Universidad Politécnica de Cartagena

EUROPEAN UNIVERSITY OF TECHNOLOGY

"Think human first"

etsit escuela técnica superior de ingeniería de telecomunicación

Aplicación de algoritmos Machine Learning en ejercicios deportivos realizados en series de repeticiones

Proyecto Fin de Estudios

Autor: Alarcón Hellín, Gonzalo

Codirector: Vales Alonso, Javier

Director: Aarnoutse Sánchez, Juan Carlos Jacobo

Cartagena, Octubre 2022

Resumen

El Machine Learning es una disciplina perteneciente al campo de la Inteligencia Artificial (IA), la cual permite identificar patrones en datos masivos y que los sistemas sean capaces de elaborar predicciones a partir de dichos datos.

En este trabajo nos vamos a enfocar en las redes neuronales recurrentes LSTM para diseñar un modelo capaz de reconocer y contabilizar las repeticiones realizadas en diferentes actividades deportivas, como son dominadas, sentadillas, flexiones, etc. Las redes de tipo Long Short Term Memory son una extensión de las redes neuronales recurrentes, que básicamente amplían su memoria para aprender de experiencias pasadas.

Los datos para hacer posible el entrenamiento de la red neuronal han sido recopilados gracias a sensores inerciales X-BIMU. Este tipo de sensores recoge los datos medidos por el acelerómetro, giroscopio y magnetómetro, con una gran ventaja, que se puede realizar de manera inalámbrica.

Por último, se ha entrenado un modelo con dichos datos y los resultados obtenidos muestran que es posible identificar con una exactitud elevada las repeticiones gracias a los patrones de movimiento extraídos de los datos en aquellos ejercicios que comparten rangos de movimiento similares. Por el contrario, existen limitaciones a la hora de extrapolar la red neuronal a otros ejercicios que no han sido entrenados previamente, siempre y cuando su rango de movimiento difiera mucho del resto. Lo mismo ocurre cuando se extrapolan los ejercicios a otros individuos, dependiendo del movimiento y su similitud con el resto de ejercicios utilizados como entrenamiento, se puede obtener una mayor precisión en la validación.

Abstract

Machine Learning is a discipline belonging to the field of Artificial Intelligence (AI), which allows patterns to be identified in massive data and systems to be able to make predictions from this data.

In this report, we are going to focus on recurrent neural networks to design a model capable of recognizing and counting repetitions performed in different sports activities, such as pull-ups, squats, push-ups and deadlifts. Long-Short Term Memory networks are an extension of Recurrent Neural Networks, which extend their memory to learn from past experiences.

The data to train the neural network has been collected by inertial sensors, in this case the X-BIMU sensors. This sensor collects the data measured by the accelerometer, gyroscope and magnetometer, with the great advantage that it can be done wirelessly.

Finally, the model has been trained with these data and the results obtained show that it is possible to identify repetitions with a high accuracy thanks to the movement patterns extracted from the data. However, on the other hand, there are many limitations when extrapolating the neural network to other exercises that have not been previously trained, if their range of motion differs from the rest. The same thing happens when the exercises are performed by other person, depending on the movement and its similarity with the rest of exercises used as training performed by another person, a greater precision in the validation can be obtained.

Agradecimientos

En este apartado más personal me gustaría agradecer a las personas que han hecho posible este Trabajo Fin de Grado.

Quisiera dar las gracias a mi tutor Juan Carlos Jacobo Sánchez Aarnautse por ofrecerme la oportunidad de trabajar y aprender más sobre el Machine Learning.

También me gustaría agradecer a Javier Vales Alonso toda su ayuda durante todos estos meses de trabajo, gracias a él, he llegado a formarme y a aprender sobre Redes Neuronales además de sus consejos para el uso de herramientas que me han sido muy útiles a la hora de realizar este trabajo.

Por último, me gustaría darle las gracias a mi familia y amigos, en especial a mis padres y hermana, que además de darme la oportunidad de estudiar esta carrera, me han apoyado desde el principio, en los momentos más difíciles, y que durante estos años en la universidad me han ayudado a esforzarme lo máximo posible.

Sin duda, hubiera sido imposible lograrlo sin ellos, os lo agradezco de corazón.

¡Muchas gracias!

Índice general

Índice de figuras

Capítulo 1. Introducción

1.1. Introducción

Gracias a la inteligencia artificial (IA), existen sistemas de cómputo capaces de aprender a partir de la experiencia acumulada y de esta forma poder realizar tareas de forma similar a cómo las realizan los humanos. La importancia radica en el aprendizaje, ya que gracias a herramientas como el Machine Learning (Aprendizaje Automático) cada vez es más común la necesidad de que los algoritmos aprendan por si solos, esto supone una verdadera revolución en el procesamiento de datos.

El Machine Learning es una disciplina perteneciente al campo de la Inteligencia Artificial (IA), la cual permite identificar patrones en datos masivos y que los sistemas sean capaces de elaborar predicciones a partir de dichos datos. El interés por estos sistemas viene dado por su capacidad para poder realizar las tareas específicas de forma autónoma, sin tener que ser programados.

A lo largo de los últimos años se ha incrementado notablemente el interés por las redes de sensores y su incorporación en diversos entornos inteligentes que permitan el modelado y la predicción del movimiento en las personas. Una de las aplicaciones con más importancia dentro del Machine Learning es el reconocimiento de la actividad humana (Human Activity Recognition), su principal objetivo es controlar, analizar y modelar el comportamiento humano para así interpretar eventos en curso correctamente.

Este Trabajo Final de Grado se ha realizado dentro del Grupo de ingeniería Telemática (GIT) de la Universidad Politécnica de Cartagena. El objetivo de este trabajo es poder reconocer el movimiento de las personas a través de sensores inerciales y diseñar una red neuronal capaz de identificar las repeticiones en ejercicios físicos, como pueden ser dominadas, sentadillas, flexiones, etc.

1.2. Objetivos y metodología

Como se ha comentado previamente, el objetivo principal de este trabajo es el diseño e implementación de un algoritmo capaz de reconocer y contar las repeticiones dentro de diferentes series en ejercicios deportivos. Para cumplir con el objetivo final se contemplan los siguientes objetivos específicos:

- Estudiar y aplicar técnicas de procesamiento de datos.
- Estudiar y optimizar arquitecturas de redes neuronales con el propósito de poder reconocer las repeticiones del ejercicio.
- Diseñar e implementar un módulo capaz de reconocer las diferentes repeticiones.

1.3. Estructura del documento

El documento se compone en ocho capítulos y están dispuestos de la siguiente manera:

- **Capítulo 1: Introducción.**
  Se presenta una introducción al Machine Learning, y se definen los objetivos de este trabajo además de una pequeña explicación de las herramientas y software utilizado.

- **Capítulo 2: Estado del arte.**
  En este capítulo se realiza un resumen de trabajos publicados que han tratado temas relacionados con el reconocimiento de actividad humana utilizando sensores inerciales y modelos de Machine Learning.

- **Capítulo 3: Machine Learning.**
  Se centra en las técnicas de Machine Learning, aborda tanto el punto de vista histórico, teórico y práctico. Se presenta su evolución a lo largo de la historia y una base teórica mostrando sus fundamentos matemáticos.

- **Capítulo 4: Redes Neuronales.**
  Se presenta una base teórica de las redes neuronales artificiales, y los diversos tipos que existen, así como las redes neuronales recurrentes, y nos centraremos en las redes con memoria a corto y largo plazo (LSTM).

- **Capítulo 5: Sensores inerciales.**
  Durante este capítulo se indican los sensores que han sido utilizados para la recopilación de datos de este trabajo, además de sus fundamentos básicos.

- **Capítulo 6: Conjunto de datos.**
  Se describe el proceso realizado en la recolección de datos, para ello, se describe las bases de datos utilizadas y las diversas técnicas utilizadas para el procesamiento de las señales resultantes de los sensores inerciales.

- **Capítulo 7: Desarrollo de la red neuronal.**
  En este capítulo se describen las características de la red neuronal utilizada durante los diferentes experimentes realizados.
  También se muestran las tasas de acierto de cada experimento, con el objetivo de conseguir que el sistema realice mejores predicciones.

- **Capítulo 8: Conclusiones.**
  En este capítulo se incluyen las principales conclusiones obtenidas en este Trabajo de Final de Grado y la propuesta de líneas futuras.

1.4. Entorno de desarrollo y herramientas utilizadas

Python

El lenguaje de programación utilizado para implementar el algoritmo en este Trabajo de Fin de Grado ha sido Python. Se trata de un lenguaje de programación de alto nivel dinámico y multiplataforma, el cual se caracteriza por la legibilidad y sencillez del código.

Se trata de un lenguaje de código abierto, por lo que no hay que pagar ninguna licencia para su utilización. Al ser un lenguaje multiparadigma, combina propiedades de diferentes paradigmas de programación, soporta programación orientada a objetos, programación imperativa y programación funcional. Lo que permite que sea muy flexible y fácil de aprender de manera independiente, además, es apto para todas las plataformas, se puede ejecutar en diferentes sistemas operativos, como Linux o Windows [1].

***Figura 1**. Logotipo de Python.*

Jupyter

El Proyecto Jupyter es una organización creada para el desarrollo de software de código abierto para la computación en diversos lenguajes de programación, entre los cuales se encuentran Julia, Python y R. Jupyter Notebook es un entorno computacional interactivo basado en la web que nos permite crear documentos jupyter. Un documento jupyter es un documento JSON que utiliza la extensión .ipynb [2].

Los Notebooks de Jupyter son muy interactivos, se puede ir ejecutando el código paso a paso y dividirlo en diferentes celdas de código. Además, tienen la ventaja de que permite hacer uso de herramientas y bibliotecas de Big Data, como son los paquetes Tensorflow, Scikit-Learn o Pandas.

Su popularidad descansa en la gran ayuda que aporta en la gestión y administración de paquetes, para lo que utiliza el sistema ANACONDA, que incluye más de 1000 paquetes de datos [3].

***Figura 2**. Logotipo de Jupyter.*

Tensorflow

Es una librería de código libre para computación numérica utilizando grafos de flujos de datos empleada para Machine Learning a través de un rango de tareas. Es desarrollado por Google Brain Team para satisfacer sus necesidades a partir de redes neuronales artificiales.

Tensorflow permite implementar redes neuronales de aprendizaje profundo, se trata de un sistema de segunda generación liberado el 9 de noviembre del 2015.

Es el que más popularidad tiene entre todas las librerías y herramientas de Machine Learning, esto es debido a que está muy optimizado para el uso de redes neuronales, y también debido a su efectividad y potencialidad [4].

Figura 3. *Logotipo de TensorFlow.*

Keras

Es una API de aprendizaje profundo implementada en Python, la cual se ejecuta sobre la plataforma TensorFlow. Fue desarrollada para permitir una experimentación rápida, siendo capaz de pasar de la idea al resultado lo más rápido posible.

La API de Keras contiene diversos bloques necesarios para diseñar y especificar las características de las redes neuronales, como por ejemplo, los layers, funciones objetivo, funciones de activación y optimizadores matemáticos.

Además de soportar las redes neuronales estándar, Keras ofrece también soporte para las Redes Neuronales Recurrentes (RNN) [5].

Figura 4. *Logotipo de Keras.*

Colaboratory

Es una herramienta de Google Research. Es un producto orientado a tareas de aprendizaje automático y análisis de datos. Se trata de un servicio de cuaderno alojado de Jupyter que no requiere configuración y que ofrece acceso libre de coste a recursos informáticos como GPUs.

Jupyter es el proyecto de código abierto en el que Colab está basado, la diferencia es que Colab permite utilizar cuadernos de Jupyter sin tener que descargar librerías ni paquetes, de esta forma, los cuadernos de Colab son almacenados en Google Drive o bien pueden ser cargados directamente desde el repositorio GitHub [6].

Figura 5. *Logotipo de Google Colaboratory.*

Capítulo 2. Estado del arte

En este capítulo se presenta una serie de trabajos relacionados con las técnicas de Machine Learning y el uso de sensores IMU para la captura de movimientos.

Es de gran importancia destacar los trabajos en los que se utilizan algoritmos de Machine Learning para el reconocimiento de actividades humanas, tal y como se realiza en [7].

El objetivo de dicho Trabajo Fin de Máster era realizar un sistema de reconocimiento de actividades mediante modelos de redes neuronales de aprendizaje profundo, a partir de datos provenientes de un dataset de movimientos capturado con sensores inerciales. Con ello se pretendía estudiar su viabilidad de inclusión en un sistema embebido y en tiempo real, con la intención de servir de base para una red neuronal que permita, mediante los sensores, el reconocimiento de movimientos o actividades físicas [7].

Otro trabajo que destacar es el artículo publicado en febrero de 2019 sobre el uso del aprendizaje profundo para el reconocimiento de actividades humanas [8].

En este estudio, crean un algoritmo capaz de distinguir entre diez tipos de ejercicios, además de contabilizar las repeticiones de cada ejercicio. En este caso, utilizan técnicas de Deep Learning y una red neuronal convolucional para el conteo de repeticiones de estos diez ejercicios complejos [8].

A diferencia de nuestro trabajo, en lugar de utilizar sensores inerciales para la recogida de datos, utilizaron dos relojes inteligentes o smartwatches, uno colocado en la muñeca y otro en el tobillo. De esta forma, registraron diez de los ejercicios más comunes del CrossFit, entre los cuales caben destacar las dominadas, flexiones, burpees, sentadillas, etc.

Se siguieron dos técnicas en la recopilación de los datos, en primer lugar, un entrenamiento restringido diseñado para etiquetar los datos para obtener una mayor precisión en el entrenamiento de la red neuronal, indicando el tipo de ejercicio y el comienzo de cada repetición. Por el contrario, la segunda técnica utilizada solamente fue utilizada para la validación de la red neuronal y no para su entrenamiento. En este caso, los datos no estaban etiquetados y no se indicaba el inicio de las repeticiones al usuario, sino que éste realizaba el ejercicio libremente.

La precisión obtenida para el reconocimiento de los ejercicios fue muy elevada, con una precisión del 99,96% al utilizar ambos relojes, y una precisión del 98,91% al utilizar solamente el reloj ubicado en la muñeca. Para el conteo de las repeticiones, llegaron a obtener un margen de error de ±1 repetición para el 91% de las series de ejercicios.

Capítulo 3. Machine Learning

La técnica en la que se basa el algoritmo en este Trabajo de Fin de Grado es el Machine Learning, o aprendizaje automático. A lo largo de este capítulo se estudian los fundamentos teóricos de este campo, así como los de Inteligencia Artificial (AI, Artificial Intelligence) el cual engloba al Machine Learning.

3.1. Introducción

Para comenzar a describir de Machine Learning es necesario especificar qué es la Inteligencia Artificial.

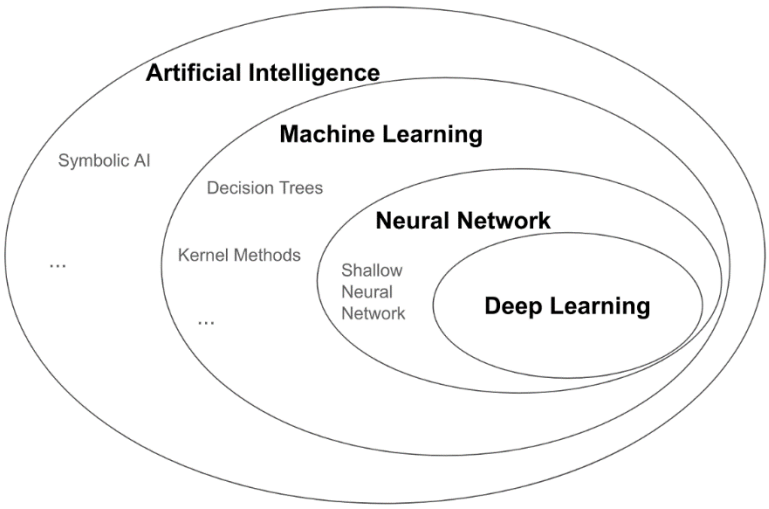

Figura 6. *Diagrama de Inteligencia Artificial.*

En la **Figura 6** se puede apreciar un diagrama de Venn en el cual se observa los diferentes subconjuntos que forman la Inteligencia Artificial, por este motivo es importante hablar primero de Inteligencia Artificial para así luego hablar de Machine Learning.

La Inteligencia Artificial es un campo compuesto de múltiples técnicas que permiten realizar software capaz de aprender modelos matemáticos sin tener que definir las infinitas combinaciones de posibilidades que se pueden dar en dicho problema. Su objetivo es conseguir que los ordenadores sean programados de tal forma que sean capaces de imitar comportamientos teniendo como referencia ejemplos previos.

En los comienzos de la Inteligencia Artificial, su objetivo era la resolución de manera rápida de tareas intelectualmente difíciles para los humanos, pero que pueden describirse mediante una serie de reglas matemáticas, lo que las hacía sencillas para los ordenadores. Sin embargo, el verdadero reto de la Inteligencia Artificial es resolver tareas que son fáciles de realizar para las personas, pero difíciles de describir formalmente por ellas, ya que las resolvemos de forma intuitiva, como puede ser reconocer ciertas palabras cuando las escuchamos o caras e imágenes cuando las vemos [9].

Hoy en día, la Inteligencia Artificial es un campo próspero con muchas aplicaciones prácticas y temas de investigación activos, como la automatización de labores rutinarias, el reconocimiento del lenguaje o imágenes o la realización de diagnósticos en medicina [9].

La complejidad a la que se enfrentan los sistemas que dependen del conocimiento codificado sugieren que los sistemas de Inteligencia Artificial necesitan la capacidad de adquirir su propio conocimiento extrayendo patrones a partir de los datos sin procesar que se les proporciona. Esta técnica se conoce como Machine Learning.

La introducción del Machine Learning permitió a los ordenadores poder abordar problemas relacionados con el conocimiento del mundo real y tomar decisiones que parecían subjetivas.

El rendimiento de estos sencillos algoritmos de aprendizaje automático depende en gran medida de la representación de los datos que se les proporcionan. Por ejemplo, cuando se utiliza la regresión logística para recomendar una operación a un paciente, el sistema no examina directamente a la paciente. En cambio, el médico le dice al sistema la información relevante, los datos del paciente, etc. Cada pieza de información incluida en la representación del paciente se conoce como "feature". La regresión logística aprende cómo cada una de estas características del paciente se correlaciona con varios resultados.

El Machine Learning es la ciencia y el arte de programar ordenadores de forma que ellas sean capaces de aprender a partir de los datos. A continuación, se da una definición de ámbito más general:

Machine Learning es el campo de estudio que da a las máquinas la habilidad para aprender sin tener que ser explícitamente programadas. [Arthur Samuel, 1959]

Una definición más orientada al punto de vista de la ingeniería:

Se dice que un programa de un ordenador aprende de la experiencia E con respecto a una tarea T y una medida de rendimiento P, si su rendimiento en T, medido por P, mejora con la experiencia E. [Tom Mitchell, 1997]

El campo de Machine Learning se utiliza para resolver problemas para los que las soluciones existentes requieren de muchos ajustes o largas listas de reglas. Un algoritmo de aprendizaje automático puede simplificar el código y funciona mejor que el enfoque tradicional. También es útil para resolver problemas con entornos cambiantes, un sistema de Machine Learning se puede adaptar a los nuevos datos, y obtener información sobre problemas complejos y grandes cantidades de datos [10].

3.2. Historia del Machine Learning

En primer lugar, el nacimiento del aprendizaje automático vino de la mano de Alan Turing en el año 1950, con la creación del Test de Turing, que servía para determinar si una máquina era realmente inteligente. Dicho test ponía a prueba la capacidad de una máquina para tener un comportamiento similar al de una persona. El test consiste en conversaciones entre una persona y la máquina, que estaba programada para generar respuestas similares a las de los humanos, mientras un examinador evaluaba dichas conversaciones. En el caso de que el examinador fuera incapaz de distinguir entre la máquina y el humano, entonces la máquina habría superado la prueba de forma exitosa.

***Figura 7**. Retrato de Alan Turing.*

Más tarde, en el año 1952 Arthur Samuel escribe el primer programa capaz de aprender. Este consistía en el juego de las damas, y cuanto más se jugaba, la computadora más aprendía. Esto fue posible gracias a estudiar los movimientos que componían las estrategias ganadoras e incorporarlos en el programa.

***Figura 8.** Fotografía de Arthur Samuel.*

No fue hasta el año 1956, donde Martin Minsky y John McCarthy, con la ayuda de Claude Shannon y Nathan Rochester, se le acuña el término “Artificial Intelligence”.

Por su parte, Frank Rosenblatt, inventó el Perceptron en el año 1958. Esta tecnología se asemeja al cerebro humano, ya que se trata de un tipo de red neuronal. El sistema Perceptron conectaba una red de puntos, los cuales toman decisiones simples que se unen para así resolver problemas más complejos. El objetivo de la máquina era el reconocimiento de patrones de activación neuronal. Esta red neuronal supuso un gran avance ya que podría reconocer letras y números, pero no dejaba de ser muy limitada.

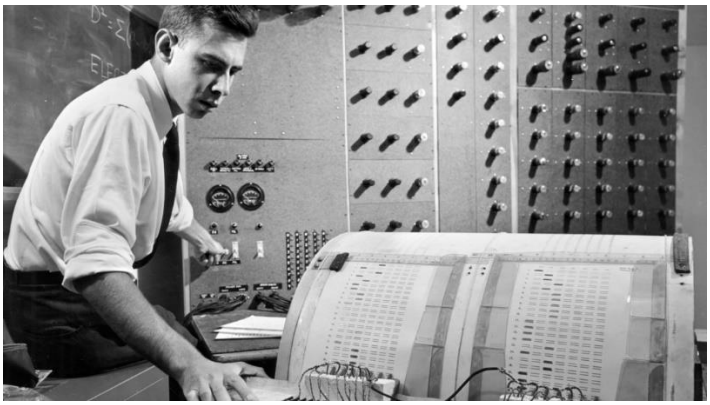

Figura 9. *Fotografía de Frank Rosenblatt.*

Más tarde, en 1979, se inventa uno de los primeros vehículos autónomos. El carro de Stanford fue un proyecto que comenzó como un estudio de cómo sería operar un rover lunar desde la Tierra y finalmente se convirtió en un vehículo capaz de esquivar obstáculos de forma autónoma.

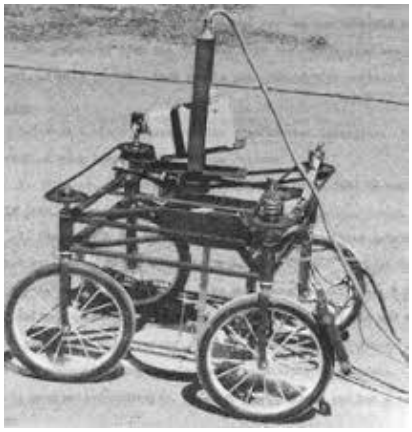

Figura 10. *Vehículo autónomo de Stanford.*

En 1981, Gerald Dejong introduce el concepto de (Explanation Based Learning, EBL) donde un ordenador crea reglas para descartar los datos menos significativos a partir del procesamiento y análisis de datos de entrenamiento.

En la década de los 90, el Machine Learning tuvo enfoques probabilísticos y se comenzó a utilizar en áreas comerciales para la minería de datos, aplicaciones web, aprendizaje de texto y aprendizaje de idiomas.

Geoffrey Hinton acuña el término “Deep Learning” en el año 2006. El uso del aprendizaje profundo hace posible entrenar a las máquinas para que sean capaces de distinguir y reconocer distintos objetos en imágenes y vídeos.

3.3. Tipos de sistemas de Machine Learning

Hay tanta variedad de sistemas de aprendizaje automático que es útil clasificarlos en categorías amplias, siguiendo los siguientes criterios:

- Sistemas que están o no entrenados con supervisión humana: supervisados, no supervisados, semi-supervisados y aprendizaje reforzado.
- Sistemas que pueden o no aprender gradualmente sobre la marcha: aprendizaje en línea o aprendizaje por lotes.
- Sistemas que funcionan simplemente comparando nuevos puntos de datos con puntos de datos conocidos, o en su lugar, mediante la detección de patrones en los datos de entrenamiento y la construcción de un modelo predictivo: aprendizaje basado en instancias versus aprendizaje basado en modelos.

Aprendizaje supervisado y no supervisado

Los sistemas de aprendizaje automático pueden ser clasificados a partir de la cantidad y del tipo de supervisión que tienen durante el entrenamiento, si se clasifican de esta manera, existen cuatro grandes categorías:

- En los sistemas con aprendizaje supervisado, el conjunto de datos de entrenamiento que alimenta al algoritmo incluye las soluciones deseadas, llamadas etiquetas (labels). Un ejemplo típico de aprendizaje supervisado es la clasificación, como puede ser el filtro de correo spam. El sistema está entrenado con muchos correos electrónicos de ejemplo con su clase, si son correos deseados o spam, y debe aprender a clasificar los nuevos correos electrónicos entrantes.

- En los sistemas de aprendizaje no supervisado, los datos de entrenamiento no tienen etiquetas (labels), el sistema trata de aprender sin un maestro. En el aprendizaje no supervisado, el sistema agrupará la información desordenada según similitudes y diferencias, aunque no se proporcionen categorías. Estos algoritmos pueden realizar tareas de procesamiento más complejas que los sistemas de aprendizaje supervisado.

- Dado que el etiquetado de datos suele llevar mucho tiempo y es costoso, a menudo se puede tener muchas instancias sin etiquetar y pocas instancias etiquetadas. Existen algoritmos que pueden manejar datos que están parcialmente etiquetados, estos sistemas se denominan aprendizaje semi-supervisado.

- Los sistemas con aprendizaje por refuerzo son diferentes, el sistema de aprendizaje llamado en este contexto como agente, puede observar el entorno, seleccionar y realizar acciones y obtener recompensas a cambio. El agente aprende a lograr un objetivo en un entorno complejo e incierto. Luego debe aprender por sí mismo cuál es la mejor estrategia, llamada política, para obtener la mayor recompensa con el tiempo. Una política define qué acción debe elegir el agente cuando se encuentra en una determinada situación.

Aprendizaje por lotes y en línea

Otro criterio que se suele utilizar para clasificar sistemas de aprendizaje automático es si el sistema puede o no aprender incrementalmente de un flujo de datos entrante.

- En el aprendizaje por lotes (Batch Learning), el sistema es incapaz de aprender de forma incremental, debe ser entrenado utilizando todos los datos disponibles. Esto generalmente requerirá mucho tiempo y recursos computacionales, por lo que normalmente se realiza fuera de línea. Primero se entrena el sistema, y luego se pone en producción y se ejecuta sin aprender más, simplemente aplica lo que ha aprendido.

- En el aprendizaje en línea (Online Learning), el sistema se entrena incrementadamente alimentándolo con instancias de datos de forma secuencial, ya sea individualmente o en pequeños grupos llamados mini lotes. Cada paso de aprendizaje es rápido y económico, por lo que el sistema puede aprender sobre nuevos datos a medida que llegan. Un gran desafío con el aprendizaje en línea es que, si el sistema se alimenta con datos incorrectos, el rendimiento del sistema disminuirá gradualmente.

Aprendizaje basado en instancias y aprendizaje basado en modelos

La mayoría de las tareas de Machine Learning consisten en hacer predicciones. Esto significa que, dado un número de ejemplos de entrenamiento, el sistema necesita ser capaz de hacer buenas predicciones para generalizar a ejemplos que nunca ha visto previamente.

Tener una buena medida de rendimiento en los datos de entrenamiento es bueno, pero insuficiente. El verdadero objetivo es tener un buen desempeño en nuevas instancias.

Hay dos enfoques para la generalización de estos nuevos datos, aprendizaje basado en instancias y aprendizaje basado en modelos.

- Los sistemas de aprendizaje automático basados en instancias (Instance-Based Learning) son los sistemas que aprenden ejemplos de entrenamiento de memoria y luego los generalizan a nuevas instancias en función de una medida de similitud. Estos sistemas también son conocidos como aprendizaje basado en memoria.

- Otra forma de generalizar a partir de un conjunto de ejemplos es construir un modelo de estos ejemplos y luego usar ese modelo para hacer predicciones. Esto se denomina aprendizaje basado en modelos (Model-based Learning).

Capítulo 4. Redes Neuronales

Una vez explicados los fundamentos del Machine Learning, es necesario profundizar en las redes neuronales, pues el algoritmo de este Trabajo de Fin de Grado se basa en una red neuronal.

4.1. Introducción

Las redes neuronales son capaces de emular ciertas características propias de los humanos, como la capacidad de memorizar y aprender con la experiencia.

En definitiva, las redes neuronales no son más que un modelo artificial y simplificado del cerebro humano, que es el ejemplo más perfecto del que disponemos para un sistema que es capaz de adquirir conocimiento a través de la experiencia [11].

Una red neuronal es un sistema para el tratamiento de la información, cuya unidad básica de procesamiento está inspirada en la célula fundamental del sistema nervioso humano, la neurona [11].

Las redes neuronales artificiales son una forma de computación, inspirada en modelos biológicos, son redes interconectadas masivamente en paralelo de elementos simples (usualmente adaptativos) y con organización jerárquica, las cuales intentan interactuar con los objetos del mundo real del mismo modo que lo hace el sistema nervioso biológico [11].

Las redes neuronales son capaces de aprender a realizar tareas basadas en un entrenamiento adquiriendo así una experiencia. Son sistemas dinámicos auto adaptativos gracias a la capacidad de autoajuste de las neuronas que componen el sistema.

Se dice que las redes neuronales son tolerantes a fallos gracias a que tienen la información distribuida entre las conexiones de las neuronas, de esta manera se consigue un cierto grado de redundancia en este tipo de almacenamiento. También son capaces de aprender a reconocer patrones con ruido, lo que se conoce como tolerancia frente a fallos en los datos, y pueden seguir realizando su función con cierta degradación, aunque se destruya parte de la red.

4.2. Elementos básicos de una Red Neuronal

Una red neuronal está compuesta por una serie de neuronas interconectadas entre sí y dispuestas de tal manera que se pueden clasificar en diferentes capas. El número de capas y el número de neuronas por capa puede variar en función de la necesidad que plantea el problema en cuestión.

Generalmente, las capas que componen la red neuronal se pueden clasificar en tres tipos, la capa de entrada, capa oculta (puede estar formada por varias capas) y capa de salida, tal y como se puede apreciar en la **Figura 11**. Los datos que ingresan a la red

entran por medio de la capa de entrada, a continuación, pasan por la capa oculta y, por último, salen por la capa de salida.

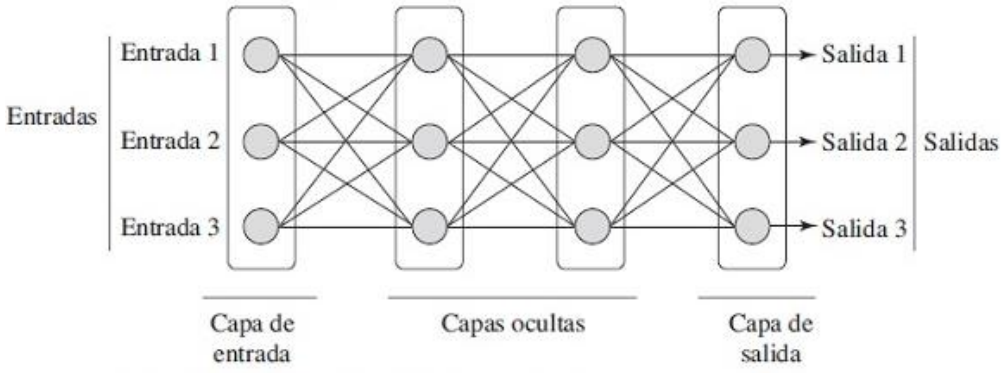

Figura 11. *Estructura de una Red Neuronal.*

Como se indica anteriormente, la distribución de las neuronas dentro de la red se realiza en diferentes niveles o capas, donde cada capa tiene un número determinado de neuronas. Las capas se pueden clasificar en tres tipos:

- Capa de entrada. Dicha capa se encarga de recibir directamente los datos que provienen de las fuentes externas.

- Capa oculta. Suele estar formada por más de una capa interna de la red por lo que no tienen contacto directo con el exterior de la red. En función del número de neuronas y del número de capas ocultas se puede determinar las diferentes topologías de redes neuronales.

- Capa de salida. Esta capa es la encargada de transferir la información de salida de la red neuronal hacia el exterior.

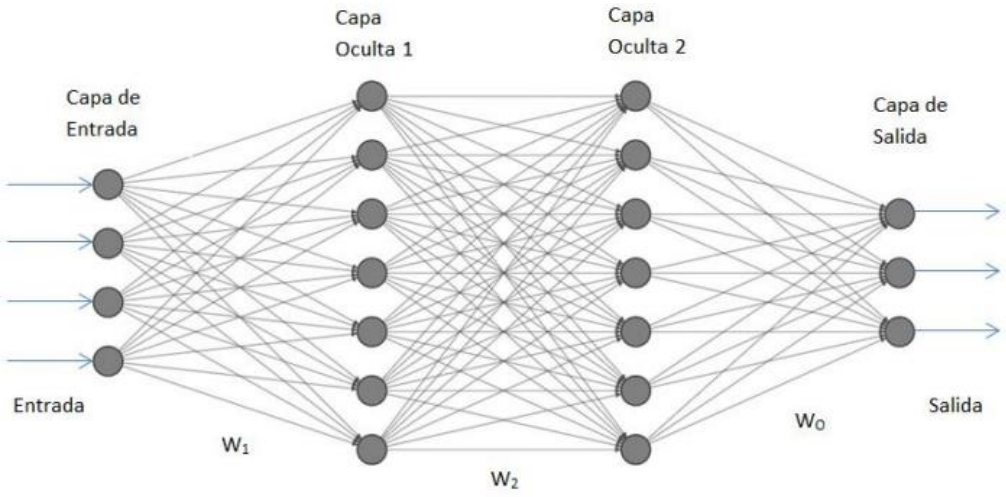

Figura 12. *Red Neuronal completamente conectada.*

Como se puede observar en la **Figura 12**, a cada una de las conexiones entre neuronas se le asigna una matriz de pesos, que determina la fuerza de dichas conexiones.

La matriz de pesos W_i viene dada por las señales que recibe cada neurona perteneciente a la capa oculta H_i de todas las neuronas de la capa anterior H_{i-1}. A su vez, las neuronas de la capa H_i emiten señales a todas las neuronas de la siguiente capa H_{i+1}indicando así la matriz de pesos W_{i+1}.

Si suponemos la neurona $O_{i,j}$ de la capa oculta H_i entonces:

$$O_{i,j} = \varphi(\sum_{k=1}^{n_i} O_{i-1,k} * W_i^{k,j} + b_{i,j})$$

Donde φ() es una función de activación que realiza una transformación no lineal de las entradas, dotando al modelo de la capacidad de aprender tareas complejas.

Para la neurona O_j de la capa de salida tenemos:

$$O_j = \varphi(\sum_{k=1}^{n_N} O_{N,k} * W_o^{k,j} + b_{o,j})$$

4.3. Redes Neuronales Recurrentes (RNN)

Hasta ahora hemos visto redes donde la función de activación va desde la capa de entrada hacia la capa de salida, las salidas de las neuronas se conectan como entrada a otras neuronas distintas. En el caso de las redes neuronales recurrentes, existe una retroalimentación entre las neuronas que componen las capas.

A continuación, en la **Figura 13** se puede observar una red RNN simple formada por una única neurona que recibe una entrada y produce una salida, que la envía a sí misma en forma de entrada.

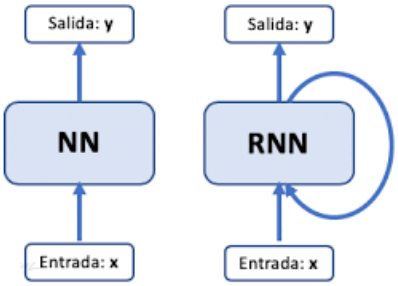

Figura 13. *Neurona de una ANN vs Neurona de una RNN.*

En cada timestep (instante de tiempo), esta neurona recurrente recibe como entrada la salida de la neurona anterior, y su propia salida del instante de tiempo anterior para generar su salida. Es decir, si suponemos que la salida de una neurona es y_t y su entrada es x_i, siendo el subíndice i el identificador de neurona, entonces su entrada en cada instante de tiempo es $x_i = y_{t-1} + y_{i-1}$. Cada neurona recibe dos entradas, la entrada de la capa anterior y a su vez la salida del instante anterior de dicha capa en cuestión.

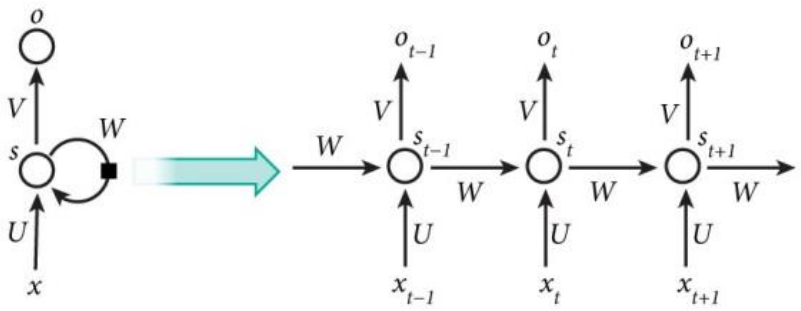

Figura 14. *Estructura de neuronas en una RNN.*

Para retener la información hay que alimentar la activación de la neurona en el instante de tiempo previo al actual. Como se observa en la figura anterior, la matriz de peso W se utiliza para cada instante de tiempo s_t.

En la red presentada en la **Figura 14**, la ecuación recursiva de la neurona en el instante de tiempo t viene dada por:

$$s_t = \varphi(Ux_t + Ws_{t-1} + b_h^t)$$

Siendo b_h^t el sesgo para la salida de la neurona. En el caso de que la salida de la capa recurrente sea un único valor, entonces se toma el valor que existe en dicha capa en el último instante de tiempo.

Una red RNN puede tomar simultáneamente una secuencia de entradas y producir una secuencia de salidas. Este tipo de redes son útiles para la predicción de series temporales como los precios de las acciones.

De forma alternativa, se puede alimentar a la red con una secuencia de entradas e ignorar todas las salidas excepto la última, en otras palabras, una red secuencia a vector. Por ejemplo, se puede alimentar a la red con la reseña de una película y la red puede generar una salida que indique una puntuación (-1 si no le gusta o +1 si le encanta).

Por el contrario, podría alimentar a la red con el mismo vector de entrada una y otra vez, una vez por cada instante de tiempo y generar una secuencia, esto se denomina una red vector a secuencia. Por ejemplo, la entrada de la red podría ser una imagen y la salida un título para dicha imagen.

Como la salida de una neurona recurrente en un instante de tiempo dado está compuesta por entradas de los instantes de tiempo anteriores, es acertado decir que una neurona recurrente posee memoria. Se le denomina *memory cell* a aquella parte de la red neuronal encargada de preservar el estado a través del tiempo. Gracias a esta memoria interna, las redes RNN son adecuadas para problemas de aprendizaje automático donde se utilizan datos secuenciales. Las redes RNN pueden recordar información relevante sobre la entrada que recibieron, lo que permite una mayor precisión en la predicción de instantes futuros a diferencia de otros tipos de redes, que no son capaces de recordar lo sucedido en el pasado.

En las redes neuronales, se realiza un Forward Propagation para aplicar el modelo y verificar si el resultado obtenido es correcto o incorrecto y obtener de esta manera la función de pérdida.

A continuación, se realiza un Backward Propagation, que se utiliza para encontrar las derivadas parciales del error con respecto a los pesos de las neuronas. Dichas derivadas son utilizadas por el algoritmo de descenso de gradiente para ajustar los pesos de estas neuronas y disminuir una función dependiendo de cómo disminuye la función de pérdida.

Un gradiente contiene las derivadas parciales respecto a los pesos, esto sirve para medir cuánto varía la salida de una función al modificar las entradas. También se puede ver como la pendiente de una función en un punto, de forma que cuanto más elevado es el gradiente, más pronunciada es la pendiente, y, por lo tanto, más rápido puede aprender un modelo. Por el contrario, si la pendiente es nula, entonces el modelo se detiene y no aprende.

4.4. Long short-term memory (LSTM)

Long Short Term Memory (LSTM) es una extensión de las redes neuronales recurrentes, que básicamente amplían su memoria para aprender de experiencias pasadas a largo plazo.

Los sistemas LSTM son útiles para el aprendizaje profundo, ya que pueden procesar tanto datos individuales como secuencias de datos. La memoria de largo y corto plazo es también una red neuronal recurrente, pero se diferencia en que mientras otras redes repiten el módulo cada vez que la entrada recibe nueva información, las redes LSTM recuerdan el problema durante más tiempo y tienen una estructura similar a una cadena para repetir el módulo.

Como se puede observar en la **Figura 15**, la estructura de una red LSTM es más compleja que la estructura de una red neuronal recurrente clásica.

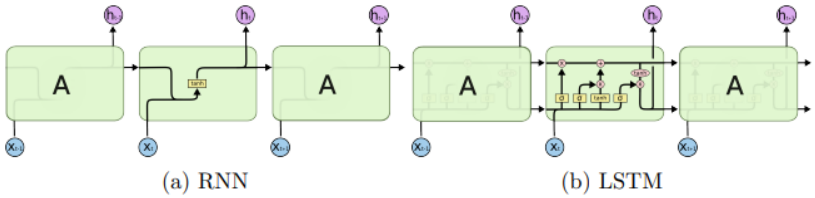

Figura 15. *Estructura RNN vs LSTM.*

En la estructura de la red LSTM se puede apreciar una línea horizontal que va desde principio a fin de las conexiones recurrentes, ésta es conocida como estado de celda, es la encargada de transportar la información a lo largo del tiempo.

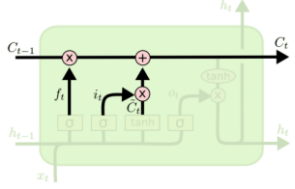

Figura 16*. Línea de estado de celda.*

Existen tres puertas encargadas de regular la información que fluye a lo largo del estado de celda, son las siguientes:

- **Puerta de olvido**. Se encarga de controlar qué información es descartada y cuál no del estado de celda.

- **Puerta de entrada.** Decide qué nueva información es añadida al estado de celda.

- **Puerta de salida**. Es la encargada de controlar la salida de la red neuronal.

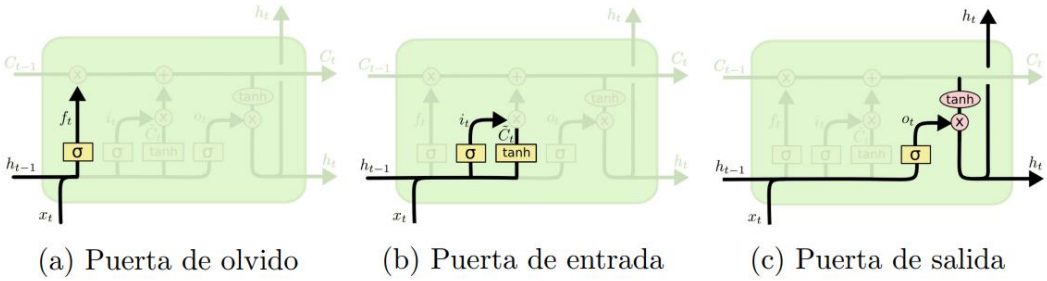

***Figura 17**. Puertas que componen una celda LSTM.*

Para entrenar una red LSTM, el primer paso es decidir qué información va a ser eliminada del estado de la celda, esta tarea recae sobre la puerta de olvido. Utiliza una función sigmoide (σ_g) para poder generar valores en el rango de 0 y 1 para cada estado de la celda C_{t-1}, de esta manera, se indica con un 0 que se deshace de un estado, por el contrario, se indica con un 1 que se mantiene el estado (**Figura 18**). La ecuación resultante de la puerta de olvido es la siguiente:

$$f_t = \sigma_g\,(U_{xf}x_t + W_{hf}h_{t-1} + b_f)$$

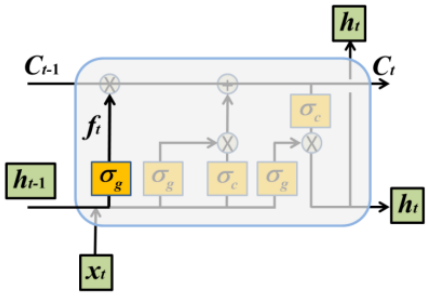

***Figura 18.** Puerta de olvido de una celda LSTM.*

A continuación, se decide qué información se añade al estado de la celda. Este proceso se realiza en dos partes, primeramente, la puerta de entrada (i_t) decide qué valores se actualizan, luego, se crean nuevos valores candidatos a partir de los pesos iniciales gracias a una capa (σ_c). Una puerta lógica los combina para generar así el nuevo estado (**Figura 19**).

De esta forma, las ecuaciones de la capa de entrada y la capa g_t son las siguientes:

$$i_t = \sigma_g\,(U_{xi}x_t + W_{hi}h_{t-1} + b_i)$$

$$g_t = \sigma_c\,(U_{xc}x_t + W_{hc}h_{t-1} + b_c)$$

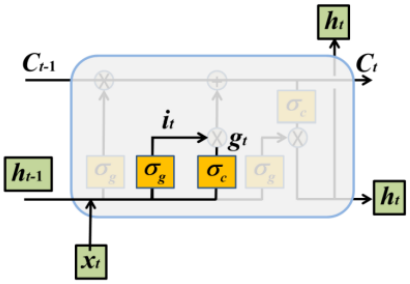

***Figura 19**. Puerta de entrada de una celda LSTM.*

Por último, se obtiene el resultado de salida que depende del estado de la celda. Se aplica una función sigmoidea; la cual decide las partes del estado de la celda que contribuyen en la salida. A continuación, se aplica una función tanh, que normaliza los valores dentro del rango -1 y 1. Finalmente, se multiplican ambas salidas entre ellas (**Figura 20**).

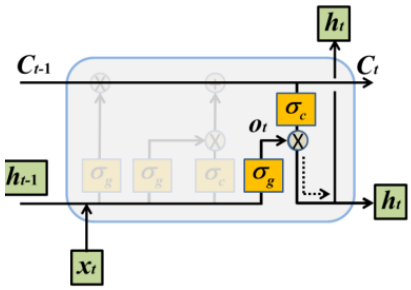

***Figura 20.** Puerta de salida de una celda LSTM.*

Capítulo 5. Sensores Inerciales

En este Trabajo de Fin de Grado se han utilizado sensores inerciales para poder determinar la orientación y posición en los ejercicios realizados. Durante este capítulo se explica el funcionamiento de estos sensores y cómo obtener los datos registrados en los movimientos.

Los sensores inerciales o Unidad de Medida Inercial (Inertial Measurement Unit, IMU) son dispositivos electrónicos que permiten medir la orientación de un objeto, a partir de los datos registrados del acelerómetro, giroscopio y magnetómetro para las tres dimensiones X, Y, Z.

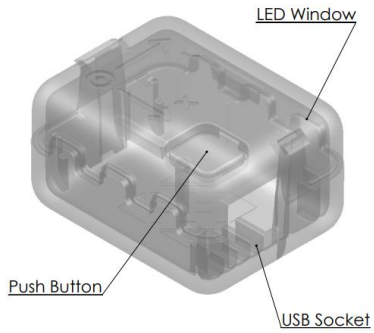

***Figura 21**. Sensor inercial X-BIMU*

Gracias a la facilidad y simplicidad de utilización de este tipo de sensores, se consigue que el proceso de recogida de datos no sea tan intrusivo, teniendo que llevar solamente el sensor, sin necesidad de cables, lo que otorga una mayor libertad en el movimiento. Esto es posible gracias a que los sensores X-BIMU se conectan de forma inalámbrica al módulo conectado al ordenador, haciendo posible que la recogida de los datos de los movimientos sea en tiempo real, lo que facilita este proceso.

Los pasos para seguir en el proceso de recogida de datos son los mismos para todos los ejercicios. Desde la aplicación (X-BIMU Terminal.exe) dada por el fabricante del sensor, se accede al puerto serie del sensor. Una vez se ha establecido la conexión, seleccionamos la opción de registrar los datos recogidos en un fichero y dejamos pasar 20 segundos como tiempo de estabilización, tras este periodo, se comienza a realizar el ejercicio en cuestión. Para finalizar, tenemos que terminar de grabar los datos en la carpeta seleccionada, obteniendo así tres archivos con extensión .csv que contienen los datos de la batería, las posiciones triaxiales, y las mediciones del acelerómetro, magnetómetro y giroscopio.

En el apartado **6.1. Recolección de datos** se explica el proceso de recogida de los datos para cada uno de los ejercicios de una forma más específica, para luego procesarlos en el capítulo **6.2. Pre-procesado de los datos** y obtener el formato adecuado de entrada a la red neuronal.

A continuación, se explica detalladamente la función de cada uno de estos componentes que conforman el sensor:

- **Acelerómetro**. Es un instrumento encargado de medir la aceleración en los tres ejes. Un acelerómetro triaxial se compone por tres acelerómetros orientados ortogonalmente entre sí, dando información de cada uno de los ejes de forma independiente.

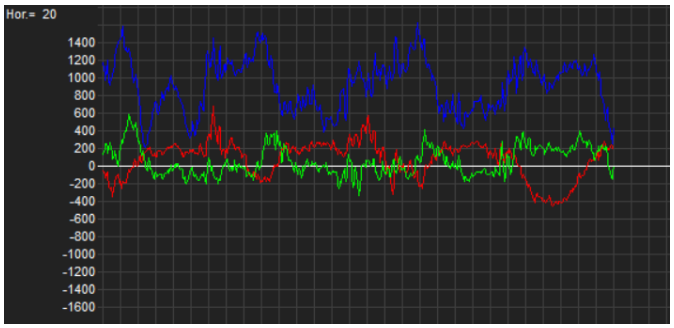

***Figura 22**. Gráfico de la aceleración medida por IMU.*

- **Giroscopio**. Es un instrumento que mide la velocidad angular de rotación de un objeto respecto al sistema de referencia inercial. Se implementa utilizando la tecnología Micro-machined Electro Mechanical System (MEMS), cuya salida es un voltaje que varía en función de la velocidad angular. Dicha salida se mide en grados por segundo y si la integramos podemos obtener el ángulo de rotación del objeto respecto de un eje.

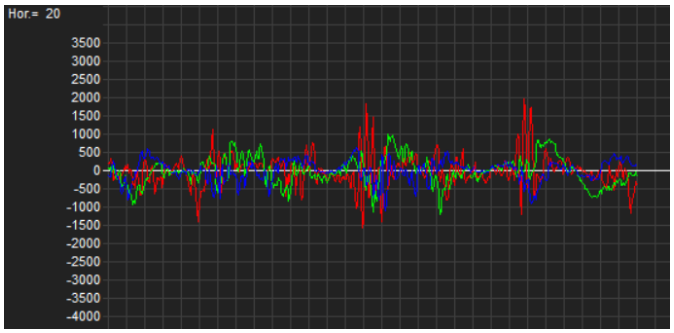

***Figura 23**. Gráfico medido por el giroscopio del IMU.*

- **Magnetómetro**. Es un instrumento capaz de medir la dirección del campo magnético y su fuerza.

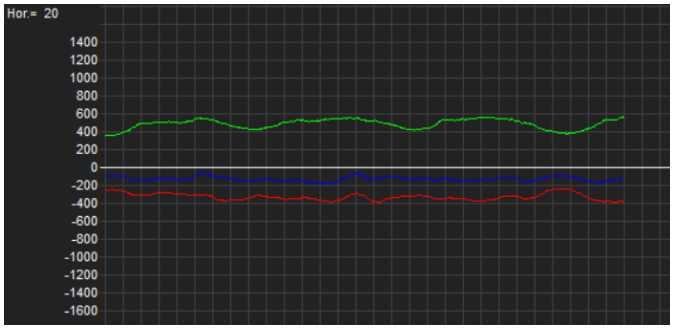

***Figura 24**. Gráfico medido por el magnetómetro del IMU.*

Capítulo 6. Conjunto de datos

En este capítulo se va a hablar de los procesos que se han seguido para la recolección de datos de los sensores y las diferentes técnicas que se han empleado para realizar un preprocesamiento de dichos datos para adecuarlos a la red neuronal.

6.1. Recolección de datos

En este caso, el conjunto de datos se basa en las lecturas realizadas con los sensores X-BIMU mientras se realizaban las diferentes actividades físicas. Las actividades físicas que se han realizado son las siguientes:

- **Sentadillas.** En este ejercicio el sensor se sujeta en la cadera. Cada repetición comienza con el cuerpo erecto y rodillas completamente bloqueadas (**Figura 25**). A continuación, se flexionan las rodillas hasta un ángulo de unos 45º y la cadera sin perder la verticalidad del movimiento (**Figura 26**), para finalmente volver a la posición inicial.

Figura 25. *Inicio de la repetición de una sentadilla.*

Figura 26. *Fin de la repetición de una sentadilla.*

- **Peso muerto.** En este caso, el sensor estaba sujeto en la barra para conseguir una mayor precisión en el rango de movimiento del ejercicio. Capada repetición comienza con la barra en el suelo y piernas flexionadas (**Figura 27**), finaliza cuando se ha completado el rango de movimiento, llevando así la barra hasta la altura de la cintura y con las rodillas totalmente bloqueadas (**Figura 28**).

Figura 27. *Inicio de la repetición de peso muerto.*

Figura 28. *Fin de la repetición de peso muerto.*

- **Flexiones.** En este ejercicio, el sensor se sujeta en el pecho, para poder tener mayor precisión en el rango de movimiento. Al comienzo de la repetición se ha de estar frente al suelo, apoyado con los pies y manos, con los codos completamente bloqueados (**Figura 29**). A continuación, se han de flexionar los codos a un ángulo de 45º aproximadamente hasta rozar el suelo con el pecho (**Figura 30**).

Figura 29. *Inicio de la repetición de una flexión.*

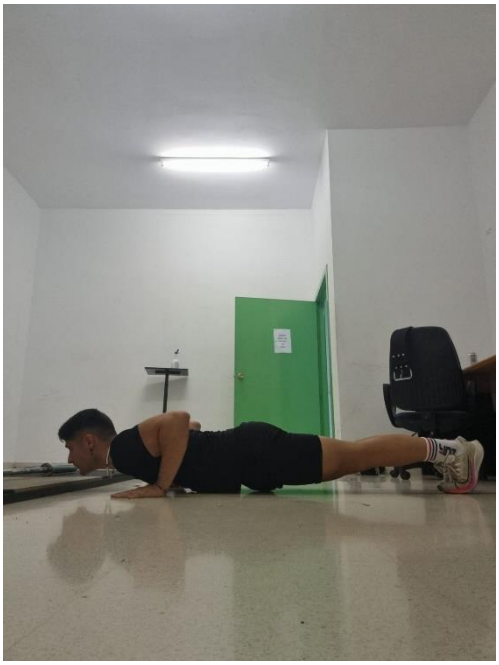

Figura 30. *Fin de la repetición de una flexión.*

- **Dominadas**. El sensor se sujeta en el pecho a la altura de la clavícula. Para realizar el ejercicio hay que colgarse de la barra agarrándose con las manos, dejando el resto del cuerpo suspendido en el aire, con los brazos totalmente bloqueados y dorsal completamente estirado (**Figura 31**). A continuación, hay que realizar un tirón hacia arriba, flexionando los brazos y llegando a superar la barra con la barbilla (**Figura 32**).

***Figura 31**. Inicio de la repetición de una dominada.*

***Figura 32.** Fin de la repetición de una dominada.*

- **Fondos.** El sensor se sujeta en el pecho a la altura de la clavícula. Para realizar el ejercicio hay que colocar las manos en la barra con la misma anchura que los hombros, en la posición inicial los brazos estarán completamente estirados, bloqueando los codos (**Figura 33**). A continuación, desciende todo el cuerpo flexionando los codos hasta llegar a un ángulo de 45º como máximo (**Figura 34**).

Figura 33. *Inicio de la repetición de fondos.*

Figura 34. *Fin de la repetición de fondos.*

Cada una de las actividades posee un conjunto de datos único. Por cada ejercicio se han realizado 10 series compuestas por 20 repeticiones cada serie, excepto en dominadas que las series se componen de 12 repeticiones. El procedimiento escogido para realizar las mediciones con el sensor es dejar un tiempo de 20 segundos al inicio de cada serie, para tener de esta forma un tiempo de establecimiento igual en todas las series.

En la **Figura 35** se puede observar la estructura de los datos recogidos, que están compuestos por las mediciones del giroscopio, acelerómetro y magnetómetro.

	time (ms)	gyroscope x (0.1 deg/s)	gyroscope y (0.1 deg/s)	gyroscope z (0.1 deg/s)	accelerometer x (mg)	accelerometer y (mg)	accelerometer z (mg)	magnetometer x (mG)	magnetometer y (mG)	magnetometer z (mG)
0	0	72	-8	56	255	-114	903	-202	435	-82
1	15	54	-11	54	250	-118	897	-205	435	-79
2	15	31	-5	55	248	-117	894	-202	435	-82
3	37	11	11	45	247	-115	886	-202	435	-84
4	37	11	5	37	255	-127	898	-202	433	-82
...	...	...	...	...	...	...	...	...	...	...
4791	75823	33	25	-12	-16	-124	895	-194	556	-15
4792	75823	19	24	-4	-36	-119	921	-200	548	-12
4793	75855	-33	-32	19	-51	-104	945	-202	551	-10
4794	75855	-27	-31	20	-21	-119	908	-202	553	-10
4795	75877	-28	13	15	-11	-146	892	-197	551	-12

Figura 35. *Estructura del conjunto de datos*

Además de las mediciones registradas del acelerómetro, giroscopio y magnetómetro, son necesarios los vectores de posición de los tres ejes X, Y, Z, para ello, es necesario implementar un programa secundario que se conecta al puerto serie del sensor y registrar estos valores. Se ha implementado de la siguiente manera (**Figura 36**).

```
ser = serial.Serial('COM1', 115200, timeout=15)

ser_io = io.TextIOWrapper(io.BufferedRWPair(ser, ser, 1),
newline = '\r',
line_buffering = True)

archivo = open('Direccion del archivo', 'w')
time.sleep(1)
```

Figura 36. *Código para registrar los vectores de posición del IMU.*

Con este código podremos registrar los datos en un archivo con extensión .csv previamente creado, para ello, hay que especificar la dirección de la ruta del archivo en cuestión.

6.2. Pre-procesado de los datos

En este caso concreto, los datos deben ser filtrados correctamente para adecuarlos a la red neuronal del algoritmo. A continuación, se habla de las modificaciones que se han realizado:

- En primer lugar, se deben realizar unas pequeñas modificaciones de los datos en crudo que ha recolectado el sensor. Se descartan las medidas triaxiales del magnetómetro, no son necesarias para este algoritmo en concreto, por lo que las eliminamos para dejar los datos lo más simples posibles. Además, es necesario unir a los datos los parámetros que indican las posiciones de cada eje X, Y, Z a la tabla vista en la figura anterior. Para ello, se crea una función llamada setDataFrame() a la cual se le pasa como argumento la ruta de los archivos .csv registrados por el sensor (**Figura 37**).

```
def setDataFrame(sensor_file, quaternion_file):
  df1 = pd.read_csv(sensor_file)
  df1.drop(['packet counter'], axis=1, inplace=True)
  df1.drop(['magnetometer x (mG)'], axis=1, inplace=True)
  df1.drop(['magnetometer y (mG)'], axis=1, inplace=True)
  df1.drop(['magnetometer z (mG)'], axis=1, inplace=True)

  df2 = pd.read_csv(quaternion_file)
  df2.drop(['packet counter'], axis=1, inplace=True)

  df = pd.merge(df1,df2)

  return df
```

Figura 37. *Función para crear el DataFrame.*

Ahora la matriz del dataset es una matriz de 11 columnas (**Figura 38**).

	time (ms)	gyroscope x (0.1 deg/s)	gyroscope y (0.1 deg/s)	gyroscope z (0.1 deg/s)	accelerometer x (mg)	accelerometer y (mg)	accelerometer z (mg)	w (10000)	x (10000)	y (10000)	z (10000)
0	31	7	16	-9	381	297	-955	-505	8446	4604	2619
1	47	28	-1	-11	384	305	-938	-502	8446	4608	2615
2	47	12	0	-16	377	313	-937	-502	8446	4608	2615
3	78	4	8	-13	372	331	-939	-499	8444	4613	2609
4	78	-23	0	8	371	333	-914	-499	8444	4613	2609
...	...	...	...	...	...	...	...	...	...	...	...
3974	66497	23	-2	18	574	-790	-158	6449	6545	3865	531
3975	66518	2	-6	16	566	-808	-158	6453	6543	3862	526
3976	66518	12	0	27	578	-806	-171	6453	6543	3862	526
3977	66549	15	10	19	571	-840	-165	6454	6545	3859	521
3978	66567	-10	2	17	568	-833	-168	6455	6546	3854	517

Figura 38. *Estructura completa del conjunto de datos.*

Para la captura de las posiciones triaxiales X, Y, Z es necesario implementar un programa capaz de registrar los datos capturados por el sensor a través del puerto serie como en la **Figura 36**.

- A continuación, es necesario calcular el módulo de la aceleración y normalizarlo, de esta forma somos capaces de observar si existe periodicidad, para ello, se crea una función llamada set_acc_accnorm() la cual toma como argumento el dataFrame anterior y crea dos features nuevos, que se corresponden con la aceleración y el módulo de la aceleración respectivamente (**Figura 39**). Esta función nos devuelve la gráfica del módulo de la aceleración normalizado. (**Figura 40**).

```
def set_acc_accnorm(df):
  df['acc'] = (df.iloc[:,4]**2+df.iloc[:,5]**2+df.iloc[:,6]**2)**0.5
  df['accnorm'] = (df.acc - df.acc.min())/(df.acc.max() - df.acc.min())

  return df.accnorm
```

Figura 39. *Función para calcular el módulo de la aceleración.*

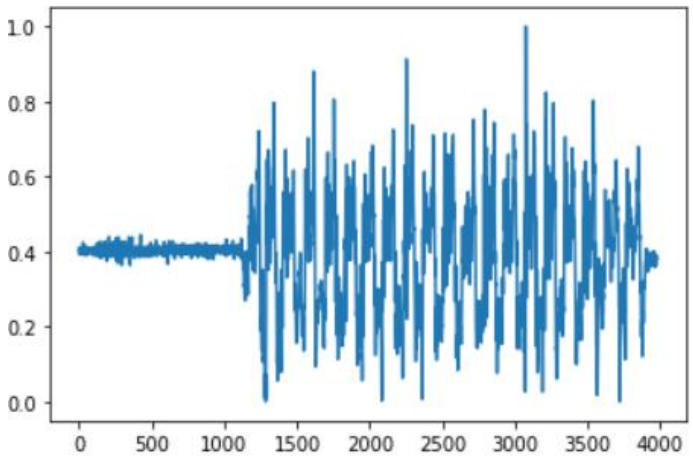

Figura 40. *Gráfica del módulo de la aceleración normalizado.*

- Una vez se ha normalizado el módulo de la aceleración, hay que calcular los puntos por donde la señal de aceleración normalizada cruza el umbral, de esta manera podemos determinar el inicio y fin de cada repetición. Para ello, primeramente, hay que filtrar la señal en paso-bajo mediante la librería scipy.signal, haciendo uso de las clases butter y lfilter. Con la clase scipy.signal.butter obtenemos los coeficientes necesarios para crear el filtro paso-bajo, el cual se calcula con la clase scipy.signal.lfilter (**Figura 41**).

```
# Primero filtramos la señal paso-bajo
x = df.accnorm.values
fs = len(df)/df.iloc[0,-1]*0.001
b, a = butter(2, 0.25, fs=fs, btype='low', analog=False)
y = lfilter(b, a, x)
```

Figura 41. *Código filtro paso bajo de la señal.*

Una vez se ha obtenido la señal filtrada, hay que calcular los puntos de cruce en el umbral, es decir, obtener en que puntos la señal corta dicho umbral. En cada repetición, existen dos puntos de corte, uno cuando la señal baja del umbral (*crossdown*) y otro cuando la señal sobrepasa el umbral (*crossup*).

Para obtener los puntos *crossup*, se genera una lista vacía y se va rellenando cuando la señal filtrada en el instante de tiempo actual se encuentra por debajo del umbral, y en el instante de tiempo futuro es mayor que dicho umbral (**Figura 42**).

```
crossup = []
jump = 0
for i in range(y.shape[0]-1):
  if jump>0:
      jump -= 1
      continue
  if y[i]<=ylimit and y[i+1]>ylimit:
      crossup.append(i)
      jump = 50
```

Figura 42. *Puntos de cruce cuando la señal sobrepasa el umbral.*

Por el contrario, para obtener los puntos *crossdown*, la lista se va rellenando cuando la señal filtrada en el instante de tiempo actual se encuentra por encima del umbral, y en el instante de tiempo futuro se encuentra por debajo de dicho umbral (**Figura 43**).

```
crossdown = []
jump = 0
for i in range(y.shape[0]-1):
  if jump>0:
      jump -= 1
      continue
  if y[i]>=ylimit and y[i+1]<ylimit:
      crossdown.append(i)
      jump = 50
```

Figura 43. *Puntos de cruce cuando la señal baja del umbral.*

La señal resultante tras realizar el filtrado paso-bajo es la señal de color naranja. Los puntos verdes indican cuándo la señal baja del umbral, y los puntos rojos indican cuándo la señal sube del umbral (**Figura 44**).

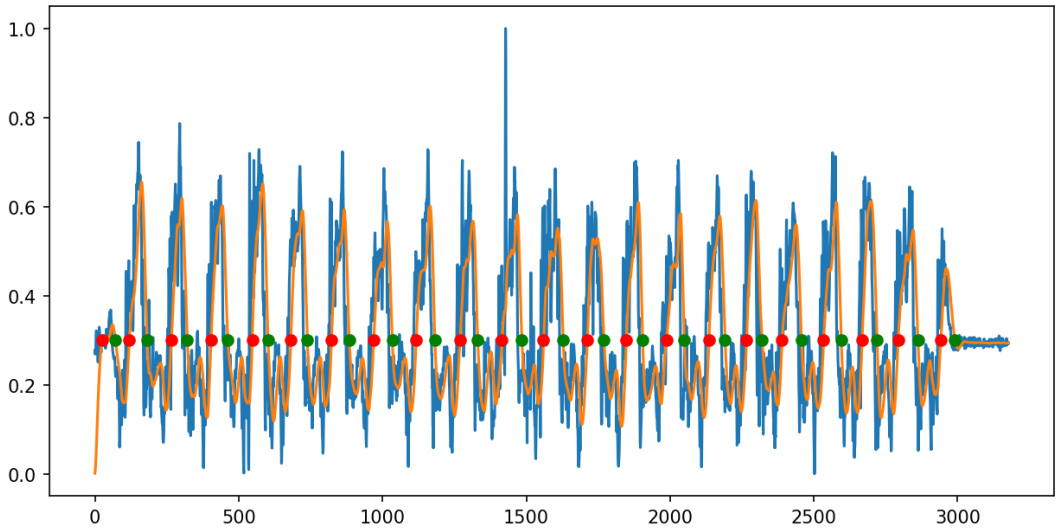

Figura 44. *Puntos donde el módulo de la aceleración cruza el umbral.*

- Por último, es necesario crear una matriz por cada ejercicio que contenga las repeticiones (**Figura 45**).

```
numreps = random.choices(range(1, REP +1), k=NUMDATOS)

for numrep in numreps:
  start = random.randint(0,REP-numrep)
  stop = start + numrep
  start_sample = crossdown[start] + int(random.gauss(mu=0,sigma=SIGMA))
  if start_sample==0:
      start_sample=0
  stop_sample = crossdown[stop] + int(random.gauss(mu=0,sigma=SIGMA))
  if stop_sample>MAX_SAMPLE:
      stop_sample=MAX_SAMPLE

  #print(crossdown[0], crossdown[i+1])
  #print(start, numrep, stop, start_sample, stop_sample)
  data = y[start_sample:stop_sample]
  data = data[1::SAMPLING]
  X.append(data)
  t.append(numrep)

X = np.array(X)
t = np.array(t)-1
```

Figura 45. *Código para obtener las matrices de las repeticiones.*

De esta manera obtenemos las matrices X, t de cada ejercicio, las cuales contienen los datos de la repetición en cuestión y el número de dicha repetición respectivamente.

En la **Figura 46** podemos observar de una manera más intuitiva lo anterior. Se representan las matrices X, t de forma aleatoria, mostrando las gráficas de cada repetición que se corresponde con los datos de la matriz X, mientras que la matriz t, la cual indica el número correspondiente a cada una de las repeticiones, queda representada en la esquina superior derecha de cada gráfica.

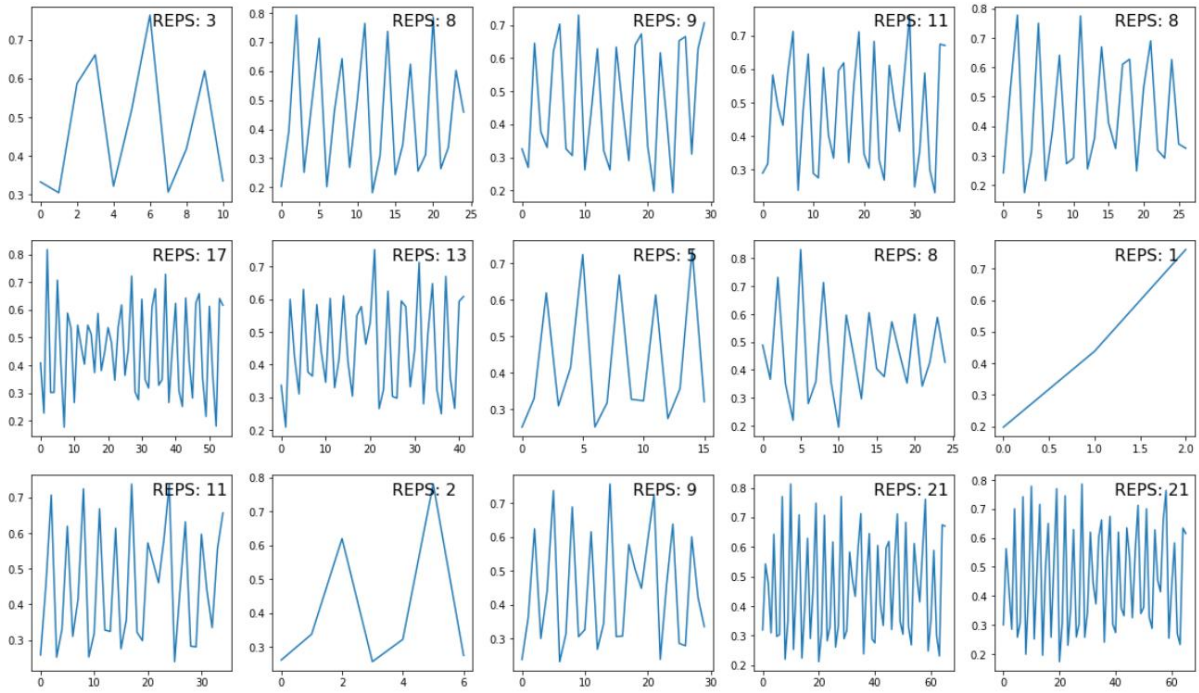

Figura 46. *Matriz con las diferentes repeticiones.*

Capítulo 7. Desarrollo de la red neuronal

Durante este capítulo se explica la arquitectura de red neuronal desarrollada en este Trabajo Fin de Grado así como las técnicas utilizadas para entrenar correctamente la red neuronal.

7.1. Unificación del dataset

Una vez que se han procesado los datos de cada ejercicio llamando a las funciones indicadas en el apartado (**6.2. Pre-procesado de los datos**), tenemos que unificar los datos de entrenamiento, de forma que podamos pasar todos los datos a la red neuronal a la misma vez. Se suman las repeticiones de cada ejercicio para obtener un total de repeticiones y se crean dos arrays X_total y t_total, de 8000 filas cada uno. X_total está compuesto por todos los datos recogidos de todos los ejercicios mientras que t_total indica las repeticiones de todos los ejercicios (**Figura 47**).

```
REP_total = REP_sentadillas + REP_dominadas + REP_pesomuerto + REP_flexiones

X = np.r_[X_sentadillas, X_dominadas, X_pesomuerto, X_flexiones]
t = np.r_[t_sentadillas, t_dominadas, t_pesomuerto, t_flexiones]

X.shape, t.shape
```

Figura 47. *Unificación del dataset.*

7.2. Enmascaramiento y relleno

El dataset es un array multidimensional, donde cada dato (muestra de las aceleraciones) es un array que tiene una dimensión variable, con un número fijo de columnas o features. La primera repetición tiene menor longitud que la segunda repetición, así sucesivamente. Esto se debe a que la última repetición engloba a todas las anteriores. Para garantizar que todos los datos de la secuencia de entrada tengan la misma longitud, es necesario rellenar o truncar los datos de entrada.

Habría que calcular cuál es la secuencia máxima que hay en cada uno de los ejercicios, pero para que no haya problemas de dimensionamiento a la hora de realizar las pruebas para diferentes ejercicios, se ha decidido poner una secuencia máxima de 100 y asegurar así el correcto funcionamiento (**Figura 48**).

```
# Padding and Masking
special_value = -1.0
max_seq_len = 100
#np.max([X[i].shape[0] for i in range(X.shape[0])])
print(f'max_seq_length: {max_seq_len}')

Xpad = np.full((X.shape[0], max_seq_len, 1), fill_value=special_value)
for s, x in enumerate(X):
    seq_len = x.shape[0]
    Xpad[s, 0:seq_len, :] = x.reshape([-1,1])
```

Figura 48. *Código de relleno de los datos.*

El enmascaramiento es una manera de contar capas de secuencia de procesamiento que ciertos timesteps en una entrada faltan, y por lo tanto, debe ser saltado al procesar los datos.

El relleno de los datos es una forma especial de enmascaramiento, donde los datos son enmascarados al principio o al final de una secuencia. El relleno proviene de la necesidad de codificar datos de secuencias en lotes contiguos. Para que todas las secuencias de datos se ajusten a una longitud estándar determinada, es necesario rellenar o truncar algunas de estas secuencias, es decir, será necesario rellenar aquellas secuencias con una longitud de datos menor para igualar la longitud de datos máxima.

Una vez que todas las muestras tienen una longitud uniforme, se debe informar al modelo de que una parte de los datos se está rellenando y debe ser ignorada. En este modelo, se ha decido introducir el enmascaramiento de entrada añadiendo una capa al modelo del tipo keras.layers.Masking [12].

7.3. Diseño y estudio de la red neuronal

En el diseño de la red neuronal es necesario estudiar la arquitectura de red que se va a seguir, en este caso particular, el modelo de red neuronal es del tipo secuencial al que se le añade una capa oculta LSTM indicando el número de neuronas de dicha capa, que en nuestro caso es 100. Además, se añaden dos capas DENSE, una con una función de activación ‘sigmoid’ y la última capa que realiza la función de capa de salida tiene una función de activación ‘softmax’, ya que elige el número de repeticiones.

La función de activación sigmoide es similar a la función de escalón, acota la señal entre los valores de 0 y 1, sin embargo, la curva que presenta es suave y si es diferenciable, por lo tanto, se puede calcular la derivada distinta de cero.

La fórmula de la función sigmoide es la siguiente:

$$\frac{1}{1 + e^{-x}}$$

Y se representa como en el gráfico (**Figura 49**).

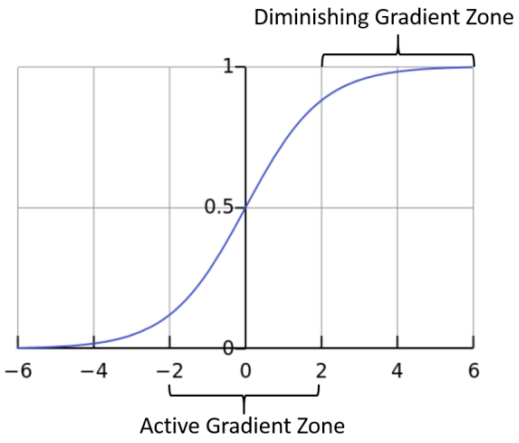

Figura 49. *Gráfico de la función sigmoide.*

La función de activación softmax suele ser utilizada en redes de clasificación, en nuestro caso queremos que la red nos indique en qué número de repetición nos encontramos. Convierte un vector de N números reales en una distribución de probabilidad con N resultados posibles. La función softmax se usa generalmente como la última función de activación de una red neuronal para normalizar la salida de una red.

La fórmula de la función softmax es la siguiente:

$$s(xi) = \frac{e^{xi}}{\sum_{j=1}^{n} e^{xj}}$$

Para compilar el modelo se utiliza métrica 'sparse_categorical_crossentropy' ya que queremos que el resultado sea una codificación de números enteros, por el contrario, si utilizáramos la métrica 'categorical_crossentropy' obtendríamos un resultado con una codificación one-hot.

Así es como se han implementado las capas a la red neuronal (**Figura 50**).

```
UNITS = 100
model = Sequential()
model.add(Masking(mask_value=special_value, input_shape=(max_seq_len, 1)))
model.add(LSTM(UNITS))
model.add(Dense(UNITS, activation='sigmoid'))
model.add(Dense(REP, activation='softmax'))
model.compile(loss='sparse_categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
```

Figura 50. *Código para implementar el modelo de Red Neuronal.*

De esta manera, la red neuronal quedaría con la estructura que se aprecia en la **Figura 51**.

```
_________________________________________________________________
 Layer (type)                 Output Shape              Param #
=================================================================
 masking_1 (Masking)          (None, 66, 1)             0

 lstm_1 (LSTM)                (None, 100)               40800

 dense_2 (Dense)              (None, 100)               10100

 dense_3 (Dense)              (None, 21)                2121

=================================================================
Total params: 53,021
Trainable params: 53,021
Non-trainable params: 0
```

Figura 51. *Estructura del modelo de la Red Neuronal.*

Para diseñar la red neuronal, además de tener en cuenta la arquitectura de red, es necesario tener presente una serie de parámetros para determinar el comportamiento de dicha red durante el entrenamiento.

- Units. Este parámetro indica el número de neuronas LSTM que tendrá la capa recurrente.

- Epochs. Indica el número de veces que itera el conjunto de entrenamiento en la red neuronal. El modelo se actualiza cada vez que se procesa un lote, lo que significa que se puede actualizar múltiples veces durante un epoch.

- Batch_size. Define el número de muestras que se van a propagar por la red, de esta forma, se puede determinar el número de iteraciones realizadas en cada época de entrenamiento.

- Learning_rate. Se conoce como tasa de entrenamiento. En el caso de que sea demasiado grande el entrenamiento fallará, por el contrario, si es demasiado pequeña no mejorará el entrenamiento de la red.

- Early Stopping. Se trata de una técnica de regularización utilizada mediante una validación durante el entrenamiento. Se encarga de parar el entrenamiento cuando la métrica utilizada ha dejado de mejorar.

A continuación, se puede observar el código necesario para configurar los parámetros explicados anteriormente (**Figura 52**).

```
earlystopping_cb = EarlyStopping(monitor='loss', patience=20)
checkpoint_cb = ModelCheckpoint(modelname, save_best_only=True, verbose=1)

model.fit(X_train, t_train, validation_data=(X_valid, t_valid), epochs=500,
          batch_size=16, callbacks=[earlystopping_cb, checkpoint_cb], verbose=1)
```

Figura 52. *Configuración de parámetros para el entrenamiento de la red.*

7.4. Entrenamiento de la red neuronal

Una vez se ha configurado correctamente el modelo con los valores de los hiperparámetros, como se ve en la figura anterior, es hora de entrenarlo pasándole los datos del set de entrenamiento.

Enseguida, en las primeras épocas, el modelo es capaz de alcanzar una precisión del 100%. Como el modelo parará de entrenar cuando ya no sea capaz de mejorar la métrica de la función de pérdida, estos son los resultados durante el entrenamiento (**Figura 53**).

```
Epoch 1/500
93/94 [============================>.] - ETA: 0s - loss: 2.4316 - accuracy: 0.1653
Epoch 1: val_loss improved from inf to 1.87288, saving model to model.h5
94/94 [==============================] - 10s 59ms/step - loss: 2.4279 - accuracy: 0.1673 - val_loss: 1.8729 - val_accuracy: 0.5040
Epoch 2/500
93/94 [============================>.] - ETA: 0s - loss: 1.5465 - accuracy: 0.5726
Epoch 2: val_loss improved from 1.87288 to 1.29658, saving model to model.h5
94/94 [==============================] - 4s 42ms/step - loss: 1.5438 - accuracy: 0.5740 - val_loss: 1.2966 - val_accuracy: 0.6920
Epoch 3/500
93/94 [============================>.] - ETA: 0s - loss: 1.0579 - accuracy: 0.7473
Epoch 3: val_loss improved from 1.29658 to 0.82881, saving model to model.h5
94/94 [==============================] - 4s 42ms/step - loss: 1.0553 - accuracy: 0.7487 - val_loss: 0.8288 - val_accuracy: 0.8600
Epoch 4/500
94/94 [==============================] - ETA: 0s - loss: 0.7493 - accuracy: 0.8753
Epoch 4: val_loss improved from 0.82881 to 0.59517, saving model to model.h5
94/94 [==============================] - 4s 42ms/step - loss: 0.7493 - accuracy: 0.8753 - val_loss: 0.5952 - val_accuracy: 0.8840
Epoch 5/500
93/94 [============================>.] - ETA: 0s - loss: 0.5182 - accuracy: 0.9483
Epoch 5: val_loss improved from 0.59517 to 0.40181, saving model to model.h5
94/94 [==============================] - 4s 41ms/step - loss: 0.5177 - accuracy: 0.9487 - val_loss: 0.4018 - val_accuracy: 1.0000
Epoch 6/500
93/94 [============================>.] - ETA: 0s - loss: 0.3504 - accuracy: 0.9906
Epoch 6: val_loss improved from 0.40181 to 0.38900, saving model to model.h5
94/94 [==============================] - 4s 42ms/step - loss: 0.3507 - accuracy: 0.9907 - val_loss: 0.3890 - val_accuracy: 0.9080
Epoch 7/500
94/94 [==============================] - ETA: 0s - loss: 0.3039 - accuracy: 0.9480
Epoch 7: val_loss improved from 0.38900 to 0.23840, saving model to model.h5
94/94 [==============================] - 4s 41ms/step - loss: 0.3039 - accuracy: 0.9480 - val_loss: 0.2384 - val_accuracy: 0.9800
Epoch 8/500
94/94 [==============================] - ETA: 0s - loss: 0.1903 - accuracy: 0.9853
Epoch 8: val_loss improved from 0.23840 to 0.14292, saving model to model.h5
94/94 [==============================] - 4s 42ms/step - loss: 0.1903 - accuracy: 0.9853 - val_loss: 0.1429 - val_accuracy: 1.0000
```

Figura 53. *Comienzo del entrenamiento de la Red Neuronal.*

Cuando la métrica que mide la función de pérdida ya no mejora según avanzan las épocas, el entrenamiento finaliza gracias a la correcta configuración de Early Stopping (**Figura 54**).

```
Epoch 101: val_loss improved from 0.00117 to 0.00100, saving model to model.h5
94/94 [==============================] - 2s 25ms/step - loss: 0.0030 - accuracy: 0.9987 - val_loss: 0.0010 - val_accuracy: 1.0000
Epoch 102/500
92/94 [============================>.] - ETA: 0s - loss: 0.0035 - accuracy: 0.9993
Epoch 102: val_loss improved from 0.00100 to 0.00100, saving model to model.h5
94/94 [==============================] - 2s 25ms/step - loss: 0.0035 - accuracy: 0.9993 - val_loss: 9.9547e-04 - val_accuracy: 1.0000
Epoch 103/500
92/94 [============================>.] - ETA: 0s - loss: 0.0037 - accuracy: 0.9986
Epoch 103: val_loss did not improve from 0.00100
94/94 [==============================] - 2s 26ms/step - loss: 0.0036 - accuracy: 0.9987 - val_loss: 0.0010 - val_accuracy: 1.0000
Epoch 104/500
94/94 [==============================] - ETA: 0s - loss: 0.0033 - accuracy: 0.9993
Epoch 104: val_loss improved from 0.00100 to 0.00088, saving model to model.h5
94/94 [==============================] - 2s 25ms/step - loss: 0.0033 - accuracy: 0.9993 - val_loss: 8.8097e-04 - val_accuracy: 1.0000
```

Figura 54. *Final del entrenamiento de la Red Neuronal.*

7.5. Resultados de los experimentos

Una vez ha concluido el entrenamiento de la red neuronal, es hora de probar su correcto funcionamiento con otros datos de entrada que no sean los utilizados para entrenarla.

Para validar el correcto funcionamiento de la red, se han seguido dos técnicas diferentes:

- Alternar de entre los cinco ejercicios recogidos el utilizado para la parte de validación y entrenar la red neuronal con los otros cuatro ejercicios restantes.
- Pedir a otra persona que realice los mismos ejercicios que los que se utilizan para entrenar la red y comprobar la tasa de acierto de la red para esos mismos ejercicios, pero con rangos de movimientos diferentes, debido a la fisionomía diferente de la persona.

En ambas técnicas los procedimientos seguidos para procesar los datos son los mismos, tal y como se explican en el apartado (**6.2. Pre-procesado de los datos**).

7.5.1. Alternar los datos de entrenamiento y validación.

Recordemos que los cinco ejercicios registrados son sentadillas, dominadas, flexiones, peso muerto y fondos de tríceps. Ahora bien, se han realizado cinco experimentos diferentes en los cuales se alterna el ejercicio que sirve para validar la red neuronal.

1. **Fondos como ejercicio de validación, el resto de los ejercicios sirven como entrenamiento de la red neuronal.**

   En este caso, al entrenamiento de la red neuronal se le pasan los datos unificados de los otros cuatro ejercicios, sentadillas, dominadas, flexiones y peso muerto. Para evaluar la red, se llama a la función evaluate a la que se pasa como argumento los datos procesados de los fondos y su matriz t_fondos con el número de repeticiones. Esta función nos devuelve la precisión con la que la red neuronal predice el nuevo moviemiento, en este caso un 23%. (**Figura 55**).

```
score, accuracy = model.evaluate(Xpad_validation, t_fondos, batch_size = 16)
print(f'Resultados de los fondos como validación')
print(f'Test score: {score}')
print(f'Test accuracy: {accuracy}')
```

```
125/125 [==============================] - 2s 20ms/step - loss: 10.6788 - accuracy: 0.2365
Resultados de los fondos como validación
Test score: 10.67884635925293
Test accuracy: 0.23649999499320984
```

Figura 55. *Resultado de la validación con fondos.*

2. **Dominadas como ejercicio de validación, el resto de los ejercicios sirven como entrenamiento de la red neuronal.**

En este caso, al entrenamiento de la red neuronal se le pasan los datos unificados de los otros cuatro ejercicios, sentadillas, flexiones, peso muerto y fondos. Para evaluar la red, se llama a la función evaluate a la que se pasa como argumento los datos procesados de las dominadas y su matriz t_dominadas con el número de repeticiones. Esta función nos devuelve la precisión con la que la red neuronal predice el nuevo moviemiento, en este caso un 35,4%. (**Figura 56**).

```
score, accuracy = model.evaluate(Xpad, t_dominadas, batch_size = 16)
print(f'Resultados de las dominadas como validación')
print(f'Test score: {score}')
print(f'Test accuracy: {accuracy}')
```

```
125/125 [==============================] - 2s 19ms/step - loss: 1.9677 - accuracy: 0.3540
Resultados de las dominadas como validación
Test score: 1.9676717519760132
Test accuracy: 0.3540000021457672
```

Figura 56. *Resultado de la validación con dominadas.*

3. **Sentadillas como ejercicio de validación, el resto de los ejercicios sirven como entrenamiento de la red neuronal.**

En este caso, al entrenamiento de la red neuronal se le pasan los datos unificados de los otros cuatro ejercicios, dominadas, flexiones, peso muerto y fondos. Para evaluar la red, se llama a la función evaluate a la que se pasa como argumento los datos procesados de las sentadillas y su matriz t_sentadillas con el número de repeticiones. Esta función nos devuelve la precisión con la que la red neuronal predice el nuevo moviemiento, en este caso un 48,95%. (**Figura 57**).

```
score, accuracy = model.evaluate(Xpad_validation, t_sentadillas, batch_size = 16)
print(f'Resultados de usar las sentadillas como validación')
print(f'Test score: {score}')
print(f'Test accuracy: {accuracy}')
```

```
125/125 [==============================] - 3s 21ms/step - loss: 3.1558 - accuracy: 0.4895
Resultados de usar las sentadillas como validación
Test score: 3.1557772159576416
Test accuracy: 0.4894999861717224
```

Figura 57. *Resultado de la validación con sentadillas.*

4. **Flexiones como ejercicio de validación, el resto de los ejercicios sirven como entrenamiento de la red neuronal.**

En este caso, al entrenamiento de la red neuronal se le pasan los datos unificados de los otros cuatro ejercicios, dominadas, sentadillas, peso muerto y fondos. Para evaluar la red, se llama a la función evaluate a la que se pasa como argumento los datos procesados de las flexiones y su matriz t_flexiones con el número de repeticiones. Esta función nos devuelve la precisión con la que la red neuronal predice el nuevo moviemiento, en este caso un 35,89%. (**Figura 58**).

```
score, accuracy = model.evaluate(Xpad, t_flexiones, batch_size = 16)
print(f'Resultados de las flexiones como validación')
print(f'Test score: {score}')
print(f'Test accuracy: {accuracy}')
```

```
125/125 [==============================] - 2s 18ms/step - loss: 4.3188 - accuracy: 0.3590
Resultados de las flexiones como validación
Test score: 4.3187971115112305
Test accuracy: 0.35899999737739563
```

Figura 58. *Resultado de la validación con flexiones.*

5. **Peso muerto como ejercicio de validación, el resto de los ejercicios sirven como entrenamiento de la red neuronal.**

En este caso, al entrenamiento de la red neuronal se le pasan los datos unificados de los otros cuatro ejercicios, dominadas, sentadillas, flexiones y fondos. Para evaluar la red, se llama a la función evaluate a la que se pasa como argumento los datos procesados de las flexiones y su matriz t_flexiones con el número de repeticiones. Esta función nos devuelve la precisión con la que la red neuronal predice el nuevo moviemiento, en este caso un 99,69%. (**Figura 59**).

```
score, accuracy = model.evaluate(Xpad, t_pesomuerto, batch_size = 16)
print(f'Resultados de peso muerto como validación')
print(f'Test score: {score}')
print(f'Test accuracy: {accuracy}')
```

```
125/125 [==============================] - 2s 18ms/step - loss: 0.0267 - accuracy: 0.9970
Resultados de peso muerto como validación
Test score: 0.02671138197183609
Test accuracy: 0.996999979019165
```

Figura 59. *Resultado de la validación con peso muerto.*

7.5.2. Validación con datos de otra persona

En este apartado se le ha pedido a otra persona que realice los mismos ejercicios para comprobar la precisión de la red neuronal cuando se le pasan estos datos, pero con rangos de movimiento diferentes, ya que la fisionomía de cada persona es diferente.

En este caso concreto, la red se entrena con los datos realizados por el autor, una persona que mide 178 cm y se va a probar la validación de la red pasándole datos realizados por una persona que mide 164 cm, la diferencia de altura hace que los rangos de movimiento en los ejercicios sean diferentes.

La red se entrena tal y como se ha visto en el apartado (**7.4. Entrenamiento de la red neuronal**) pasando a la red neuronal los datos unificados en el apartado (**7.1. Unificación del dataset**).

Para esta técnica de validación, se ha decidido probar la red pasando los ejercicios de forma independiente y por último pasándole todos los datos unificados, a continuación, se muestran los resultados.

1. **Validación con dominadas realizadas por otra persona**

   En este caso, la precisión alcanzada por la red neuronal pasándole los datos de las dominadas es del 64,6%. (**Figura 60**).

```
score, accuracy = model.evaluate(Xpad_validation, t_dominadas_2, batch_size = 16)
print(f'Resultados de las dominadas realizadas por otra persona como validación')
print(f'Test score: {score}')
print(f'Test accuracy: {accuracy}')
```

```
125/125 [==============================] - 2s 19ms/step - loss: 1.8578 - accuracy: 0.6460
Resultados de las dominadas realizadas por otra persona como validación
Test score: 1.8577783107757568
Test accuracy: 0.6460000276565552
```

Figura 60. *Validación con dominadas realizadas por otra persona.*

2. **Validación con sentadillas realizadas por otra persona**

   En este caso, la precisión alcanzada por la red neuronal pasándole los datos de las sentadillas es del 31,54%. (**Figura 61**).

```
score, accuracy = model.evaluate(Xpad_validation, t_sentadillas_2, batch_size = 16)
print(f'Resultados de las sentadillas realizadas por otra persona como validación')
print(f'Test score: {score}')
print(f'Test accuracy: {accuracy}')
```

```
125/125 [==============================] - 2s 19ms/step - loss: 6.5859 - accuracy: 0.3155
Resultados de las sentadillas realizadas por otra persona como validación
Test score: 6.585939884185791
Test accuracy: 0.3154999911785126
```

Figura 61. *Validación con sentadillas realizadas por otra persona.*

3. Validación con flexiones realizadas por otra persona

En este caso, la precisión alcanzada por la red neuronal pasándole los datos de las flexiones es del 83,15%. (**Figura 62**).

```
score, accuracy = model.evaluate(Xpad_validation, t_flexiones_2, batch_size = 16)
print(f'Resultados de las flexiones realizadas por otra persona como validación')
print(f'Test score: {score}')
print(f'Test accuracy: {accuracy}')
```

```
125/125 [==============================] - 2s 19ms/step - loss: 0.6815 - accuracy: 0.8315
Resultados de las flexiones realizadas por otra persona como validación
Test score: 0.6814817190170288
Test accuracy: 0.8314999938011169
```

Figura 62. *Validación con flexiones realizadas por otra persona.*

4. Validación con peso muerto realizado por otra persona

En este caso, la precisión alcanzada por la red neuronal pasándole los datos del peso muerto es del 28,85%. (**Figura 63**).

```
score, accuracy = model.evaluate(Xpad_validation, t_pesomuerto_2, batch_size = 16)
print(f'Resultados del peso muerto realizado por otra persona como validación')
print(f'Test score: {score}')
print(f'Test accuracy: {accuracy}')
```

```
125/125 [==============================] - 2s 19ms/step - loss: 6.1952 - accuracy: 0.2885
Resultados del peso muerto realizado por otra persona como validación
Test score: 6.19521427154541
Test accuracy: 0.28850001096725464
```

Figura 63. *Validación con peso muerto realizado por otra persona.*

5. Validación del dataset unificado realizado por otra persona

En este caso, se han unido todos los datos de los ejercicios tal y como se hace en el apartado (**7.1. Unificación del dataset**) y se han paso para probar la validación del modelo, alcanzo una precisión del 52%. (**Figura 64**).

```
score, accuracy = model.evaluate(Xpad_validation, t_validacion, batch_size = 16)
print(f'Resultados de usar la unificación de los ejercicios realizados por otra persona como validación')
print(f'Test score: {score}')
print(f'Test accuracy: {accuracy}')
```

```
500/500 [==============================] - 9s 19ms/step - loss: 3.8301 - accuracy: 0.5204
Resultados de usar la unificación de los ejercicios realizados por otra persona como validación
Test score: 3.8301024436950684
Test accuracy: 0.5203750133514404
```

Figura 64. *Validación de todos los datos realizados por otra persona.*

Capítulo 8. Conclusiones

8.1. Conclusiones sobre los resultados de los experimentos.

En el apartado (**7.5.1. Alternar los datos de entrenamiento y validación.**) podemos observar que la precisión varía según el tipo de ejercicio empleado para la validación, esto se debe a las diferencias y similitudes entre ejercicios. Por ejemplo, en el caso de peso muerto como validación, se obtiene una precisión muy elevada del 99,69%, mientras que para el caso de los fondos de tríceps solamente se llega a una precisión del 23%. Esto se debe a que el rango de movimiento de peso muerto es muy similar al rango de movimiento realizado en sentadillas, por el contrario, los fondos difieren mucho del resto de ejercicios.

Ahora bien, podemos pensar que, si las sentadillas y el peso muerto comparten un rango de movimiento similar, como es posible que en el caso de utilizar las sentadillas como validación se obtenga una precisión del 48,95%, la mitad que obtenemos con el peso muerto. Esto es debido a que las sentadillas tienen un mayor rango de movimiento, es decir, el sensor recorre más distancia que en el caso de peso muerto, esto nos lleva a observar que el movimiento de peso muerto se encuentra casi por completo en el rango de movimiento de las sentadillas, por este motivo se consigue una mayor precisión.

En el apartado (**7.5.2. Validación con datos de otra persona**) podemos observar que se obtiene una mayor precisión en los ejercicios de dominadas y flexiones, esto se debe a que los rangos de movimiento de los ejercicios realizados por ambas personas no son tan distintos. Esto es debido a que, a pesar de la diferencia de altura, en estos ejercicios en concreto no influye tanto, ya que son ejercicios del tren superior y, por lo tanto, al realizarse con los brazos, no existen tantas diferencias entre los rangos de movimiento de ambas personas.

Por el contrario, los ejercicios con menor precisión son las sentadillas y peso muerto, lo cual no es de extrañar, pues si la diferencia de altura es muy elevada, entonces los rangos de movimiento en estos ejercicios también lo serán.

En conclusión, podemos determinar que cuanto más similar es el rango de movimiento entre dos ejercicios diferentes, más precisión tendrá la red neuronal a la hora de predecir el rango de movimiento del segundo ejercicio cuando se le pasa como validación. Si estos ejercicios son muy diferentes, como es el caso de las dominadas y sentadillas, entonces la red neuronal no será capaz de obtener una precisión elevada en su predicción.

8.2. Conocimientos adquiridos

Una vez se ha finalizado el proyecto, se puede hacer una recopilación de todos los conocimientos aprendidos:

- Aprender a utilizar sensores inerciales X-BIMU.
- Aprender desde cero sobre el Machine Learning y sus diferentes usos.
- Aprender los diversos tipos de redes neuronales y cómo configurarlas.
- Uso de librerías como keras, TensorFlow y Pandas.
- Uso de herramientas como Jupyter Notebooks que facilitan la implementación y ejecución del código.
- Procesamiento de datos para adecuarlos a la entrada de la red neuronal.

8.3. Líneas futuras

Como trabajo a futuros se pueden realizar algunas mejoras tanto en la recogida de datos como en el modelo de red neuronal, además de darle otro enfoque al algoritmo:

- Como mejora, se puede incrementar el número de sensores utilizados en la recogida de datos. Si tenemos varios sensores inerciales, por ejemplo, en las articulaciones que realizan el ejercicio, puede ayudar a tomar más datos de los movimientos y de esta manera obtener resultados más precisos.
- También se puede extrapolar el algoritmo a otros ejercicios físicos y añadir una nueva funcionalidad, diferenciar e identificar el tipo de ejercicio que se está realizando, es decir, realizar un clasificador de movimientos que además cuente las repeticiones realizadas. Para este caso se tendría que predecir los patrones de movimiento de cada ejercicio, además del de las repeticiones.
- Una modificación más compleja pero también interesante, es prescindir de los sensores y utilizar una serie de librerías de Python para poder reconocer los movimientos y contabilizar las repeticiones mediante una cámara web integrada. Gracias a la librería MediaPipe [13], se pueden identificar las articulaciones y tomar datos de posición de estas. Es un caso más complejo, ya que, aunque la recogida de datos es casi inmediata y no hay que realizar un preprocesamiento como en el caso de los sensores, sí que habría que diseñar un programa aparte para la utilización de la cámara web.

Bibliografía

[1] Python, «Python,» [En línea]. Available: https://www.python.org/.

[2] Jupyter, «Jupyter,» [En línea]. Available: https://jupyter.org/.

[3] Anaconda, «Anaconda,» [En línea]. Available: https://www.anaconda.com/.

[4] TensorFlow, «TensorFlow,» [En línea]. Available: https://www.tensorflow.org/.

[5] Keras, «Keras,» [En línea]. Available: https://keras.io/.

[6] Google, «Google Colab,» [En línea]. Available: https://colab.research.google.com/.

[7] D. S. S. Bombín, «Sistema de Aprendizaje Profundo para reconocimiento de actividades con sensores con captura de movimientos.,» Valladolid, 2020.

[8] G. B. S. T. a. R. W. Andrea Soro, «Recognition and Repetition Counting for Complex Physical Exercises with Deep Learning,» Suiza, Febrero 2019.

[9] I. G. a. Y. B. a. A. Courville, Ian Goodfellow and Yoshua Bengio and Aaron Courville, MIT Press, 2016.

[10] A. Geron, Hands-on Machine Learning with Scikit-Learn, Keras, and TensorFlow, Canada: O'Reilly Media, 2019.

[11] D. J. Matich, «Redes Neuronales: Conceptos Básicos y Aplicaciones,» 2001.

[12] Keras, «Masking Layer,» [En línea]. Available: https://keras.io/api/layers/core_layers/masking/.

[13] MediaPipe, «MediaPipe,» [En línea]. Available: https://mediapipe.dev/.