



TRABAJO FIN DE GRADO EN INGENIERIA ELÉCTRICA Y  
ELÉTRÓNICA

# **Clasificación de movimiento de animales mediante Redes Neuronales**

Autor: Carlos Martín Cañal

Tutor: Dr. Alejandro Linares Barranco

Fecha de presentación: Marzo 2017

---

TRABAJO FIN DE GRADO EN INGENIERIA ELÉCTRICA Y  
ELÉTRÓNICA

**Clasificación de movimiento de  
animales mediante Redes  
Neuronales**

Autor: Carlos Martín Cañal

Autorización del Tutor:

Agradecimientos:

Al grupo del Equipo Minerva y al tutor de este TFG, por su inestimable ayuda.

Título del trabajo: **Clasificación de movimiento de animales mediante Redes Neuronales**

Autor: Carlos Martín Cañal

Tutor Académico: Dr. Alejandro Linares Barranco

Resumen del trabajo:

En este trabajo, realizado dentro del proyecto MINERVA, se pretende establecer un método para llevar a cabo la clasificación de comportamientos de animales, como andar, trotar o estar parado, haciendo uso de redes neuronales y los datos proporcionados por una IMU.

Palabras clave

MINERVA, redes neuronales, FFT, GPU.

Abstract

In this work, carried out inside the MINERVA Research excellence project of the Andalusian Council, it is intended to establish a method to perform the classification of animals behaviors as walking, standing and trotting, through neuronal networks, using data provided by an IMU at different visits to Doñana National Park..

Keywords

MINERVA, neural networks, FFT, GPU

**INDICE DE CONTENIDOS**

<b>1 RESUMEN Y OBJETIVOS.....</b>	<b>5</b>
<b>2 INTRODUCCIÓN .....</b>	<b>7</b>
2.1 MINERVA.....	7
2.2 ANTECEDENTES DE MINERVA .....	9
2.3 .PREPROCESADO DE LA RED NEURONAL MEDIANTE FFT.....	11
2.3.1 Transformadas de Fourier.....	11
2.3.2 Transformada Discreta de Fourier.....	13
2.3.3 FFT (Fast Fourier Transform).....	15
2.3.4 Papel de la FFT en el preprocesamiento de la red neuronal del Proyecto .....	19
<b>3 REDES NEURONALES ARTIFICIALES.....</b>	<b>20</b>
3.1 INTRODUCCIÓN .....	20
3.2 HISTORIA DE LAS REDES NEURONALES[11].....	20
3.3 TIPOS DE REDES NEURONALES.....	22
3.3.1 Perceptron.....	22
3.3.2 Adaline .....	23
3.3.3 Redes Neuronales Multicapa.....	24
3.3.4 Redes neuronales recurrentes.....	29
3.4 MATLAB. REDES NEURONALES .....	31
3.4.1 Introducción .....	31
3.4.2 Neuronal Network Toolbox.....	31
3.4.3 Modelo de neurona.....	33
3.4.4 Arquitectura de una red multicapa.....	34
3.4.5 FreeIMU.....	35
3.5 ACCELERACIÓN MEDIANTE GPU.....	37
3.5.1 Introducción .....	37
3.5.2 GPU en MATLAB.....	37
<b>4 LA RED NEURONAL.....</b>	<b>39</b>
4.1 TIPO DE RED ELEGIDA .....	39
4.2 REDES NEURONALES EN MATLAB .....	42
4.2.1 Algoritmo de entrenamiento. ....	42
4.2.2 Chequeo de la red en MATLAB.....	43
<b>5 OBTENCIÓN DE LA BASE DATOS PARA ENTRENAMIENTO Y TEST .....</b>	<b>45</b>
5.1 COLLARES .....	45
5.2 EXPERIMENTOS.....	48
5.3 ARCHIVOS EN RAW.....	51
5.4 OBTENCIÓN DE LOS DATOS EN RAW. ....	52
5.5 OBTENCIÓN DE LA BASE DE DATOS PARA ENTRENAMIENTO Y TEST .....	53
5.6 PREPROCESADO DE LAS MUESTRAS.....	54
<b>6 RESULTADOS .....</b>	<b>56</b>
6.1 RESULTADOS CON ACCELERÓMETRO .....	56
6.2 RESULTADOS CON FREEIMU .....	58
6.3 GRAFICAS DE COMPARACIÓN.....	59
<b>7 CONCLUSIONES .....</b>	<b>64</b>
<b>8 APENDICE :CÓDIGO EN MATLAB.....</b>	<b>66</b>
<b>9 BIBLIOGRAFÍA .....</b>	<b>70</b>
<b>10 LISTADO DE ILUSTRACIONES .....</b>	<b>72</b>
10.1 FIGURAS.....	72
10.2 TABLAS .....	73
10.3 ECUACIONES.....	73

## 1 RESUMEN Y OBJETIVOS

El estudio y supervisión de la vida salvaje es siempre un tema de gran interés, pero plantea la dificultad del seguimiento de la actividad de los animales para clasificar su comportamiento a través de sensores instalados para ello.

El uso de redes neuronales permite el reconocimiento de patrones de comportamiento de los animales pero el ruido de los sensores puede afectar al funcionamiento de la red.

En el presente Trabajo de Fin de Grado se expondrá el método de clasificación de una parte del comportamiento animal, en concreto de sus patrones de movimiento, aplicando la metodología concretamente a los caballos.

Dentro de este proyecto se realizará la clasificación de las distintas muestras de los sensores de la IMU (acelerómetro, giróscopo y magnetómetro), y de distintos patrones de movimiento del caballo.

Para resolver el problema de la clasificación, se ha hecho uso de las redes neuronales y en concreto de las del tipo *backpropagation*. Así, entrenando dicha red, se consiguen clasificar los distintos patrones de movimiento del animal como "parado", "andando" y "trotando".

Como las muestras están en RAW, a la red neuronal le resultaría complicado aprender los patrones de movimientos con estos datos, así que se ha hecho uso de la FFT (*Fast Fourier Transform*), que ha dado resultados muy esperanzadores en campos como el de reconocimiento de voz.

Para trabajar con redes neuronales se ha hecho uso de la aplicación MATLAB y más concretamente de la herramienta Neural Networks Toolbox, que incluye varios tipos de redes neuronales y con la que se puede realizar un modelo de red a medida. MATLAB también incluye el algoritmo de la FFT por lo que es la mejor opción para realizar las simulaciones.

Mediante el uso de estas herramientas se han realizado una serie de pasos en el Proyecto para conseguir la clasificación del movimiento del animal mediante una Red Neuronal:

- Realizar pre-procesamiento de los datos para entrenar y testear la red neuronal.
- Entrenar la red con distintos parámetros como son el número de neuronas en la capa oculta y el número de entradas.
- Comprobar los resultados con muestras de testeo previamente pre-procesadas.
- Conclusiones de los resultados obtenidos.

## 2 INTRODUCCIÓN

### 2.1 Minerva.

Este Trabajo de Fin de Grado ha sido realizado dentro del Proyecto Minerva, que tiene como objetivo el estudio y seguimiento de la fauna salvaje o en semi-libertad. Este seguimiento ha tenido siempre un gran interés dentro del estudio del comportamiento animal.

En el caso de la fauna salvaje el Proyecto Minerva aporta un mecanismo de conocimiento así como una herramienta para controlar la evolución y conservación de la misma.

Por lo que respecta a los animales en semi-libertad se establece un mecanismo de control y optimización de los recursos en las explotaciones de este tipo en nuestro entorno. Es remarcable esta actividad, sobre todo, en la cría de caballos y de ganado vacuno bravo, pero no es descartable su utilización en otras variedades de explotación, ya sea ganadera, avícola o acuícola.

Desde hace unos años la tecnología nos brinda elementos, de relativo bajo coste, con los que diseñar nuevos sistemas que podrían facilitar estas tareas de forma notable: microcontroladores, redes inalámbricas de bajo consumo, nuevas generaciones de sensores.

En este proyecto se plantea la utilización de estos elementos para diseñar un sistema novedoso, de bajo coste, que permita la monitorización de la fauna. Se pretende utilizar múltiples sensores y fusionar la información de los mismos de forma local para reducir los datos a transmitir.

Para realizar dicha transmisión, y cubrir todas las posibles necesidades e incidencias en la monitorización de este tipo de fauna, se diseñan dos sistemas de adquisición: uno móvil de monitorización manual y otro mediante la utilización de un nodo fijo transportable más un conjunto de motas 802.15.4 para adaptar la cobertura, que podrán ser estratégicamente situados en los lugares de acceso rutinario de los animales a monitorizar (lugar de descanso, de comida, agua, etc)[1].

El sistema a diseñar se compone de cuatro subsistemas, como se puede ver en la figura 1:

1. Un collar o arnés para el animal a monitorizar. Se diseñará un prototipo en sistema empotrado, basado en un microcontrolador de bajo consumo de potencia más un transceptor RF 802.15.4 al que se le añadirán una serie de sensores, como una IMU así como algunos otros para la monitorización de las constantes biológicas del animal.

La novedad que se plantea en este trabajo es el procesamiento de los datos de los sensores mediante redes neuronales para almacenar comportamientos en la memoria local, evitando la necesidad de almacenar todas las muestras de los sensores, y de esta forma pueden ser transmitidos los comportamientos vía RF, optimizando la transmisión y por tanto la duración de las baterías.

2. Subsistemas de seguimiento y monitorización manual mediante nodo móvil, que consiste en un dispositivo electrónico conectable a un portátil, PDA u otro dispositivo de visualización que, junto a un software específico, permitirá recolectar los datos de los nodos cercanos.

También podrá servir como ayuda a la colocación del nodo fijo y las motas y establecer su cobertura por el territorio.

3. Subsistema de monitorización mediante nodo fijo transportable y red de motas: Para que no haya ningún impacto visual en el entorno ni tampoco el animal detecte ningún tipo de vigilancia, se camufla el nodo fijo que hará de puente con la wifi.
4. Software de gestión de la información. Tanto la red compuesta por el nodo transportable y sus motas como los dispositivos móviles de seguimiento manual, volcarán sus datos capturados al servidor central que tendrá 2 objetivos: por una parte programar los patrones de actividad de los collares, y por otra organizar la información capturada para que pueda ser usada de manera eficaz.



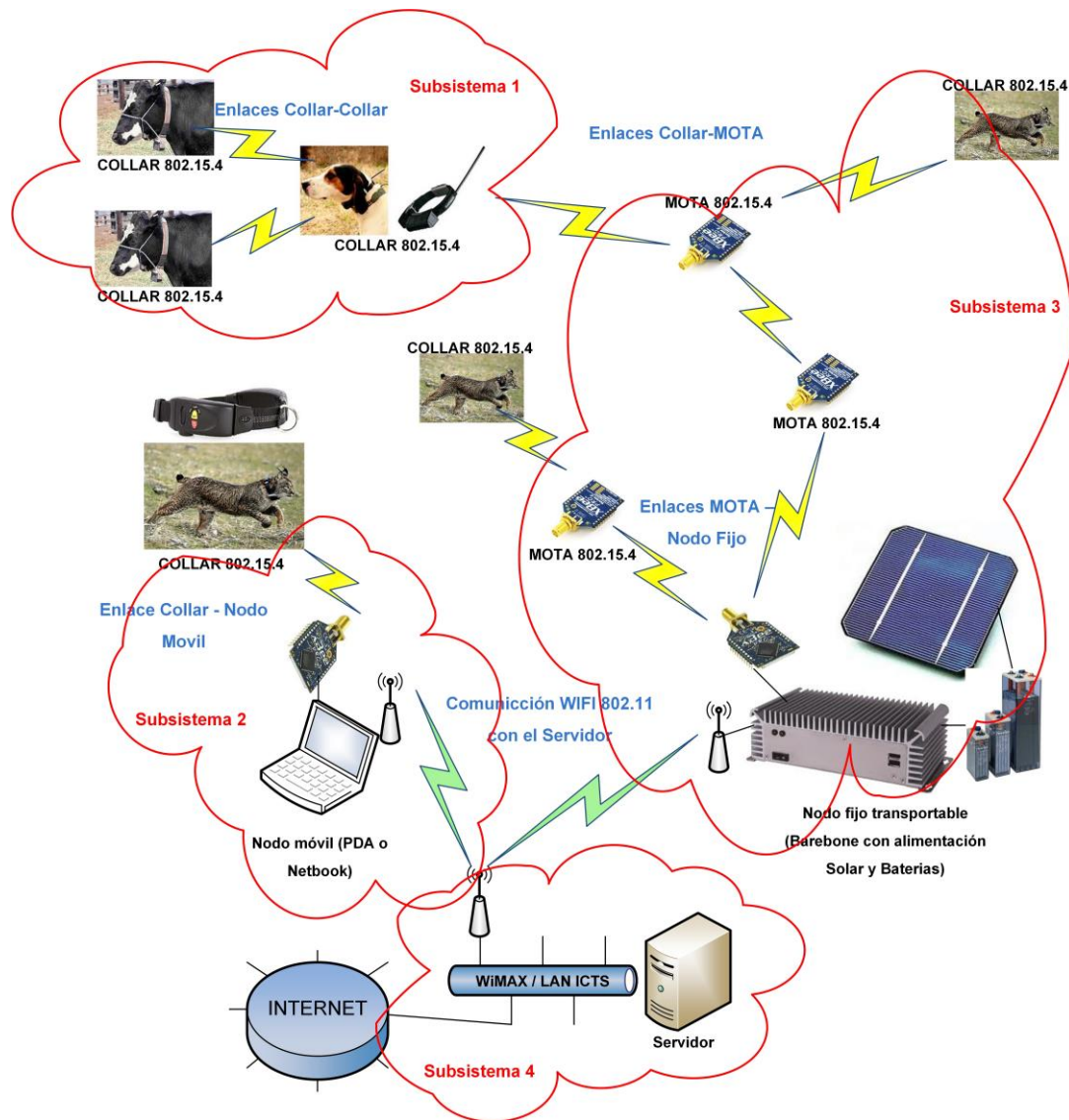


Figura 1: Arquitectura de Minerva

## 2.2 Antecedentes de Minerva

Durante la segunda mitad del siglo veinte se emplearon 3 tipos de métodos para el seguimiento de la fauna salvaje [1]:

- seguimiento por VHF
- mediante satélite
- GPS.

El sistema de VHF tradicional colapsa en cuanto se empiezan a utilizar múltiples collares debido a la escasez de frecuencias asignadas, lo que hace necesario codificar las señales que cada collar transmite para optimizar la localización.

Los mecanismos de localización por satélite (Argos sobre todo) son tan costosos que sólo se empleaban en animales migratorios a nivel continental.

Ya en los años noventa se implantaron mecanismos de localización basados en nodos fijos que cubrían una extensa área mediante triangulación sin necesidad de GPS con una precisión de 50 metros, a los que se les añadió posteriormente GPS para mayor precisión.

Uno de los sistemas más referenciados es el ZebraNET que, además de utilizar GPS para la localización, implementa otros sensores para detectar la actividad del animal monitorizado. La principal característica de este sistema es que almacena los datos recolectados localmente y que estos pueden ser transferidos entre diferentes collares hasta que son recuperados por el investigador.

Con respecto a la monitorización de animales en explotaciones en semi-libertad y pastoreo, la utilización de mecanismos electrónicos ha sido más reciente. A finales de los noventa y principios de este siglo se utilizaba de forma experimental la conjunción del GPS con la telefonía móvil GSM.

Desde el año 2004 se viene proponiendo por diversos investigadores la utilización, tanto para el caso de la fauna salvaje como el de explotaciones en semi-libertad, de redes tipo ZigBee. Las principales ventajas que aporta este tipo de tecnología son el coste, el tamaño y la duración de las baterías.

La utilización de mecanismos GPS para la localización, aunque generalizada, presenta diversos problemas. El primero de ellos hace referencia al consumo: es necesario estudiar con detenimiento cada cuanto tiempo “encender” el GPS, puesto que su consumo es relativamente alto. El otro problema que plantea la utilización de este tipo de tecnología es la cobertura sobre todo en zonas boscosas: para cierto tipo de fauna puede que sea difícil tener cobertura GPS.

Con respecto a la tecnología de transmisión GSM/GPRS tiene los mismos problemas: consumo energético relativamente alto y cobertura reducida en las zonas rurales y parques naturales.

El sistema Zebanet describe mecanismos de recolección de datos de diversos sensores con objeto de procesar posteriormente la información.

Esto permite obtener patrones en la información que se pueden asociar a las diversas actividades del animal: comer, cazar, dormir, ...

La posibilidad de que este reconocimiento de patrones se realice de forma local en cada collar colocado en el animal es muy atractiva, puesto que reduce los tiempos de transmisión de la información y del post-procesado.

Eliminar completamente el sistema GPS es posible si se tiene una infraestructura de nodos fijos que puedan localizar el animal, lo cual se puede realizar con relativa facilidad. Por otra parte si no tenemos nodos fijos de localización y no es importante el seguimiento continuo del animal, puede omitirse el uso del GPS, recogiendo sólo la actividad y el lugar dónde se encuentra cada vez que se localizase para transferir los datos. Esto puede ser equivalente a los sistemas primitivos de localización VHF pero con registro de la actividad.

## 2.3 .Preprocesado de la red neuronal mediante FFT.

### 2.3.1 *Transformadas de Fourier*

#### Introducción

La historia del análisis de Fourier tiene más de 200 años. Sus orígenes se inician unos 60 años antes del momento en que Jean Baptiste Joseph Fourier presentó la primera versión de su trabajo sobre la teoría de la conducción del calor a la Academia de París (1807) [2].

En 1755 Daniel Bernoulli propuso una solución en función de ondas estacionarias para resolver el problema de la cuerda vibrante.

Su argumento se comprende mejor al considerar la solución:

$$y(t, x) = \sin k\pi x \cos ak\pi t$$

**Ecuación 1. Ecuación de la cuerda**

Euler no estaba de acuerdo con este planteamiento y sus principales objeciones fueron dos: En primer lugar, las hipótesis de Bernoulli implicarían que cualquier función  $f(x)$  podría representarse como [2]:

$$f(x) = \sum \alpha_k \sin k\pi x$$

#### Ecuación 2. Ecuación de Euler para la cuerda

la segunda objeción es que la Ec.2 no puede ser una solución general del problema, ya que el miembro de la derecha (aunque sea una serie infinita) es una función analítica y por lo tanto una “función” (en el sentido que se entendía entonces, en el siglo XVIII).

Pero todo cambió con los trabajos de Joseph Fourier en 1804, sobre la ecuación del calor.

En 1807 Fourier presentó su trabajo *Mémoire sur la propagation de la chaleur* [3] al Instituto de París. El trabajo recibió apenas una tenue acogida, y los jueces recomendaron a Fourier que puliera su trabajo, y lo presentara para el gran premio de 1812. El panel de jueces de la Academia para este concurso incluía a Lagrange, Laplace, y Legendre, y dieron a Fourier el gran premio, pero no sin reservas.

El gran premio contenía el siguiente comentario:

*"La forma en la que el autor llega a sus ecuaciones no está exenta de dificultades, y su análisis deja algo que desear, bien en generalidad o incluso en rigurosidad."*

En definitiva, el artículo nunca fue publicado, y el trabajo de Fourier sobre la conducción del calor, y la teoría de las series trigonométricas que lo soportaba, vio la luz con la publicación en 1822 de *Théorie analytique de la chaleur*. Desde ese año, el trabajo de Fourier recibió comentarios muchos más entusiastas que los de la Academia de París. James Clerk Maxwell lo llamó un “gran poema matemático”. La carrera entera de William Thompson (Lord Kelvin) estuvo marcada por la teoría del calor de Fourier:[3]

*"El teorema de Fourier no es sólo uno de los más bellos resultados del análisis moderno, sino que se ha vuelto un instrumento indispensable en el tratamiento de casi cualquier recóndito problema de la física moderna".[2]*

### Aplicación

Cuando se desarrolló la transformada de Fourier, inicialmente fue un instrumento para la solución de ecuaciones diferenciales en derivadas parciales, como el problema del calor o el problema de la cuerda vibrante.

Pero actualmente tiene varios usos entre los que se destacan: el análisis espectral (en frecuencia) de las señales digitales, para comunicaciones, análisis de vibraciones de elementos de máquinas, en decodificación y codificación en MP3, etc.

Para este Trabajo de Fin de Grado se usará la FFT, que realiza la misma función de la transformada de Fourier pero es un algoritmo más optimizado, para realizar una descomposición espectral de las muestras de los sensores en bruto y tratar de averiguar el patrón de movimiento a partir del patrón de frecuencias que se obtiene mediante la FFT.

#### 2.3.2 Transformada Discreta de Fourier

La transformada discreta de Fourier o **DFT** (del inglés, *Discrete Fourier Transform*) es un tipo de transformada discreta utilizada en el análisis de Fourier. Transforma una función matemática en otra, obteniendo una representación en el dominio de la frecuencia, siendo la función original una función en el dominio del tiempo.

Pero la DFT requiere que la función de entrada sea una secuencia discreta y de duración finita. Dichas secuencias se suelen generar a partir del muestreo de una función continua, como puede ser la voz humana.

Al contrario que la transformada de Fourier en tiempo discreto (DTFT), esta transformación únicamente evalúa suficientes componentes frecuenciales para reconstruir el segmento finito que se analiza. Utilizar la DFT implica que el segmento que se analiza es un único período de una señal periódica que se extiende de forma infinita; si esto no se cumple, se debe utilizar una ventana para reducir los espurios del espectro.

Por la misma razón, la DFT inversa (IDFT) no puede reproducir el dominio del tiempo completo, a no ser que la entrada sea periódica indefinidamente. Por estas razones, se dice que la DFT es una transformada de Fourier para

análisis de señales de tiempo discreto y dominio finito. Las funciones sinusoidales base que surgen de la descomposición tienen las mismas propiedades.

La entrada de la DFT es una secuencia finita de números reales o complejos, de modo que es ideal para procesar información almacenada en soportes digitales.

En particular, la DFT se utiliza comúnmente en procesamiento digital de señales y otros campos relacionados dedicados a analizar las frecuencias que contiene una señal muestreada, también para resolver ecuaciones diferenciales parciales, y para llevar a cabo operaciones como convoluciones o multiplicaciones de grandes números enteros.[6]. Su ecuación se define como:

$$X(k) = \sum_{n=0}^{N-1} x(n) \cdot e^{\frac{-j \cdot 2 \cdot \pi \cdot k \cdot n}{N}}$$

*Ecuación 3: Ecuación de la DFT*

Donde:

$x(n)$ : son las muestras de la señal.

$X(k)$ : son las muestras de la DFT

$e$ : base de los logaritmos neperianos,

$N$ : números de muestras.

$n$ : número de la muestra actual.

Para el cálculo de la señal original a partir de sus componentes discretos en el dominio de la frecuencia, se utiliza la transformada inversa discreta:

$$x(n) = \frac{1}{N} \cdot \sum_{k=0}^{N-1} X(k) \cdot e^{\frac{-j \cdot 2 \cdot \pi \cdot k \cdot n}{N}}$$

*Ecuación 4: Ecuación de la IDFT*

Esta se utiliza en gran parte para temas de filtrado de señales y, en algunos casos, de predicciones en series temporales.

### 2.3.3 FFT (*Fast Fourier Transform*)

#### Introducción

Es un eficiente algoritmo que permite calcular la transformada de Fourier discreta (DFT) y su inversa. La FFT es de gran importancia en una amplia variedad de aplicaciones, desde el tratamiento digital de señales y filtrado digital en general a la resolución de ecuaciones en derivadas parciales o los algoritmos de multiplicación rápida de grandes enteros.

El algoritmo pone algunas limitaciones en la señal y en el espectro resultante. Por ejemplo: la señal de la que se tomaron muestras y que se va a transformar debe consistir de un número de muestras igual a una potencia de dos.

La mayoría de los analizadores de frecuencia permiten la transformación de 512, 1024, 2048 o 4096 muestras. El rango de frecuencias cubierto por el análisis TRF depende de la cantidad de muestras recogidas y de la proporción de muestreo.

La transformada rápida de Fourier es de importancia fundamental en el análisis matemático y ha sido objeto de numerosos estudios. La aparición de un algoritmo eficaz para esta operación fue una piedra angular en la historia de la informática.

Las aplicaciones de la transformada rápida de Fourier son múltiples. Es la base de muchas operaciones fundamentales del procesamiento de señales, donde tiene amplia utilización.

Además, proporciona un medio oportuno para mejorar el rendimiento de los algoritmos para un conjunto de problemas aritméticos comunes.

Como se ha dicho anteriormente, este algoritmo realiza el cálculo de la DFT más eficientemente. A continuación mostraremos un algoritmo de cálculo de la FFT:

La DFT se define como [10]:



$$X(k) = \sum_{n=0}^{N-1} x(n) \cdot e^{-j \cdot 2\pi \cdot k \cdot n / N} = \sum_{n=0}^{N-1} x(n) \cdot W_N^{k \cdot n}$$

*Ecuación 5: Ecuacion de la DFT modificada*

Donde:

$$W_N = e^{-j \cdot \frac{2\pi}{N}}$$

Y

$$W_N^{k \cdot n} = e^{-j \cdot \frac{2\pi}{N} \cdot k \cdot n}$$

Antes de entrar en detalles sobre la FFT comprobemos la naturaleza periódica del término  $W_N^{k \cdot n}$ , de hecho la periodicidad y simetría de  $W_N^{k \cdot n}$  contribuyen a la redundancia de la DFT.

$$W_N^{k \cdot n} = e^{-j \cdot \frac{2\pi}{N} \cdot k \cdot n} = \cos\left(\frac{2\pi}{N} \cdot k \cdot n\right) - j \sin\left(\frac{2\pi}{N} \cdot k \cdot n\right)$$

*Ecuación 6 Redundancia  $W_N$  en forma trigonométrica*

En la siguiente tabla se da el caso para N=8:

Valores de n

		0	1	2	3	4	5	6	7
Va	0	1	1	1	1	1	1	1	1
lor	1	1	$1 - j/\sqrt{2}$	-j	$-1 - j/\sqrt{2}$	-1	$-1 + j/\sqrt{2}$	j	$1 + j/\sqrt{2}$
de	2	1	-j	-1	j	1	-j	-1	j
k	3	1	$-1 - j/\sqrt{2}$	j	$1 - j/\sqrt{2}$	-1	$1 + j/\sqrt{2}$	-j	$-1 + j/\sqrt{2}$
	4	1	-1	1	-1	1	-1	1	-1
	5	1	$-1 + j/\sqrt{2}$	-j	$1 + j/\sqrt{2}$	-1	$1 - j/\sqrt{2}$	j	$-1 - j/\sqrt{2}$
	6	1	j	-1	-j	1	j	-1	-j
	7	1	$1 + j/\sqrt{2}$	j	$-1 + j/\sqrt{2}$	-1	$-1 - j/\sqrt{2}$	-j	$1 - j/\sqrt{2}$

*Tabla 1: Valores de  $W_N^{k \cdot n}$*

Para mostrar el algoritmo de la FFT empecemos por elegir un número de muestras  $N = 2^m$  donde m es un número entero. Estudiemos el caso de N=4:



$$\left. \begin{aligned} X[0] &= x[0] \cdot e^{-j \cdot \frac{2 \cdot \pi}{4} \cdot 0} + x[1] \cdot e^{-j \cdot \frac{2 \cdot \pi}{4} \cdot 0} + x[2] \cdot e^{-j \cdot \frac{2 \cdot \pi}{4} \cdot 0} + x[3] \cdot e^{-j \cdot \frac{2 \cdot \pi}{4} \cdot 0} \\ X[1] &= x[0] \cdot e^{-j \cdot \frac{2 \cdot \pi}{4} \cdot 0} + x[1] \cdot e^{-j \cdot \frac{2 \cdot \pi}{4} \cdot 1} + x[2] \cdot e^{-j \cdot \frac{2 \cdot \pi}{4} \cdot 2} + x[3] \cdot e^{-j \cdot \frac{2 \cdot \pi}{4} \cdot 3} \\ X[2] &= x[0] \cdot e^{-j \cdot \frac{2 \cdot \pi}{4} \cdot 0} + x[1] \cdot e^{-j \cdot \frac{2 \cdot \pi}{4} \cdot 2} + x[2] \cdot e^{-j \cdot \frac{2 \cdot \pi}{4} \cdot 4} + x[3] \cdot e^{-j \cdot \frac{2 \cdot \pi}{4} \cdot 6} \\ X[3] &= x[0] \cdot e^{-j \cdot \frac{2 \cdot \pi}{4} \cdot 0} + x[1] \cdot e^{-j \cdot \frac{2 \cdot \pi}{4} \cdot 3} + x[2] \cdot e^{-j \cdot \frac{2 \cdot \pi}{4} \cdot 6} + x[3] \cdot e^{-j \cdot \frac{2 \cdot \pi}{4} \cdot 9} \end{aligned} \right\}$$

En este ejemplo se ve un cálculo que implica 12 sumas y 9 multiplicaciones complejas.

Observando que  $e^{-j \cdot \frac{2 \cdot \pi}{4} \cdot k}$ , el cálculo de  $x[2]$  y  $x[3]$ , se expresa:

$$\begin{aligned} X[2] &= x[0] \cdot e^{-j \cdot \frac{2 \cdot \pi}{4} \cdot 0} + x[1] \cdot e^{-j \cdot \frac{2 \cdot \pi}{4} \cdot 2} + x[2] \cdot e^{-j \cdot \frac{2 \cdot \pi}{4} \cdot 0} + x[3] \cdot e^{-j \cdot \frac{2 \cdot \pi}{4} \cdot 2} \\ X[3] &= x[0] \cdot e^{-j \cdot \frac{2 \cdot \pi}{4} \cdot 0} + x[1] \cdot e^{-j \cdot \frac{2 \cdot \pi}{4} \cdot 3} + x[2] \cdot e^{-j \cdot \frac{2 \cdot \pi}{4} \cdot 2} + x[3] \cdot e^{-j \cdot \frac{2 \cdot \pi}{4} \cdot 1} \end{aligned}$$

Y el anterior sistema de ecuaciones se expresa como:

$$\begin{aligned} X[0] &= (x[0] + x[2]) + (x[1] + x[3]) \\ X[1] &= (x[0] + x[2] \cdot e^{-j \cdot \frac{2 \cdot \pi}{4} \cdot 2}) + (x[1] + x[3] \cdot e^{-j \cdot \frac{2 \cdot \pi}{4} \cdot 2}) \cdot e^{-j \cdot \frac{2 \cdot \pi}{4}} \\ X[2] &= (x[0] + x[2]) + (x[1] + x[3]) \cdot e^{-j \cdot \frac{2 \cdot \pi}{4} \cdot 2} \\ X[3] &= (x[0] + x[2] \cdot e^{-j \cdot \frac{2 \cdot \pi}{4} \cdot 2}) + (x[1] + x[3] \cdot e^{-j \cdot \frac{2 \cdot \pi}{4} \cdot 2}) \cdot e^{-j \cdot \frac{2 \cdot \pi}{4} \cdot 3} \end{aligned}$$

cuyo cálculo implica 12 sumas y 5 multiplicaciones complejas.

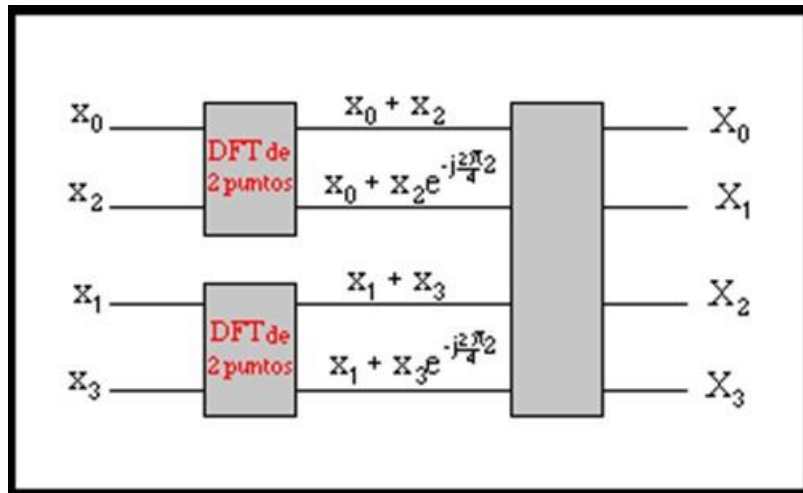


Figura 2: Estructura de la FFT

Esta interpretación de la FFT de 4 puntos sugiere que la organización de la FFT se obtuvo dividiendo la DFT de 4 puntos en dos TDF de 2 puntos y combinando sus coeficientes. Hay que observar que en el desarrollo del algoritmo, el dato  $x[0]$  se empareja con  $x[2]$ , y  $x[1]$  con  $x[3]$ .

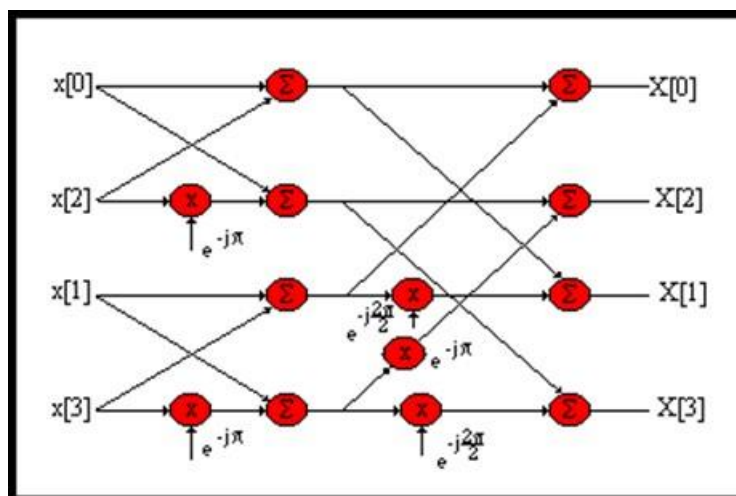


Figura 3: ejemplo desarrollado de FFT

Esto implica que la organización de una FFT de 8 puntos se puede dividir en dos DFT de 4 puntos y a continuación una combinación de ambos conjuntos de coeficientes.

Esto es el algoritmo de **Cooley y Tukey**.

#### 2.3.4 *Papel de la FFT en el preprocesamiento de la red neuronal del Proyecto*

Cuando se obtienen los datos en RAW de los sensores, se realiza un primer pre-procesamiento, agrupando las muestras en grupos de un cierto número de muestras y después a estas muestras se les realiza una FFT, porque así se pueden definir las características del movimiento en el conjunto de grupos de muestras y de esta forma realizar una mejor clasificación del grupo de muestra obtenida de los sensores

### **3 REDES NEURONALES ARTIFICIALES.**

#### **3.1 Introducción**

Los primeros modelos de redes neuronales datan de los trabajos realizados en 1943 por los neurólogos Warren McCulloch y Walter Pitts. Años más tarde, en 1949, Donald Hebb desarrolló sus ideas sobre el aprendizaje neuronal, quedando reflejado en la "regla de Hebb". En 1958, Rosenblatt desarrolló el perceptrón simple, y en 1960, Widrow y Hoff desarrollaron el ADALINE, que fue la primera aplicación industrial real.

En los años siguientes, se redujo la investigación, debido a la falta de modelos de aprendizaje y el estudio de Minsky y Papert sobre las limitaciones del perceptrón. Sin embargo, en los años 80, volvieron a resurgir las RNA gracias al desarrollo de la red de Hopfield, y en especial, al algoritmo de aprendizaje de retropropagación (BackPropagation) ideado por Rumelhart y McClelland en 1986 que fue aplicado en el desarrollo de los perceptrones multicapa.

#### **3.2 Historia de las Redes Neuronales[11]**

**1936** Alan Turing fue el primero en estudiar el cerebro como una forma de ver el mundo de la computación.

Sin embargo, los primeros teóricos que concibieron los fundamentos de la computación neuronal fueron Warren McCulloch, un neurofisiólogo, y Walter Pitts, un matemático, quienes, en 1943, lanzaron una teoría acerca de la forma de trabajar de las neuronas (Un Cálculo Lógico de la Inminente Idea de la Actividad Nerviosa - Boletín de Matemática Biofísica 5: 115-133). Ellos modelaron una red neuronal simple mediante circuitos eléctricos.

**1949.**-Donald Hebb. Fue el primero en explicar los procesos del aprendizaje (que es el elemento básico de la inteligencia humana) desde un punto de vista psicológico, desarrollando una regla de como el aprendizaje ocurría. Aun hoy, este es el fundamento de la mayoría de las funciones de aprendizaje que pueden hallarse en una red neuronal.

Su idea fue que el aprendizaje ocurría cuando ciertos cambios en una neurona eran activados. También intentó encontrar semejanzas entre el

aprendizaje y la actividad nerviosa. Los trabajos de Hebb formaron las bases de la Teoría de las Redes Neuronales.

**1950.**- Karl Lashley. En sus series de ensayos, encontró que la información no era almacenada en forma centralizada en el cerebro sino que era distribuida encima de él.

**1956** - Congreso de Dartmouth. Este Congreso es mencionado frecuentemente para indicar el nacimiento de la inteligencia artificial.

**1957** - Frank Rosenblatt. Comenzó el desarrollo del Perceptrón. Esta es la red neuronal más antigua; utilizándose hoy en día para aplicación como identificador de patrones.

Este modelo era capaz de generalizar, es decir, después de haber aprendido una serie de patrones podía reconocer otros similares, aunque no se le hubiesen presentado en el entrenamiento.

Sin embargo, tenía una serie de limitaciones, por ejemplo, su incapacidad para resolver el problema de la función OR-exclusiva y, en general, era incapaz de clasificar clases no separables linealmente.

**1959** - Frank Rosenblatt: Principios de Neurodinámica. En este libro confirmó que, bajo ciertas condiciones, el aprendizaje del Perceptrón convergía hacia un estado finito (Teorema de Convergencia del Perceptrón).

**1960.**-Bernard Widroff/MarcianHoff. Desarrollaron el modelo Adaline (ADaptive LINear Elements). Esta fue la primera red neuronal aplicada a un problema real (filtros adaptativos para eliminar ecos en las líneas telefónicas) que se ha utilizado comercialmente durante varias décadas.

**1961.**-Karl Steinbeck: Die Lernmatrix. Red neuronal para realizaciones técnicas simples (memoria asociativa).

**1969.**-Marvin Minsky/Seymour Papert. En este año casi se produjo la “muerte abrupta” de las Redes Neuronales; ya que Minsky y Papert probaron (matemáticamente) que el Perceptrón no era capaz de resolver problemas relativamente fáciles, tales como el aprendizaje de una función no-lineal.

Esto demostró que el Perceptrón era muy débil, puesto que las funciones no-lineales son extensamente empleadas en computación y en los problemas del mundo real.

**1974.**-Paul Werbos. Desarrolló la idea básica del algoritmo de aprendizaje de propagación hacia atrás (backpropagation); cuyo significado quedó definitivamente aclarado en 1985.

**1977** - Stephen Grossberg: Teoría de Resonancia Adaptada (TRA). La Teoría de Resonancia Adaptada es una arquitectura de red que se diferencia de todas las demás previamente inventadas. La misma simula otras habilidades del cerebro: memoria a largo y corto plazo.

**1985.**-John Hopfield. Provocó el renacimiento de las redes neuronales con su libro: "Computación neuronal de decisiones en problemas de optimización."

**1986.**-David Rumelhart/G. Hinton. Redescubrieron el algoritmo de aprendizaje de propagación hacia atrás (backpropagation).

A partir de 1986, el panorama fue alentador con respecto a las investigaciones y el desarrollo de las redes neuronales.

En la actualidad, son numerosos los trabajos que se realizan y publican cada año, las aplicaciones nuevas que surgen (sobre todo en el área de control) y las empresas que lanzan al mercado productos nuevos, tanto hardware como software (sobre todo para simulación).

### **3.3 Tipos de Redes Neuronales.**

#### **3.3.1 *Perceptron***

Este modelo se concibió como un sistema capaz de realizar tareas de clasificación de forma automática.

La idea era disponer de un sistema que, a partir de un conjunto de ejemplos de clases diferentes fuera capaz de determinar las ecuaciones de frontera entre dichas clases.

Estos conjuntos de ejemplos son los llamados patrones de entrenamiento, que son los que utilizaban el sistema para formar las superficies

discriminatorias y además funcionase como modelo discriminatorio de nuevas clases de patrones de entradas.

Un ejemplo de la neurona del perceptrón lo vemos en la siguiente figura:

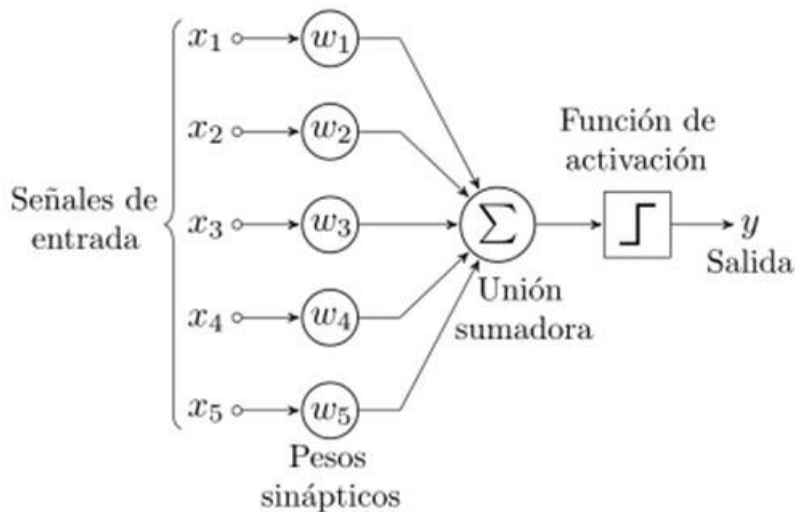


Figura 4: Diagrama neurona perceptrón

### 3.3.2 Adaline

Existe un gran número de ejemplo de aprendizaje que requiere que las entradas sean valores reales, en vez de discretos. Así que en 1960, Widrow y Hoff, propusieron un sistema de aprendizaje que tenía en cuenta el error producido por la neurona en el aprendizaje y diseñaron lo que se denominó ADActiveLinearNEuron o en su acrónimo ADALINE.

Este es el modelo de neurona de las red ADALINE:

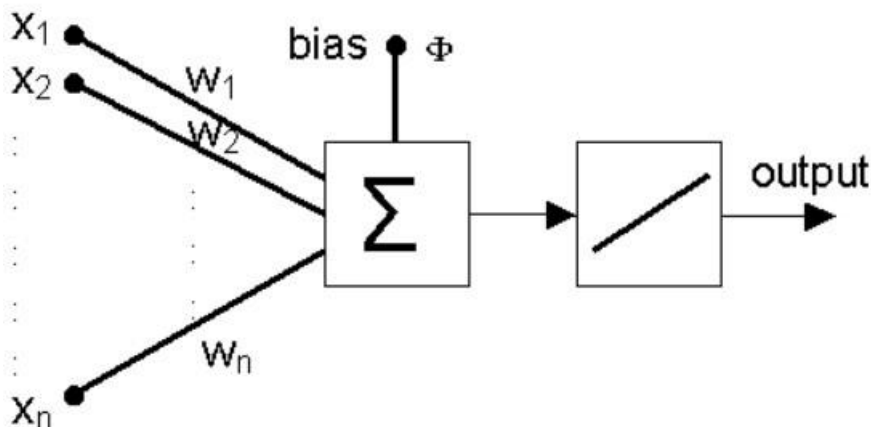
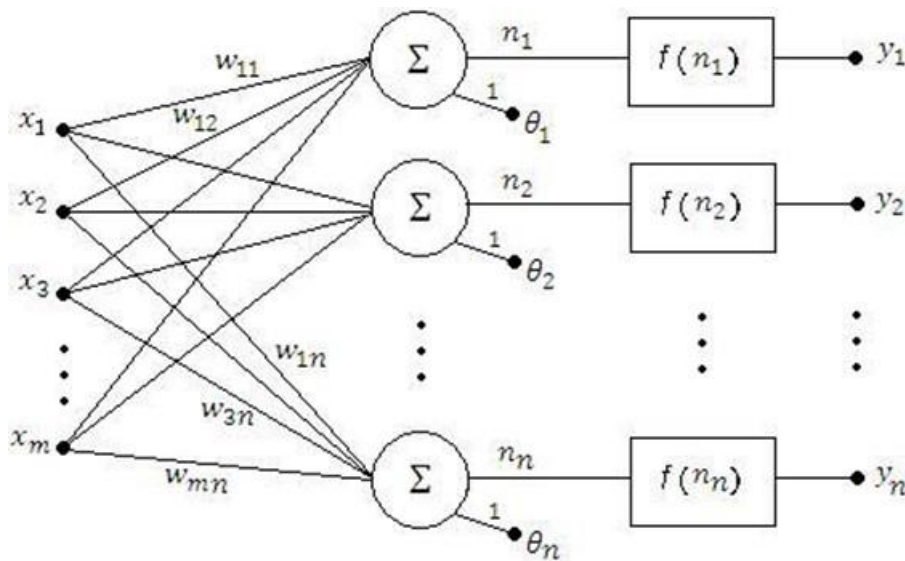


Figura 5: Modelo de neurona ADALINE

La estructura básica de esta red se puede ver en la siguiente figura:



*Arquitectura general de la red neuronal Adaline. En general  $f(n_i) = n_i$*

Figura 6: Arquitectura de red ADALINE

Para entrenar esta red, se utiliza el algoritmo "normalize LMS" (least mean square) [14], también indicada como la regla delta. Este algoritmo de entrenamiento ha sido utilizado en otras aplicaciones como los filtros adaptativos LMS, para eliminar el ruido de las líneas telefónicas.

### 3.3.3 Redes Neuronales Multicapa

#### Introducción

El perceptrón multicapa es una red neuronal artificial (RNA) formada por múltiples capas, lo que le permite resolver problemas que no son linealmente separables. Esta es la principal limitación del perceptrón (también llamado perceptrón simple).

El perceptrón multicapa puede ser totalmente o localmente conectado. En el primer caso cada salida de una neurona de la capa "i" es entrada de todas las neuronas de la capa "i+1", mientras que en el segundo cada neurona de la capa "i" es entrada de una serie de neuronas (región) de la capa "i+1" [15].



El modelo de neurona típico para esta red es el siguiente:

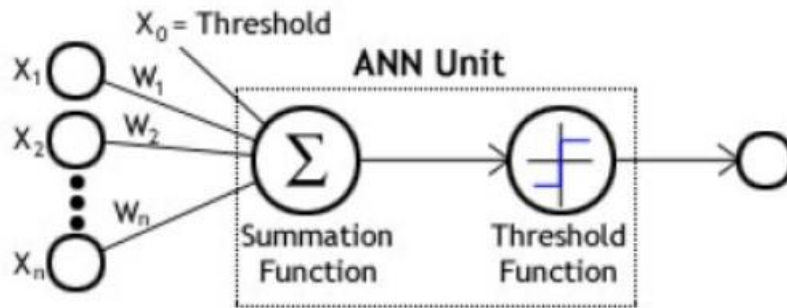


Figura 7: Neurona de red perceptrón multicapa

La arquitectura normal de este tipo de redes es:

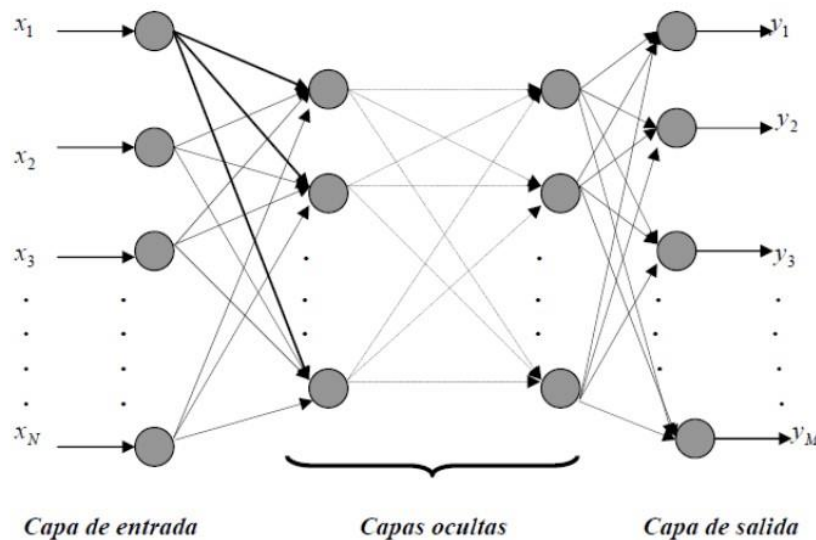


Figura 8: Arquitectura de perceptrón multicapa

### Tipos de capas

Las capas pueden clasificarse en tres tipos:

- Capa de entrada: Constituida por aquellas neuronas que introducen los patrones de entrada en la red. En estas neuronas no se produce procesamiento.
- Capas ocultas: Formada por aquellas neuronas cuyas entradas provienen de capas anteriores y cuyas salidas pasan a neuronas de capas posteriores.
- Capa de salida: Neuronas cuyos valores de salida se corresponden con las salidas de toda la red.

La propagación hacia atrás (retropropagación del error o regla delta generalizada), es un algoritmo utilizado en el entrenamiento de estas redes, por ello el perceptrón multicapa también es conocido como red de retropropagación.

### Tipos de funciones de transferencias

Toda función debe de ser continua y derivable en todo su dominio. Las funciones más normales son las siguientes:

- Función logística o función sigmoidea, de la forma:  $f_{log}(z) = \frac{1}{1 + e^{-z}}$  cuya gráfica es la siguiente

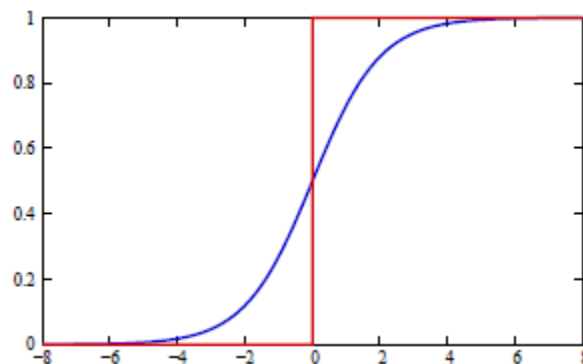


Figura 9: Gráfica función logística

En esta figura, se observa la diferencia entre la curva logística (en azul) y la curva "todo o nada" en rojo".

- Todo o nada o función Threshold: es la función que se utiliza en los perceptrón y algunas veces en este tipo de redes, su gráfica es como la que sigue:

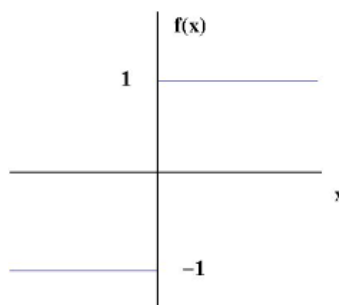


Figura 10: Gráfica función "todo o nada"

- Función lineal es aquella cuyo formato es  $f(x)=x$ , se utiliza algunas veces cuando la red se le utiliza en aproximaciones de funciones. Su gráfica es:

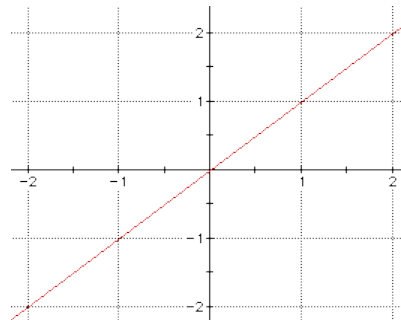


Figura 11: Gráfica función lineal

### Entrenamiento del perceptrón multicapa

El entrenamiento de las redes neuronales sigue un cierto algoritmo, según el tipo de las muestras, y/o el tipo de red en cuestión. Para esta red se utilizará el entrenamiento de backpropagation [17,18].

La estructura de dichas redes es como sigue:

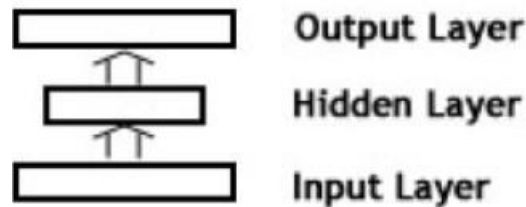


Figura 12: Estructura en bloques

El input layer, es la capa de la red donde se incluyen los datos recogidos del mundo externo, la hidden layer, es la capa oculta de la red, normalmente llevarán 1, aunque pueden ser más de una, y la output layer, es la capa de la red donde se producen las salidas de la red.

El backpropagation es un algoritmo de aprendizaje supervisado, es decir se modifica los parámetros de la red, para que las salidas de la red sean lo más próximas posible a la salida del supervisor [12].

Entonces el error de las salidas se podría definir como:

$$e(n) = \frac{1}{2} * \sum_{i=1}^{nc} (s_i(n) - y_i(n))^2$$

*Ecuación 7: Error de Salidas*

Debido a esto, el aprendizaje del perceptrón multicapa es la búsqueda del mínimo de la función de error, y como las funciones de transferencias son no lineales, tendremos que basarnos en técnicas de optimización no lineales para resolverlo.

Para llevar a cabo el algoritmo del backpropagation se basa en unas fórmulas matemáticas para la actualización de los pesos como puede ser la regla delta deducida a partir de las funciones de transferencias **sigmoidales**.

Esta regla delta se demuestra que es:

$$w_{ij}(k+1) = w_{ij}(k) + \Delta w_{ij}(k)$$

*Ecuación 8: Ecuación de la regla delta*

Donde

$$\Delta w_{ij}(k) = -\eta \frac{\partial E}{\partial w_{ij}(k)} = \eta [z_i(k) - y_i(k)] \cdot g'_1(h_i) s_j(k)$$

$$h_i = \sum_{j=1}^L w_{ij}(k) s_j(k)$$

$$g'_1(h_i) = 2\beta g_1(h_i)[1 - g_1(h_i)]$$

si tomamos la función sigmoidea y:

$$g'_1(h_i) = h_i$$

si se toma la función lineal

Ahora se va a mostrar el algoritmo de entrenamiento del backpropagation:

- ❖ Determinar la arquitectura:
  - ⇒ Cuántas entradas y salidas.
  - ⇒ Número de neuronas en la capa oculta y número de capas.
- ❖ Inicializar los pesos y bias con valores aleatorios comprendidos entre -1 y 1.
- ❖ Repetir hasta que se cumpla algún criterio de finalización:
  - ⇒ Presentar el ejemplo de entrenamiento y propagarlo a través de la red.
  - ⇒ Calcular la salida actual de la red.
  - ⇒ Adaptar los pesos, empezando por las neuronas de la capa de salida e ir hacia atrás, hasta la capa de entrada. Aquí es cuando se empleará la regla delta explicada más arriba.

### 3.3.4 Redes neuronales recurrentes

Para tareas como la predicción y búsqueda de patrones en series temporales, las redes neuronales han demostrado un mejor desempeño.

Su arquitectura es parecida a las redes multicapas, excepto en que tienen las salidas conectadas a las entradas.

Dependiendo de cuantas entradas estén conectadas a las salidas (feedback), tenemos redes parcialmente recurrentes si sólo algunas entradas están conectadas a la salidas, como es el caso de las redes Hopfield, o redes totalmente recurrentes, como las Elmar.

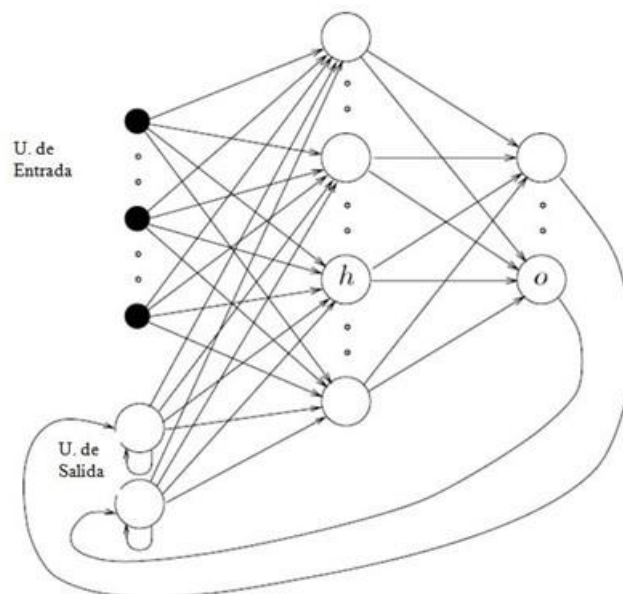


Figura 13 : Arquitectura de red recurrente

En la Figura de arriba se muestra dicha arquitectura. Estas redes se utilizan además como memorias asociativas, ya que al mostrarles distintos patrones a las entradas, son capaces de recordarlos, y clasificarlos, aun cuando exista ruido.

### 3.4 MATLAB. Redes Neuronales

#### 3.4.1 Introducción

MATLAB fue desarrollado por el matemático y programador de computadoras Cleve Moler en 1984, surgiendo la primera versión con la idea de emplear paquetes de subrutinas escritas en Fortran en los cursos de álgebra lineal y análisis numérico, sin necesidad de escribir programas en dicho lenguaje.

Las aplicaciones de MATLAB se realizan en un lenguaje de programación propio denominado M, creado en 1970 para proporcionar un sencillo acceso al software de matrices *LINPACK* y *EISPACK* sin tener que usar Fortran.

Este lenguaje es interpretado, y puede ejecutarse tanto en el entorno interactivo, como a través de un archivo de script (archivos \*.m).

Este lenguaje permite operaciones de vectores y matrices, funciones, cálculo lambda, y programación orientada a objetos [22,21].

Cada funcionalidad de MATLAB, se agrupa en diferentes toolboxes. En este Trabajo sólo se usará uno de ellos: Neuronal Networks Toolbox.

#### 3.4.2 Neuronal Network Toolbox.

Este toolbox de MATLAB, se usa para el diseño, entrenamiento y test de diferentes tipos de redes neuronales

Cuenta con varios tipos de funciones de transferencia como:

##### HardLim

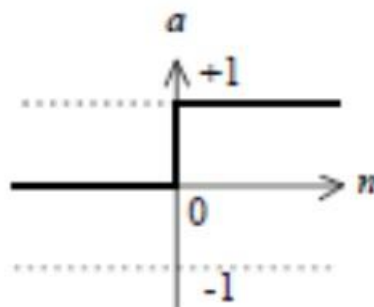
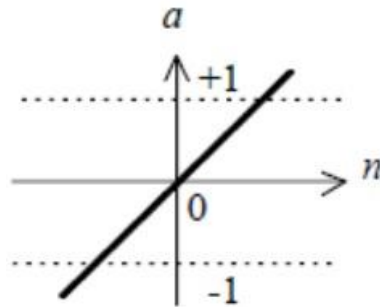


Figura 14: Función Hardlim

Esta función tiene la forma  $\text{hardlim}(a)$ , siendo  $a$  el valor de entrada. Si el valor entregado a esta función es menor que 0, la salida será 0 y si es mayor,

entrega un 1. Es utilizado sobretodo en perceptrons para tareas de clasificación.

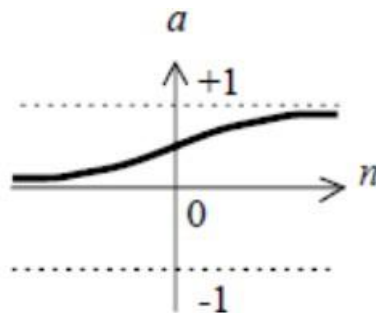
### Purelin



*Figura 15: Función lineal*

En MATLAB el comando `pureline(a)`, hace referencia a  $f(x)=x$ , definido en el intervalo  $[-\infty, \infty]$ . Se utiliza para aproximadores lineales como las redes ADALINE y en filtros lineales.

### Sigmoide



*Figura 16: Función Sigmoide*

Esta función se define en MATLAB como `logsig(n)`, con n como cualquier número. Es utilizada para la backpropagation. Es la función elegida para este Proyecto.



## 3.4.3 Modelo de neurona

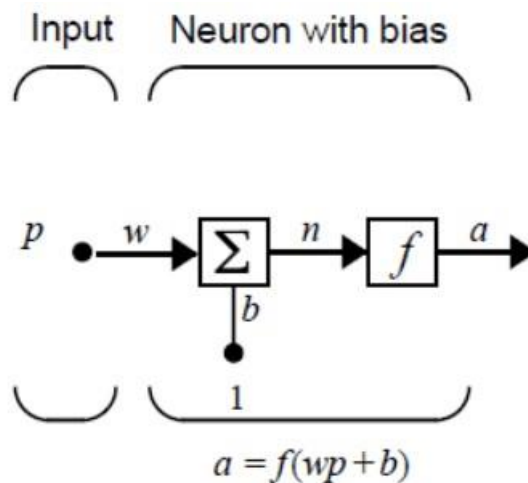


Figura 17: Modelo de neurona en MATLAB

Este es el modelo que se utilizará para desarrollar el Proyecto. A la suma del producto de los pesos por las entradas, se le suma una bias, que sirve para disminuir el nivel de disparo de la neurona.

Este es el modelo más desarrollado de la neurona que utiliza MATLAB:

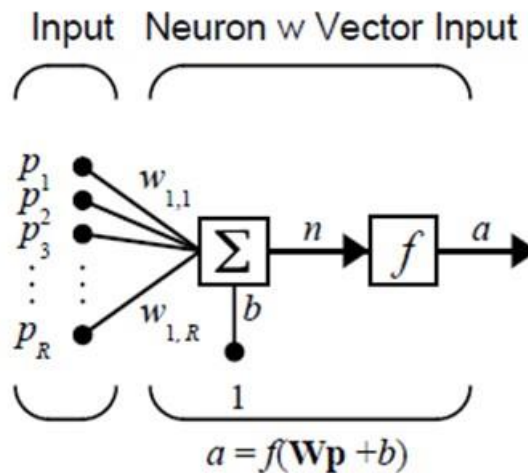


Figura 18: Modelo desarrollado

Las entradas  $p_R$  se multiplican por los distintos elementos de la matriz  $W$  de pesos, a los que se suma la bias ( $b$ ) que sirve para aumentar o disminuir el umbral de disparo. Después, este resultado será la entrada de la función de transferencia  $f$  de la neurona.

Como se ve en la figura, la ecuación  $a=f(Wp+b)$  es una ecuación matricial, en el que  $f$  es la función de transferencia,  $W$  la matriz de pesos,  $p$  es el vector de entrada, y  $b$  es el vector de bias, que generalmente se multiplica por el elemento único '1'.

Aquí se muestra un poco más claro el modelo de neurona:

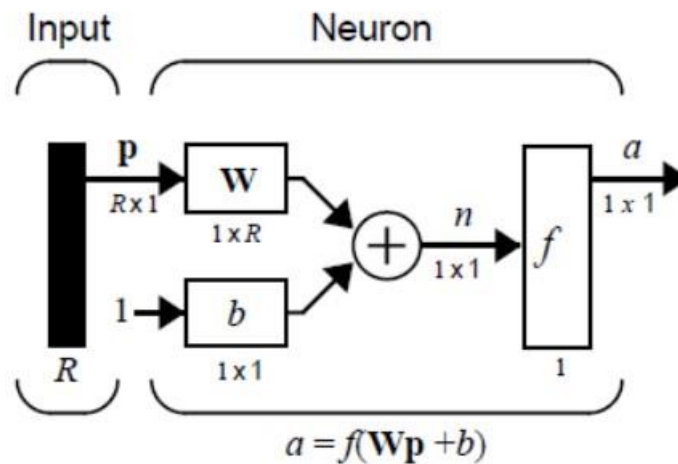


Figura 19: Arquitectura de red multicapa en MATLAB

#### 3.4.4 Arquitectura de una red multicapa

En MATLAB, cuando se quiere implementar una red backpropagation, dicha red tiene la siguiente estructura:

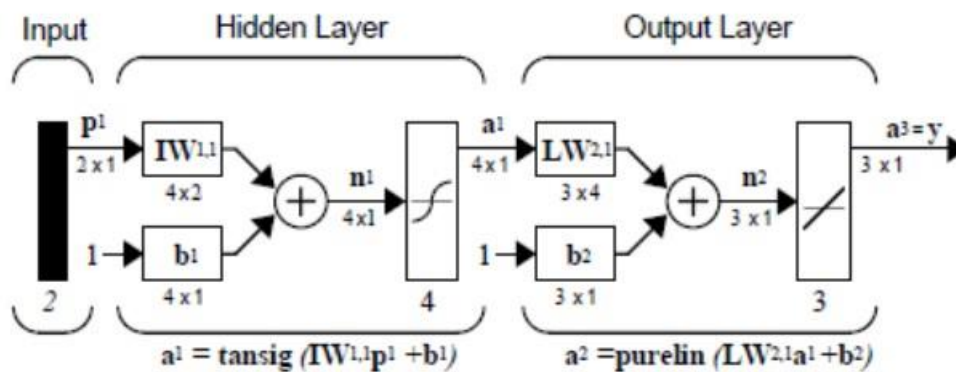


Figura 20: Implementación de una red Backpropagation en MATLAB.

Para crear una red con este toolbox se utilizan los siguientes comandos:

- ❖ Comando `net=newff(p,t,3)`, siendo **net** el objeto red propiamente dicha, **p** es la matriz con los valores de inicio de pesos y bias, **t** es el

rango de las salidas y **3** son las neuronas en la capa oculta (hiddenlayer).

- ❖ La simulación de la red se hace gracias al comando `out=sim(net,in)`, siendo **in** el vector de entradas, **out** el de salidas y **net** la estructura de datos de la red.
- ❖ Para entrenar la red, se utiliza el comando **train**, cuya sintaxis es:  
`[net,tr]=train(net,in,outdesire)`,  
siendo **outdesire** las salidas deseadas, y **tr** la información de entrenamiento.

#### 3.4.5 FreeIMU

Esta librería es utilizada para extraer de los datos en raw(en bruto), las coordenadas de la orientación de la IMU mediante el cálculo de cuaterniones. Un cuaternión es una entidad algebraica formada por un conjunto de 4 de números:

Se puede ver que los cuaterniones tienen la forma:

$q=a+bi+cj+dk$ , siendo **i,j,k** vectores unitarios cuyo producto se demuestra que es: **ijk=-1**

En cada iteración, se actualiza el cuaternión. Se compara el valor actual de éste con el que se tenía anteriormente. En un paso del algoritmo se aplica el método del descenso de gradiente para integrar el magnetómetro y el acelerómetro.

En otro paso también se añade el giróscopo para estimar la deriva de los datos [2].

A continuación se muestra, de forma esquematizada, el funcionamiento de dicho algoritmo [2].

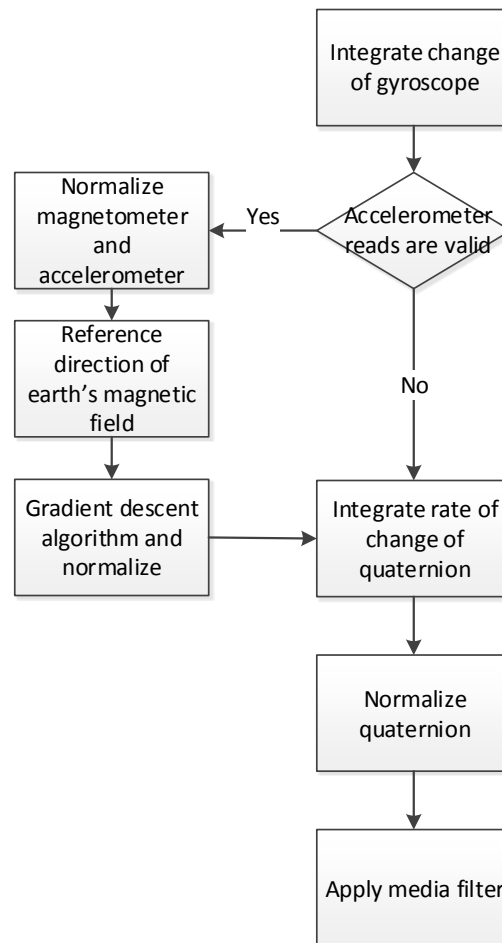


Figura 21: Algoritmo de FreeImu

En esta figura se muestra como el algoritmo comienza integrando los cambios del giróscopo y si los datos son válidos, normaliza los datos del magnetómetro y el acelerómetro.

El siguiente paso es referenciar una dirección magnética a la dirección de tierra para evitar perturbaciones externas.

A continuación se fusionan los datos del acelerómetro con el del magnetómetro, mediante el algoritmo de descenso del gradiente y después se normaliza. Finalmente este algoritmo (FreeIMU) integra estos datos, que están en cuaterniones, al dato anterior, lo que reduce el error producido por algún fallo en los datos del acelerómetro. El último paso es normalizarlos y reducir los posibles ruidos mediante un filtro de media [2].

En esta ocasión a estos datos se le aplicó un filtro de media [2] cuya ecuación es la que sigue:

$$B = \alpha * S_{act} + (1 - \alpha) * S_{ant}$$

La B es el resultado del filtro,  $S_{act}$  es la muestra actual,  $S_{ant}$  es la muestra anterior y  $\alpha$  es un factor ajustable en el rango de [0,1]. Su mejor valor ha demostrado ser de 0.65 [2].

Este filtro logra reducir los picos en los datos debido a la gran sensibilidad de la IMU.

### 3.5 Aceleración mediante GPU

#### 3.5.1 Introducción

La **GPU** es un coprocesador dedicado al procesamiento de gráficos u operaciones de coma flotante, para aligerar la carga de trabajo del procesador central en aplicaciones como los videojuegos o aplicaciones 3D interactivas.

De esta forma, mientras gran parte de lo relacionado con los gráficos se procesa en la *GPU*, la unidad central de procesamiento (*CPU*) puede dedicarse a otro tipo de cálculos [24].

Para aprovechar este poder de computación de las GPU, se han desarrollado técnicas y librerías, como **CUDA** y **OpenCL**, que facilitan el uso de la GPU al programador.

Todo esto forma parte de lo que se llama GPGPU (General Purpose GPU) o GPU de propósito general, debido a que la GPU es un procesador basado en un modelo circulante, lo que implica una gran segmentación en las tareas y facilita el procesamiento paralelo.

#### 3.5.2 GPU en MATLAB

Para desarrollar algunas tareas eficientemente como el entrenamiento de las redes neuronales, MATLAB, tiene la opción de procesar los datos mediante la GPU. Tiene un inconveniente y es que necesita una tarjeta compatible con Nvidia para aprovechar esta opción.

Para realizar por ejemplo el entrenamiento en MATLAB de las redes neuronales mediante GPU, sólo tenemos que proporcionar el comando `useGPU` a la función de entrenamiento:

```
net=train(net,datosentrenamiento,targetout1,'useGPU','yes','showResources','yes');
```

Como se ve en este ejemplo, para entrenar esta red, tendremos sólo que asegurarnos que MATLAB acepta la tarjeta gráfica, y después proporcionar los comandos antes mencionados para que el entrenamiento se desarrolle en la GPU.

## 4 LA RED NEURONAL.

### 4.1 Tipo de red elegida

Para este trabajo se ha elegido una red perceptrón multicapa, debido al desempeño que esta tiene para la clasificación, mucho mejor que otros métodos estadísticos como se demuestran en los trabajos de Emiliano [25].

Además va a formar parte de un sistema empujado, lo que limita la extensión del algoritmo y el consumo de energía, así como su dificultad a la hora de ser procesado, ya que se van a disponer de muy pocos recursos.

Las redes neuronales son una opción viable, sobre todo teniendo en cuenta que se va a procesar un número limitado de muestras y sólo un sensor, el acelerómetro.

Se pensó en un principio, realizar la clasificación con lógica difusa, pero debido a que se tienen que entrenar con un conjunto elevado de muestras, y puesto que el algoritmo de entrenamiento que posee MATLAB no es lo bastante eficaz cuando se trata de conjuntos de muestras grandes, finalmente se escogió la red neuronal.

Se eligió inicialmente el perceptrón multicapa debido a la facilidad de entrenamiento y a la gran disponibilidad que se tenía de herramientas y de documentación al respecto.

Además con una capa oculta ha demostrado ser un aproximador de funciones universal además de un buen clasificador lineal.

Otras redes neuronales, por ejemplo las redes recurrentes no habrían servido para este fin, ya que requieren demasiados recursos computacionales que un sistema empujado con un microcontrolador con una memoria tan limitada para este fin no sería capaz de llevar a cabo la tarea de procesar dicha red.

Pero lo que no se puede saber a priori es el número de neuronas que debe tener en la capa oculta para un determinado problema.

Para eso se ha desarrollado un script en MATLAB que varía, junto a otros parámetros, el número de neuronas en la capa oculta y muestra los

resultados del entrenamiento, para hacer un análisis de los resultados y así elegir el número óptimo de neuronas.

El aspecto que tiene dicha red es el siguiente:

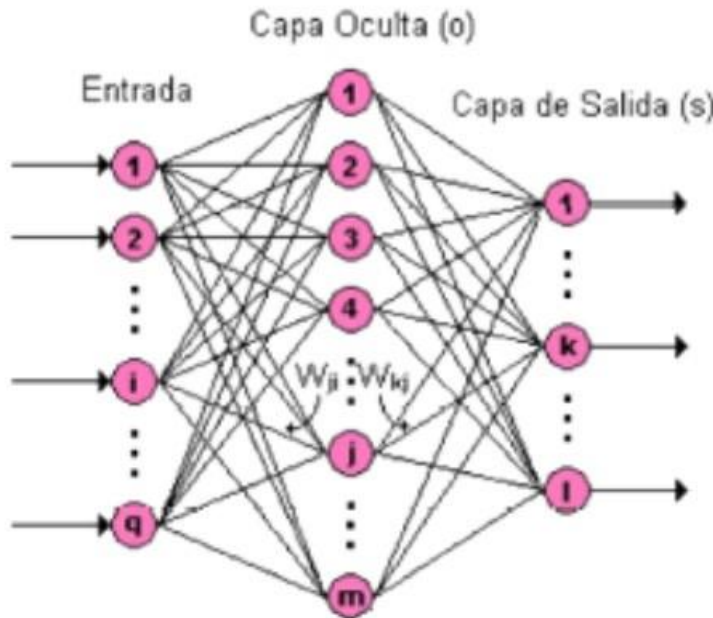


Figura 22: Red BackPropagation

Para esta red en concreto, se han realizado distintas versiones.

En una primera versión de este proyecto se eligió entrenar la red con todos los sensores e ir probando con varias redes, variando el número de neuronas en la capa oculta.

Pero dado que dio unos resultados poco satisfactorios, se fueron quitando sensores, hasta que se dejó sólo con el acelerómetro. En este caso se hicieron varias pruebas, como se verá en el capítulo 5:

- ❖ Para 10 neuronas en la capa oculta y con 50 muestras por estado daba buenos resultados con el fichero más desfavorable (andando2) con un 60% de aciertos para andando.
- ❖ Ahora bien, cambiando el frame de muestras, que es la ventana de muestras que se consideró para crear una muestra preprocesada, a 25 se consiguen mejores resultados que con 30. Asimismo cambiando "andando2" a fichero de entrenamiento, al ser el



desfavorable, se obtienen mejores resultados con el resto de ficheros "andando".

- ❖ También se comprobó que con las muestras del FreeIMU [27] no da tampoco buenos resultados, ya que este algoritmo proporciona datos sólo de la orientación de la IMU en un instante dado y para clasificar los movimientos del animal, hacen falta también los movimientos de translación de éste, no sólo los de la orientación.

Después de que se demostrara que los datos del acelerómetro son los más adecuados para el entrenamiento de la red, se propuso un script para ir variando tanto los frames (número de muestras en bruto que se cogen para realizar una muestra preprocesada), como el número de neuronas en la capa oculta.

Así se obtienen los resultados para distintas versiones de las redes, variando distintos parámetros de la red neuronal como en este caso, el número de neuronas en la capa ocultas y el tamaño del frame de muestras en RAW.

Se decidió tener una única capa oculta, ya que se demuestra que para tareas de clasificación es más que suficiente, y puede generalizar bastante bien para otros casos desconocidos, pero que estén en los márgenes de las entradas con las que se entrenó dicha red. En otro caso la red no podría generalizar, cuando la muestra que se le proporciona se sale de los márgenes de las entradas proporcionadas en los entrenamientos.

En cuanto a los tiempos de training que se han obtenidos, han sido muy cortos, llegando incluso a 30 segundos aproximadamente por caso.

Esto es debido a que se ha elegido la opción "useGPU" que implementa MATLAB en su toolbox de redes neuronales, lo cual acelera el procesamiento del entrenamiento hasta tal punto que mejora sustancialmente la "performance" del entrenamiento y la posterior ejecución de la red cuando esta sea verificada.

## 4.2 Redes neuronales en MATLAB

Como ya se indicó en la introducción, MATLAB dispone de un toolbox de redes neuronales con el que se puede trabajar con varios modelos, incluido el perceptrón multicapa utilizado en este trabajo.

### 4.2.1 Algoritmo de entrenamiento.

En MATLAB existen varios algoritmos de entrenamiento que utilizan el descenso de gradiente como el explicado en este trabajo o métodos Jacobianos.

Para una extensa explicación de los métodos utilizados en MATLAB véase Hagan, M.T., H.B. Demuth, and M.H. Beale, *Neural Network Design*, Boston, MA: PWS Publishing, 1996, Capítulos 11 and 12. En la tabla de abajo se muestra algunos de los principales algoritmos que utiliza MATLAB para entrenar este tipo de redes:

Function	Algorithm
<code>trainlm</code>	Levenberg-Marquardt
<code>trainbr</code>	Bayesian Regularization
<code>trainbfg</code>	BFGS Quasi-Newton
<code>trainrp</code>	Resilient Backpropagation
<code>trainscg</code>	Scaled Conjugate Gradient
<code>traincgb</code>	Conjugate Gradient with Powell/Beale Restarts
<code>traincgf</code>	Fletcher-Powell Conjugate Gradient

*Tabla 2: Funciones del ToolBox de Matlab*

El algoritmo "trainlm" es el que se utiliza en este trabajo junto con la red "patternnet" y ha dado buenos resultados en las pruebas con la red neuronal y fue bastante rápida en la convergencia del entrenamiento de la red con los datos dados.

Para entrenar la red para la clasificación se ha tenido que diseñar un algoritmo basado en la FFT para realizar el preprocesamiento de las muestras y facilitar la tarea de clasificación de la red neuronal. Todo este algoritmo se explicará de una manera más extensa en el capítulo 5.

En MATLAB existe una opción para acelerar el entrenamiento que usa la GPU como acelerador de entrenamiento. Para usar esta opción hace falta

que el ordenador donde se tenga instalado MATLAB disponga de una tarjeta gráfica NVIDIA Gforce compatible con CUDA.

De esta manera se puede usar el entrenamiento en MATLAB usando el comando 'useGPU', dentro de las funciones de ***train*** y ***net***, y proporciona entrenamientos rápidos sin usar CPU apenas.

Un ejemplo de entrenamiento se puede ver en el fragmento que se ha reproducido del código en MATLAB:

```
net = patternnet(n_neuronas);  
  
net = configure(net,datosentrenamiento,targetout1);  
  
net=train(net,datosentrenamiento,targetout1,'useGPU'  
, 'yes', 'showResources', 'yes');
```

La función "patternnet" es la que crea e inicializa los pesos de la red. Tiene un argumento, el número de neuronas en la capa oculta. En este caso como utilizamos la red para clasificación, se utiliza este comando.

La función "configure" configura la red propiamente dicha, ajustando el número de entradas y salidas a las características de los datos (número de elementos en los vectores de entradas y salidas)

La función "train" es la que se encarga de entrenar la red, aquí es donde podemos ajustar parámetros de entrenamientos como el algoritmo de entrenamiento, aparte de los datos de entrada y salida deseadas.

#### 4.2.2 Chequeo de la red en MATLAB

Hay diversas maneras para chequear la red en MATLAB, se ha optado por el siguiente método:

Después del entrenamiento, se prueba la red con el comando 'net', que tiene como argumento el vector de entrada de la red. En el código en MATLAB, queda: **sal=net(datosverificacion1)**,

siendo "**sal**" el vector de salida y "**datosdeverificacion1**" el vector de entrada recogido del preprocesamiento del archivo de muestras de test. El

preprocesamiento es necesario, ya que la red se ha entrenado con datos previamente preprocesados.

En segundo lugar, se obtienen las salidas de todas las muestras de test y se normalizan a valores de 1 ó 0 dependiendo si algunos de los valores del vector de salida es el máximo o no propiamente dicho.

Por último, y ya con las salidas normalizadas se comprueba cuantas muestras de test es capaz la red neuronal de clasificar apropiadamente.

Este método ha sido muy satisfactoria a la hora de testear la red, ya que así se puede comparar con los distintos casos que se han realizados con las distintas versiones de la red realizadas para comparar los resultados y elegir así la que obtenga mejores tasas de aciertos.

El toolbox de redes neuronales de MATLAB también incluye varios comandos para testear la red, y ver la "performance" que se obtienen con un conjunto dado de muestras de test. Por ejemplo:

"plotperf (tr)" que muestra la performance de la red que se ha entrenado, donde tres la estructura devuelta por la función "train", donde se puede ver por ejemplo los resultados de la validación o test con "tr.valInd".

Pero debido a que sólo necesitamos valorar la red según el nivel de aciertos y como un ejercicio puramente académico se ha desarrollado el algoritmo de test explicado en este trabajo de forma más extensa.

En el próximo capítulo se hablará de cómo se obtuvo la base de datos de muestras de entrenamiento y de test y se dará una extensa explicación de cómo se realizaron los experimentos y con qué instrumentos se obtuvo las muestras en raw utilizadas en el presente trabajo.

## 5 OBTENCIÓN DE LA BASE DATOS. ENTRENAMIENTO Y TEST

### 5.1 Collares

Hoy en día, la monitorización del comportamiento de los animales es un gran desafío para la tecnología actual. Existen en el mercado muchos equipos que son capaces de realizar seguimiento por GPS y algunos tienen incluso IMU, para seguir sus cambios en la actividad del animal.

Normalmente se ha centrado la atención en sistemas que sólo envían datos a sistemas computarizados, que se encargaban de procesar los parámetros bajo estudio, como son la cópula, los movimientos migratorios, los movimientos para la obtención de alimento, etc.

Minerva, es un proyecto que tiene como objetivo el desarrollo de sistemas empotrados con requerimientos mínimos de energía, que serán capaces de procesar información combinada de sensores, como la IMU y sensores biométricos que midan por ejemplo el pulso cardiaco, para la clasificación de comportamientos en tiempo real [27].

Como el comportamiento del animal requiere una observación constante y a tiempo real, se necesitan unos collares que puedan ser usados para la recogida efectiva de datos a la vez que sean cómodos para el animal. Un ejemplo se puede ver en la fotografía de abajo:



*Figura 23: Caballo con collar*

Aplicando los mismos principios que los "wearables" para el seguimiento de la salud de una persona, se ha desarrollado un equipo como el descrito que permite conseguir mucha información acerca de animales en libertad y en semi-libertad.

Estos sistemas se basan en la tecnología MEMS (Microelectromechanical systems) para los sensores que conforman la IMU.

Estos sistemas se caracterizan por tener un consumo bajo de energía y tener un pequeño tamaño. En la siguiente figura se aprecia mejor:

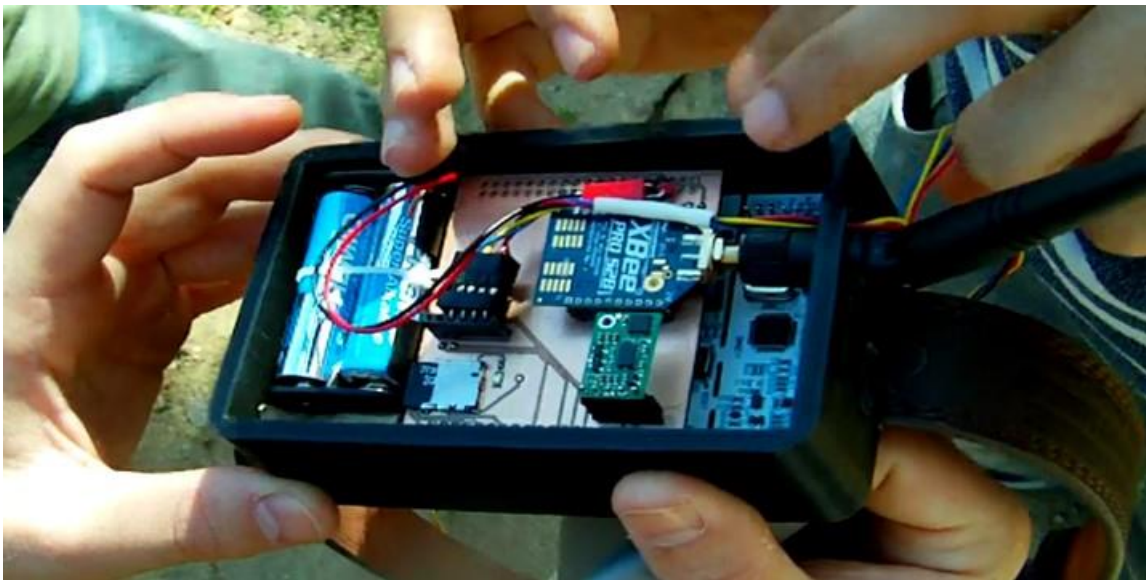


Figura 24: Interior del collar

El collar está formado por varios sistemas como se muestra a continuación en la siguiente figura:

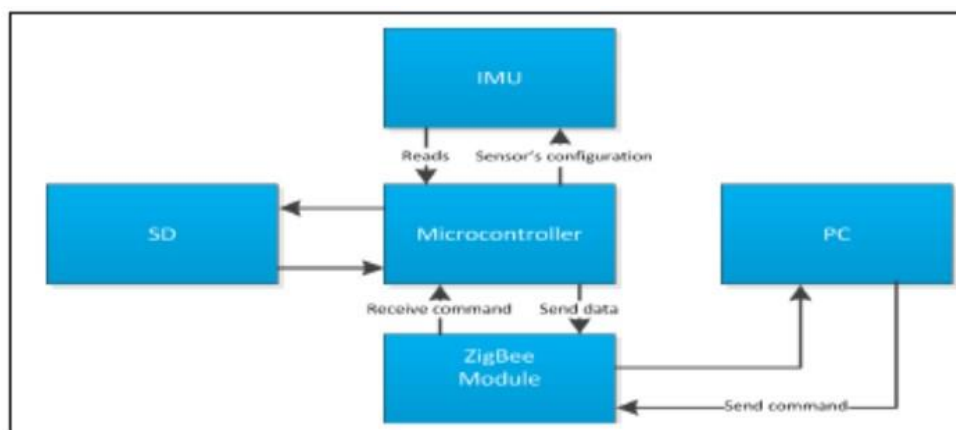


Figura 25: Arquitectura del collar

En esta figura se puede apreciar la presencia de un PC, que no es más la estación receptora que se encarga de recibir los datos emitidos por el módulo de Zigbee del collar.

Los diversos componentes de hardware que se aprecian en la figura son:

❖ Inertial Measurement Unit( IMU)

Esta unidad incluye: un acelerómetro, un giróscopo y un magnetómetro. En este trabajo se usó la IMU MinIMU-9V2 IMU. Cada uno de los sensores tiene una resolución de 12 bit y se comunica por I2C . El giróscopo tiene una escala de 250 a 2000 dps (grados por segundos), ya que el giróscopo mide la velocidad angular. El acelerómetro mide entre 2g y 16g, siendo g la aceleración normal de la gravedad (en condiciones ideales) y el magnetómetro tiene una escala de 1.3 a 8.1 gauss.

❖ Zigbee

Es una especificación de protocolos orientada a WPAN (wireless personal area network), que se basa en el estándar 802.15.4 un estándar, cuyo principal objetivo es reducir al máximo posible el consumo energético para prolongar la vida de las baterías.

Tiene dos características que lo hace adecuado para el collar: consumo de energía bajo y red mallada, ya que lo que se desea es hacer una red de mota, en la que cada mota es un animal.

En el collar se usa para emitir los resultados de la clasificación de la red neuronal, es decir, emitir el comportamiento del animal predicho por la red en ese instante, para configurar el collar y calibrar los sensores.

❖ Microcontrolador

Para este propósito se ha utilizado un microcontrolador ARM Cortex M4, de la marca ST, alimentado por una batería externa.

El microcontrolador es utilizado para configurar la IMU y el módulo de ZigBee, realizar la clasificación del comportamiento del animal en base a los datos recogidos por la IMU y configurar ZigBee para la transmisión de los datos. Tanto el módulo de ZigBee como la IMU se comunican con el microcontrolador mediante I2C.



### ❖ Tarjeta SD

La principal tarea de la tarjeta SD será de guardar la configuración del collar así como los datos de los sensores en sí.

La información se guarda en archivos de tipo CSV, caracterizado por filas y columnas. Cada fila representa una muestra de todos los sensores y las columnas son los distintos campos de la muestra tomada.

## 5.2 Experimentos

Para realizar la tarea de la recogida de datos se realizaron una serie de experimentos con los animales.

El lugar donde tuvo lugar dicho experimentos fue en una pista de rodamiento para caballos en la Estación Biológica de Doñana, sita en el Parque Nacional de Doñana en Huelva, España.

En la siguiente foto por satélite se muestra el lugar donde se realizaron dichos experimentos:





Para el estado "parado", se puso el collar a adquirir datos con el caballo parado y guardándolo en la SD que tiene el collar incorporado. En la siguiente figura se muestra el caballo en reposo y el collar en su cuello:



*Figura 26: Caballo en reposo*

Para el estado "andando", se guio al caballo a través de la pista de rodamiento. Una persona guio el caballo a través de la pista a paso lento, mientras se adquirían los datos de la IMU:



*Figura 27: Caballo andando*

En el “trote”, como cuando se adquirirían los datos para el estado "andando", se procedió de la misma manera, pero esta vez a paso rápido, de tal manera que el caballo trotara y así poder recoger datos del trote:



*Figura 28 Caballo al trote*

Cuando ya se han obtenido los datos de la IMU de los distintos experimentos ya se puede empezar a realizar la base de datos de muestras para el entrenamiento tal como se ve en la fotografía de abajo.



*Figura 29: Adquisición de datos del collar*

### 5.3 Archivos en RAW

De los experimentos se extrajeron una serie de archivos con los resultados de la IMU en cada uno de los estados del animal, corriendo, parado y trotando.

En este caso se han obtenido unos cuantos de archivos por cada estado del animal, cada uno con una serie de muestras. En la tabla siguiente se expone para cada archivo en formato CSV el número de muestras que contiene, entendiéndose por muestra aquí muestra cada conjunto de todos los resultados de los sensores de la IMU en cada instante de tiempo:

Archivo	Numero de muestras
andando1	2634
andando2	5259
andando3	4691
parado1	5991
trotando1	3412
trotando2	2845
trotando3	2096

*Tabla 3: Número de muestras por cada archivo de muestras en raw*

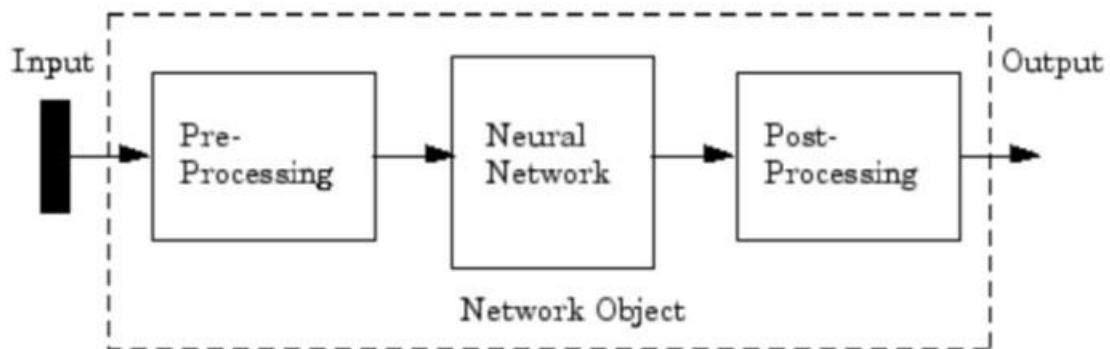
Cada columna de cada archivo corresponde a un sensor y cada fila a un instante de tiempo. El número de muestras escogidas en cada archivo es el número de muestras que tiene el archivo con menores muestras, en este caso, trotando3, para que sea coherente.

#### 5.4 Obtención de los datos en RAW.

En MATLAB, cuando queremos trabajar con redes neuronales, se debe realizar un pre-procesamiento a las muestras, en el que primero se hace una subdivisión en el conjunto de muestras y después se le aplicará el algoritmo que convenga para procesar las muestras y poder así facilitar la tarea a las redes neuronales.

Este algoritmo es la FFT, algoritmo explicado más detalladamente en la introducción, cuyo papel en el pre-procesamiento se explicará posteriormente.

El diagrama que viene a continuación, muestra como es la arquitectura seguida para utilizar cualquier red neuronal en MATLAB:



*Figura 30: estructura para manejar las redes neuronales*

Para realizar el sistema con redes neuronales que sean capaz de clasificar los distintos patrones que queremos distinguir en los datos, la primera etapa es recopilar un conjunto de datos en un entorno controlado, y caracterizar los distintos estados, que en el caso de este trabajo, son los estados del animal como trotar, andar y correr.

Para cada "estado", se recogen datos pertinentes de un conjunto de sensores, a saber, acelerómetro, magnetómetro y giroscopo.

Para realizar dicha tarea es necesario disponer de los collares, descritos en la introducción de este capítulo, que es el dispositivo que contiene la IMU y un microcontrolador que se dedica a recoger datos de la IMU y almacenarlos en una SD.

### 5.5 Obtención de la base de datos para entrenamiento y test

Una vez que ya tenemos un conjunto de muestras por cada estado a clasificar, como por ejemplo 3000 muestras por cada sensor de la IMU, ya podremos empezar a realizar las muestras de entrenamiento.

Para realizar las muestras pre-procesadas, se siguió el siguiente procedimiento:

Para empezar, se han tomado los datos de cada sensor que se ha considerado conveniente, que inicialmente fueron todos.

Por cada sensor se han dividido el conjunto de muestras que se han recogido en cada experimento. La forma que se ha dividido es la siguiente.

Del conjunto total de las muestras de cada sensor, se han tomado subconjuntos de muestras consecutivos, cuyo número de muestras que forman cada subconjunto son iguales en todos los subconjuntos. Como se muestra a continuación en el diagrama:

Conjunto de muestras de cada sensor
Subconjunto de $n$ muestras
Subconjunto de $n$ muestras
Subconjunto de $n$ muestras
...
Subconjunto de $n$ muestras

*Tabla 4: Ejemplo de muestra pre-procesada*

$n$  es el número de muestras por cada subconjunto considerado.

A cada subconjunto de  $n$  muestras se le introduce en una FFT, cuyo resultado, se dividirá posteriormente por la mitad quedándose así con  $(n/2)-1$  muestras y después se normalizarán los resultados en el intervalo  $[-1,1]$  cogiendo el máximo del conjunto de todos los subconjuntos calculados y dividiendo ese máximo por todos los valores de cada subconjunto para que el algoritmo de entrenamiento converja más fácilmente.

Ahora ya tenemos la muestra hecha y ya podemos entrenar y testear la red.

Como se ha dispuesto de varios archivos de muestras por cada estado, se ha podido utilizar un archivo para el entrenamiento y otro para el test para cada estado.

## 5.6 Preprocesado de las muestras

Para realizar la clasificación de las muestras en el collar, se han desarrollado dos puntos de vistas distintos:

- ❖ realizar el filtrado con los datos en RAW (bruto), realizarles la FFT, y después introducirlos en la red para hacer la clasificación, y
- ❖ pasar los datos de los 3 sensores (acelerómetro, giróscopo, y magnetómetro) por el algoritmo FreeIMU, después por la FFT y por último, por la red para la clasificación de las muestras.



Para ejemplificarlos se muestran un esquema a continuación:

*Figura 31: Estructura de pre-procesamiento sin FreeIMU*

Este ejemplo es para la arquitectura sin FreeIMU. El número de entradas de la red varía, ya que no se sabe cuál será el número óptimo de entradas que dará el resultado óptimo. Con la FFT se descomponen los datos del movimiento en un patrón de frecuencia que la red ya entrenada puede reconocer y clasificar.

Para la arquitectura con FreeIMU, tenemos la siguiente figura:



*Figura 32: Estructura de pre-procesamiento con FreeIMU*

En todas estas arquitecturas resalta el hecho de que se hace un pre-procesado con la FFT.



Cuando se recogen los movimientos mediante un acelerómetro, se puede pensar que existe en la señal unos determinados patrones periódicos, ya que en el movimiento de todo animal bípedo o cuadrúpedo, sus movimientos son cíclicos y responden a una serie de patrones característicos como por ejemplo los movimientos que realiza el animal en el trote contiene componentes de mayor frecuencia que cuando está andando.

Cuando el animal está parado, los datos del acelerómetro contiene componentes de baja frecuencia, pero en muy pocas muestras se aprecian componentes de alta frecuencia, ya sea por movimientos bruscos de la cabeza o por otros efectos.

Las salidas de la red, en este caso, son constantes porque representan los estados del animal recogidos en campo y son: andando, parado, y trotando. En el futuro se recogerán más datos del animal como son pulso cardíacos y se tendrán más datos acerca de los distintos estadios en la vida del animal, como son cuando se está alimentando, apareando, etc.

Para representar las salidas, se ha optado por poner de salida objetivo un vector de 3 componentes de forma  $[e_1, e_2, e_3]$ ,  $e_n$  que depende del estado que se esté considerando. Cuando se esté en un determinado estado, uno de los componentes estará a 1, mientras que los otros serán 0.

Cuando se valide la red, se tendrá en cuenta que están o dará unos resultados exactos tales como  $[0 \ 0 \ 1]$  o  $[1 \ 0 \ 0]$  sino que sus salidas serán más difusas y habrá que comprobar cuáles de las componentes de dicho vector es la mayor para establecer el estado que predice la red.

## 6 RESULTADOS

En este capítulo, se presentarán los resultados y como se han conseguido, además de explicar los distintos resultados obtenidos con las redes neuronales.

Para la obtención de los resultados, primero se han reprocesado las muestras, y después se han realizados los distintos entrenamientos para cada caso, según sea el número de entradas de datos, si es FreeIMU o en RAW o el número de neuronas en la capa oculta.

### 6.1 Resultados con acelerómetro

Ahora se presentarán los resultados de la red neuronal entrenada y testada con datos pre-procesados del acelerómetro.

Para la validación de la red neuronal, se ha realizado 2 casos, en un caso se ha hecho con 80 muestras y en el otro caso con 100 muestras, a fin de caracterizar mejor los porcentajes de error cometidos por la red neuronal.

En el caso de las 80 muestras de validación se ha obtenido:

Tabla de Resultados (para 80 muestras de validación, acelerómetro)										
Estados		Andando			Parado			Trote		
Resultados por estado		Andando	Parado	Trote	Andando	Parado	Trote	Andando	Parado	Trote
10 Neuronas	5 frames	71,25	22,5	6,25	11,76	87,06	1,18	22,5	7,5	70
	10 frames	77,5	15	7,5	5,882	91,765	2,353	3,75	0	96,25
	15 frames	81,25	13,75	5	8,24	87,06	4,71	5	0	95
	20 frames	81,25	7,5	11,25	7,06	87,06	5,88	7,5	1,25	91,25
	25 frames	80	12,5	7,5	7,06	89,41	3,53	10	0	90
20 Neuronas	5 frames	66,25	25	8,75	8,23	89,41	2,35	21,25	8,75	70
	10 frames	78,75	17,5	3,75	4,71	94,12	1,18	6,25	0	93,75
	15 frames	63,75	31,25	5	7,06	88,24	4,71	11,25	1,25	87,5
	20 frames	86,25	10	3,75	7,06	89,41	3,53	6,25	0	93,75
	25 frames	75	17,5	7,5	8,24	88,24	3,53	12,5	0	87,5
30 Neuronas	5 frames	71,25	22,5	6,25	11,76	87,06	1,18	16,25	8,75	75
	10 frames	76,25	20	3,75	2,35	96,47	1,18	3,75	0	96,25
	15 frames	80	16,25	3,75	10,59	85,88	3,53	6,25	0	93,75
	20 frames	82,5	12,5	5	3,53	89,41	7,06	11,25	1,25	87,5
	25 frames	73,75	16,25	10	2,35	91,76	5,88	6,25	0	93,75

Tabla 5: Resultados %de aciertos a partir de 80 muestras (sin FreeIMU)

Como se ve en la Tabla 5 se han obtenido buenos resultados en la clasificación de las distintas muestras directamente obtenidas del acelerómetro. Todos los valores están en porcentaje.



Cada frame es el subconjunto de muestras que se cogen para ser pre-procesadas. Para cada valor de frame y del número de neuronas en la capa oculta se ha obtenido un valor de error. Posteriormente se elige, de entre las distintas configuraciones, la que obtenga mejores resultados.

Para elegir en esta tabla los mejores valores, se debe considerar una mayor tasa de éxito en los estados andando y trotando, al ser movimientos muy parecidos y algunas veces pueden confundir a la red.

Los valores que mejor se ajustan se muestran en la siguiente tabla, en la que, para el caso de 100 muestras de validación, los resultados obtenidos son parecidos a los anteriores:

Tabla de Resultados (para 100 muestras de validación, con acc)										
Estados		Andando			Parado			Trote		
Resultados por estado		Andando	Parado	Trote	Andando	Parado	Trote	Andando	Parado	Trote
10 Neuronas	5 frames	75	20	5	15	84	1	31	7	62
	10 frames	77	15	8	6	91	3	21	3	76
	15 frames	80	14	6	8	88	4	12	2	86
	20 frames	79	9	12	7	87	6	11	2	87
	25 frames	82	11	7	6	91	3	9	0	91
20 Neuronas	5 frames	67	26	7	11	87	2	30	9	61
	10 frames	78	16	6	6	93	1	22	1	77
	15 frames	68	28	4	6	90	4	11	5	84
	20 frames	84	9	7	6	89	5	15	1	84
	25 frames	79	15	6	7	90	3	7	0	93
30 Neuronas	5 frames	73	22	5	15	84	1	28	9	63
	10 frames	76	19	5	3	95	2	18	4	78
	15 frames	82	14	4	10	87	3	13	0	87
	20 frames	78	14	8	3	89	8	13	1	86
	25 frames	76	15	9	2	93	5	8	0	92

Tabla 6: Resultados % aciertos a partir de 100 muestras (sinFreeIMU)

Como se puede ver, los resultados son muy buenos, incluso con 100 muestras de validación, y con esto se nota que utilizar una red neuronal es una buena opción para clasificar los movimientos del animal, y con sólo un sensor de la IMU se han conseguido buenos resultados.

## 6.2 Resultados con FreeIMU

Los resultados que se obtuvieron con el Free IMU, son peores que los que se han obtenidos con el acelerómetro.

A continuación se mostrarán los resultados obtenidos al entrenar la red con el esquema, en el que los datos del FreeIMU son tomados como entradas del algoritmo de pre-procesado de las muestras.

Para el caso de 100 muestras tenemos:

Tabla de Resultados (para 100 muestras de validación, con freeimu)										
Estados		Andando			Parado			Trote		
Resultados por estado		Andando	Parado	Trote	Andando	Parado	Trote	Andando	Parado	Trote
10 Neuronas	5 frames	49	43	8	18	80	2	28	55	17
	10 frames	73	27	0	31	69	0	45	54	1
	15 frames	30	26	44	14	49	37	13	56	31
	20 frames	35	20	45	28	51	21	17	37	46
	25 frames	12	2	86	10	0	90	6	4	90
20 Neuronas	5 frames	30	52	18	10	88	2	6	91	3
	10 frames	21	40	39	3	81	16	19	65	16
	15 frames	43	39	18	16	64	20	15	64	21
	20 frames	48	43	9	38	54	8	29	53	18
	25 frames	35	57	8	22	64	14	26	61	13
30 Neuronas	5 frames	27	48	25	0	81	19	15	84	1
	10 frames	46	40	14	5	71	24	21	57	22
	15 frames	58	30	12	25	57	18	16	54	30
	20 frames	38	29	33	30	39	31	12	57	31
	25 frames	44	34	22	22	54	24	18	57	25

Tabla 7: % de acierto con FreeIMU y calculadas a partir de 100 muestras

Como se puede ver en esta tabla, los resultados son peores que los obtenidos cuando se entrenó y testeó la red con muestras pre-procesadas del acelerómetro.

Con unas 80 muestras de validación se obtienen resultados similares, aunque se mejora los resultados:

Tabla de Resultados (para 80 muestras de validación, con freeimu)										
Estados		Andando			Parado			Trote		
Resultados por estado		Andando	Parado	Trote	Andando	Parado	Trote	Andando	Parado	Trote
10 Neuronas	5 frames	46,25	45	8,75	21,18	77,65	1,18	46,25	45	8,75
	10 frames	66,25	33,75	0	30,59	69,41	0	58,75	41,25	0
	15 frames	32,5	22,5	45	11,76	54,11	34,12	13,75	40	46,25
	20 frames	30	22,5	47,5	29,41	48,24	22,35	32,5	23,75	43,75
	25 frames	15	2,5	82,5	11,76	0	88,24	11,25	2,5	86,25
20 Neuronas	5 frames	30	55	15	11,76	85,88	2,35	12,5	85	2,5
	10 frames	17,5	47,5	35	2,35	82,35	15,29	15	63,75	21,25
	15 frames	45	35	20	15,29	68,23	16,47	22,5	57,5	20
	20 frames	47,5	43,75	8,75	40	50,58	9,41	45	46,25	8,75
	25 frames	37,5	53,75	8,75	21,18	62,35	16,47	31,25	51,25	17,5
30 Neuronas	5 frames	22,5	50	27,5	0	77,65	22,35	13,75	68,75	17,5
	10 frames	45	42,5	12,5	3,53	72,94	23,53	18,75	56,25	25
	15 frames	58,75	27,5	13,75	23,53	61,17	15,29	41,25	42,5	16,25
	20 frames	31,25	30	38,75	32,94	40	27,06	28,75	48,75	22,5
	25 frames	46,25	32,5	21,25	23,53	50,59	25,88	23,75	35	41,25

Tabla 8: % de acierto con FreeIMU y calculadas a partir de 80 muestras

En estos resultados apenas se aprecian diferencias significativas, incluso se observan varios casos en donde la red no ha podido clasificar bien los datos.

Para resaltar mejor las diferencias entre los distintos estados, se han realizados graficas que comparan, por cada estado y según el número de muestras de validación.

### 6.3 Graficas de comparación

Las gráficas realizadas tomando como eje de abscisas distintos números de frames por cada número de neuronas en la capa oculta:

Para 80 muestras de validación, tenemos los siguientes resultados para los distintos estados del animal:

Para "andando", obtenemos la siguiente gráfica:

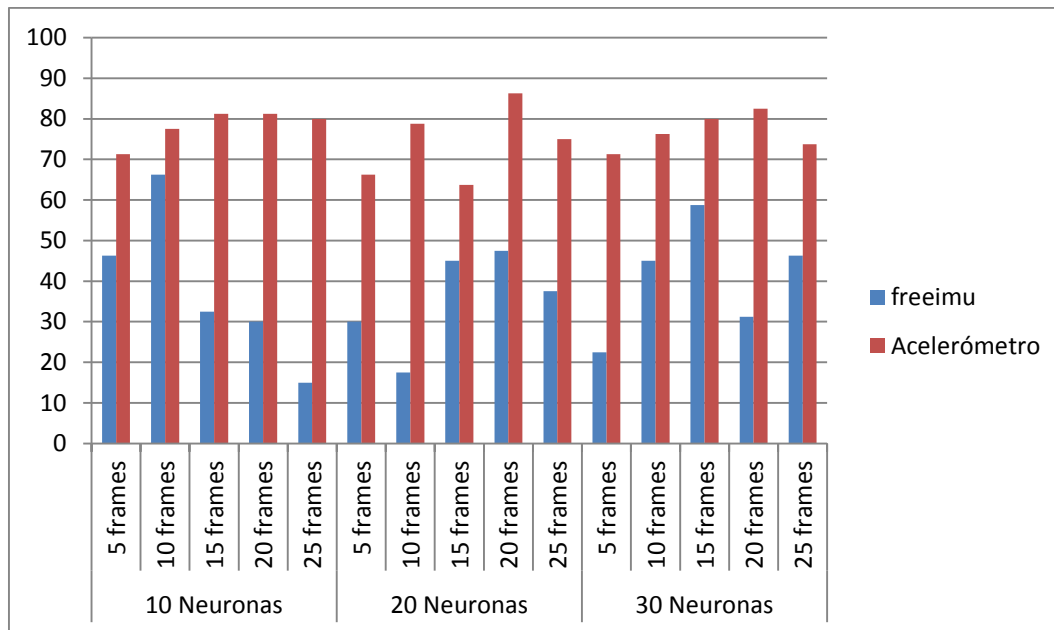


Figura 33: Gráfica para "andando", 80 muestras de validación

Como se puede ver en esta gráfica, los mejores resultados son para 30 neuronas y 20 frames y para 20 neuronas y 20 frames.

Para el estado "parado" obtenemos la siguiente gráfica

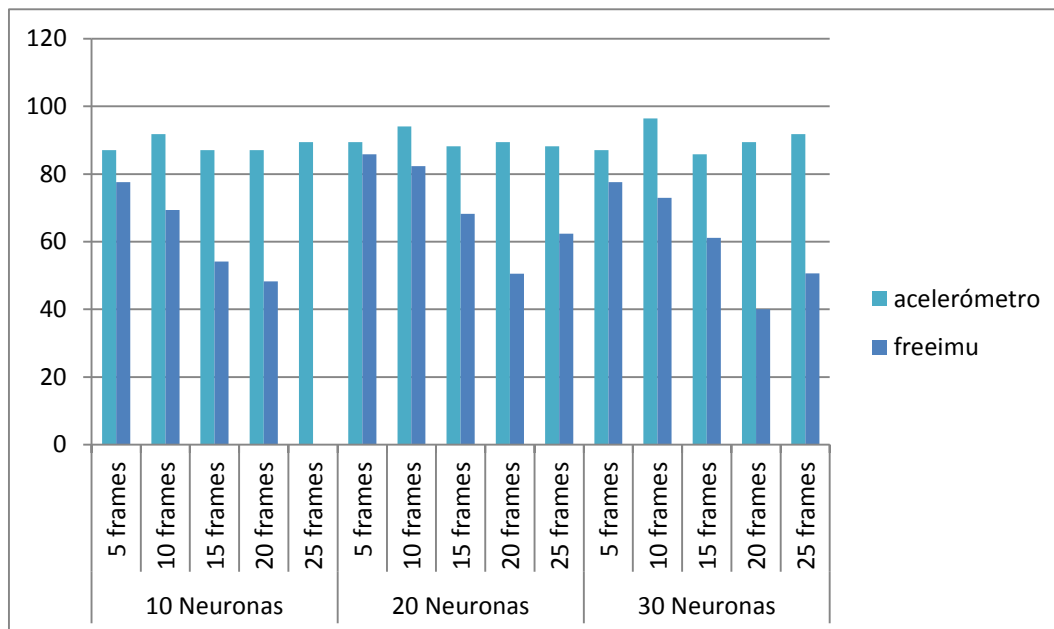


Figura 34: Gráfica para "parado", 80 muestras de validación

Esta vez los resultados son algo distintos para este estado, los mejores resultados son para acelerómetro y 30 neuronas con 10 frames, aunque los resultados son similares al caso anterior.

Para "trote" obtenemos similares resultados,

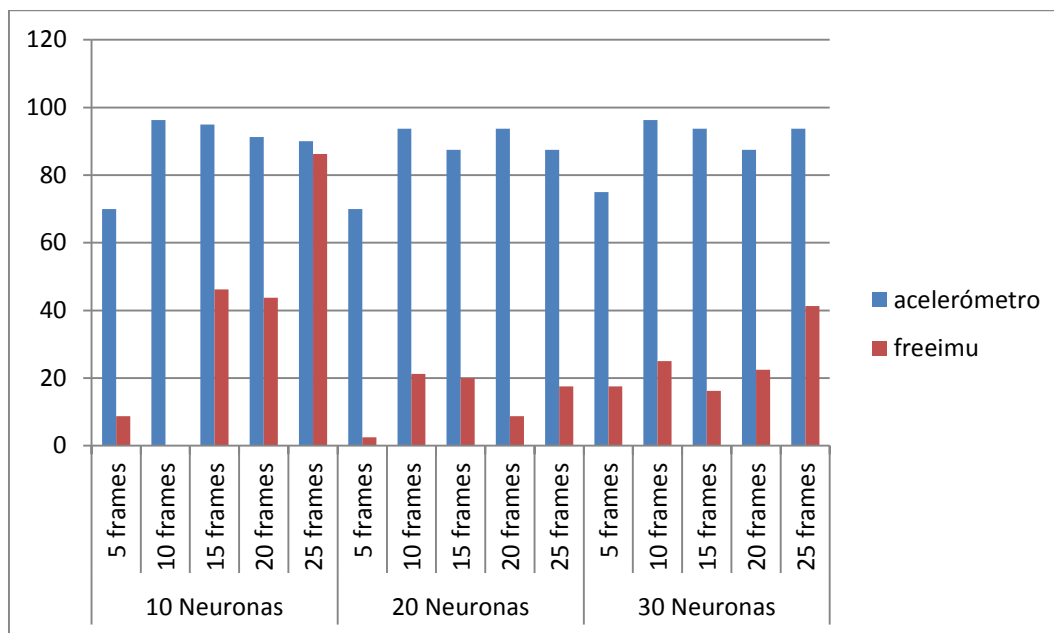


Figura 35: Gráfica para "trote", 80 muestras de validación

Esta vez 30 neuronas y 10 frames de entrada, y 20 neuronas con 10 frames también tiene resultados bastantes altos.

Para 100 muestras de validación, los resultados son bastantes similares a la anterior pero se han calculados dichos resultados para tener una mejor visión estadística del error que se tiene al clasificar los datos en la red neuronal.

Para el estado "andando" se ha obtenido la siguiente gráfica:

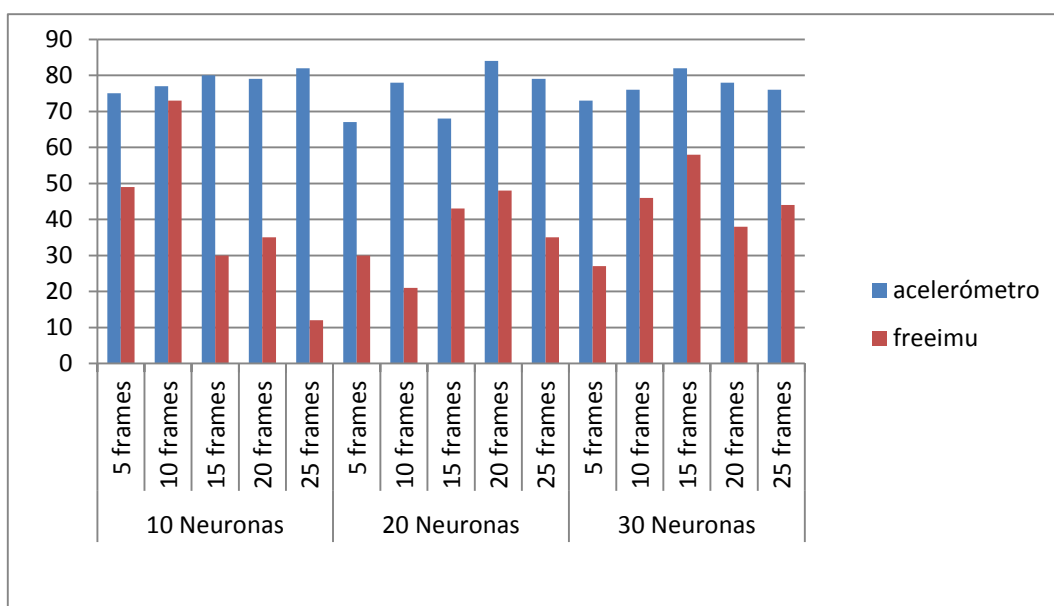


Figura 36: Gráfica para "andando", 100 muestras de validación

Aquí se obtiene casi los mismos resultados que el anterior, teniendo 20 neuronas con 20 frames como mejor resultado.

Para el caso de "parado", se ha obtenido la siguiente gráfica:

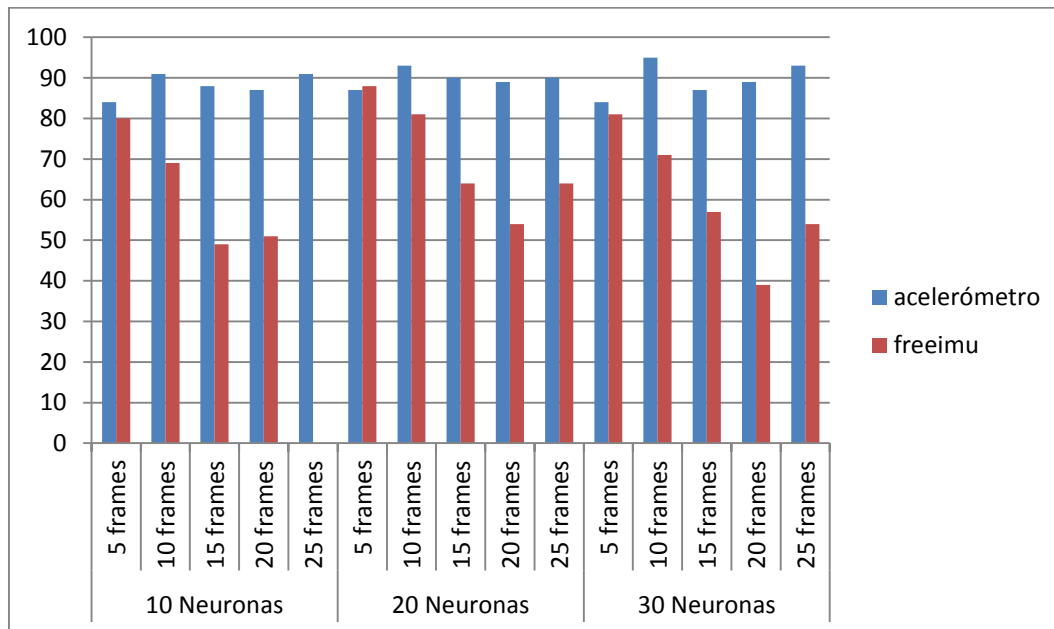


Figura 37: Gráfica para "parado", 100 muestras de validación

Ahora los mejores resultados se han obtenidos para 20 neuronas con 10 frames y 30 neuronas con 10 frames de entrada, parecido al caso de 80 muestras de validación.

Para "trote", el resultado que se ha obtenido ha sido :

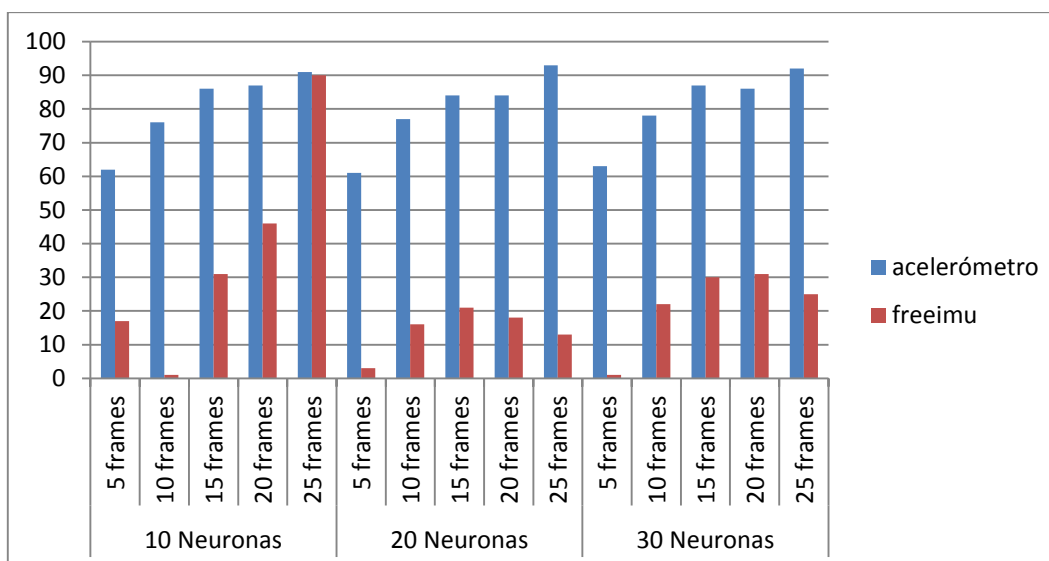


Figura 38: Gráfica para "trote", 100 muestras de validación

Aquí los mejores resultados se han obtenido en 10 neuronas con 25 frames, 20 neuronas con 25 frames y 30 neuronas con 25 frames.

Ahora se mostrará una tabla para resumir los mejores resultados obtenidos:

Mejores Resultados		
Estado	Neuronas ocultas	Frames
Andando	20	20
	30	20
	20	20
Parado	30	10
	20	10
	30	10
Trote	30	10
	20	10
	10	25
	20	25
	30	25

Figura 39: Mejores resultados de la red

Para la implementación de la red para este propósito se puede escoger 20 neuronas en la capa oculta y 10 frames ó 30 neuronas en la capa oculta y 10 frames, que son las configuraciones de la red que mejores resultados han dado.

Como se ha explicado anteriormente, el algoritmo de FreeIMU, es un algoritmo utilizado para la fusión de los sensores de la IMU y calcular los ángulos de Pitch, Roll y Yaw. Por este motivo, la clasificación llevada a cabo por la red es deficiente.

Por este motivo se desaconseja el uso del FreeIMU, puesto que ocupa espacio en la memoria del microcontrolador y quita recursos computacionales a otros subsistemas del microcontrolador como puede ser la red neuronal, ya que el FreeIMU es muy costoso computacionalmente.



## 7 CONCLUSIONES

En este trabajo se ha realizado un estudio de la posibilidad de clasificar, mediante redes neuronales artificiales de tipo perceptrón multicapa, los comportamientos de animales de datos obtenidos de una IMU.

Como se ha visto, los datos proporcionados por el acelerómetro son más que suficientes para entrenar dicha red, así que se pueden obviar los datos de los demás sensores, con el ahorro en coste en hardware que esto supone, ya que no se necesitan más sensores que un acelerómetro para la clasificación efectiva de los datos.

Se han probado diversas arquitecturas en el pre-procesamiento, y la que mejor resultado ha dado, es la que sólo tiene el acelerómetro como entrada más la FFT como algoritmo de pre-procesamiento.

En torno al trabajo MINERVA, se ha implementado una red como la utilizada en este trabajo, pero con distinto pre-procesamiento, ya que al ser utilizado en un microcontrolador con pocos recursos computacionales, no se pudo implementar la FFT.

Para implementar dicha red, se ha utilizado la librería FANN. Esta librería es Open Source, que implementa redes neuronales multicapa, con la posibilidad de configurar libremente todos los parámetros de dicha red. Soporta modos de redes totalmente conectada o parcialmente conectada.

También puede funcionar con coma fija o flotante, haciéndola deseable para los sistemas empotrados. También tiene funciones para el entrenamiento y está bien documentada, por lo que su utilización es fácil. Como está escrita en C, puede funcionar en entornos empotrados como el STM32, utilizado en el proyecto MINERVA.

Para utilizarla en dicho microcontrolador, se ha reescrito dicha librería y se ha optimizado para que no consuma los limitados recursos del microcontrolador.

En la librería optimizada no ha hecho uso de las funciones de entrenamiento de la librería original, ya que en el micro sólo se correrá la red. Se ha utilizado



coma fija, porque el micro no tiene FPU, y se ha optimizado el código de las rutinas de la red para así ahorrar memoria en el micro y tiempo de cálculo.

## 8 APENDICE :CÓDIGO EN MATLAB

El script en MATLAB preprocesa las muestras de la IMU, entrena y verifica la red neuronal y extrae los resultados en un archivo .txt para que la lectura sea cómoda.

El archivo m en MATLAB es el siguiente:

```
%datosAndando=xlsread('c:\DatosSensores\andando','Hoja1','A1:I87');
%datosTrote=xlsread('c:\DatosSensores\trote','Hoja1','A1:I87');
%datosParado=xlsread('c:\DatosSensores\parado','Hoja1','A1:I87');
%seleccionar archivos y tipo de preprocesamiento , entre FFT y sin FFT.

datosAnd1=xlsread('c:\DatosSensores\andando2','andando2','A2:C3400');
datosTr1=xlsread('c:\DatosSensores\trotando1','trotando1','A2:C3400');
datosP1=xlsread('c:\DatosSensores\parado1','parado1','A2:C3400');
datosAnd2=xlsread('c:\DatosSensores\andando2','andando2','J2:L3400');
datosTr2=xlsread('c:\DatosSensores\trotando1','trotando1','J2:L3400');
datosP2=xlsread('c:\DatosSensores\parado1','parado1','J2:L3400');
datosAndandol=[datosAnd1];
datosTrotel=[datosTr1];
datosParadol=[datosP1];
%datosAndandol=[datosAnd2];
%datosTrotel=[datosTr2];
%datosParadol=[ datosP2];
datosVal1=xlsread('c:\DatosSensores\trotando2','trotando2','A2:C2845');
datosVal2=xlsread('c:\DatosSensores\trotando2','trotando2','J2:L2845');% con freeimu
datosTrote2=[datosVal1];
%datosTrote2=[datosVal2];
%se va variando
file1=fopen('trotando1(solo acc).txt','w'); % abrimos archivo
% Para cada caso debemos de descomentar la linea que corresponda segun el
% estado que estemos tratando
% file1=fopen('parado1(solo acc).txt','w');
% file1=fopen('trotando1(solo acc).txt','w');

n_muestras=3390;%Se mantienen constante ya que el valor de las muestras
n_muestras_val=2840;
sens=3;
% Para ordenar mejor los datos en archivos de salida txt ,
for n_muestras_proc=80:5:100
    z=['Para un numero:',num2str(n_muestras_proc),'muestras de validacion','\n'];
    fprintf(file1,z);
for n_neuronas=10:10:30
    y=['Para un numero:',num2str(n_neuronas),'neuronas','\n'];
    fprintf(file1,y);
for n=5:5:25
    % utiliza para verificar.
    %n numero de muestras por frames.

n_columnas=fix(n_muestras/(n)); % por estado.
%variables temporales
datosdiezPar=[];
datosdiezspecPar=[];
    datosdiezT=[];
    datosdiezspecT=[];
    datosdiezAnd=[];
    datosdiezAnd1=[];
    datosdiezspecAnd=[];
    datosdiezspecAnd1=[];
    datosentrenamiento=[];
    datosverificacion=[];
    datosverificacion1=[];

for m=1:sens
for i= 1:(n_columnas) %cuantas columnas
    for j=1:n
        datosdiezPar(j,((i+(m-1)*n_columnas)))=datosParadol(((n*(i-1))+j),m); %Por cada
n columnas en un sensor y
```

```

    datosdiezT(j, ((i+(m-1)*n_columnas)))=datosTrotel((n*(i-1))+j),m); %Partimos
muestras de n en n en una matriz
    datosdiezAnd(j, ((i+(m-1)*n_columnas)))=datosAndandol((n*(i-1))+j),m);
end;
    datosdiezspecPar(:, ((i+(m-1)*n_columnas)))=abs((2/n)*fft(datosdiezPar(:, ((i+(m-
1)*n_columnas)))));
    datosdiezspecT(:, ((i+(m-1)*n_columnas)))=abs((2/n)*fft(datosdiezT(:, ((i+(m-
1)*n_columnas)))));
    datosdiezspecAnd(:, ((i+(m-1)*n_columnas)))=abs((2/n)*fft(datosdiezAnd(:, ((i+(m-
1)*n_columnas)))));
    %Normalizamos los valores de todos los vectores con respecto al maximo del vector.
    datosdiezspecPar(:, ((i+(m-1)*n_columnas)))= datosdiezspecPar(:, ((i+(m-
1)*n_columnas)))./max(max(datosdiezspecPar(:, ((i+(m-1)*n_columnas)))));
    datosdiezspecAnd(:, ((i+(m-1)*n_columnas)))=datosdiezspecAnd(:, ((i+(m-
1)*n_columnas)))./max(max(datosdiezspecAnd(:, ((i+(m-1)*n_columnas)))));
    datosdiezspecT(:, ((i+(m-1)*n_columnas)))= datosdiezspecT(:, ((i+(m-
1)*n_columnas)))./max(max(datosdiezspecT(:, ((i+(m-1)*n_columnas)))));
end;
end;

datosdiezspecT=datosdiezspecT(1:fix(n/2)+1,:);
datosdiezspecPar=datosdiezspecPar(1:fix(n/2)+1,:);
datosdiezspecAnd=datosdiezspecAnd(1:fix(n/2)+1,:);
%El siguiente paso es la seleccion de conjunto de entrenamiento y validación a
partir de las
%matrices que contiene los datos de las fft
%Hay que tener cuidado, ya que los subconjuntos de cada eje de cada sensor
%están repartidos todos juntos en las variables finales , y para
%seleccionarlos hay que ver cuantos son por eje de cada sensor para
%separlos adecuadamente , como la fft se hace a cada subconjunto de ellos ,
%el número de subconjuntos no varía , pero si el numero de elementos de
%cada subconjunto que es la mitad.
%columnas por estado, para hacer cte el numero de
%muestras por estados que se le muestran a la red para entrenarlas.
n_colum_entr=fix(n_muestras/(n));
for col=1:n_colum_entr
for j=0:sens-1
    for i=1:(n/2)+1
datosentrenamiento((j*(fix(n/2)+1))+i,col)= datosdiezspecAnd(i, (j*n_columnas)+col);
%Comprobado ,parece que está bien
    end
end
end
for col=(n_colum_entr+1):(n_colum_entr*2)
for j=0:sens-1
    for i=1:(n/2)+1

datosentrenamiento((j*(fix(n/2)+1))+i,col)= datosdiezspecPar(i, (j*n_columnas)+col-
n_colum_entr); %Comprobado ,parece que está bien//SI
% si se quiere cambiar a datos solo en bruto, solo hay que cambiar datosdiezspecPar
por datosdiezPar, igual con datosvalidacion.
    end
end
end
for col=((n_colum_entr*2)+1):(3*n_colum_entr)
for j=0:sens-1
    for i=1:(n/2)+1

datosentrenamiento((j*(fix(n/2)+1))+i,col)= datosdiezspecT(i, (j*n_columnas)+col-
(n_colum_entr*2)); %Comprobado ,parece que está bien
    end
end
end
% Ahora viene los datos de verificacion
% son 3 bucles , cada uno mete los datos de las matricas diezmadasy
% procesadas.
%
%
%datosentrenamiento=datosentrenamiento';

%-----
% Parte en la que se produce los targets
%-----
for i=1:n_colum_entr

```

```

        targetout1(:,i)=[1;0;0];
    end
    for j=(n_colum_entr+1):(n_colum_entr*2)
        targetout1(:,j)=[0;1;0];
    end
    for k=((n_colum_entr*2)+1):(3*n_colum_entr)
        targetout1(:,k)=[0;0;1];
    end
    %net = feedforwardnet(5);
    %net = configure(net,datosentrenamiento,targetval);
    %net = train(net,datosentrenamiento,targetval);
    %-----
    %Hasta aquí ,crear la red, inicializarla , entrenarla y verificarla.
    %-----
    %Procesado de nuevos datos , se sigue el mismo proceso que en los
    %anteriores archivos , pero con la diferencia que son más muestras
    %-----
    n_col_comp=fix(n_muestras_val/(n));%IMPORTANTE! Cambiar si el archivo de verificación
    contiene diferentes numeros de muestras
    for m=1:sens
        for i= 1:n_col_comp %cuantas columnas
            for j=1:n

                datosdiezAnd1(j,((i+(m-1)*n_col_comp)))=datosTrote2(((n*(i-1))+j),m); %Partimos
                muestras de n en n en una matriz
            end;

            datosdiezspecAnd1(:,((i+(m-1)*n_col_comp)))=abs(((2/n)*fft(datosdiezAnd1(:,((i+(m-1)*n_col_comp))))));
            %Normalizamos los valores de todos los vectores con respecto al maximo del vector.

            datosdiezspecAnd1(:,((i+(m-1)*n_col_comp)))=datosdiezspecAnd1(:,((i+(m-1)*n_col_comp)))./max(max(datosdiezspecAnd1(:,((i+(m-1)*n_col_comp)))));

        end;
    end;
    datosdiezspecAnd1=datosdiezspecAnd1(1:fix(n/2)+1,:);

    for col=1:n_muestras_proc
        for j=0:sens-1
            for i=1:(n/2)+1

                datosverificacion1((j*(fix(n/2)+1))+i,col)= datosdiezspecAnd1(i,(j*n_col_comp)+col);
            end
        end
    end
    %IMPORTANTE:Cambiar datosdiezspecAnd1 por datosdiezAnd1 para el caso sin FFT
    %Conclusión: Hay que entrenar la red para más casos de cada uno de los
    %comportamientos , si no puede haber ambigüedades en las salidas!
    net = patternnet(n_neuronas);
    %net = configure(net,datosentrenamiento,targetout1);
    net = train(net,datosentrenamiento,targetout1,'useGPU','yes','showResources','yes');
    sal=net(datosverificacion1);

    for i=1:n_muestras_proc
        salm=max(sal(:,i));
        for j=1:3
            if (salm==sal(j,i)) % dice si es andando (1) , parado(2) ó trote(3)
                proc(i)=j;
            end
        end
    end
    %Si estoy buscando andando ; "2"
    contador=0;
    contador1=0;
    contador2=0;
    for i=1:n_muestras_proc % para cada
        if (proc(i)==1)
            contador=contador+1;
        end
        if (proc(i)==2)
            contador1=contador1+1;
        end
        if (proc(i)==3)
            contador2=contador2+1;
        end
    end

```

```
end
x=['Para un numero',num2str(n),'frames','\n'];
porcentajeand=(contador/n_muestras_proc)*100;%Introducimos en el archivo
porcentajePar=(contador1/n_muestras_proc)*100;
porcentajeTrote=(contador2/n_muestras_proc)*100;
fprintf(file1,x);
fprintf(file1,'%f\n',porcentajeand);
fprintf(file1,'%f\n',porcentajePar);
fprintf(file1,'%f\n',porcentajeTrote);
targetout1=[];
end
end
end
fclose(file1);
```

---

## 9 BIBLIOGRAFÍA

- [1] Alejandro Linares Barranco, Manuel García León "*Mota infraestructura de sensado y transmisión inalámbrica para la observación y análisis de la pauta de animales salvajes o en semilibertad (Minerva)*"
- [2] R. Tapiador, Juan P. Dominguez-Morales, A. Rios-Navarro, D. Gutierrez-Galan, C. Martín-Cañal, D. Cascado-Caballero, A. Linares-Barranco "*Pattern recognition based on sensory fusion and Neural Networks*"
- [3] <http://www.ingelec.uns.edu.ar/pds2803/materiales/cap02/02-cap02-01-historia.pdf>
- [4] [https://www.ugr.es/~dpto\\_am/docencia/Apuntes/Historia\\_series\\_Fourier\\_Canada.pdf](https://www.ugr.es/~dpto_am/docencia/Apuntes/Historia_series_Fourier_Canada.pdf)
- [5] Wikipedia: "*Series de Fourier*"
- [6] [http://www.uv.es/~soriae/tema\\_5\\_pds.pdf](http://www.uv.es/~soriae/tema_5_pds.pdf)
- [7] Wikipedia: "*Transformada discreta de Fourier*".
- [8] [http://www.uv.es/~soriae/tema\\_5\\_pds.pdf](http://www.uv.es/~soriae/tema_5_pds.pdf)
- [9] [https://en.wikipedia.org/wiki/Fast\\_Fourier\\_transform](https://en.wikipedia.org/wiki/Fast_Fourier_transform)
- [10] Wikipedia: "*FFT*"
- [11] <http://www.ehu.eus/Procesadodesenales/tema7/ty3.html>
- [12] [https://www.frro.utn.edu.ar/repositorio/catedras/quimica/5\\_anio/orientadora1/monograias/matich-redesneuronales.pdf](https://www.frro.utn.edu.ar/repositorio/catedras/quimica/5_anio/orientadora1/monograias/matich-redesneuronales.pdf)
- [13] Isasi Viñuela, Pedro Gálván, León Inés: "*Redes Neuronales Artificiales. Un enfoque Práctico*"
- [14] <http://ldc.usb.ve/~gabro/teaching/MachineL/ANN2.pdf>
- [15] [http://www.myreaders.info/02-Fundamentals\\_of\\_Neural\\_Network.pdf](http://www.myreaders.info/02-Fundamentals_of_Neural_Network.pdf)
- [16] Wikipedia: "*Perceptron multicapa*"
- [17] [http://ml.informatik.unireiburg.de/\\_media/teaching/ss10/05\\_mlps.printer.pdf7](http://ml.informatik.unireiburg.de/_media/teaching/ss10/05_mlps.printer.pdf7)

- 
- [18] A. Tagliarini: "*PhD An Introduction To The Backpropagation Algorithm*"
- [19] <http://www.cs.sfu.ca/~oschulte/king.pdf>(english)
- [20] <http://sydney.edu.au/engineering/it/courses/comp4302/ann4-6s.pdf>
- [21] *MathWorks*
- [22] Wikipedia: *Matlab*
- [23] Howard Demuth, Mark Beale, Martin Hagan: "*Neuronal Network Toolbox User guide*".
- [24] Wikipedia *GPU*
- [25] Emiliano Aldabas-Rubira: "*Introducción al reconocimiento de patrones mediante redes neuronales*"
- [26] [http://catarina.udlap.mx/u\\_dl\\_a/tales/documentos/lep/mejia\\_s\\_ja/capitulo3.pdf](http://catarina.udlap.mx/u_dl_a/tales/documentos/lep/mejia_s_ja/capitulo3.pdf)
- [27] R.Tapiador Morales, A.Rios Navarro, A.Jimenez Fernandez, J.Dominguez Morales, A.Linares Barranco: "*System based on inertial sensors for behavioral monitoring of wildlife*"
- [28] J. P. Dominguez Morales, A. Rios Navarro, M. Dominguez Morales, R. Tapiador Morales, D.Gutierrez-Galan, D.Cascado Caballero, A. Jimenez Fernandez y A. Linares-Barranco "*Wireless sensor network for wildlife tracking and behavior classification of animals in Doñana*"

## 10 LISTADO DE ILUSTRACIONES

### 10.1 Figuras

Figura 1:	Arquitectura de Minerva .....	9
Figura 2:	Estructura de la FFT .....	18
Figura 3:	Ejemplo desarrollado de FFT .....	18
Figura 4:	Diagrama neurona perceptron .....	23
Figura 5:	Modelo de neurona ADALINE .....	23
Figura 6:	Arquitectura de red ADALINE .....	24
Figura 7:	Neurona de red perceptron multicapa .....	25
Figura 8:	Arquitectura de perceptrón multicapa .....	25
Figura 9:	Gráfica función logística .....	26
Figura 10:	Gráfica función "todo o nada" .....	26
Figura 11:	Gráfica función lineal .....	27
Figura 12:	Estructura en bloques.....	27
Figura 13 :	Arquitectura de red recurrente .....	29
Figura 14:	Función Hardlim.....	31
Figura 15:	Función lineal.....	32
Figura 16:	Función Sigmoide .....	32
Figura 17:	Modelo de neurona en MATLAB.....	33
Figura 18:	Modelo desarrollado .....	33
Figura 19:	Arquitectura de red multicapa en MATLAB .....	34
Figura 20:	Implementación de una red Backpropagation en MATLAB. ....	34
Figura 21:	Algoritmo de Freemu .....	36
Figura 22:	Red BackPropagation.....	40
Figura 23:	Caballo con collar .....	45
Figura 24:	Interior del collar .....	46
Figura 25:	Arquitectura del collar.....	46
Figura 26:	Caballo en reposo .....	49
Figura 27:	Caballo andando.....	49
Figura 28	Caballo al trote.....	50
Figura 29:	Adquisición de datos del collar.....	51
Figura 30:	Estructura para manejar las redes neuronales .....	52
Figura 31:	Estructura de preprocesamiento sin FreeIMU.....	54



Figura 32: Estructura de preprocesamiento con FreeIMU .....	54
Figura 33: Gráfica para "andando", 80 muestras de validación .....	60
Figura 34: Gráfica para "parado", 80 muestras de validación .....	60
Figura 35: Gráfica para "trote", 80 muestras de validación .....	61
Figura 36: Gráfica para "andando", 100 muestras de validación .....	61
Figura 37: Gráfica para "parado", 100 muestras de validación .....	62
Figura 38: Gráfica para "trote", 100 muestras de validación .....	62
Figura 39: Mejores resultados de la red.....	63

## 10.2 Tablas

Tabla 1: Valores de $w_N^{k,n}$ .....	16
Tabla 2: Funciones del ToolBox de Matlab .....	42
Tabla 3: Número de muestras por cada archivos de muestras en raw .....	51
Tabla 4: Ejemplo de muestra preprocesada .....	53
Tabla 5: Resultados % de aciertos a partir de 80 muestras (sin FreeIMU) .....	56
Tabla 6: Resultados % aciertos a partir de 100 muestras (sin FreeIMU) ...	57
Tabla 7: % de acierto con FreeIMU y calculadas a partir de 100 muestras	58
Tabla 8: % de acierto con FreeIMU y calculadas a partir de 800 muestras	59

## 10.3 Ecuaciones

Ecuación 1. Ecuación de la cuerda .....	11
Ecuación 2. Ecuación de Euler para la cuerda .....	12
Ecuación 3: Ecuación de la DFT .....	14
Ecuación 4: Ecuación de la IDFT .....	14
Ecuación 5: Ecuación de la DFT modificada .....	16
Ecuación 6 Redundancia $W_n$ en forma trigonométrica .....	16
Ecuación 7 Error de Salidas .....	28
Ecuación 8: Ecuación de la regla delta .....	28