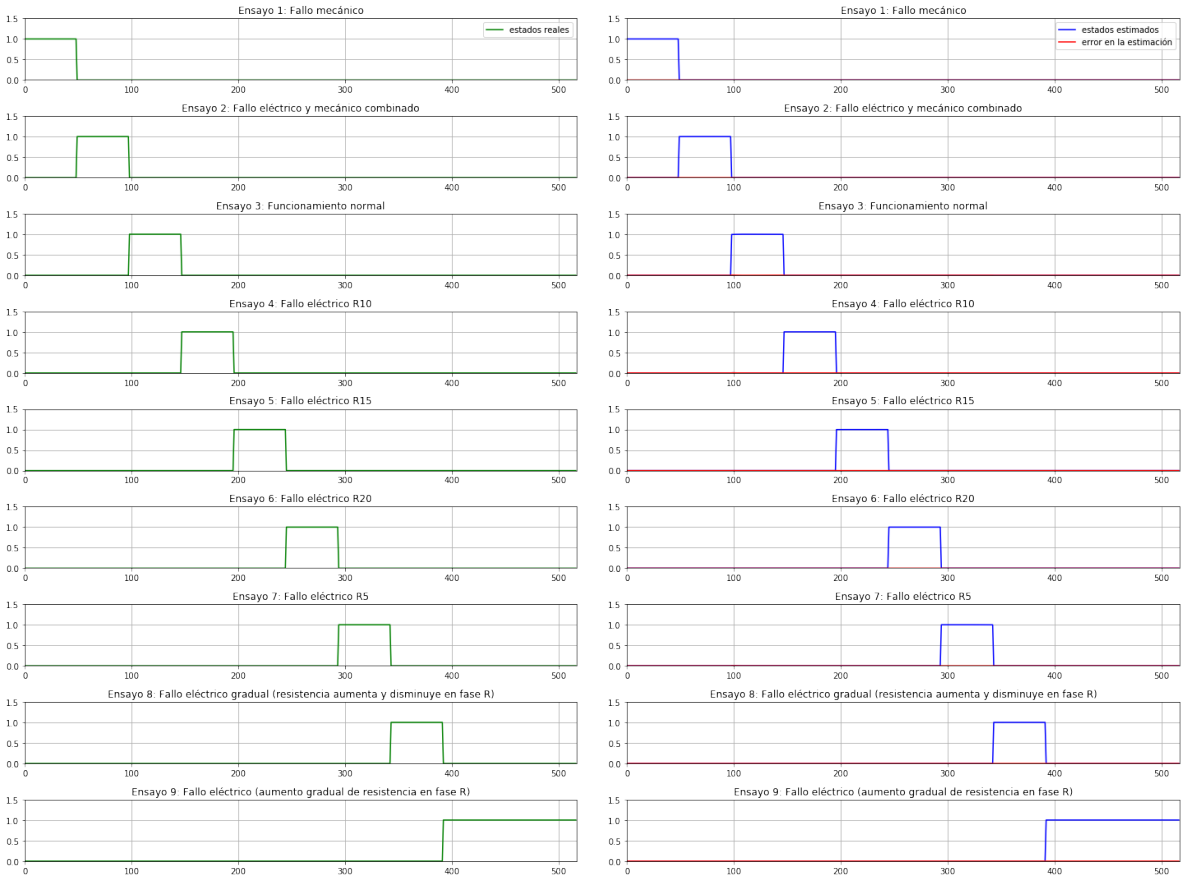

Figura 3.7. Clasificación de estados del motor con arquitectura convolucional.

Adicionalmente, el análisis de los filtros de convolución aprendidos por la red proporciona valiosa información acerca del funcionamiento del motor, como se expone en la sección 3.4.1 de este documento.

3.3. Predicción

En esta sección se propone analizar el conjunto de datos de **consumo energético** (datosConsumos) con el propósito de **predecir la demanda eléctrica** en un instante de tiempo futuro. Para ello se plantean tres alternativas: una red *feedforward*, una red LSTM y una red convolucional.

Las tres arquitecturas reciben como **entrada** un vector de tamaño $[(N-1)+3]$ elementos, que contiene para cada instante k: los **consumos** horarios de las últimas N horas, la **semana** del año (W), el **día** de la semana (D) y la **hora** del día (H). Como **salida**, las arquitecturas devuelven el **consumo previsto** dentro de n horas.

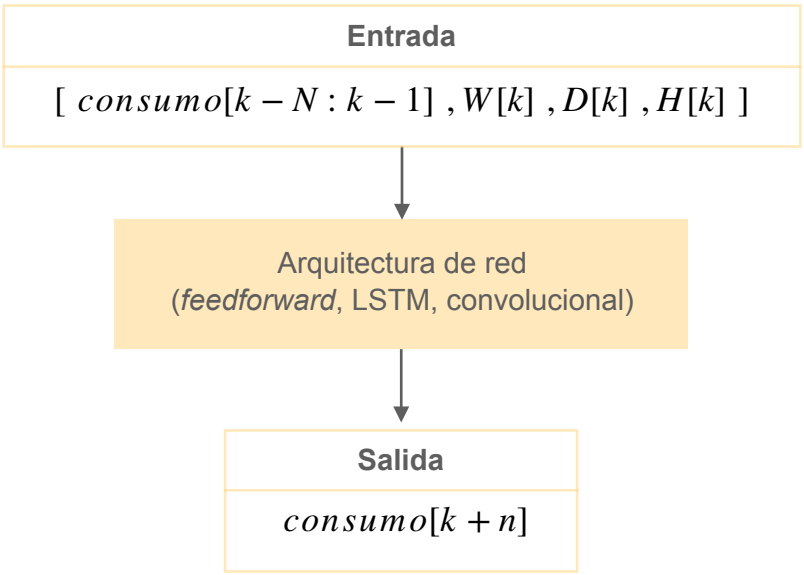

Figura 3.8. Esquema arquitectura de red para predicción.

A continuación se plantean las tres arquitecturas propuestas y los resultados de la predicción para cada una de ellas. Para los parámetros (N, n) se han elegido unos valores de $(N = 48)$ y $(n = 24)$, con lo que se estimará el consumo energético dentro de 24 horas en base a los consumos horarios de los últimos dos días.

3.3.1. Predicción con arquitectura *feedforward*

En primer lugar, se propone resolver el problema de predicción mediante la arquitectura *feedforward* expuesta en la Figura 3.9. Esta arquitectura consta una **capa de entrada** de 50 neuronas (tamaño del vector de entrada) y una **capa de salida** de 1 neurona (predicción de consumo). En cuanto a las **capas ocultas**, se ha optado por una combinación de 7 capas ocultas con 80 neuronas cada una.

Para evitar el **sobreajuste** del modelo, se ha limitado la norma del vector de pesos, en las capas ocultas y de salida, a un valor máximo de 3. Las **activaciones** empleadas son de tipo ReLU para las capas ocultas e identidad para la capa de salida.

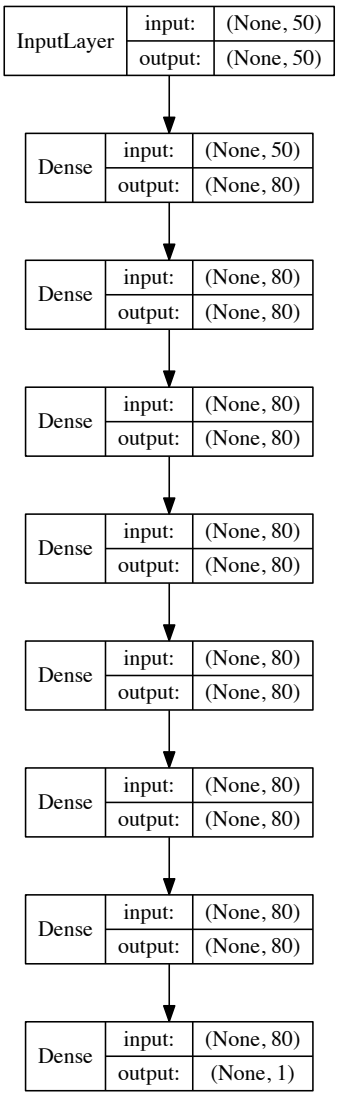

Figura 3.9. Arquitectura *feedforward* propuesta para la predicción de consumo.

3.3.2. Predicción con arquitectura LSTM

Esta segunda arquitectura es una ampliación de la anterior. Como novedad, incorpora una **capa LSTM** entre la capa de entrada y las capas ocultas (Figura 3.10). Dicha capa, al igual que las ocultas, consta de 80 neuronas, utiliza una función de activación tipo ReLU y tiene limitada la norma del vector de pesos a un valor máximo de 3.

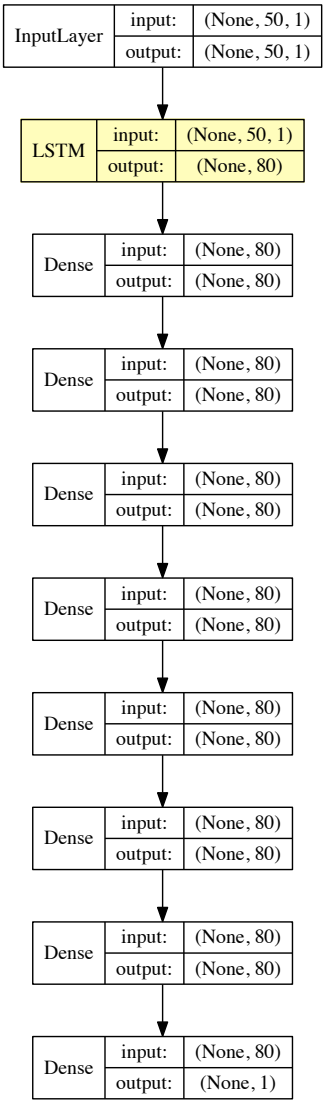

Figura 3.10. Arquitectura LSTM propuesta para la predicción de consumo. Se sombrean en amarillo las novedades introducidas respecto a la arquitectura *feedforward* expuesta en la Figura 3.9.

3.3.3. Predicción con arquitectura convolucional

Por último, se recurre a una red convolucional para llevar a cabo la predicción. Esta propuesta consiste en aplicar una operación de **convolución** sobre el vector de **consumos previos**, en busca de patrones de comportamiento, y replicar a continuación la arquitectura *feedforward* expuesta en la sección 3.3.1.

En la Figura 3.11 se expone la arquitectura de red, que presenta **dos ramas de entrada**:

- A la izquierda se encuentra la rama encargada de realizar la convolución. Consta de una **capa de entrada** de 47 neuronas (vector de consumos previos), una **capa de**

convolución (con un filtro unidimensional de tamaño 47 elementos) y una **capa de *flatten*** que convierte la salida de la rama en un vector plano. Para la capa de convolución se utiliza una función de activación tipo ReLU y se aplica una restricción de no negatividad a sus pesos, lo cual ayuda a evitar el sobreajuste y obliga a que la contribución de los consumos previos sobre la predicción sea siempre positiva.

- A la derecha se observa la rama que recibe el resto de elementos del vector de entrada (semana, día y hora). Consta de una **capa de entrada** de 3 neuronas y su correspondiente **capa de *flatten***.

A continuación, se **concatenan** las dos ramas y se repite la arquitectura de la red *feedforward*.

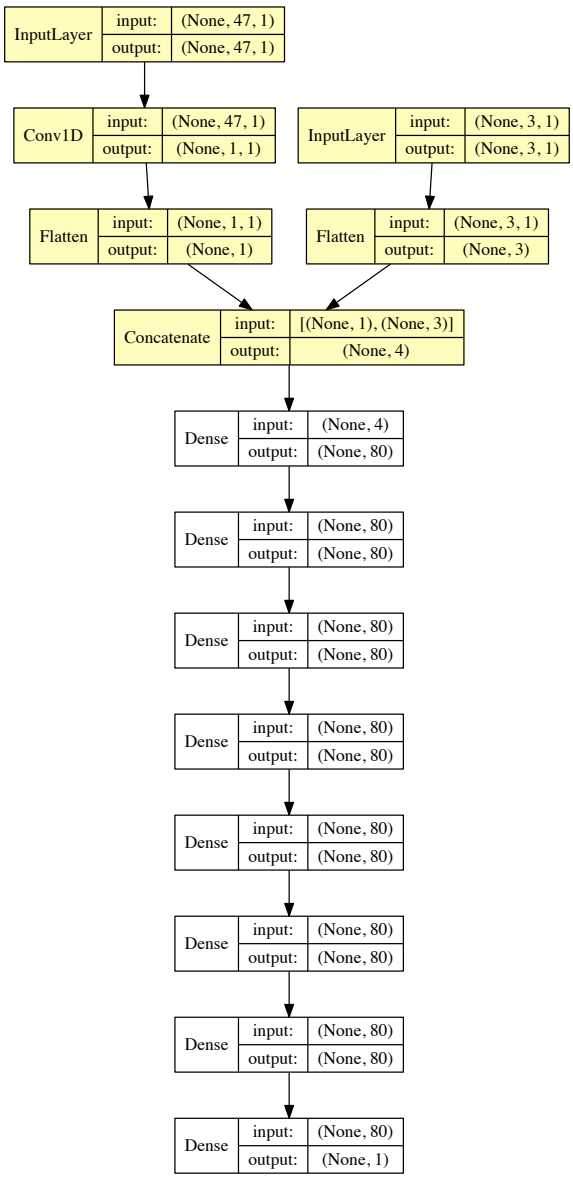

Figura 3.11. Arquitectura convolucional propuesta para la predicción de consumo. Se sombrean en amarillo las novedades introducidas respecto a la arquitectura *feedforward* expuesta en la Figura 3.9.

3.3.4. Resultados

Las arquitecturas descritas se han puesto en práctica sobre el conjunto **datosConsumos**, previamente normalizado. Para el **proceso de aprendizaje**, se ha escogido el optimizador adam, completando el entrenamiento en 200 épocas, con un tamaño del *batch* de 100 muestras y empleando para ello el 70% de los datos de trabajo (el 30% restante se ha utilizado a modo de test).

En la Figura 3.12 se muestra el comportamiento de los tres modelos propuestos a lo largo del entrenamiento.

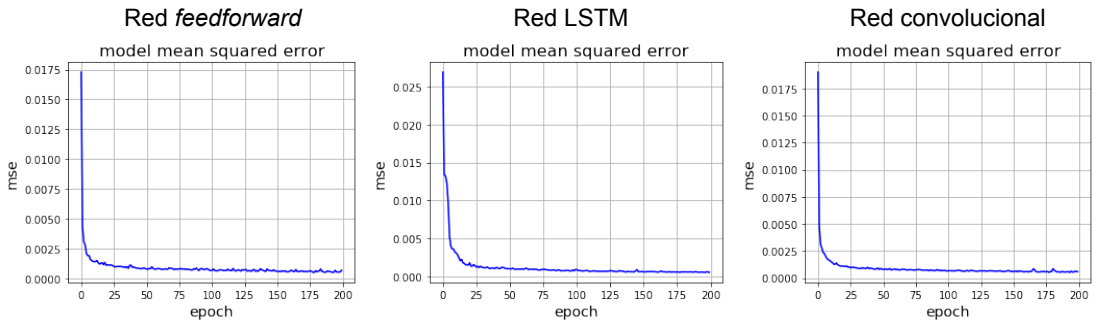

Figura 3.12. Evolución de los modelos de predicción a lo largo del entrenamiento (evaluados en base al error cuadrático medio).

Como se aprecia en la Tabla 3.6, los tres modelos aportan buenos resultados, destacando especialmente la **arquitectura LSTM**. Mientras, la red *feedforward* es la que ofrece peores estimaciones y tiene menor capacidad de generalización.

Adicionalmente, se incluyen en esta misma tabla los resultados de la predicción empleando otras técnicas de análisis, como son la predicción lineal o la predicción con vectores de soporte (*Support Vector Regression*, SVR). Esta última técnica demuestra un gran comportamiento pero, aún así, no es capaz de mejorar las estimaciones aportadas por la arquitectura LSTM.

	Error cuadrático medio	
	Entrenamiento	Test
Red *feedforward*	6.88×10^{-4}	9.04×10^{-4}
Red LSTM	5.55×10^{-4}	5.99×10^{-4}
Red convolucional	5.90×10^{-4}	6.00×10^{-4}
Modelo lineal (Ridge Regression)	29.83×10^{-4}	31.51×10^{-4}
SVR (Kernel rbf)	5.71×10^{-4}	7.58×10^{-4}

Tabla 3.6. Resultados de los modelos de predicción ante los dos conjuntos de datos (entrenamiento y test). Para generar los resultados del modelo lineal y de la regresión SVR, se ha recurrido a la librería gratuita scikit-learn (http://scikit-learn.org/stable/#).

En la Figura 3.13 se muestra el comportamiento de la arquitectura LSTM, para un tramo de datos de un mes. Aquí se observan: la demanda real (en verde), la demanda estimada (en azul) y el error cometido en la estimación (en rojo).

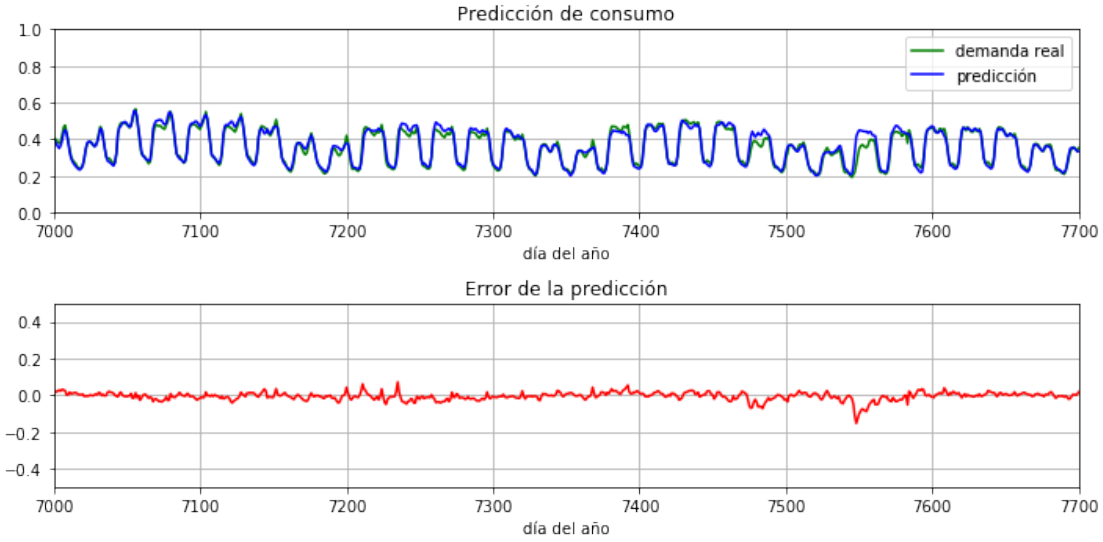

Figura 3.13. Predicción de consumo con arquitectura LSTM.

A la vista de los resultados expuestos, cualquiera de las redes propuestas sería capaz de **abordar con éxito la tarea de predicción**, permitiendo realizar estimaciones de futuro en cuanto al comportamiento de procesos y sistemas de ingeniería. Adicionalmente, el análisis de la máscara de convolución aprendida por la red permite extraer información adicional acerca del sistema, como se expone en la sección 3.4.2 de este documento.

3.4. Extracción de características relevantes

Dado que las redes profundas tienen la particularidad de aprender las fases de extracción de características, se propone en esta sección el **estudio de las características aprendidas por las redes**, en busca de **información desconocida** acerca de los procesos.

En concreto, se plantea el análisis de las máscaras y filtros aprendidos por las **capas de convolución**, que contienen gran cantidad de información y son fácilmente interpretables. Para ello se proponen dos casos de estudio: el análisis de los filtros utilizados en la **clasificación** de estados del motor de inducción y el análisis de la máscara empleada para la **predicción** de consumo energético.

3.4.1. Filtros de la arquitectura de clasificación

En la sección 3.2.2 se proponía una red convolucional capaz de determinar el estado de funcionamiento de un motor de inducción. Esta arquitectura constaba de tres

ramas de entrada, en cada una de las cuales se incluía una capa de convolución, como se señala en la Figura 3.14.

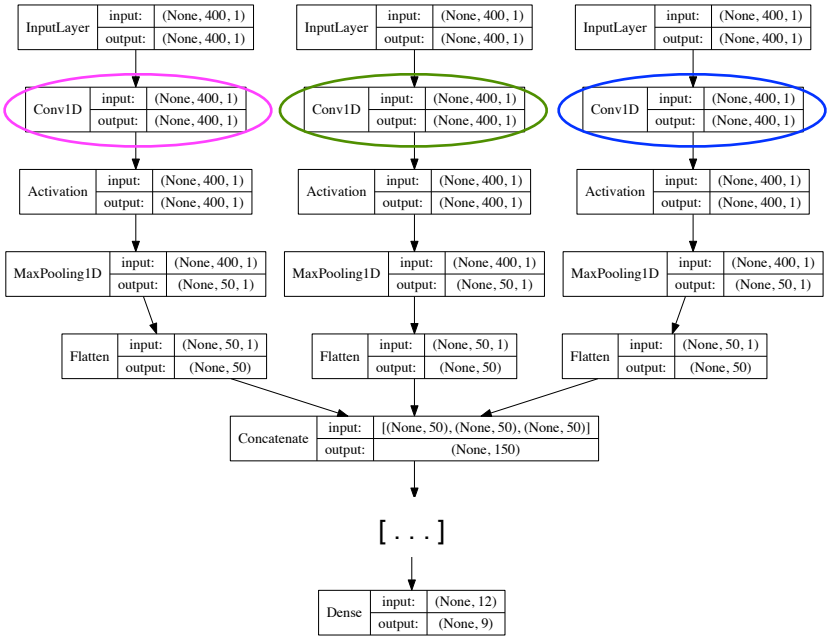

Figura 3.14. Reproducción parcial de la arquitectura de clasificación convolucional (la arquitectura completa se muestra en la Figura 3.5). Se señalan en color las capas de convolución, en relación con los filtros presentes en la Figura 3.15.

Finalizado el aprendizaje, cada capa de convolución queda definida por un vector de pesos, que es equivalente al vector de coeficientes del filtro de convolución. La representación en frecuencia de estos coeficientes (Tabla 3.7) permite reconstruir los filtros aprendidos por la red y extraer de ellos información relevante acerca del proceso, como se expone a continuación.

Representación de los filtros de convolución

Parámetros conocidos

```
N - Número de neuronas (N=400)
fm - Frecuencia de muestreo (fm=5000Hz)
coef - Vector de coeficientes del filtro
```

```
01: DEFINIR vector de frecuencias x:
    Generamos un vector de tamaño N/2 y paso fm/N
    x = arange(N)*(fm/N)

02: DEFINIR vector de amplitudes y:
    Calculamos la FFT (Fast Fourier Transform) del vector coef
    y = abs(fft(coef,N/2))

03: REPRESENTAR el filtro:
    Ploteamos el conjunto de puntos definidos por los vectores x, y
    plot(x,y)
```

Tabla 3.7. Representación en frecuencia de los filtros de convolución a partir de sus coeficientes.

3.4.1.1. Resultados

De acuerdo al planteamiento expuesto en la Tabla 3.7, se representan a continuación los filtros de convolución aprendidos por la red, donde cada uno está ligado a una variable diferente: ax (aceleración horizontal), ay (aceleración vertical) e ir (corriente en fase R).

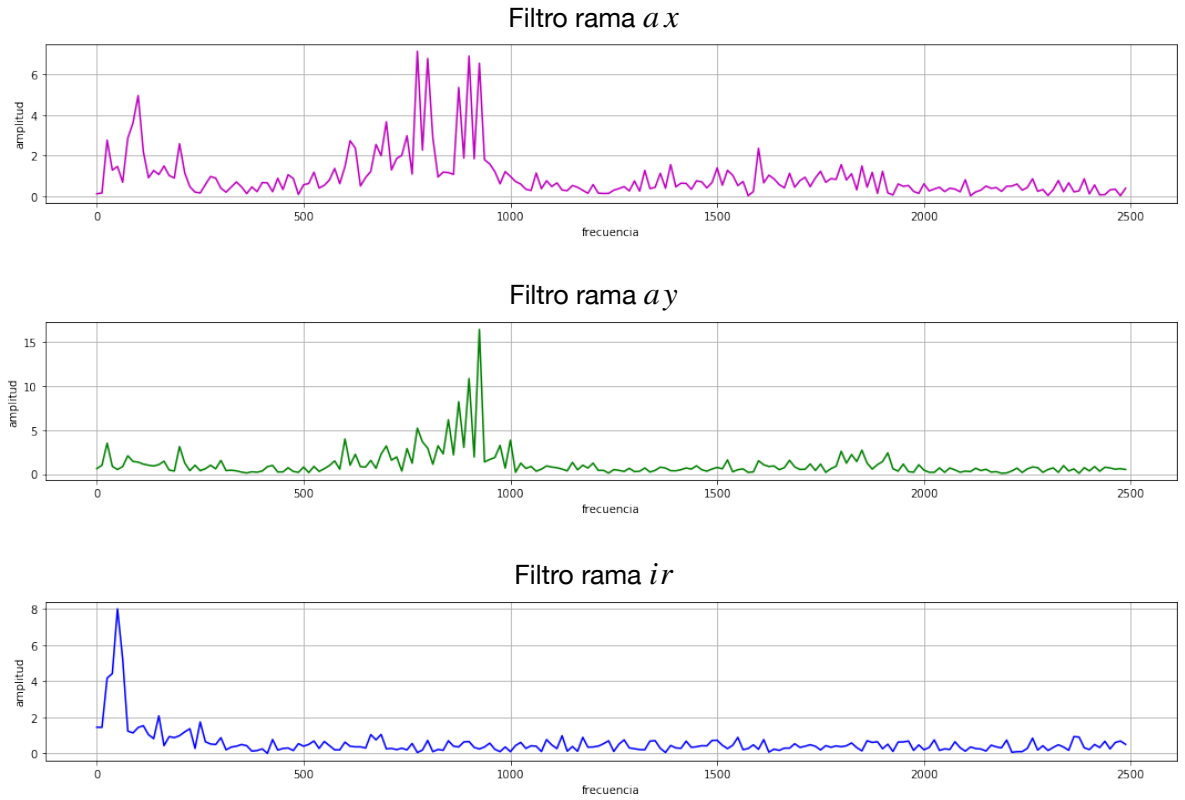

Figura 3.15. Filtros de convolución aprendidos por la red.

Estos filtros recogen aquellas frecuencias que resultan **relevantes** para llevar a cabo la **tarea de clasificación** y que, por tanto, contienen una enorme riqueza de información acerca del proceso bajo estudio (entre ellas se encuentran, por ejemplo, la velocidad de giro del motor o las frecuencias de paso del rodamiento).

A continuación se detallan distintos parámetros de funcionamiento del motor, así como constructivos, extraídos a partir del análisis de estos filtros de convolución.

- **Filtro rama ir**

La componente fundamental de este filtro permite identificar la **frecuencia de alimentación** del motor, situada a **50Hz** (frecuencia de red). También se aprecian sus armónicos impares a 150Hz y 250Hz.

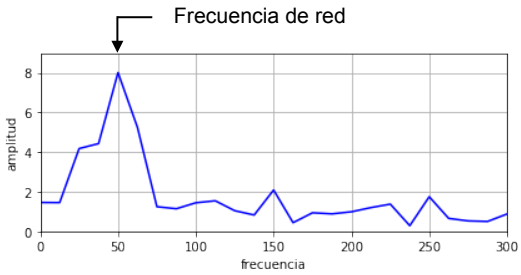

Figura 3.16. Frecuencia de red, presente en el filtro de la rama ir.

- **Filtro rama ax**

El filtro de la rama ax permite identificar la **velocidad de giro** del motor (RPM) y la **frecuencia de paso en la pista interior del rodamiento** (BPFI), gracias a las correspondencias observadas entre este filtro y el comportamiento típico de un motor ante fallo en la pista interior (Figura 3.17).

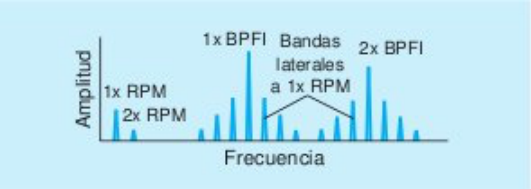

Figura 3.17. Espectro de frecuencia en caso de fallo en la pista interior. (Fuente: Imagen extraída de http://www.sinais.es/curso-vibraciones.html)

Dichas similitudes tienen lugar en dos bandas de frecuencia:

- A baja frecuencia (Figura 3.18), se detecta la presencia de la **velocidad de giro (RPM)**, que toma un valor de **25Hz** (equivalente a 1500rpm).

- A alta frecuencia (Figura 3.19), se aprecian dos modulaciones (en 775 y 900Hz) separadas una distancia **BPFI** de **125Hz** y con bandas laterales situadas a la velocidad de giro.

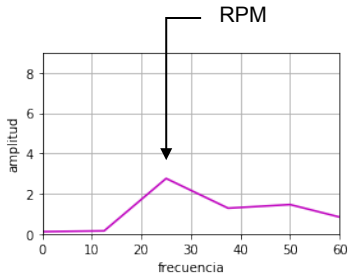

Figura 3.18. Frecuencia de giro, presente en el filtro de la rama ax.

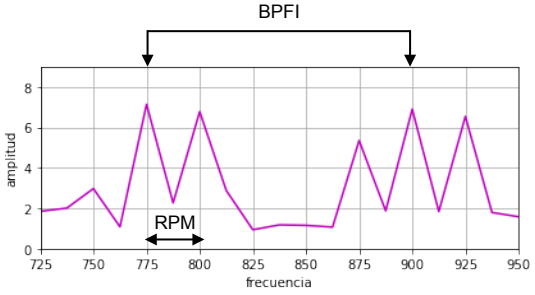

Figura 3.19. Frecuencia de giro y frecuencia de paso en la pista interior, presentes en el filtro de la rama ax.

- **Filtro rama ay**

En tercer lugar, se detectan similitudes entre el comportamiento ante fallo en la pista exterior (Figura 3.20) y el filtro de la rama ay (Figura 3.21). Aquí se aprecia de nuevo la presencia de la velocidad de giro (25Hz) y, también, la aparición de la **frecuencia de paso en la pista exterior del rodamiento** (BPFO), que toma un valor de **75Hz**.

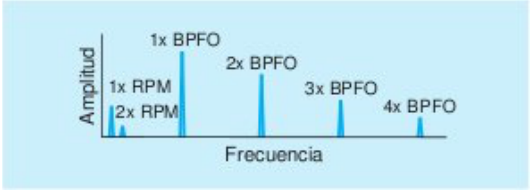

Figura 3.20. Espectro de frecuencia en caso de fallo en la pista exterior. (Fuente: Imagen extraída de http://www.sinais.es/curso-vibraciones.html)

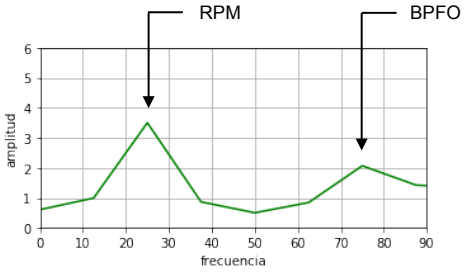

Figura 3.21. Frecuencia de giro y frecuencia de paso en la pista exterior, presentes en el filtro de la rama ay.

- **Parámetros constructivos**

A partir de las frecuencias detectadas en los filtros de convolución (RPM, $BPFI$, $BPFO$) es posible determinar ciertos parámetros constructivos del motor, como el **número de bolas del rodamiento** N_b o la **relación entre el diámetro de las bolas y el diámetro del rodamiento** d/D. Para ello, se recurre a las expresiones 3.1, 3.2, 3.3 y 3.4, ampliamente utilizadas en el análisis de vibraciones en máquinas [54].

$$BPFO = 0.4 \cdot N_b \cdot RPM \tag{3.1}$$

$$BPFI = 0.6 \cdot N_b \cdot RPM \tag{3.2}$$

$$BPFO = 0.5 \cdot N_b \cdot RPM \cdot \left[1 - \frac{d}{D}\right] \tag{3.3}$$

$$BPFI = 0.5 \cdot N_b \cdot RPM \cdot \left[1 + \frac{d}{D}\right] \tag{3.4}$$

Por un lado, las fórmulas experimentales 3.1 y 3.2, relacionan las frecuencias de paso exterior e interior con el parámetro N_b. Despejando dicha incógnita tenemos que, de acuerdo a ambas expresiones, los rodamientos del motor tienen un total de **ocho bolas**.

$$N_b = \frac{BPFO}{0.4 \cdot RPM} = \frac{75Hz}{0.4 \cdot 25Hz} = 7.5 \approx 8 \tag{3.5}$$

$$N_b = \frac{BPFI}{0.6 \cdot RPM} = \frac{125Hz}{0.6 \cdot 25Hz} = 8.3 \approx 8 \tag{3.6}$$

Por el otro lado, en las ecuaciones 3.3 y 3.4, se relacionan las frecuencias de paso exterior e interior con el parámetro d/D. Despejando dicha incógnita tenemos que, en los dos casos, la **relación de proporción** entre el diámetro de las bolas y el diámetro del rodamiento es de **0.25**.

$$\frac{d}{D} = -\left[\frac{BPFO}{0.5 \cdot N_b \cdot RPM} - 1\right] = -\left[\frac{75Hz}{0.5 \cdot 8 \cdot 25Hz} - 1\right] = 0.25 \tag{3.7}$$

$$\frac{d}{D} = \left[\frac{BPFI}{0.5 \cdot N_b \cdot RPM} - 1\right] = \left[\frac{125Hz}{0.5 \cdot 8 \cdot 25Hz} - 1\right] = 0.25 \tag{3.8}$$

Finalmente se comprueba, sobre el motor objeto de estudio, que tanto el número de bolas N_b como la relación de proporción d/D se corresponden con los parámetros reales.

Figura 3.22. Motor sometido a los ensayos dataicann.

- **Comparativa**

Como cierre de esta sección, se expone la información extraída a partir de los filtros de convolución, estableciendo la comparativa con los parámetros presentes en el *dataset* de trabajo y los extraídos del propio motor.

		Información acerca del motor					
		Frecuencia de alimentación	RPM	BPFI	BPFO	N_b	d/D
Fuente de información	Filtros de convolución	50 Hz	1500 rpm	125 Hz	75 Hz	8 bolas	0.25
	Dataset	-	1500 rpm	-	-	-	-
	Motor	-	-	-	-	8 bolas	0.25

Tabla 3.8. Resultados de la extracción de características para el modelo de clasificación convolucional.

A la vista de la Tabla 3.8, se puede afirmar que el análisis de los filtros de convolución **aporta valiosa información** acerca del proceso bajo estudio. En este caso, permite determinar tanto **parámetros de funcionamiento** como **parámetros constructivos** del motor que inicialmente eran desconocidos, con una mínima desviación respecto a los parámetros reales.

3.4.2. Máscara de la arquitectura de predicción

Para abordar la tarea de predicción, se proponía en la sección 3.3.3 una red convolucional de dos ramas de entrada. En una de ellas, la rama encargada de tratar los datos de consumos previos, se incluía una capa de convolución, como se aprecia en la Figura 3.23.

Finalizado el entrenamiento de la red, esta capa de convolución queda definida por un vector de pesos, el cual define la contribución de los consumos previos sobre el consumo futuro, como se representa en la Figura 3.24.

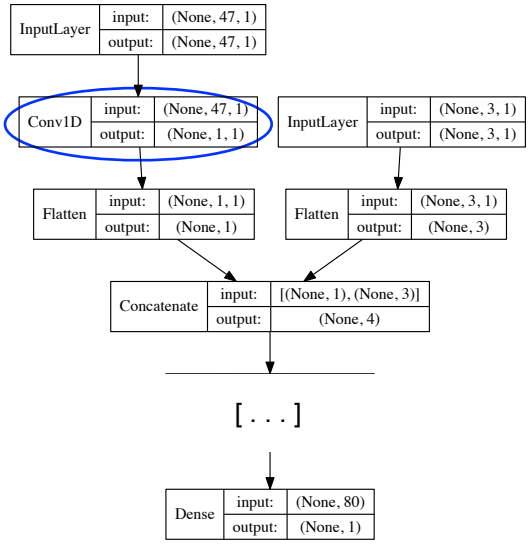

Figura 3.23. Reproducción parcial de la arquitectura de predicción convolucional (la arquitectura completa se muestra en la Figura 3.11).

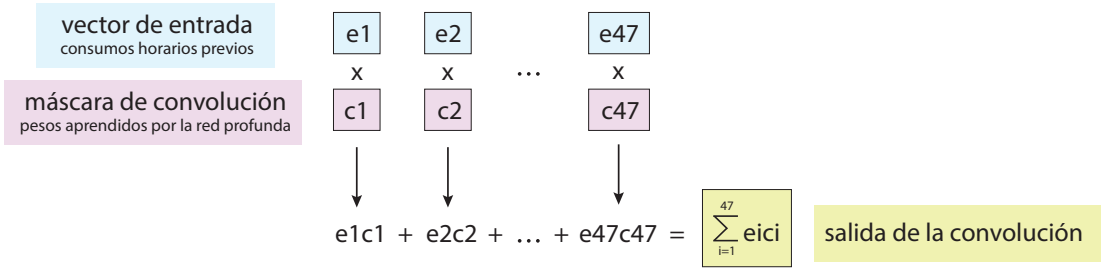

Figura 3.24. Operación de convolución sobre los consumos previos.

Aquí se aprecia como la capa de convolución transforma el vector de entrada de 47 elementos en un único valor de salida. Para ello, convoluciona la máscara de pesos aprendida durante el entrenamiento sobre el vector de entrada, estableciendo una correspondencia uno a uno entre sus elementos. Como resultado, el vector de pesos pondera la influencia de los consumos previos sobre el consumo futuro, es decir, pone en evidencia qué horas son más y menos relevantes para la predicción de consumo.

3.4.2.1. Resultados

En la Figura 3.25 se muestra la máscara de convolución aprendida por la red, representando los pesos en escala de grises.

En ella se observa, en primer lugar, que los **pesos de la convolución** son siempre **nulos o positivos**, lo cual entra dentro de lo esperado pues, como se menciona en la sección 3.3.3, se ha impuesto una restricción de no negatividad sobre el vector de pesos.

En segundo lugar, se aprecian las **horas relevantes** para la predicción de consumo, que son aquellas asociadas a los pesos más elevados (en la figura, menor nivel de gris). Se concentran, principalmente, en **tres periodos de tiempo**: consumo en la última hora, consumo hace 24 horas y consumo hace 48 horas.

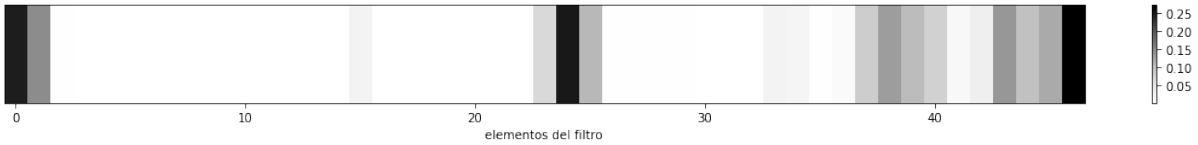

Figura 3.25. Máscara de convolución de la arquitectura de predicción.

Por tanto, podemos afirmar que la evolución de consumo a lo largo de los últimos dos días resulta poco relevante para la predicción. En su lugar, prevalecen tan solo los consumos puntuales de los días previos que hayan tenido lugar a la misma hora para la cual se pretende realizar la estimación.

De esta manera, el análisis de la máscara de convolución permite **profundizar en la comprensión del proceso** bajo estudio y **detectar correspondencias** no evidentes en los datos.

3.5. Visualización de datos del proceso

En último lugar, se aborda el estudio de procesos industriales con un enfoque de analítica visual. En este contexto, se propone trabajar de nuevo sobre los datos de corriente y aceleración del **motor de inducción** (dataicann), con el fin de **proyectar** este espacio de datos sobre un **espacio de baja dimensión 2D**, elaborando así un "mapa" visual de los estados del proceso que permita su interpretación y visualización.

Para llevar a cabo esta reducción de la dimensión, se propone recurrir a un d*eep convolutional autoencoder*. Esta arquitectura, por su estructura de tipo ***autoencoder***, permite incluir un cuello de botella 2D sobre el que proyectar los datos de trabajo. Al mismo tiempo, gracias a su naturaleza **convolucional**, no requiere ningún tipo de información frecuencial acerca del motor para llevar a cabo la proyección. Por tanto, se trata de una red profunda capaz de encontrar las frecuencias relevantes y en base a ellas generar la mejor representación de los datos.

3.5.1. *Deep convolutional autoencoder*

Esta arquitectura recibe como **entrada**, y devuelve como **salida**, **tres vectores** con los valores históricos de las variables bajo estudio (vectores de N muestras, tomadas desde el instante actual k).

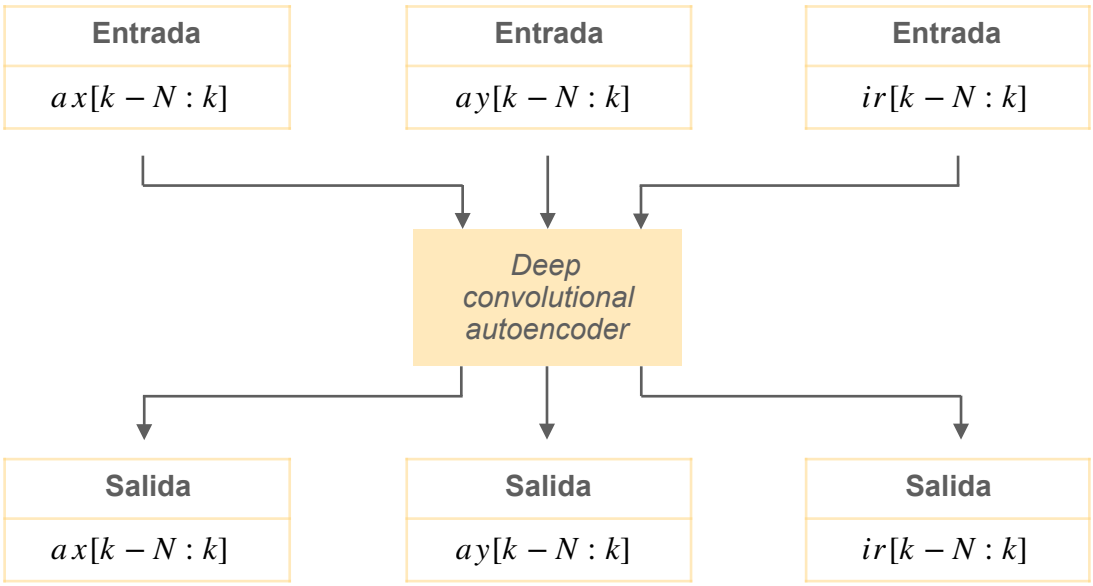

Figura 3.26. Esquema del *deep convolutional autoencoder* para reducción de la dimensión.

Para establecer esta correspondencia, se propone la arquitectura profunda expuesta en la Figura 3.27, que consta de tres ramas de entrada y salida, las cuales

permiten filtrar de manera independiente las tres variables de trabajo. En esta figura se observa el cuello de botella (señalado en verde) y los dos bloques que componen el *autoencoder*: el *encoder*, que proyecta el espacio de entrada sobre el cuello de botella; y el *decoder*, que a partir del cuello de botella reconstruye el espacio de salida.

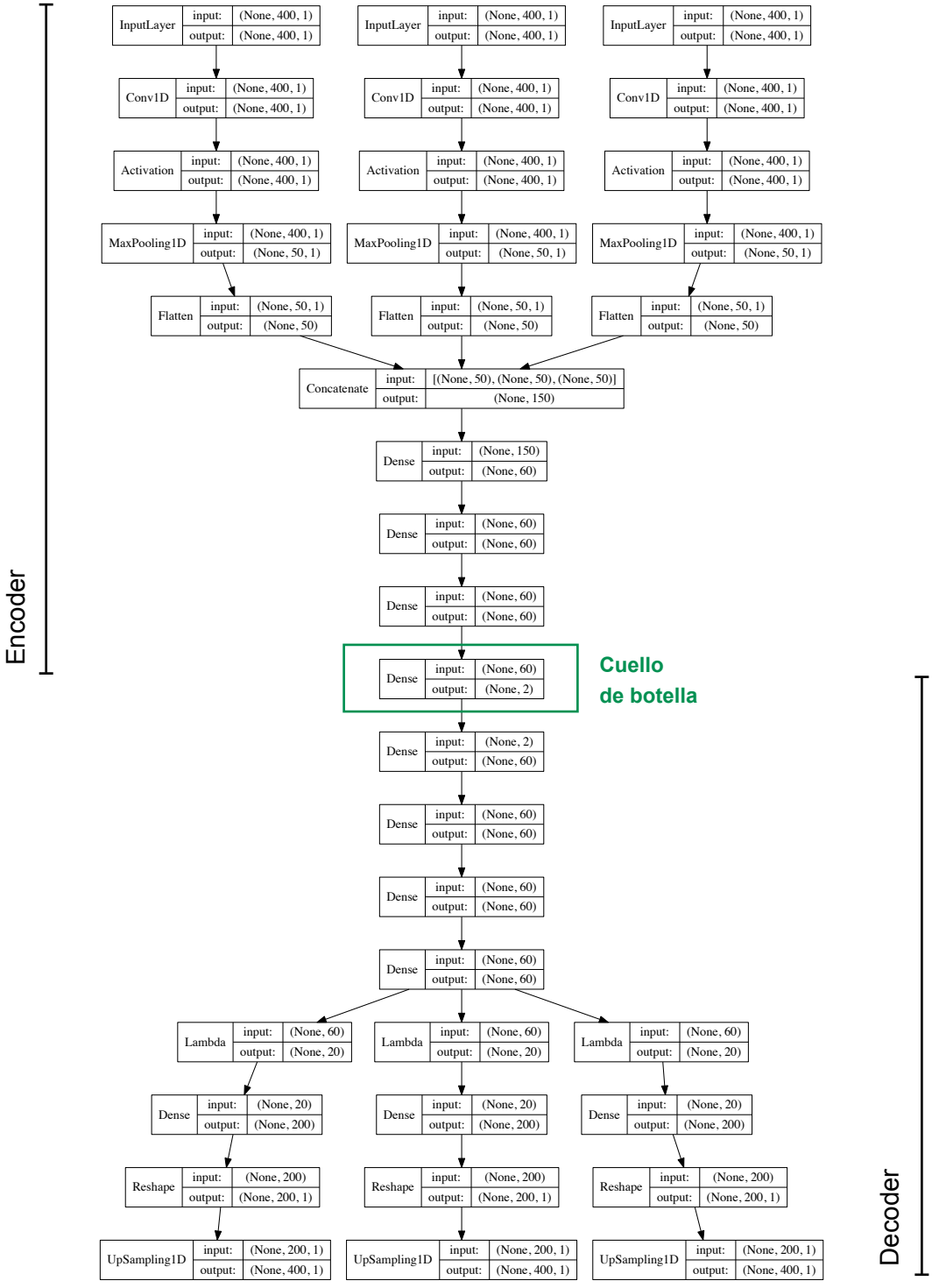

Figura 3.27. Arquitectura propuesta para la reducción de la dimensión.

El *encoder* consta de tres ramas, cada una de ellas con una **capa de entrada** de N neuronas (tamaño del vector de entrada), una **capa de convolución** (con un filtro unidimensional de tamaño 512 elementos), una **capa de *pooling*** (operación *max-pooling*, con una máscara de submuestreo de 8 elementos) y una **capa de *flatten*** para convertir el resultado de la rama en un vector plano. A continuación, se **concatenan** las tres ramas y se añaden tres **capas ocultas** de 60 neuronas cada una. Finalmente, el *encoder* termina con un **cuello de botella** de 2 neuronas (espacio de dos dimensiones).

Tras el cuello de botella, el *decoder* comienza con cuatro **capas ocultas** de 60 neuronas cada una. La última de ellas se subdivide en tres bloques de neuronas (capas Lambda), creando así las tres ramas de salida. Cada rama consta de una **capa oculta** de 200 neuronas, una **capa de *reshape*** para recuperar las dimensiones de entrada y una **capa de *upsampling*** para obtener a la salida las N neuronas de entrada.

Para evitar el **sobreajuste** del modelo, se han tomado varias medidas. En las capas de convolución, se establece una penalización de tipo L2 para los pesos elevados (con una constante de penalización de 0.001), mientras que en las capas ocultas se limita la norma del vector de pesos a un valor máximo de 3.

Finalmente, se utiliza una función de **activación** tipo ReLU para las capas convolucionales y ocultas, con excepción de las últimas capas del *decoder* (las capas ocultas de las ramas de salida y la capa oculta que da lugar a estas ramas) para las que se recurre a una función identidad.

3.5.1.1. Resultados

La arquitectura expuesta en la sección anterior se ha puesto en práctica sobre el conjunto **dataicann**, utilizando vectores de entrada de tamaño $N = 400$ muestras y sin solapamiento. Para el **proceso de aprendizaje**, se ha escogido el optimizador adam, completando el entrenamiento en 100 épocas, con un tamaño del *batch* de 64 muestras y empleando para ello el 70% de los datos de trabajo (el 30% restante se ha utilizado a modo de test).

En la Tabla 3.9 se muestran los resultados aportados por el *autoencoder* ante los dos conjuntos de trabajo (entrenamiento y test).

		Rama ax	Rama ay	Rama ir
Error cuadrático medio	Entrenamiento	5.04×10^{-2}	6.37×10^{-2}	2.99×10^{-3}
	Test	5.72×10^{-2}	7.35×10^{-2}	2.77×10^{-3}

Tabla 3.9. Resultados del *autoencoder* convolucional evaluados de acuerdo al error cuadrático medio.

Al mismo tiempo, se expone el comportamiento del modelo en las siguientes figuras. A la derecha (Figura 3.28), se muestra el conjunto de datos de trabajo, de tres dimensiones, donde se entremezclan los nueve ensayos del motor. Debajo, en la Figura 3.29, se expone la reducción de la dimensión aprendida por la red profunda, donde se observa la proyección de los datos de entrada sobre el espacio 2D del cuello de botella.

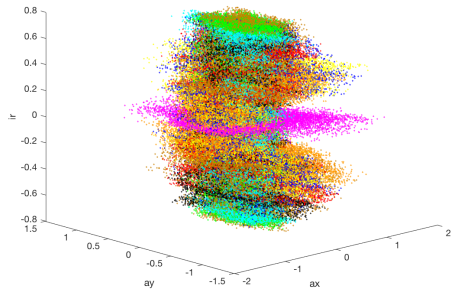

Figura 3.28. Conjunto de datos de trabajo (se asigna un color diferente a cada uno de los nueve ensayos del motor).

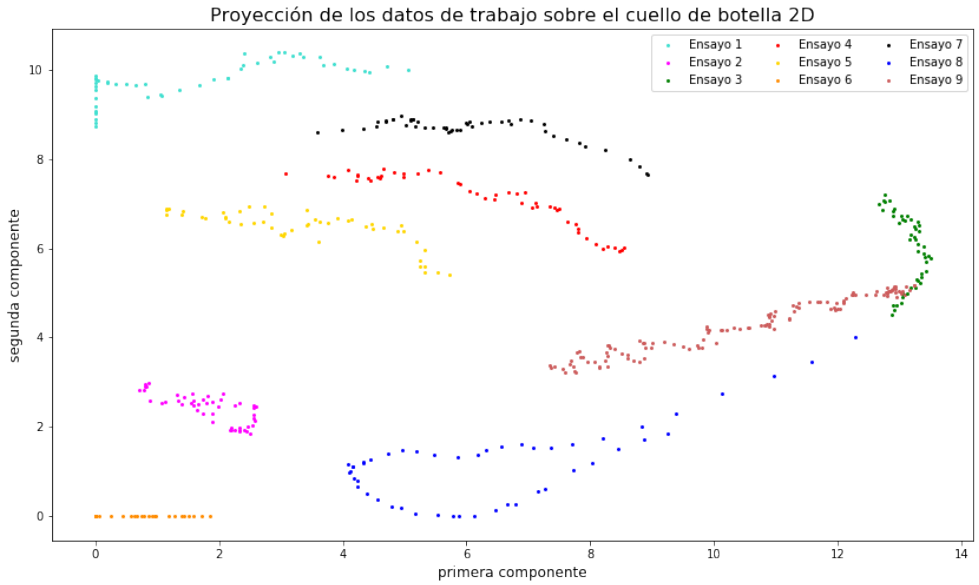

Figura 3.29. Reducción de la dimensión aportada por la arquitectura profunda.

En esta proyección, **cada uno de los ensayos** ocupa una **región diferente** del espacio, resultando fácilmente identificables. Se obtiene así un mapa de estados que permite monitorizar gráficamente el comportamiento del motor. Por tanto, se puede afirmar que el *autoencoder* propuesto es capaz de **alcanzar satisfactoriamente el objetivo** de partida.

- **Correspondencia con la arquitectura convolucional de clasificación**

Adicionalmente, se propone enriquecer el mapa de estados proporcionado por el *autoencoder*, utilizando para ello la correspondencia existente entre esta arquitectura y la red de clasificación convolucional expuesta en la sección 3.2.2.

Para ello, en primer lugar, se despliega una retícula de puntos sobre el cuello de botella, como se expone en la Figura 3.30.

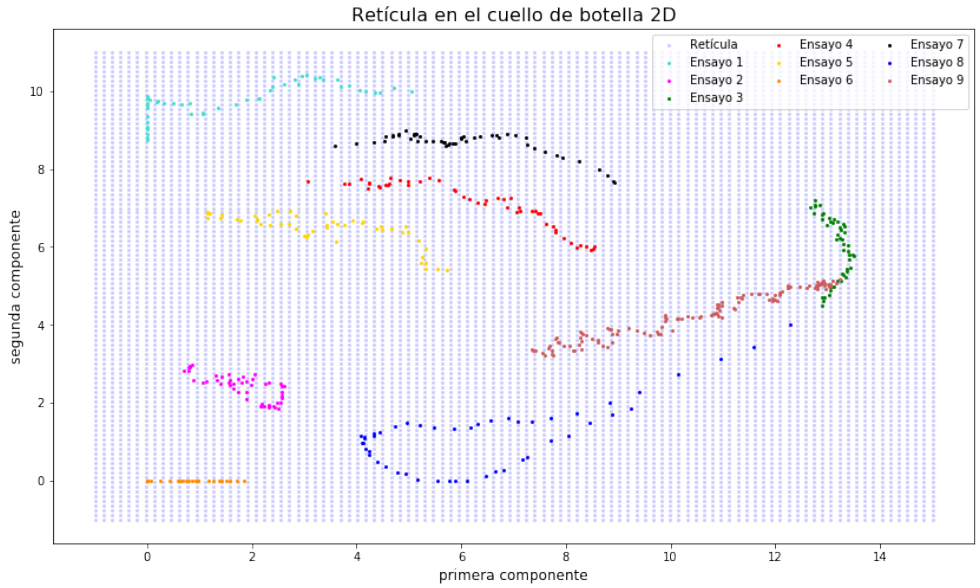

Figura 3.30. Retícula de puntos sobre el cuello de botella.

A continuación, utilizando la transformación aprendida por el *decoder*, se reconstruye la retícula sobre el espacio 3D de partida (Figura 3.31). Como resultado, se obtiene una superficie tridimensional cuyos puntos pueden ser clasificados de acuerdo a la arquitectura convolucional de clasificación. Siguiendo esta línea, se muestra de nuevo la retícula en la Figura 3.32, asignando un color diferente a cada uno de los puntos, de acuerdo al estado determinado por la arquitectura de clasificación.

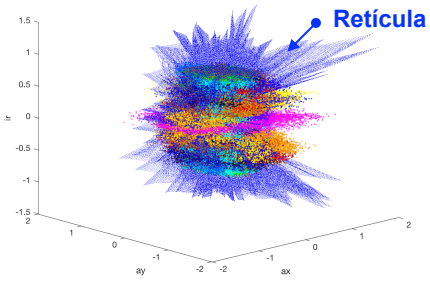

Figura 3.31. Retícula aprendida por el *deep convolutional autoencoder*.

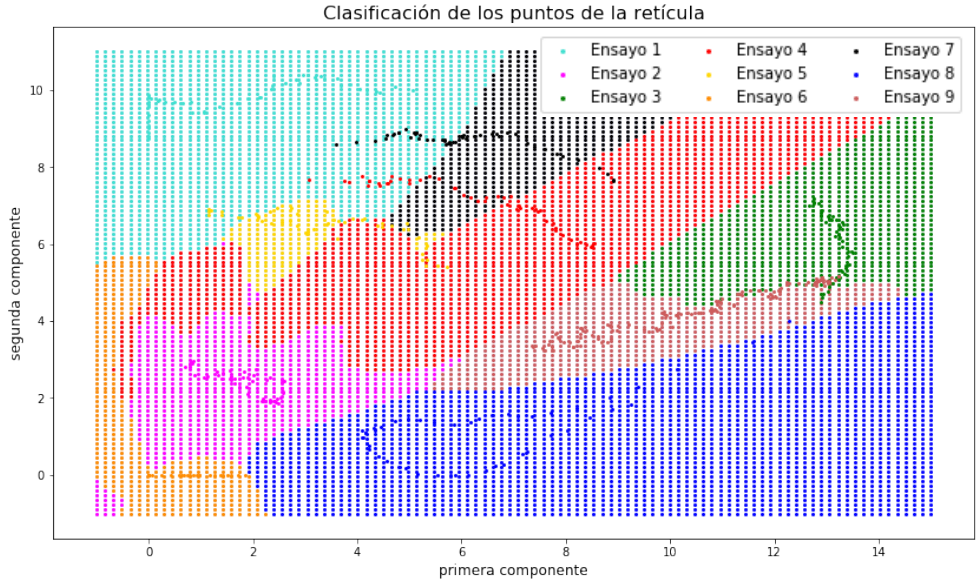

Figura 3.32. Clasificación de los puntos de la retícula.

Como resultado, se obtiene un **mapa de estados más completo** que el de partida. Adicionalmente, se visualiza el equilibrio entre la representación y la clasificación de estados, que se manifiesta en la **correspondencia entre las regiones** definidas por ambas arquitecturas.

No obstante, se aprecia que para tres de los ensayos (números 4, 5 y 7), las trayectorias representadas por el *autoencoder* no se encuentran, en su totalidad, dentro de las regiones definidas por la arquitectura de clasificación, lo cual refleja el margen de mejora de ambas redes profundas. Ante esta situación y dados los excelentes resultados obtenidos en la tarea de clasificación (sección 3.2.2.1), se concluye que el *autoencoder* no es capaz de reconstruir los datos con la suficiente precisión como para equiparar su comportamiento al de la arquitectura de clasificación convolucional.

4
Conclusiones

En este capítulo final, se exponen las conclusiones obtenidas, las aportaciones del trabajo desarrollado y, por último, posibles líneas de futuro trabajo.

4.1. Discusión general

A lo largo de los últimos años, las **técnicas de aprendizaje profundo** han demostrado su éxito en gran cantidad de aplicaciones, aportando resultados muy superiores a los obtenidos hasta el momento con otras técnicas tradicionales de análisis. El *deep learning* se ha convertido así en **una de las ramas más destacadas de la Inteligencia Artificial**. Sin embargo, su uso **no ha llegado aún al sector industrial**, donde constituye un interesante campo de estudio todavía por explotar. Ante esta situación y en el contexto del presente Trabajo Fin de Máster, se han investigado las posibilidades de aplicación de las **técnicas de *deep learning* en el análisis y mejora de la eficiencia de procesos industriales**.

De forma más concreta, esta investigación se ha centrado en resolver cuatro tipos de problemas de ámbito industrial: la **clasificación** de estados, la **predicción** de comportamiento, la **extracción de características** y, por último, la **visualización** de datos de un proceso. Para ello se han manejado dos contextos de trabajo: el análisis de funcionamiento de un motor de inducción y el análisis de la demanda energética en Gran Bretaña.

La **clasificación** de estados se ha puesto en práctica sobre el motor de inducción y se ha resuelto por medio de dos arquitecturas distintas: una red *feedforward* y una red convolucional. Ambas arquitecturas han aportado buenos resultados, destacando especialmente la red convolucional, que es capaz de determinar el estado de

funcionamiento del motor analizado con una tasa de aproximadamente el 100% de acierto. Adicionalmente, el análisis de los filtros de convolución aprendidos por la red ha permitido **extraer características desconocidas** acerca del motor, como su velocidad de giro o el número de bolas de los rodamientos.

Para la **predicción** de comportamiento se ha recurrido a la escena de consumo energético, tratando de predecir la demanda prevista para un instante de tiempo futuro. De nuevo, se han planteado varias alternativas para resolver este problema: una red *feedforward*, una red convolucional y una red LSTM. Las tres arquitecturas han aportado buenas estimaciones, aunque el mejor comportamiento ha sido el de la red LSTM, con un error en la predicción en torno al 0.06%. En cuanto a la **extracción de características**, el estudio de los vectores de pesos aprendidos por la red convolucional ha sacado a la luz la influencia de los consumos previos sobre el consumo energético futuro.

En último lugar, la tarea de **visualización** se ha puesto en práctica sobre el caso del motor de inducción, a través de una arquitectura de tipo *deep autoencoder*. Esta red profunda permite reducir la dimensión del espacio de datos de entrada, generando como resultado un mapa de estados 2D en el que cada estado del motor tiene asociada una región diferente del espacio. De esta forma, el *deep autoencoder* proporciona la posibilidad de monitorizar visualmente el comportamiento del motor de inducción.

Con la resolución de estos problemas, se alcanza satisfactoriamente el objetivo de partida, pues queda **demostrada la aplicabilidad de las técnicas de *deep learning* en el ámbito industrial**. En cuanto a las arquitecturas propuestas, cabe destacar la variedad de las mismas, dado que se han manejado cuatro tipos de redes profundas: *feedforward*, convolucionales, LSTM y *deep autoencoders*. Se trata de las cuatro arquitecturas más empleadas en la literatura, por lo que se ha decidido experimentar con todas ellas y comparar su comportamiento ante los distintos problemas abordados.

En este sentido, las **redes *feedforward*** han destacado por ser las más simples y fáciles de interpretar, aunque también han sido las que han demostrado un peor comportamiento. Por su parte, las **redes LSTM**, debido a su capacidad para conservar memoria a corto y largo plazo, representan la arquitectura más compleja y, por tanto, más costosa de entrenar. No obstante, los excelentes resultados aportados en la predicción son una muestra del gran potencial de estas redes en el análisis de secuencias temporales. En lo que respecta a las **redes convolucionales**, son las que han demostrado un mejor comportamiento en las fases de extracción de características, lo que se ha traducido en exitosos resultados, por ejemplo, en la tarea de clasificación. Adicionalmente, se trata de arquitecturas fácilmente interpretables, cuyo análisis permite extraer valiosa información acerca de los procesos. En último lugar, la arquitectura ***deep autoencoder*** destaca por su singular estructura que, con un cuello de botella, es la única de las redes propuestas que permite abordar el problema de reducción de la dimensión.

Finalmente, desde un punto de vista técnico, este TFM ha servido a la proyectante para iniciarse en el manejo de la combinación de librerías **Keras-Tensorflow**. Estas valiosas herramientas, *open source* y con un gran crecimiento en fiabilidad y potencial, son

las que han hecho posible la implementación de las arquitecturas de red expuestas a lo largo de la investigación.

4.2. Aportaciones

Esta investigación se ha centrado en la consecución del objetivo planteado al comienzo de la misma, que consistía en estudiar **las posibilidades de aplicación de las técnicas de *deep learning* en el análisis y mejora de la eficiencia de procesos industriales**. Las aportaciones realizadas en relación a este objetivo son las siguientes:

1. Se investigó el estado del arte en técnicas de *deep learning*.

2. Se aplicaron con éxito distintas técnicas de *deep learning* en la resolución de problemas industriales:

    2.1. Se detectaron fallos en un proceso industrial (en concreto, sobre un motor de inducción, mediante análisis de corrientes y vibraciones), empleando distintos tipos de arquitecturas profundas (red *feedforward* y red convolucional).

    2.2. Se consiguió predecir el comportamiento de un sistema de ingeniería (predicción de demanda eléctrica), empleando distintos tipos de arquitecturas profundas (red *feedforward*, red convolucional y red LSTM).

    2.3. Se visualizó el comportamiento de un proceso industrial (de nuevo, un motor de inducción, en base al análisis de corrientes y vibraciones) mediante “mapas” de estados del mismo, empleando para ello una arquitectura de tipo *deep convolutional autoencoder*.

3. Se estudiaron los resultados obtenidos, extrayendo información desconocida acerca de los procesos analizados (parámetros constructivos y de funcionamiento de los sistemas).

4.3. Líneas de futuro trabajo

Este proyecto puede servir como trampolín para futuros trabajos de investigación. Entre los posibles caminos a seguir, destacan los señalados a continuación.

En primer lugar, dados los buenos resultados aportados por las redes convolucionales, parece interesante profundizar en su estudio. Una opción es poner en práctica las **convoluciones 2D** y abordar a través de ellas el tratamiento de imágenes, que es uno de los nichos por excelencia de las redes profundas [45]. Esto permitiría analizar imágenes de los procesos y realizar tareas de inspección sobre ellas. Una posible aplicación es la detección del defecto de *chatter* (defecto típico de la laminación en frío del acero) mediante el tratamiento de los espectrogramas del proceso.

Más allá del aprendizaje puramente supervisado, esta tomando fuerza el **aprendizaje por refuerzo** (*Reinforcement Learning*, RL) [55]. Este enfoque propone entrenar las arquitecturas de red con los resultados de sus propias acciones, recibiendo como observación las consecuencias de esas acciones y una recompensa que mide el éxito en la tarea a realizar. Aplicaciones como el AlphaGo Zero, presentado por DeepMind en 2017 [23], utilizan este tipo de aprendizaje. En la industria, este enfoque podría extenderse a gran cantidad de problemas como, por ejemplo, la navegación autónoma de robots.

Finalmente, otro caso de interés es el de las **arquitecturas generativas** (*Generative Adversarial Networks*, GANs), que plantean el entrenamiento como un equilibrio entre dos redes: una red generadora (encargada de generar muestras sintéticas) y una red discriminatoria (encargada de valorar la calidad de las muestras generadas, comparándolas con muestras reales), que son entrenadas hasta que la red generadora es capaz de “engañar” a la discriminatoria [56]. Como resultado, estas redes aportan nuevas muestras realistas y genuinas, nunca antes vistas. Así es posible generar audios, imágenes, vídeos o, incluso, diseños farmacológicos (en [57] se muestra un ejemplo en el que se generan moléculas candidatas con potenciales propiedades para combatir el cáncer). Esta idea trasladada al ámbito industrial permitiría explorar territorios desconocidos en los procesos, en cuanto a, por ejemplo, nuevas condiciones de funcionamiento, con las que mejorar el rendimiento y eficiencia de los mismos.

5
Planificación y presupuesto

5.1. Planificación

En el siguiente cronograma se observan las tareas realizadas y los tiempos orientativos dedicados a cada una de ellas.

	2017/2018																			
	Septiembre				Octubre				Noviembre				Diciembre				Enero			
TAREAS	S1	S2	S3	S4	S1	S2	S3	S4	S1	S2	S3	S4	S1	S2	S3	S4	S1	S2	S3	S4
1. ESTUDIO DEL ESTADO DEL ARTE																				
- Técnicas básicas de *deep learning*																				
- Herramientas: Keras/Tensorflow																				
- Redes *feedforward*																				
- Redes convolucionales																				
- Redes recurrentes																				
- *Deep autoencoders*																				
- Otros aspectos (sobreajuste, activaciones, etc.)																				
2. APLICACIÓN AL ÁMBITO INDUSTRIAL																				
- Problema de clasificación																				
- Problema de predicción																				
- Problema de extracción de características																				
- Problema de visualización de datos																				
3. DOCUMENTACIÓN																				
- Redacción del borrador																				
- Revisiones continuas del borrador																				

Tabla 5.1. Planificación del TFM.

5.2. Presupuesto

En base a las horas de trabajo requeridas y a los materiales empleados, se calcula a continuación el presupuesto del trabajo desarrollado.

CONCEPTO	COSTE UNITARIO	MEDICIÓN	COSTE
Mano de obra	25 €/hora	450 horas	11250 €
Ordenador (Procesador: 3,3 GHz Intel Core i7; Memoria: 16 GB 2133 MHz LPDDR3; Gráficos: Intel Iris Graphics 550 1536 MB; Disco duro: 500GB)	40 €/mes	5 meses	200 €
macOS High Sierra	0 €/hora	1 ud	0 €
Jupyter Notebook	0 €/hora	1 ud	0 €
Keras/Tensorflow	0 €/hora	1 ud	0 €
Datasets de trabajo	0 €/hora	2 ud	0 €
TOTAL			11450 €
IVA (21%)			2405 €
TOTAL + IVA			13855 €

Tabla 5.2. Presupuesto del TFM.

Para las mediciones y precios dados, se calcula un presupuesto total de TRECE MIL OCHOCIENTOS CINCUENTA Y CINCO EUROS.

Ana González Muñiz

26 de Enero de 2018

6
Listado de anexos

Este TFM incluye un total de ocho anexos, que recogen la implementación de las arquitecturas de red expuestas a lo largo de la investigación. El contenido de cada uno de ellos se detalla en la siguiente tabla.

Nº Anexo	Contenido
1	Arquitectura de clasificación *feedforward* expuesta en la sección 3.2.1.
2	Arquitectura de clasificación convolucional expuesta en la sección 3.2.2. Extracción de características expuesta en la sección 3.4.1.
3	Arquitectura de predicción *feedforward* expuesta en la sección 3.3.1.
4	Arquitectura de predicción LSTM expuesta en la sección 3.3.2.
5	Arquitectura de predicción convolucional expuesta en la sección 3.3.3. Extracción de características expuesta en la sección 3.4.2.
6	Técnicas alternativas de predicción presentes en la Tabla 3.6.
7	Arquitectura *deep convolutional autoencoder* expuesta en la sección 3.5.1.
8	Clasificación de los puntos de la retícula para la generación de la Figura 3.32.

Tabla 6.1. Anexos del TFM.

7
Bibliografía

[1] Hilbert, Martin, and Priscila López. "The world's technological capacity to store, communicate, and compute information." *science* 332.6025 (2011): 60-65.

[2] Arel, Itamar, Derek C. Rose, and Thomas P. Karnowski. "Deep machine learning-a new frontier in artificial intelligence research [research frontier]." *IEEE computational intelligence magazine* 5.4 (2010): 13-18.

[3] LeCun, Yann, Yoshua Bengio, and Geoffrey Hinton. "Deep learning." *Nature* 521.7553 (2015): 436-444.

[4] Bernard, Marr. "The amazing ways Google Uses Deep Learning AI." *Forbes*: Tech #Big data. (2017)

[5] Bernard, Marr. "4 Mind-Blowing Ways Facebook Uses Artificial Intelligence". *Forbes*: Tech #Big data. (2016)

[6] Deng, Li, et al. "Recent advances in deep learning for speech research at Microsoft." Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on. IEEE, 2013.

[7] Wing Kosner, Anthony. "Deep Learning And Machine Intelligence Will Eat The World" *Forbes*: Tech 2015 (2014)

[8] McCulloch, Warren S., and Walter Pitts. "A logical calculus of the ideas immanent in nervous activity." *The bulletin of mathematical biophysics* 5.4 (1943): 115-133.

[9] Hebb, Donald O. "The organization of behavior: A neuropsychological theory." (1949).

[10] Rosenblatt, Frank. "The perceptron: A probabilistic model for information storage and organization in the brain." *Psychological review* 65.6 (1958): 386.

[11] Widrow, B., and M. E. Hoff. "IRE WESCON Convention Record, volume 4, chapter Adaptive Switching Circuits." *IRE, New York* (1960).

[12] Minsky, Marvin, Seymour A. Papert, and Léon Bottou. “Perceptrons: An introduction to computational geometry.” MIT press, 2017.

[13] Rumelhart, David E., Geoffrey E. Hinton, and James L. McClelland. "A general framework for parallel distributed processing." *Parallel distributed processing: Explorations in the microstructure of cognition* 1 (1986): 45-76.

[14] Lecun, Yann. "PhD thesis: Modeles connexionnistes de l'apprentissage (connectionist learning models)." (1987).

[15] Fukushima, Kunihiko, and Sei Miyake. "Neocognitron: A self-organizing neural network model for a mechanism of visual pattern recognition." *Competition and cooperation in neural nets*. Springer, Berlin, Heidelberg, 1982. 267-285.

[16] LeCun, Yann, et al. "Gradient-based learning applied to document recognition." *Proceedings of the IEEE* 86.11 (1998): 2278-2324.

[17] Hochreiter, Sepp. "Untersuchungen zu dynamischen neuronalen Netzen." *Diploma, Technische Universität München* 91 (1991).

[18] Bengio, Yoshua, Patrice Simard, and Paolo Frasconi. "Learning long-term dependencies with gradient descent is difficult." *IEEE transactions on neural networks* 5.2 (1994): 157-166.

[19] Hochreiter, Sepp, and Jürgen Schmidhuber. "Long short-term memory." *Neural computation* 9.8 (1997): 1735-1780.

[20] Schaeffer, Jonathan, and Aske Plaat. "Kasparov versus deep blue: The re-match." *ICCA Journal* 20.2 (1997): 95-101.

[21] Hinton, Geoffrey E., Simon Osindero, and Yee-Whye Teh. "A fast learning algorithm for deep belief nets." *Neural computation* 18.7 (2006): 1527-1554.

[22] Le, Quoc V. "Building high-level features using large scale unsupervised learning." Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on. IEEE, 2013.

[23] Silver, David, et al. "Mastering the game of go without human knowledge." *Nature* 550.7676 (2017): 354.

[24] Taigman, Yaniv, et al. "Deepface: Closing the gap to human-level performance in face verification." *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2014.

[25] Goodfellow, Ian, Bengio, Yoshua. and Courville, Aaron. (2016). “Deep Feedforward Networks”. In: “Deep Learning”, MIT Press.

[26] Montufar, Guido F., et al. "On the number of linear regions of deep neural networks." *Advances in neural information processing systems*. 2014.

[27] LeCun, Yann, and Yoshua Bengio. "Convolutional networks for images, speech, and time series." *The handbook of brain theory and neural networks* 3361.10 (1995): 1995.

[28] Sainath, Tara N., et al. "Deep convolutional neural networks for LVCSR." Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on. IEEE, 2013.

[29] Graves, Alex, Abdel-rahman Mohamed, and Geoffrey Hinton. "Speech recognition with deep recurrent neural networks." *Acoustics, speech and signal processing (icassp), 2013 ieee international conference on*. IEEE, 2013.

[30] Sutskever, Ilya, James Martens, and Geoffrey E. Hinton. "Generating text with recurrent neural networks." *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*. 2011.

[31] Hinton, Geoffrey E., and Ruslan R. Salakhutdinov. "Reducing the dimensionality of data with neural networks." *science* 313.5786 (2006): 504-507.

[32] Bengio, Yoshua, Aaron Courville, and Pascal Vincent. "Representation learning: A review and new perspectives." *IEEE transactions on pattern analysis and machine intelligence* 35.8 (2013): 1798-1828.

[33] Hornik, Kurt, Maxwell Stinchcombe, and Halbert White. "Multilayer feedforward networks are universal approximators." *Neural networks* 2.5 (1989): 359-366.

[34] Cybenko, George. "Approximation by superpositions of a sigmoidal function." *Mathematics of Control, Signals, and Systems (MCSS)* 2.4 (1989): 303-314.

[35] Glorot, Xavier, and Yoshua Bengio. "Understanding the difficulty of training deep feedforward neural networks." *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*. 2010.

[36] Hahnloser, Richard HR, et al. "Digital selection and analogue amplification coexist in a cortex-inspired silicon circuit." *Nature* 405.6789 (2000): 947-951.

[37] Glorot, Xavier, Antoine Bordes, and Yoshua Bengio. "Deep sparse rectifier neural networks." Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics. 2011.

[38] Rumelhart, David, Hinton, Geoffrey, and Ronald Williams. (1986a). “Learning representations by back-propagating errors”. *Nature*, 323, 533–536.

[39] Kingma, Diederik, and Jimmy Ba. "Adam: A method for stochastic optimization." arXiv preprint arXiv:1412.6980 (2014).

[40] Morgan, Nelson, and Hervé Bourlard. "Generalization and parameter estimation in feedforward nets: Some experiments." *Advances in neural information processing systems*. 1990.

[41] Prechelt, Lutz. "Early stopping-but when?." *Neural Networks: Tricks of the trade* (1998): 553-553.

[42] Srivastava, Nitish, et al. "Dropout: a simple way to prevent neural networks from overfitting." *Journal of machine learning research* 15.1 (2014): 1929-1958.

[43] Ioffe, Sergey, and Christian Szegedy. "Batch normalization: Accelerating deep network training by reducing internal covariate shift." *International Conference on Machine Learning*. 2015.

[44] Salimans, Tim, et al. "Improved techniques for training gans." *Advances in Neural Information Processing Systems*. 2016.

[45] Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton. "Imagenet classification with deep convolutional neural networks." *Advances in neural information processing systems*. 2012.

[46] "Convolución." *Wikipedia, La enciclopedia libre*. 4 dic 2017. Consultado el 26 dic 2017. [https://es.wikipedia.org/wiki/Convolución]

[47] Turchenko, Volodymyr, Eric Chalmers, and Artur Luczak. "A Deep Convolutional Auto-Encoder with Pooling-Unpooling Layers in Caffe." arXiv preprint arXiv: 1701.04949 (2017).

[48] Noh, Hyeonwoo, Seunghoon Hong, and Bohyung Han. "Learning deconvolution network for semantic segmentation." Proceedings of the IEEE International Conference on Computer Vision. 2015.

[49] Qian, Feng, et al. "Seismic facies recognition based on prestack data using deep convolutional autoencoder." arXiv preprint arXiv:1704.02446 (2017).

[50] Sutskever, Ilya, Oriol Vinyals, and Quoc V. Le. "Sequence to sequence learning with neural networks." Advances in neural information processing systems. 2014.

[51] Venugopalan, Subhashini, et al. "Translating videos to natural language using deep recurrent neural networks." arXiv preprint arXiv:1412.4729 (2014).

[52] Christopher Olah, “Understanding LSTM Networks.” *Colah’s blog*. 27 Aug 2015. Consultado el 26 dic 2017. [http://colah.github.io/posts/2015-08-Understanding-LSTMs/]

[53] Donahue, Jeffrey, et al. "Long-term recurrent convolutional networks for visual recognition and description." Proceedings of the IEEE conference on computer vision and pattern recognition. 2015.

[54] Departamento de Ingeniería Mecánica, Energética y de Materiales. “Vibraciones en máquinas. Mantenimiento Predictivo.” *Universitas Navarrensis*. 2015. Consultado el 08 ene 2018. [http://www.imem.unavarra.es/EMyV/pdfdoc/vib/vib_predictivo.pdf]

[55] Sutton, Richard S., and Andrew G. Barto. *Reinforcement learning: An introduction*. Vol. 1. No. 1. Cambridge: MIT press, 1998.

[56] Goodfellow, Ian, et al. "Generative adversarial nets." *Advances in neural information processing systems*. 2014.

[57] Kadurin, Artur, et al. "The cornucopia of meaningful leads: Applying deep adversarial autoencoders for new molecule development in oncology." *Oncotarget* 8.7 (2017): 10883.