



UNIVERSIDAD LOYOLA ANDALUCÍA  
ESCUELA SUPERIOR DE INGENIERÍA  
GRADO EN INGENIERÍA INFORMÁTICA Y  
TECNOLOGÍAS VIRTUALES

TRABAJO FIN DE GRADO:

---

Desarrollo de Algoritmos Bioinspirados para el  
entrenamiento de Redes Neuronales Extreme  
Learning Machine

---

MANUAL TÉCNICO

**Autor:**

• David Pineda Peña

**Tutor:**

• Antonio Manuel Durán Rosal

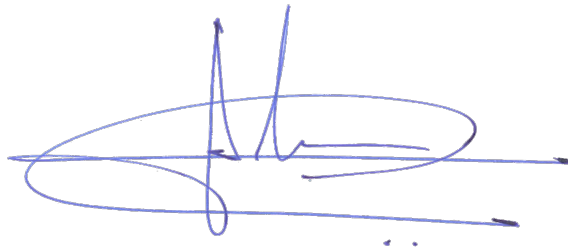
Febrero, 2023



# DESARROLLO DE ALGORITMOS BIOINSPIRADOS PARA EL ENTRENAMIENTO DE REDES NEURONALES EXTREME LEARNING MACHINE

Firmado en Sevilla, Febrero de 2023

Tutor/Director:



Fdo: Dr. Antonio Manuel Durán Rosal

Autor:



Fdo: David Pineda Peña



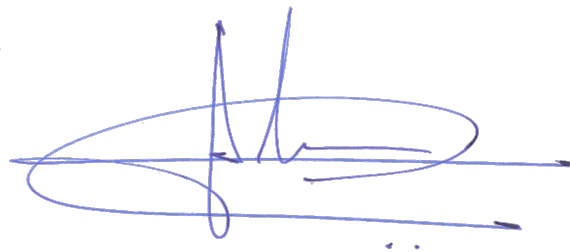
**Antonio Manuel Durán Rosal**, Profesor Adjunto del Dpto. de Métodos Cuantitativos y la Escuela Superior de Ingeniería de la Universidad Loyola Andalucía.

### **Informa:**

Que el presente Trabajo Fin de Grado titulado *Desarrollo de Algoritmos Bioinspirados para el entrenamiento de Redes Neuronales Extreme Learning Machine*, constituye la memoria presentada por **David Pineda Peña** para aspirar al título de Graduado en Ingeniería Informática y Tecnologías Virtuales, y ha sido realizado bajo mi dirección en la Escuela Superior de Ingeniería de la Universidad Loyola Andalucía reuniendo, a mi juicio, las condiciones necesarias exigidas en este tipo de trabajos.

Y para que conste, se expide y firma el presente certificado en Sevilla, Febrero 2023.

El Director:



Fdo: Prof. Dr. Antonio Manuel Durán Rosal



# AGRADECIMIENTOS

No podría abrir este apartado de agradecimientos sin hacer mención a mi madre Isabel María. Sin ella, no habría escogido el camino de realizar una carrera universitaria y, a día de hoy, no estaría poniéndole punto y final a este Trabajo Fin de Grado.

Gracias a todos los profesores, y especialmente a todos aquellos del Departamento de Método Cuantitativos por haber sido los auténticos impulsores de mi pasión por este mundo, *'include'* a mi tutor, Antonio Manuel Durán Rosal, quien con su sabiduría e infinita paciencia me ha guiado en el desarrollo de este TFG.

A mis compañeros de clase, los cuales más bien son ya mi segunda familia, por haber apoyado a aquel alumno de ciencias sociales que dejaba mucho que desear en áreas como las matemáticas y la física en el primer curso de la carrera. Estoy seguro de que sin vuestra ayuda éste camino habría sido mucho más cuesta arriba.

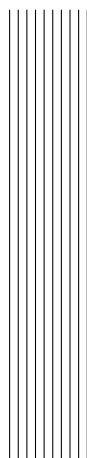
Mencionar también a todos mis hermanos, y al auténtico padre que ha estado ahí para mi educación, Mike.

Para finalizar, quiero dedicar unas líneas a la persona que me ha acompañado sobre todo este último año, Elena. Gracias por estar a mi lado cada semana, por estudiar a mi lado mientras programaba, y en definitiva, animarme en todo momento para que me pusiese las pilas.

Por todo eso y más, gracias.







# ÍNDICE GENERAL

Índice General	IX
Índice de Figuras	XV
Índice de Tablas	XVII
<b>I Introducción</b>	<b>1</b>
<b>1. Introducción</b>	<b>3</b>
<b>2. Definición del Problema</b>	<b>7</b>
2.1. Identificación del Problema Real . . . . .	7
2.2. Problema Técnico . . . . .	9
2.2.1. Funcionamiento . . . . .	9
2.2.2. Entorno . . . . .	10
2.2.3. Vida esperada . . . . .	11
2.2.4. Ciclo de Mantenimiento . . . . .	11
2.2.5. Competencia . . . . .	12
2.2.6. Aspecto Externo . . . . .	12
2.2.7. Estandarización . . . . .	13

2.2.8. Calidad y Fiabilidad . . . . .	14
2.2.9. Programa de Tareas . . . . .	14
2.2.10. Pruebas . . . . .	16
2.2.11. Seguridad . . . . .	17
<b>3. Objetivos</b>	<b>19</b>
3.1. Objetivos de Formación . . . . .	19
3.2. Objetivos Operacionales . . . . .	20
<b>4. Antecedentes</b>	<b>23</b>
4.1. Redes Neuronales Artificiales . . . . .	23
4.1.1. Funciones de Activación . . . . .	25
4.1.2. Aprendizaje supervisado en las RNAs . . . . .	28
4.2. Extreme Learning Machine . . . . .	33
4.3. Algoritmos Bioinspirados . . . . .	35
4.3.1. Algoritmo Genético . . . . .	35
4.3.2. Particle Swarm Optimization . . . . .	37
4.3.3. Coral Reef Optimization . . . . .	39
4.4. ELM con algoritmos evolutivos . . . . .	41
4.4.1. GA-ELM . . . . .	41
4.4.2. PSO-ELM . . . . .	43
<b>5. Restricciones</b>	<b>47</b>
5.1. Factores Dato . . . . .	48
5.2. Factores Estratégicos . . . . .	49
<b>6. Recursos</b>	<b>51</b>
6.1. Recursos Humanos . . . . .	51
6.2. Recursos Hardware . . . . .	52
6.3. Recursos Software . . . . .	52
6.3.1. Sistemas Operativos . . . . .	52
6.3.2. Lenguajes y Herramientas de Programación . . . . .	53
6.3.3. Herramientas de Documentación . . . . .	53

<b>II</b>	<b>Análisis y Especificación de Requisitos</b>	<b>55</b>
<b>7.</b>	<b>Descripción General del Problema</b>	<b>57</b>
7.1.	Algoritmo Propuesto . . . . .	57
7.1.1.	Representación de los Corales . . . . .	58
7.1.2.	Generación del Arrecife . . . . .	59
7.1.3.	Evaluación . . . . .	59
7.1.4.	Reproducción . . . . .	59
7.1.5.	Asentamiento de la larva . . . . .	61
7.1.6.	Depredación . . . . .	62
7.1.7.	Criterio de Parada . . . . .	63
7.1.8.	Pseudocódigo del Algoritmo . . . . .	63
<b>8.</b>	<b>Especificación de Requisitos</b>	<b>65</b>
8.1.	Introducción . . . . .	65
8.2.	Participantes en el TFG . . . . .	67
8.3.	Catálogo de Objetivos del Sistema . . . . .	68
8.4.	Catálogo de Requisitos del Sistema . . . . .	73
8.4.1.	Requisitos de la Información . . . . .	73
8.4.2.	Requisitos Funcionales . . . . .	75
8.4.3.	Requisitos no Funcionales . . . . .	77
8.5.	Matriz de Trazabilidad . . . . .	80
8.6.	Formato de Entrada/Salida de Datos . . . . .	81
8.6.1.	Formato de Entrada de Datos . . . . .	81
8.6.2.	Formato de Salida de Datos . . . . .	82
<b>9.</b>	<b>Herramientas Utilizadas</b>	<b>85</b>
9.1.	Lenguaje de Programación Python . . . . .	85
9.1.1.	Historia . . . . .	86
9.1.2.	Características Principales . . . . .	87
9.2.	Librería Panel . . . . .	89
9.2.1.	Historia . . . . .	89

9.2.2. Características Principales . . . . .	90
<b>III Diseño del Sistema</b>	<b>93</b>
<b>10.Arquitectura del Sistema</b>	<b>95</b>
10.1. Módulos . . . . .	95
10.1.1. Relación Algoritmo-Usuario . . . . .	96
10.1.2. Relación Interfaz Gráfica-Usuario . . . . .	98
<b>11.Diagramas de Clases</b>	<b>99</b>
11.1. Diagramas de Clases . . . . .	99
11.2. Diagrama de Clases del Sistema Completo . . . . .	100
11.3. Clase Individual . . . . .	102
11.4. Clase Particle . . . . .	105
11.5. Clase Larvae . . . . .	109
11.6. Clase Population . . . . .	112
11.7. Clase Swarm . . . . .	114
11.8. Clase Reef . . . . .	117
<b>12.Diseño de la Interfaz Gráfica</b>	<b>121</b>
12.1. Estructura de la Interfaz Gráfica . . . . .	123
12.1.1. Sección de Parámetros del Algoritmo . . . . .	124
12.1.2. Sección de Gráfico Interactivo . . . . .	127
12.1.3. Sección de Eventos . . . . .	127
12.2. Diagrama de Secuencia de la Interfaz . . . . .	127
<b>IV Pruebas</b>	<b>129</b>
<b>13.Pruebas</b>	<b>131</b>
13.1. Estrategia de Pruebas . . . . .	132
13.2. Pruebas Unitarias . . . . .	132
13.2.1. Pruebas de Caja Blanca . . . . .	133

## ÍNDICE GENERAL

---

13.2.2. Pruebas de Caja Negra . . . . .	134
13.3. Pruebas de Integración . . . . .	136
13.4. Pruebas del Sistema . . . . .	137
13.5. Pruebas de Aceptación . . . . .	138
<b>14.Resultados Experimentales</b>	<b>141</b>
14.1. Condiciones de los Experimentos . . . . .	142
14.1.1. Bases de datos . . . . .	142
14.1.2. Parámetros de los Experimentos . . . . .	144
14.2. Resultados . . . . .	145
14.3. Discusión . . . . .	148
<b>V Conclusiones</b>	<b>149</b>
<b>15.Conclusiones del Autor</b>	<b>151</b>
15.1. Objetivos de Formación Alcanzados . . . . .	151
15.2. Objetivos Operacionales Alcanzados . . . . .	152
<b>16.Futuras Mejoras</b>	<b>155</b>
16.1. Mejoras de la propuesta . . . . .	155
16.1.1. Librerías de Optimización . . . . .	155
16.1.2. Técnicas de Regularización . . . . .	155
16.1.3. Función de Activación . . . . .	156
16.1.4. Métricas de evaluación . . . . .	156
16.2. Mejoras de la interfaz gráfica . . . . .	157
16.2.1. Despliegue Servidor . . . . .	157
16.2.2. <i>Multithreading</i> . . . . .	157
<b>Bibliografía</b>	<b>159</b>
<b>VI Apéndices</b>	<b>163</b>
<b>A. Manual de Usuario</b>	<b>165</b>

A.1. Descripción del Producto . . . . .	165
A.2. Requisitos del Sistema . . . . .	166
A.2.1. Requisitos Mínimos . . . . .	167
A.2.2. Requisitos Recomendados . . . . .	167
A.2.3. Requisitos Software . . . . .	167
A.3. Instalación del Software . . . . .	168
A.3.1. Instalación Anaconda Distribution . . . . .	168
A.4. Desinstalación del Software . . . . .	169
A.4.1. Desinstalación Anaconda . . . . .	169
A.5. Ejecución del Software . . . . .	169
A.6. Módulo Interfaz Gráfica . . . . .	170



## ÍNDICE DE FIGURAS

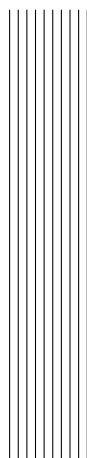
4.1. Modelo de perceptrón con cinco señales de entrada, FUENTE: <a href="#">Wikipedia [2023d]</a> . . . . .	25
4.2. Gráfica de la función sigmoide, FUENTE: <a href="#">Wikipedia [2023b]</a> . . . . .	26
4.3. Gráfica de la función ReLU, FUENTE: <a href="#">Wikipedia [2023a]</a> . . . . .	27
4.4. Gráfica de la función tangente hiperbólica, FUENTE: <a href="#">Wikipedia [2023c]</a> . . . . .	28
4.5. Diagrama de flujo GA, FUENTE: <a href="#">Durán Rosal [2019]</a> . . . . .	37
4.6. Diagrama de flujo PSO, FUENTE: <a href="#">Durán Rosal [2019]</a> . . . . .	39
4.7. Diagrama de flujo CRO, FUENTE: <a href="#">Durán Rosal [2019]</a> . . . . .	41
4.8. Representación de la solución, FUENTE: <a href="#">Kennedy and Eberhart [1995]</a> . . . . .	44
4.9. Proceso iterativo DPSO-PSO-FFNN para la optimización conjunta de la topología de red y los parámetros de FFNN, FUENTE: <a href="#">Kennedy and Eberhart [1995]</a> . . . . .	45
8.1. Esquema del OBJ-001 relacionado con la interfaz gráfica. . . . .	69
8.2. Formato de fichero .csv de Entrada de Datos. . . . .	82
8.3. Vista del fichero desde una hoja de cálculo. . . . .	84
9.1. Logo Python. . . . .	85
9.2. Logo Panel. . . . .	89

## ÍNDICE DE FIGURAS

---

10.1. Diagrama de interacción Algoritmo-Usuario. . . . .	97
10.2. Diagrama de interacción Interfaz Gráfica-Usuario. . . .	98
11.1. Diagrama de clases del sistema completo. . . . .	101
11.2. Diagrama de clases de la clase Individual. . . . .	104
11.3. Diagrama de clases de la clase Particle. . . . .	108
11.4. Diagrama de clases de la clase Larvae. . . . .	111
11.5. Diagrama de clases de la clase Population. . . . .	113
11.6. Diagrama de clases de la clase Swarm. . . . .	116
11.7. Diagrama de clases de la clase Reef. . . . .	119
12.1. Secciones de la interfaz gráfica. . . . .	122
12.2. Estructura de la interfaz de usuario. . . . .	123
12.3. Diagrama de secuencia de la interacción usuario-sistema- algoritmo. . . . .	128
A.1. Apariencia de la interfaz gráfica. . . . .	170





## ÍNDICE DE TABLAS

8.1. Participante David Pineda Peña. . . . .	67
8.2. Participante Antonio Manuel Durán Rosal. . . . .	67
8.3. Objetivo perteneciente a la implementación de la interfaz gráfica. . . . .	68
8.4. Objetivo perteneciente a la implementación de los algoritmos bioinspirados en el modelo ELM. . . . .	69
8.5. Objetivo perteneciente a las opciones de configuración. . . . .	70
8.6. Objetivo perteneciente a la entrada de datos. . . . .	70
8.7. Objetivo perteneciente al almacenamiento de resultados. . . . .	70
8.8. Objetivo perteneciente a la implementación del modelo ELM. . . . .	71
8.9. Objetivo perteneciente a la implementación del algoritmo genético. . . . .	71
8.10. Objetivo perteneciente a implementación del algoritmo PSO. . . . .	71
8.11. Objetivo perteneciente a implementación del algoritmo CRO. . . . .	72
8.12. Objetivo perteneciente a la representación gráfica de los resultados. . . . .	72
8.13. Requisito de Información perteneciente a la representación de los experimentos. . . . .	73

## ÍNDICE DE TABLAS

---

8.14. Requisito de Información perteneciente a la configuración de los experimentos. . . . .	74
8.15. Requisito funcional perteneciente a aplicar la configuración de entrada. . . . .	75
8.16. Requisito funcional perteneciente a la ejecutar el algoritmo. . . . .	75
8.17. Requisito funcional perteneciente a almacenar los resultados de los algoritmos. . . . .	76
8.18. Requisito funcional perteneciente a la representación gráfica de rendimiento del algoritmo. . . . .	76
8.19. Requisito no funcional relativo al lenguaje de programación Python. . . . .	77
8.20. Requisito no funcional perteneciente a la usabilidad. . .	78
8.21. Requisito no funcional perteneciente a los formatos de almacenamiento para los resultados. . . . .	78
8.22. Requisito no funcional perteneciente a la fiabilidad y optimización. . . . .	79
8.23. Requisito no funcional perteneciente a los formatos válidos para la entrada de texto. . . . .	79
8.24. Matriz de trazabilidad para relacionar objetivos del TFG.	80
10.1. Módulo implementado de la interfaz gráfica. . . . .	96
10.2. Módulo implementado de algoritmos bioinspirados. . .	96
11.1. Ejemplo general para la especificación de clases. . . . .	100
11.2. Especificación de la clase Individual. . . . .	103
11.3. Especificación de la clase Particle. . . . .	107
11.4. Especificación de la clase Larvae. . . . .	110
11.5. Especificación de la clase Population. . . . .	113
11.6. Especificación de la clase Swarm. . . . .	115
11.7. Especificación de la clase Reef. . . . .	118
14.1. Bases de datos empleadas en los experimentos. . . . .	143
14.2. Rendimiento promedio de los algoritmos. . . . .	146

## ÍNDICE DE TABLAS

---

14.3. Tiempos de ejecución promedio (en segundos) de los algoritmos. . . . .	147
---	-----

# Parte I

## Introducción





# 1 Introducción

A lo largo de este primer capítulo se expone de forma introductoria y general el problema que pretende abordar el presente Trabajo Fin de Grado (TFG), así como los objetivos planteados para su correcta resolución.

Para fundamentar la importancia y el avance del algoritmo desarrollado en este TFG, se realizará un análisis de las capacidades de varios algoritmos bioinspirados y sus múltiples aplicaciones en el campo de la Inteligencia Artificial (IA).

La capacidad de recopilar y retener grandes volúmenes de datos ha aumentado enormemente en las últimas décadas a medida que la tecnología ha ido evolucionando. La IA es un campo de la informática que se centra en la creación y desarrollo de máquinas capaces de realizar operaciones que, a priori, requieren de cierta inteligencia humana como aprender, pensar y tomar decisiones. Como resultado, la IA se ha convertido en un campo de estudio en auge.

Dentro de esta disciplina surge el Aprendizaje Automático o *Machine Learning* (ML) y las Redes Neuronales Artificiales (RNAs). Las RNAs son una forma de modelo matemático inspiradas en el funcionamiento del cerebro humano. Dichas redes están constituidas por un conjunto de neuronas interconectadas, capaces de transmitir informa-

---

ción que atraviesa distintas capas de la red neuronal, como la capa de entrada, las capas ocultas y la capa de salida, sometiéndose a diferentes operaciones durante el proceso. Al final se obtiene un valor de salida que puede tener distintos significados dependiendo del problema que se esté abordando. Una vez obtenido este valor de salida, el objetivo del modelo frente al problema planteado es minimizar una función de pérdida. Bajo esta premisa de optimización, los valores de las conexiones de las neuronas se van ajustando de manera iterativa, haciendo que el modelo sea capaz de dar un valor de salida cada vez más fiable y cercano a la realidad.

Dado este enfoque y su enorme complejidad matemática, surgen infinitud de modelos neuronales y métodos que buscan maximizar el rendimiento de las RNAs con el menor coste computacional posible.

Un caso especial son las redes *Extreme Learning Machine* (ELM), que fueron propuestas por el profesor **Guang-Bin Huang** en el artículo [Huang et al. \[2006\]](#), como medio para superar algunos de los retos y limitaciones de las RNAs tradicionales. A diferencia de éstas, que requieren un proceso de entrenamiento complejo y largo, las redes ELM pueden entrenarse fácilmente y de manera rápida a la vez que consiguen resultados altamente precisos.

Para el ajuste de los parámetros de las RNAs se han desarrollado técnicas de optimización numérica así como metaheurísticas. Dentro de estas últimas, destacan los algoritmos bioinspirados. La idea de imitar comportamientos que suceden en la naturaleza para intentar resolver problemas complejos se remonta a la antigüedad y ha sido un tema central en muchos campos, como la ingeniería, la informática y la biología.

En los últimos años, el desarrollo de herramientas cada vez más potentes, capaces de ejecutar operaciones y técnicas computacionales más complejas ha permitido a los investigadores estudiar y modelizar con mayor eficacia los sistemas naturales, resultando en un gran interés por los algoritmos metaheurísticos con base evolutiva, sobre todo, los basados en poblaciones de individuos y enjambres.

## 1. Introducción

---

Ambos paradigmas juntos, las redes ELM y los algoritmos bioinspirados tienen el potencial de revolucionar la forma en que se resuelven problemas complejos en una amplia gama de dominios dentro del ML. Combinando la sencillez y eficiencia de las redes ELM con la adaptabilidad y robustez de los algoritmos bioinspirados, se pueden desarrollar sistemas de IA capaces de abordar algunos de los problemas más complejos a los que se enfrenta la sociedad actual.







## 2 Definición del Problema

En el siguiente capítulo se procede a definir el ámbito del Trabajo Fin de Grado, identificando tanto las necesidades del mismo como estableciendo los objetivos a cumplir.

- **Identificación del problema real:** El punto de vista del usuario final es quien define el problema.
- **Identificación del problema técnico:** El punto de vista del desarrollador es quien define el problema.

### 2.1. Identificación del Problema Real

Se plantea, para este TFG, implementar diversos algoritmos bioinspirados aplicados al entrenamientos de redes Extreme Learning Machine, aumentando las capacidades predictivas del modelo sobre unos datos de entrada, independientemente de su naturaleza.

Los algoritmos bioinspirados que constituyen este trabajo son:

- **Algoritmo GA (*Genetic Algorithm*)**, siguiendo las bases de la Teoría de la Evolución de **Darwin**, este algoritmo parte de una población de individuos, donde cada individuo representa

## 2.1. Identificación del Problema Real

---

una posible solución del espacio de búsqueda. Con el paso de distintas generaciones, este tipo de algoritmos convergen hacia una solución optimizada. Se explicará en profundidad en el Capítulo 4 de este TFG.

- **Algoritmo PSO (*Particle Swarm Optimization*)**, basado en enjambre de partículas. Atribuido a los investigadores **Kennedy, Eberhart y Shi**, [Kennedy and Eberhart \[1995\]](#). Este tipo de algoritmo sigue un método heurístico, bajo la idea del comportamiento de las partículas en la naturaleza, las cuales se desplazan por el espacio de búsqueda y van actualizando su posición en base a unos criterios matemáticos.
- **Algoritmo CRO (*Coral Reef Optimization*)**, basado en los procesos naturales que ocurren en los arrecifes de corales. Propuesto por el Catedrático **Sancho Salcedo-Sanz** ([Salcedo-Sanz et al. \[2014a\]](#)), este algoritmo trata de imitar de manera computacional y artificial, las fases por las que un arrecife de coral se desarrolla, dando lugar a nuevas larvas de coral que compiten por el espacio en la colonia.

El problema real por tanto podría dividirse en tres grandes apartados:

- Implementación de los algoritmos bioinspirados.
- Desarrollo de un método que automatice el proceso de prueba de los tres algoritmos mencionados y al mismo tiempo haga un volcado de los resultados extraídos de cada uno, para así poder realizar una comparativa seria y concluyente en un futuro apartado.
- Desarrollo de un medio cómodo y de fácil uso que permita la interacción del usuario final con los algoritmos implementados durante el TFG, pudiendo realizar experimentos y configurar los distintos parámetros de las pruebas.

## 2. Definición del Problema

---

### 2.2. Problema Técnico

Tras definir el problema real al cual se pretende dar solución con este TFG, se procede a estudiar el punto de vista del desarrollador con un enfoque más técnico. Para ello, se hará siguiendo la metodología PDS (*Product Design Specification*).

#### 2.2.1. Funcionamiento

La herramienta software desarrollada constará de varios algoritmos bioinspirados aplicados al entrenamiento del modelo neuronal de clasificación-regresión ELM en la parte no lineal del sistema. A partir de unos datos de entrada proporcionados por una base de datos, se llevará a cabo dicho entrenamiento ajustando los sesgos y pesos de las conexiones de las redes neuronales y realizando una selección de características. Los resultados del sistema permitirán al usuario comparar la eficacia del modelo junto con el entrenamiento de cada uno de los algoritmos, usando métricas de evaluación como el **CCR** (*Correct Classification Rate*), y teniendo en cuenta los tiempos de ejecución.

A su vez, se dará la posibilidad al usuario de elegir los datos de entrada del experimento, así como modificar los parámetros asociados a los algoritmos, por medio de una interfaz gráfica fácil de manejar y desplegada en un servidor *localhost* local.

En concreto, los pasos más generales del funcionamiento de la herramienta son:

- **Introducción de Datos:** El usuario deberá escoger entre un repositorio de bases de datos codificadas en ficheros *.csv*, los datos de entrada del experimento a los que posteriormente se les realizará un pre-procesamiento. Finalmente, se hará una ejecución del modelo neuronal ELM con la variante de entrenamiento del algoritmo bioinspirado que haya seleccionado el usuario.

- ***Establecer la Configuración:*** El usuario podrá realizar diversos cambios en la configuración de los datos de la aplicación, mediante la interfaz gráfica que se proporciona. Se tendrá acceso a elegir el número de iteraciones e introducir el valor de varios hiperparámetros influyentes en el proceso de aprendizaje de los algoritmos.
- ***Ejecución:*** Con los datos introducidos por el usuario, se aplicará el algoritmo con la base de datos y configuración seleccionada.
- ***Mostrar Resultados:*** La aplicación irá realizando una visualización de datos al usuario al mismo tiempo que se está ejecutando. Adicionalmente, una vez haya acabado se seguirá mostrando los resultados del experimento.
- ***Almacenamiento Resultados:*** La herramienta *software* permitirá al usuario poder exportar los datos de la ejecución a formato *.xlsx* (hoja de cálculo).

### 2.2.2. Entorno

El entorno de programación será el impuesto por el lenguaje de alto nivel, orientado a objetos e interpretado ***Python***. Esta decisión cuenta con la ventaja de permitir ejecutar la aplicación en entornos *cloud*, tales como **Google Colaboratory**, un producto de *Google Research* que ya cuenta de serie con las librerías principales del lenguaje. Entre ellas la librería **Panel**, con la que posteriormente se implementará la interfaz gráfica para el usuario.

En el presente TFG se utilizará la distribución de **Windows Pro 10**, aunque al tratarse de un entorno de desarrollo del navegador, es posible ejecutarlo en cualquier otro Sistema Operativo.

Los usuarios a los que va dirigida esta aplicación forman un grupo más o menos reducido, puesto que se necesitan tener unos conocimientos básicos sobre la temática de la aplicación (algoritmos bioinspirados,

## 2. Definición del Problema

---

redes neuronales, hiperparámetros, ...) para poder utilizar el programa correctamente, haciendo una correcta introducción de los datos y comprendiendo los resultados de la propia ejecución. También se requiere de unos mínimos conocimientos informáticos para instalarla y ejecutar los archivos necesarios.

### 2.2.3. Vida esperada

Es difícil deducir el tiempo de vida esperada del TFG ya que al estar el englobado en el área de investigación se prevé que haya un continuo desarrollo del área. Cada año aparecen gran cantidad de algoritmos nuevos o mejora de algoritmos antiguos (y no tan antiguos), lo cual hace muy previsible que aparezcan algoritmos que mejoren significativamente los resultados obtenidos en este TFG. Bajo estas premisas y marco tecnológico en el que se encuadra, se puede estimar que la vida esperada sea corta.

Cabe remarcar que este algoritmo siempre tendrá opción a ser mejorado mediante la aplicación de nuevas técnicas estadísticas o aplicando otros métodos numéricos que reduzcan la dificultad de cálculo y no supongan una caída de la eficiencia del mismo.

Es por todo esto que el TFG se debe tratar como una base para un posible futuro desarrollo de una aplicación o aplicaciones más completas, más que como un trabajo terminado totalmente.

### 2.2.4. Ciclo de Mantenimiento

Enlazando con el punto anterior, ya que la vida esperada se prevé que sea limitada, es probable que sea necesario llevar a cabo en un futuro cercano una serie de posibles mejoras y/o ampliaciones tales como:

- Cambios en el algoritmo de selección de individuos.

- Mejora del algoritmo con nuevos operadores de mutación y cruce en las fases de reproducción de los algoritmos **GA** y **CRO**.
- Aplicación de nuevas técnicas desarrolladas en un futuro.

### 2.2.5. Competencia

Existen algunos trabajos que se detallarán en profundidad en el Capítulo 4 que proponen el uso de algoritmos evolutivos aplicados a las redes ELM y pueden ser considerados competencia al algoritmo propuesto en este TFG.

Actualmente, se están realizando investigaciones sobre el entrenamiento de redes neuronales en otros modelos, los cuales a pesar de que incluyen un mayor número de capas ocultas y por tanto no se iguala con este TFG, tienen base evolutiva y siguen la misma premisa de optimización de pesos y selección de características.

En referencia a la interfaz gráfica no se ha observado que exista ningún otro software que aplique algoritmos bioinspirados junto con el modelo ELM, por lo que no se contempla competencia en este aspecto.

### 2.2.6. Aspecto Externo

La aplicación se pretende que sea presentada al usuario tanto en formato físico como es una memoria USB, como de forma *online* alojado en un repositorio de **GitHub** en el perfil del alumno.

Se recomienda la última opción dado que, con perspectiva de futuro y sujeto a posibles mejoras, se tiene una automatización por completo de las versiones y actualizaciones que puedan realizarse, además de tener acceso desde cualquier equipo sin necesidad de portar nada físico.

Junto con la herramienta software también se le hará poseedor al usuario del presente manual técnico en formato **.PDF**, incluyendo el manual de usuario del mismo.

## 2. Definición del Problema

---

### 2.2.7. Estandarización

Durante la implementación del TFG se plantea seguir los estándares de programación del propio lenguaje Python, declarados en el **PEP** (***Python Enhancement Proposal***), más concretamente el PEP 8, que describe la guía de estilo para código Python.

Se buscará hacer siempre una correcta declaración tanto de clases de objetos, como de métodos y de atributos.

Por lo que al código respecta, se seguirán las convenciones recomendadas de nombramiento tanto general como específico. Algunas de ellas son:

- Nombres autoexplicativos, y cuanto más lo sean mejor.
- Todas las clases deben ser declaradas con una mayúscula al inicio siguiendo la convención *CapWords*.
- Los nombres de las variables y funciones deben estar en minúscula. En caso de tener que declarar una variable o función con varias palabras se deberá dividir las palabras mediante un guión bajo "\_", y todas las palabras deberán estar escritas en minúscula.
- Cualquier constante que aparezca en el código debe ir escrita en su totalidad de letra mayúscula, y en caso de ser una palabra compuesta, se usará guión bajo para separar palabras.
- Las anotaciones para: variables a nivel de módulo, variables de clase e instancia y variables locales deben tener un solo espacio después de los dos puntos. No debe haber ningún espacio antes de los dos puntos.

Junto con las convenciones en cuanto a la escritura de nombres, existen otras referentes a la estructuración de los ficheros, de las clases, etc. que también se tendrán en cuenta.



También se seguirán los estándares de documentación para cualquier código escrito en lenguaje Python, como es el ejemplo de las *Documentation Strings*.

Para la realización de la interfaz se intentará seguir una distribución de los elementos así como una apariencia de éstos lo más similar posible a aplicaciones profesionales, y primando la sencillez sobre todo para facilitar el uso de esta.

### 2.2.8. Calidad y Fiabilidad

En primera instancia, la calidad del TFG deberá ser evaluada y asegurada directamente por el alumno y el director del mismo. Una vez finalizado, será el Tribunal quien evalúe el resultado final. En todo momento se intentará cumplir con los estándares de Ingeniería de Software para favorecer la mayoría de los factores que influyen en el desarrollo y calidad de una aplicación.

Con respecto a la fiabilidad, se deberá diseñar de tal forma que alerte en todo momento al usuario en caso de que se produzca algún fallo, ya sea por motivo de introducir erróneamente los parámetros, o por fallo de lectura y/o inexistencia de algún archivo. También, durante la fase de almacenamiento de datos, se deberá informar en ambos casos, éxito y fracaso, del estado de la aplicación.

### 2.2.9. Programa de Tareas

Desde un punto de visto global, se pueden definir las siguientes etapas del desarrollo del TFG:

1. **Fase de preparación:** Durante la primera fase de desarrollo se estudiará el problema desde un punto de vista teórico, así como los algoritmos evolutivos y modelos de redes neuronales de clasificación y regresión existentes hasta el momento, para poder hacer una comparación de la precisión y complejidad algorítmica.

## 2. Definición del Problema

---

ca. Además, también se estudiará el lenguaje de programación Python, junto con sus distintas librerías.

2. **Fase de diseño:** Tras la fase de preparación y teniendo claros los requisitos y objetivos del TFG, se realizará un diseño previo a la implementación. Se supervisará la funcionalidad del sistema, el diseño de los datos, el flujo de control que éstos seguirán, etc.
3. **Fase de implementación:** Con el diseño terminado y teniendo claros los pasos a seguir, se comenzará a implementar los algoritmos de manera óptima y aplicando la mejor práctica posible sobre el lenguaje Python. A su vez, se implementará la interfaz gráfica por medio de la librería Panel facilitando al usuario la interacción con el algoritmo.
4. **Fase de pruebas:** Terminada la implementación, se procederá a determinar la capacidad y el rendimiento de los algoritmos. Esta fase es crucial en el desarrollo de una aplicación ya que marca la diferencia entre un software depurado y operativo, de otro cuya calidad sea pobre, o su uso sea incluso inviable. Además, adquiere una mayor importancia esta fase al tratarse de un TFG donde los algoritmos expuestos, suponen una novedad en el campo y hay que validar su correcta efectividad, ya que puede dar pie a futuros paradigmas a seguir.
5. **Fase de documentación:** De forma transversal a todas las etapas de desarrollo del TFG, se irá documentando tanto la memoria final, como el manual de usuario.
6. **Fase de aplicación:** La fase final del TFG incluye la ejecución sobre un conjunto de bases de datos que prueben de la correcta metodología y verifiquen los resultados esperados en pasos anteriores.

### 2.2.10. Pruebas

En esta sección de pruebas se establecen las estrategias y los criterios que nos permitan certificar la calidad del software y de los resultados obtenidos durante las mismas. La estrategia de pruebas que se van a llevar a cabo son las siguientes:

- Pruebas unitarias de *caja negra* y *caja blanca* a nivel de módulos.
- Pruebas de integración para comprobar el ensamblado de los módulos.
- Pruebas de validación para asegurar el correcto funcionamiento del software.
- Pruebas de sistema para garantizar el cumplimiento de los objetivos.
- Pruebas de aceptación que verifiquen el cumplimiento de especificaciones y requisitos impuestos por el usuario.
- Además, se probará la validez de los algoritmos bioinspirados y el modelo ELM sobre diferentes bases de datos para así poder determinar su desempeño y funcionamiento.

Tras finalizar esta fase y cuando se verifique la corrección de todos los posibles errores que hayan surgido (o puedan surgir) durante la ejecución de las pruebas y el cumplimiento de todos los requisitos para los que la aplicación fue diseñada, se considerará que la aplicación es válida.

El algoritmo propuesto pasará por una serie de pruebas que acreditarán su correcto funcionamiento. Se comprobará si los resultados finales cumplen los objetivos planteados acerca de: la realización de estimaciones predictivas correctas, y además, el proceso de entrenamiento de las redes ELM usando algoritmos bioinspirados suponen una mejoría en la salida del modelo ELM.

## **2. Definición del Problema**

---

### **2.2.11. Seguridad**

Dado que el entorno de ejecución se encuentra en un ámbito privado y personal, la seguridad se sugestiona totalmente al usuario en cuestión o al administrador del servidor.

Y debido a que el tipo de datos manejados no requieren de ocultación o privacidad de ningún carácter, no se requiere de ninguna actividad de privacidad durante el desarrollo del TFG.





## 3 Objetivos

El siguiente capítulo expone los objetivos finales de este Trabajo Fin de Grado. La idea es implementar varios algoritmos bioinspirados que lleven a cabo una fase de entrenamiento de las redes neuronales de la capa de entrada del modelo Extreme Learning Machine, y estudiar si existe una mejoría en la precisión de la red, a cambio de un coste computacional asumible.

En concreto, los objetivos a conseguir son los siguientes:

### 3.1. Objetivos de Formación

Como objetivos de formación se priorizan los siguientes:

- Estudio en profundidad del lenguaje de programación Python y el paradigma orientado a objetos.
- Adquirir una base sólida de las diferentes librerías para el manejo de datos y funciones matemáticas de alto nivel implementadas en el lenguaje Python.
- Aprender a desarrollar una medio gráfico interactivo para la visualización de datos en Python.

### 3.2. Objetivos Operacionales

---

- Expandir los conocimientos del alumno en materia de las meta-heurísticas estudiadas (*state of the art*).
- Estudio en profundidad del modelo neuronal ELM, así como el resto de modelos existentes para la resolución de problemas de clasificación y regresión.
- Evaluar la eficacia del algoritmo mediante un conjunto de series de prueba para refinarlo hasta obtener unos resultados consistentes, según los estudios previos existentes en la bibliografía.
- Profundizar en el conocimiento de todas las fases de desarrollo de un TFG, desde la planificación y la investigación hasta el diseño y la implementación, pasando por la elaboración de guías de usuario, manuales técnicos, etc.
- Estudiar la herramienta  $\text{\LaTeX}$  para la realización de trabajos y artículos de alta calidad tipográfica.

### 3.2. Objetivos Operacionales

Como objetivos operacionales se identifican los siguientes:

- Proporcionar un algoritmo de aprendizaje automático construido sobre el modelo neuronal ELM, con la variante de un algoritmo evolutivo basado en el comportamiento de los arrecifes de corales (CRO), resultando en la variante CRO-ELM. Este objetivo conlleva dichas mejoras gracias a los siguientes subobjetivos:
  1. Iterar durante un número óptimo de generaciones los valores de los sesgos y pesos de las neuronas de capa oculta.
  2. Hacer una selección de individuos mediante una *roulette wheel* (método de ruleta). Dicho método escoge a los miembros que estarán presentes en la siguiente generación, a

### 3. Objetivos

---

partir de la población actual y de una manera aleatoria, a la vez que se premia a aquellos individuos con mejor rendimiento. En definitiva, se está cumpliendo con el objetivo de exploración del espacio de búsqueda, a la vez que se hace una explotación dentro del abanico de soluciones.

3. Añadir operadores de cruce y mutaciones para incorporar nuevas variedades en la población, a la vez que existe una probabilidad de depredación para dejar espacios libres a nuevos individuos.
- Implementar y adaptar otros algoritmos del estado del arte como el GA-ELM y el PSO-ELM para la posterior comparación de rendimientos con el algoritmo propuesto en este TFG.
  - Proveer al usuario de la aplicación de una interfaz gráfica que le permita modificar los datos de entrada e interactuar con los algoritmos durante su ejecución.
  - Ofrecer la posibilidad de guardar en ficheros *.xlsx* los resultados obtenidos por la aplicación. Los datos que se van a almacenar en dicho fichero son:
    - Base de datos seleccionada.
    - Configuración de los parámetros generales de la aplicación e hiperparámetros propios del algoritmo seleccionado.
    - Métrica de rendimiento del modelo, más conocido como *Correct Classification Rate*.
    - Tiempos de ejecución.







## 4 Antecedentes

A lo largo de este capítulo se hará un repaso por todos aquellos estudios y conocimientos previos que asientan las bases de nuestro Trabajo Fin de Grado. Este es un apartado de vital importancia pues será donde se profundizará en las principales ideas de este TFG, en concreto, las especificaciones y requisitos que deberá tener el sistema dado todo el conocimiento previo que se ha recogido y analizado.

Debido a que este TFG se basa en el desarrollo algoritmos bioinspirados aplicados en el modelo de *Extreme Learning Machine*, se comenzará haciendo una explicación del concepto de Redes Neuronales Artificiales. Seguidamente, se explicarán los principios del modelo ELM y tres algoritmos bioinspirados del estado del arte, para así finalizar con los trabajos de investigación realizados hasta la fecha, que combinan ambas ideas.

### 4.1. Redes Neuronales Artificiales

El concepto de RNAs se remonta a la década de 1940, cuando **Warren McCulloch** y **Walter Pitts** [McCulloch \[1943\]](#) propusieron la idea de utilizar unidades computacionales simples, llamadas neuronas,

#### 4.1. Redes Neuronales Artificiales

---

para modelar la forma en que el cerebro procesa la información. Sin embargo, no fue hasta las décadas de 1980 y 1990 cuando las RNAs empezaron a ganar atención y popularidad, gracias a los avances en el hardware informático y al desarrollo de algoritmos de aprendizaje eficaces.

Desde entonces, y gracias a éste crecimiento en la capacidad computacional de los ordenadores, las RNAs han sido de vital importancia en un campo de investigación como es el de la Inteligencia Artificial, más concretamente el aprendizaje automático o *Machine Learning*, e incluso otro campo más complejo como el del *Deep Learning* (DL) o aprendizaje profundo si se trata de varias capas de complejidad dentro de un mismo modelo de red neuronal.

Cada neurona en una RNA se basa en una unidad computacional que se activa o se desactiva en función de los valores de entrada recibidos. Estas neuronas se conectan entre sí mediante enlaces que reciben el nombre de sinapsis, que transmiten señales entre las neuronas, y cada enlace tiene un peso asociado que indica la importancia de la conexión.

En una RNA, la información se propaga a través de estas conexiones a lo largo de varias capas donde cada una consta de un conjunto de neuronas. Cada capa se encarga de realizar una operación matemática específica en la información de entrada antes de transmitirla a la siguiente. La última capa en la red se conoce como la capa de salida, y proporciona una respuesta o una predicción basada en la información de entrada.

En la Figura 4.1 se puede ver de forma general cómo se estructura un modelo básico de RNA.

## 4. Antecedentes

---

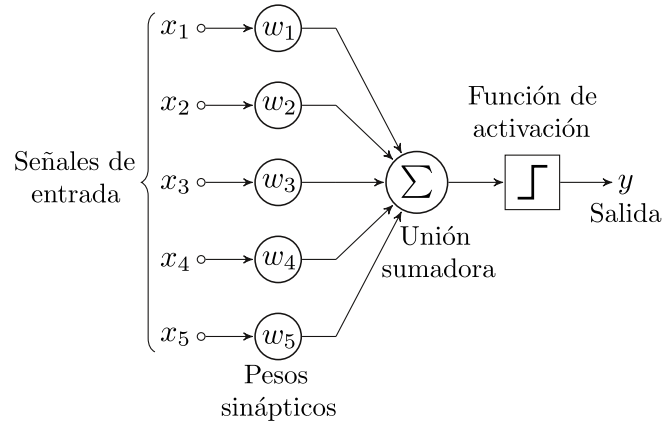


Figura 4.1: Modelo de perceptrón con cinco señales de entrada, FUENTE: [Wikipedia \[2023d\]](#).

La activación de las neuronas viene determinada por una función de activación. En el caso de la Figura 4.1 se emplea una función de tipo escalón.

### 4.1.1. Funciones de Activación

En los modelos de redes neuronales existen funciones de activaciones lineales y no lineales. Especialmente se hace uso de las no lineales ya que facilita que el modelo se adapte a una variedad de datos, y diferencie entre los resultados cuyos patrones y relaciones entre las entradas y las salidas son más complejas. Además en modelos de perceptrón multicapa (*MultiLayer Perceptron*, *MLP*) se busca que la función de activación sea derivable ya que es una condición necesaria si se quiere emplear el algoritmo de optimización de descenso del gradiente.

Las funciones se dividen principalmente en función de sus rangos o curvas.

#### Función Sigmoide

Esta función muestra una progresión temporal, la cual describen muchos sistemas en la naturaleza y toma una entrada en el rango

#### 4.1. Redes Neuronales Artificiales

$(-\infty, +\infty)$ , con asíntotas horizontales en dichos extremos y la mapea a un valor en el rango  $(0, 1)$ . La función sigmoide describe una gráfica representada en la Figura 4.2, y sigue la forma general definida en GALIPIENSO et al. [2003]:

$$f(x, p) = \frac{1}{1 + e^{-xp}} \quad (4.1)$$

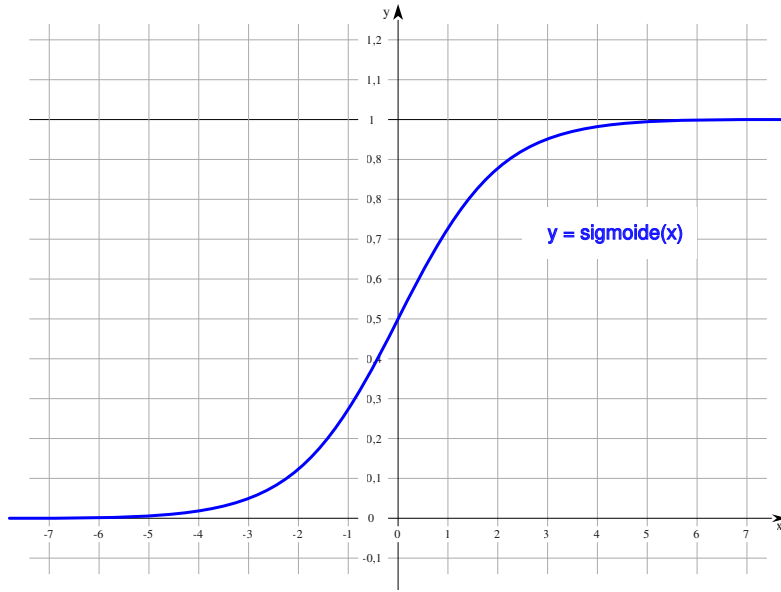


Figura 4.2: Gráfica de la función sigmoide, FUENTE: Wikipedia [2023b].

#### Rectified Linear Unit (ReLU)

Esta función toma una entrada en el rango  $(-\infty, +\infty)$  y la mapea a un valor en el rango  $(0, +\infty)$ . El valor de la función de activación es igual a su entrada mientras ésta sea mayor que cero. Al final, su trabajo se limita a deshacerse de los valores negativos. La función ReLU describe una gráfica representada en la Figura 4.2, y sigue la forma general definida en GALIPIENSO et al. [2003]:

$$f(x) = \max(0, x) \quad (4.2)$$

#### 4. Antecedentes

---

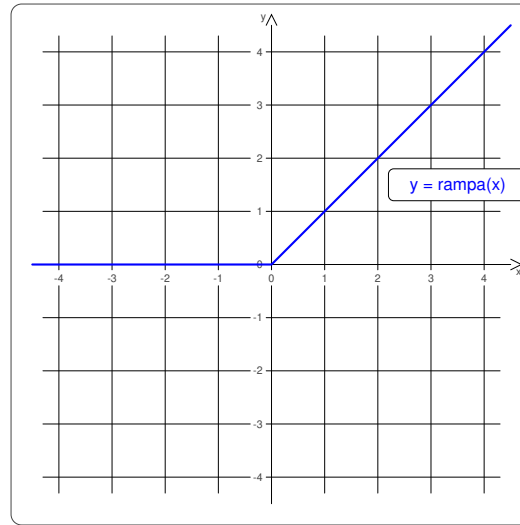


Figura 4.3: Gráfica de la función ReLU, FUENTE: [Wikipedia \[2023a\]](#).

#### Función tangente hiperbólica

Esta función es similar a la sigmoide pero acepta mapea los valores de entrada en el rango  $(-1, +1)$ , admitiendo valores negativos en la salida. La función tangente hiperbólica describe una gráfica representada en la Figura 4.4, y sigue la forma general definida en [GALIPIENSO et al. \[2003\]](#):

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \quad (4.3)$$

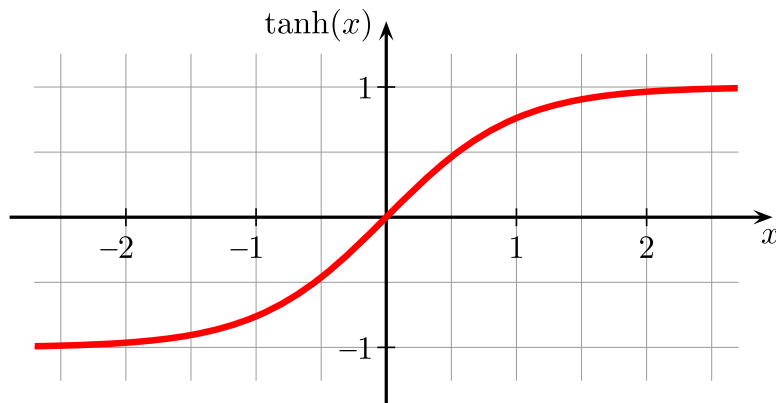


Figura 4.4: Gráfica de la función tangente hiperbólica, FUENTE: [Wikipedia \[2023c\]](#).

### 4.1.2. Aprendizaje supervisado en las RNAs

El aprendizaje en una RNA se realiza mediante la optimización de los pesos de las conexiones entre las neuronas, de esta manera se ajusta la respuesta de la red para un conjunto de datos de entrenamiento, y más tarde se valida su capacidad predictiva con un conjunto de prueba (*test*) aplicando una función de coste.

#### Funciones de coste en problemas de regresión

En problemas de regresión se intenta predecir las etiquetas de variable continua del conjunto  $Y = [y_1, y_2, \dots, y_n] \in \mathbb{R}^n$  ajustando la función  $f(x)$  que transforma los datos de entrada  $X = [x_1, x_2, \dots, x_n]$ .

Para poder cuantificar el error producido por la función  $f(x)$  existen varias funciones de coste:

- **Error cuadrático medio (MSE).** Es una función de coste comúnmente utilizada en problemas de regresión y es fácil de optimizar. Sin embargo, es sensible a valores atípicos (outliers) ya que penaliza fuertemente los errores grandes.

#### 4. Antecedentes

---

$$MSE = \frac{\sum_{i=1}^n (f(x_i) - y_i)^2}{n}. \quad (4.4)$$

- **Raíz del error cuadrático medio (RMSE).** Es una versión normalizada del MSE que toma en cuenta la escala de las etiquetas. Sin embargo, sigue siendo sensible a valores atípicos.

$$RMSE = \sqrt{\frac{\sum_{i=1}^n (f(x_i) - y_i)^2}{n}}. \quad (4.5)$$

- **Error absoluto medio (MAE).** Es una función de coste robusta que no penaliza tanto los errores grandes como el MSE y el RMSE. Sin embargo, puede resultar en una optimización más lenta debido a la naturaleza discontinua de la función.

$$MAE = \frac{\sum_{i=1}^n |y_i - f(x_i)|}{n}. \quad (4.6)$$

- **Huber loss.** Es una función de coste que combina la robustez del MAE y la sensibilidad a los errores pequeños del MSE. El parámetro  $\delta$  controla el nivel de sensibilidad a los errores grandes.

$$L_\delta = \begin{cases} \frac{1}{2}(y - f(x_i))^2 & \text{if } |(y - f(x_i))| < \delta, \\ \delta((y - f(x_i)) - \frac{1}{2}\delta) & \text{e.o.c.} \end{cases} \quad (4.7)$$

#### Funciones de coste en problemas de clasificación

En problemas de clasificación se busca predecir el valor categórico de las instancias. Para ello, se utiliza la teoría de la probabilidad y se mide la diferencia entre la distribución predicha y la distribución verdadera a través de diversas funciones de coste.

- **Entropía cruzada (CE).** Formulada en [Cybenko et al. \[2012\]](#), es una función de coste comúnmente utilizada en problemas de



## 4.1. Redes Neuronales Artificiales

---

clasificación y se utiliza para medir la discrepancia entre la distribución real  $P$  de etiquetas y la distribución predicha por la función  $y$ .

$$L_{\log}(y, p) = -(y \log(p) + (1 - y) \log(1 - p)). \quad (4.8)$$

- **Divergencia de Kullback-Leibler (KL)** [Kullback and Leibler \[1951\]](#). Es una medida de la diferencia entre una distribución de probabilidad  $P$  de otra  $Q$ . Se utiliza para comparar la distribución predicha y la distribución verdadera.

$$D_{KL}(P||Q) = \sum_{i=1}^n P(i) \log \frac{P(i)}{Q(i)}. \quad (4.9)$$

- **Negative Loglikelihood (NLL)**. Es una función de coste utilizada en clasificación que mide la probabilidad de la clase verdadera dada la entrada.

$$NLL(i) = -\log(P(y_i|x_i)), \quad (4.10)$$

donde  $P(y_i|x_i)$  es la probabilidad predicha de la clase correcta  $y_i$  dada la entrada  $x_i$  y  $\log$  es el logaritmo natural.

La pérdida total NLL para un conjunto de entrada se puede obtener tomando el promedio de la pérdida NLL para cada entrada:

$$NLL = -\frac{1}{N} * \sum_{i=1}^N [\log(P(y_i|x_i))]. \quad (4.11)$$

### Backpropagation

Una vez aplicada la función de coste, con los datos de entradas y la propia RNA, se realiza el ajuste de los pesos, con el objetivo de que el modelo sea capaz de generar respuestas más precisas y útiles.

## 4. Antecedentes

---

Existen muchas técnicas de aprendizaje pero la más utilizada y conocida es el algoritmo de *backpropagation* (propagación hacia atrás). Dicho término, backpropagation, fue propuesto por Rumelhart et al. [1986] y más tarde popularizado en McClelland et al. [1987].

El algoritmo parte de calcular el gradiente de la función de pérdida en el espacio de los pesos de la red neuronal, y comienza de manera inversa con el gradiente de pesos de la capa final hasta llegar al gradiente de los pesos de la capa de entrada.

La siguiente ecuación describe el cálculo de los gradientes de la función de pérdida con respecto a los pesos:

$$\frac{\partial E}{\partial w_{ij}^{(l)}} = \frac{\partial E}{\partial a_j^{(l)}} \cdot \frac{\partial a_j^{(l)}}{\partial z_j^{(l)}} \cdot \frac{\partial z_j^{(l)}}{\partial w_{ij}^{(l)}} \quad (4.12)$$

La ecuación 4.12 nace de aplicar la conocida *chain rule* (regla de la cadena) a la composición de funciones:

$$E(a(Z^L)) \quad (4.13)$$

donde  $L$  hace referencia a la última capa,  $Z^L$  es la suma ponderada de los pesos en la última capa,  $a(Z^L)$  es el resultado de las activaciones de las neuronas en la última capa, y  $E$  es la función de coste.

Esta composición de funciones se resuelve mediante la multiplicación de las derivadas intermedias, tal y como se muestra en la ecuación 4.12.

Lo siguiente es desarrollar y resolver las derivadas parciales de la ecuación, tales que:

$$\frac{\partial E}{\partial a_j^{(l)}} \quad (4.14)$$

es la derivada del coste respecto a la activación.

#### 4.1. Redes Neuronales Artificiales

---

$$\frac{\partial a_j^{(l)}}{\partial z_j^{(l)}} \quad (4.15)$$

describe la derivada de la activación respecto a la suma ponderada de los pesos. En este punto es donde juega un papel importante la función de activación y la condición necesaria que se imponía sobre su derivabilidad.

Y por último se tiene:

$$\frac{\partial z_j^{(l)}}{\partial w_{ij}^{(l)}} \quad (4.16)$$

que relaciona la derivada la suma ponderada teniendo en cuenta los valores de pesos. Esto tiene todo el sentido del mundo ya que si se recuerda la expresión de  $z(l)$ :

$$z(l) = \sum_{i=1} a_i^{(l-1)} \cdot w_i^l + b^l \quad (4.17)$$

se puede relacionar la entrada de la neurona en la capa  $l$  con la salida de las neuronas en la capa  $l - 1$ .

Finalmente, el gradiente se propaga hacia atrás, de tal manera que se computa el error de la última capa:

$$\delta^{(l)} = \frac{\partial E}{\partial a_j^{(l)}} \cdot \frac{\partial a_j^{(l)}}{\partial z_j^{(l)}} \quad (4.18)$$

y se retropropaga el error imputado  $\delta^{(l)}$  a la capa anterior:

$$\delta^{(l-1)} = W^{(l)} \cdot \delta^{(l)} \cdot \frac{\partial a_j^{(l-1)}}{\partial z_j^{(l-1)}} \quad (4.19)$$

siendo  $W^{(l)}$ , la matriz de pesos de la capa  $l$ . Este proceso se repite hasta que se llega a la primera capa, donde se han computado los gradientes de todas las variables de entrada y se pueden realizar las actualizaciones necesarias en los pesos para minimizar el error, ha-

## 4. Antecedentes

---

ciendo uso de algún algoritmo de optimización, como el descenso del gradiente.

### 4.2. Extreme Learning Machine

El modelo ELM es un algoritmo de ML basado en RNAs. Fue propuesto por Huang et al. [2006] como una forma de entrenar RNAs de manera eficiente y rápida.

La principal diferencia entre el ELM y otros algoritmos de RNAs es que, en ELM, los pesos de la capa oculta se seleccionan aleatoriamente y se fijan durante el proceso de entrenamiento, mientras que en otros algoritmos los pesos se actualizan iterativamente a través del proceso de aprendizaje, tal como se ha detallado en la sección anterior. Esto es lo que se conoce como *Single Layer Feedforward Network* (SLFN), y la característica principal de este modelo de alimentación de capa única es que no requiere de un proceso de optimización de los pesos como el descenso de gradiente o algoritmos similares, lo que lo hace más rápido y fácil de implementar.

En el modelo ELM se busca llegar a un sistema lineal en la capa de salida donde se pueda computar y realizar una clasificación por clases, a partir de un sistema no lineal en el conjunto de datos en la capa de entrada.

Por tanto, la salida de la red ELM se calcula mediante la siguiente fórmula:

$$\mathbf{y} = \mathbf{H} \cdot \beta \quad (4.20)$$

siendo  $\mathbf{y}$  la salida predicha por la red,  $\mathbf{H}$  la matriz de activación de la capa oculta y  $\beta$  es el vector de pesos de las conexiones entre la capa oculta y la capa de salida.

Dicha matriz  $\mathbf{H}$  vendrá condicionada por las funciones de activación que se emplee en las neuronas pertenecientes a la capa oculta, y  $\beta$  son los coeficientes resultantes de aplicar la inversa de Moore-Penrose,

## 4.2. Extreme Learning Machine

---

descrita en Moore [1920]:

$$\beta = H^+Y \quad (4.21)$$

Debido a que ELM se engloba dentro del aprendizaje supervisado, su objetivo de entrenamiento es encontrar el vector de pesos óptimo  $\beta$  que minimice el error entre la salida de la red, más conocido como la predicción del modelo, y las etiquetas reales. Este error se calcula mediante la siguiente función de pérdida:

$$\mathcal{L} = \|\mathbf{H}\beta - \mathbf{Y}\| \quad (4.22)$$

donde  $\mathbf{H}\beta$  es la predicción de la red e  $\mathbf{Y}$  es la salida deseada.

La problemática reside en la ausencia de dicho entrenamiento y optimización de pesos, ya que ELM es un modelo de capa única de alimentación, por tanto, se hace una sola instanciación de los pesos en el rango  $[-1, 1]$ . De esta manera se obtienen resultados de una manera mucho más rápida, y eficientes tal y como se ha demostrado en Huang et al. [2006].

### 4.3. Algoritmos Bioinspirados

Existen una gran variedad de algoritmos cuyo funcionamiento se inspira en procesos que ocurren en la naturaleza. En esta sección, se abordan los tres algoritmos que guardan relación con el presente TFG: algoritmos genéticos, algoritmos de enjambre y algoritmos de arrecife de coral.

#### 4.3.1. Algoritmo Genético

Los GAs son una clase de algoritmos de optimización que se inspiran en el proceso de selección natural de **Charles Darwin**. Estos algoritmos se utilizan para encontrar la mejor solución a un problema dado, a través de la iteración y la mejora continua de una población de individuos que representan las posibles soluciones de dicho problema.

Fueron desarrollados a partir de 1970, comenzando por el trabajo de investigación de **John Henry Holland** [Holland \[1975\]](#), y que más tarde se aplicarían al ámbito de la IA [Golberg \[1989\]](#), [Holland \[1992\]](#).

El funcionamiento de un algoritmo genético es el siguiente:

1. **Inicialización:** Se genera de manera aleatoria una población de individuos, donde cada individuo supone una posible solución al problema planteado.
2. **Evaluación:** A cada uno de los individuos de dicha población se les aplica una función para calcular su rendimiento y se determina si su solución al problema es buena o no.
3. **Condición de parada:** En este paso se abre toda una rama de investigación ya que, los dos pasos anteriores son triviales e inamovibles, pero la importancia y calidad del GA reside en las condiciones y criterios que se apliquen a la hora de evolucionar a los individuos. Durante un número determinado de generaciones se va iterando sobre la población y se aplican operadores

### 4.3. Algoritmos Bioinspirados

---

con inspiración biológica, dando como resultado una población con nuevos individuos con variedad genética distinta a la de la generación anterior. Los operadores que se aplican son:

- **Selección:** De la población de individuos se escoge una fracción de aquellos con mejor rendimiento, manteniendo un nivel de rendimiento en el algoritmo, y dando la posibilidad de creación de nuevos individuos con ligeras mutaciones genéticas que tengan un rendimiento similar o incluso mejor.

La selección de individuos se puede llevar a cabo de muchas maneras:

- **Selección directa:** toma individuos de acuerdo a un criterio objetivo, como son "los  $x$  mejores", o "los  $x$  peores".
  - **Selección aleatoria:** puede ser realizado por selección de igual probabilidad o selección estocástica.
  - **Selección neutra:** selecciona los individuos en el orden que le llegan, sin realizar ningún tipo de procesamiento sobre ellos.
  - **Selección por torneos:** realiza torneos entre individuos de la población escogidos al azar, quedando como vencedor aquel con mayor aptitud. El tamaño del torneo es variable.
  - **Selección por ruleta:** selecciona individuos utilizando una ruleta, en la que el área correspondiente a cada individuo es proporcional a su aptitud.
- **Cruce:** Es el principal comportamiento que se observa en la naturaleza y en el cual, mediante la reproducción de dos individuos, se obtienen dos descendientes, productos de la combinación genética de los padres.
  - **Mutación:** Este proceso surge de la variación genética aleatoria que se produce en una parte del cromosoma del in-

## 4. Antecedentes

---

dividuo, permitiendo explorar espacios de búsqueda no cubiertos por el resto de individuos.

En la Figura 4.5 se muestra un diagrama de flujo del GA.

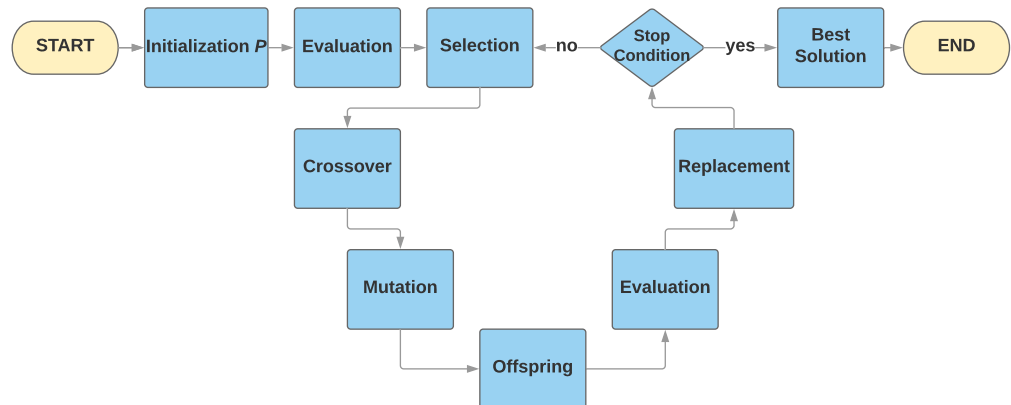


Figura 4.5: Diagrama de flujo GA, FUENTE: [Durán Rosal \[2019\]](#).

En conclusión, los algoritmos genéticos pertenecen a la categoría más amplia de algoritmos evolutivos y es una metaheurística inspirada en el proceso de selección natural. Al basarse en operadores de inspiración biológica como la mutación, el cruce y la selección, los algoritmos genéticos se emplean con frecuencia para producir soluciones de alta calidad a problemas de optimización y búsqueda [Golberg \[1989\]](#).

### 4.3.2. Particle Swarm Optimization

El algoritmo PSO se atribuye originalmente a **Kennedy, Eberhart** y **Shi** [Kennedy and Eberhart \[1995\]](#) y se pensó en un principio para simular el comportamiento social, como una representación estilizada del movimiento de los organismos en una bandada de pájaros o un banco de peces. En [Kennedy and Eberhart \[1995\]](#), los autores describen muchos aspectos filosóficos de PSO y la inteligencia de enjambre.



### 4.3. Algoritmos Bioinspirados

---

En el algoritmo de PSO se crea una población, o más conocido como enjambre, de partículas virtuales que representan diferentes soluciones posibles al problema de optimización. Cada partícula tiene una posición y una velocidad en el espacio de búsqueda, y se mueve a través del espacio de búsqueda en busca de la solución óptima. Cada partícula tiene también una “mejor posición” conocida, que representa la solución más óptima que ha encontrado hasta el momento.

Profundizando más en estos conceptos, la posición de la partícula representa la solución actual de dicha partícula en una iteración  $t$ , mientras que la velocidad representa la tendencia de la partícula a moverse hacia una nueva solución.

Estos dos atributos, velocidad  $v_i(t)$  y posición  $x_i(t)$ , de la partícula se actualizan para cada generación  $t$  del algoritmo en base a las siguientes ecuaciones:

$$\mathbf{v}_i(t+1) = \omega \mathbf{v}_i(t) + c_1 r_1 (\mathbf{p}_i(t) - \mathbf{x}_i(t)) + c_2 r_2 (\mathbf{p}_g(t) - \mathbf{x}_i(t)), \quad (4.23)$$

y

$$\mathbf{x}_i(t+1) = \mathbf{x}_i(t) + \mathbf{v}_i(t+1), \quad (4.24)$$

donde  $\omega$  es el coeficiente de inercia de la partícula,  $c_1$  y  $c_2$  son los coeficientes de aceleración cognitiva y social, respectivamente,  $r_1$  y  $r_2$  son números aleatorios  $\mathbf{U}(0, 1)$ ,  $\mathbf{p}_i(t)$  es la mejor solución encontrada por la partícula  $i$  hasta el momento  $t$ , y  $\mathbf{p}_g(t)$  es la mejor solución encontrada por el enjambre hasta el momento  $t$ .

Ejecutándose de forma iterativa, para cada iteración del algoritmo, las partículas se mueven hacia la mejor posición conocida de toda la población, así como hacia su propia mejor posición conocida. De este modo, las partículas aprenden de las soluciones óptimas encontradas por otras partículas, al mismo tiempo que intentan mejorar su propia solución.

## 4. Antecedentes

---

En la Figura 4.6 se muestra un diagrama de flujo del algoritmo PSO.

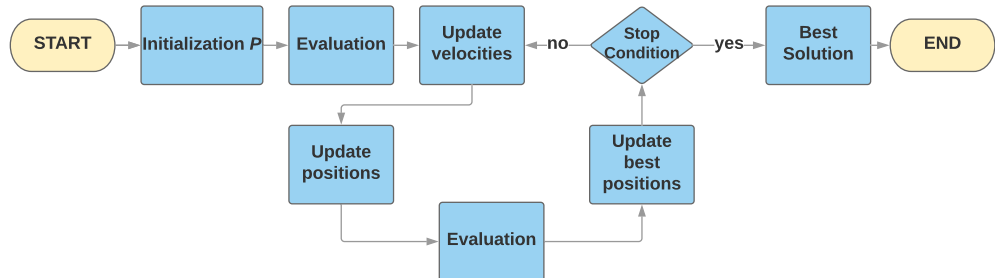


Figura 4.6: Diagrama de flujo PSO, FUENTE: [Durán Rosal \[2019\]](#).

De forma general, PSO es un algoritmo de optimización muy útil para problemas en los que es difícil encontrar una solución óptima con precisión, debido a la complejidad del problema o a la incapacidad de evaluar completamente el espacio de búsqueda.

### 4.3.3. Coral Reef Optimization

El algoritmo CRO es una metaheurística de optimización inspirada en el proceso natural de formación de los arrecifes de coral. Fue presentado por primera vez en [Salcedo-Sanz et al. \[2014a\]](#).

El algoritmo se basa en el comportamiento de los arrecifes de coral y su adaptación al medio. Los arrecifes de coral son ecosistemas complejos formados por la acumulación de pólipos de coral, que son pequeños animales marinos. El ecosistema de los arrecifes de coral es autoorganizativo y autoadaptativo, y los pólipos de coral son capaces de adaptarse a los cambios del entorno alterando sus patrones de crecimiento.

El algoritmo CRO se trata de un algoritmo basado en un arrecife que utiliza un conjunto de corales representando a la población. Cada arrecife de coral es una solución potencial al problema de optimiza-

### 4.3. Algoritmos Bioinspirados

---

ción. El algoritmo comienza con una población de corales generada aleatoriamente y mejora iterativamente la población aplicando diversos operadores de crecimiento y reproducción de corales.

El algoritmo CRO incluye varios procesos que son clave en su desarrollo:

- **Reproducción del coral:** Durante este proceso los arrecifes de coral se fragmentan por medio de la reproducción asexual, o bien se reproducen sexualmente mediante *broadcast spawning* y *brooding*, tal y como hace un algoritmo genético cuando lleva a cabo la etapa de cruce y mutación.
- **Asentamiento de la larva:** Cuando ya se han liberado al agua las larvas resultantes del proceso de reproducción, se puede producir un asentamiento de la larva en el arrecife de coral, que llegará a convertirse en un coral con mayor aptitud al medio.
- **Depredación:** Debido a múltiples factores, ya sean por actividades relacionadas con el humano o por otros animales marinos que son depredadores de los corales, algunas de las larvas que flotan en el agua e incluso algunos de los corales ya existentes en el arrecife pueden ser depredados y eliminados.

En el capítulo 7.1 se explica con un mayor nivel de detalle cada uno de estos procesos, ya que la combinación de este algoritmo con las redes ELM, CRO-ELM es un algoritmo de nueva implementación desarrollado en este TFG.

En la Figura 4.7 se muestra un diagrama de flujo del algoritmo CRO.

## 4. Antecedentes

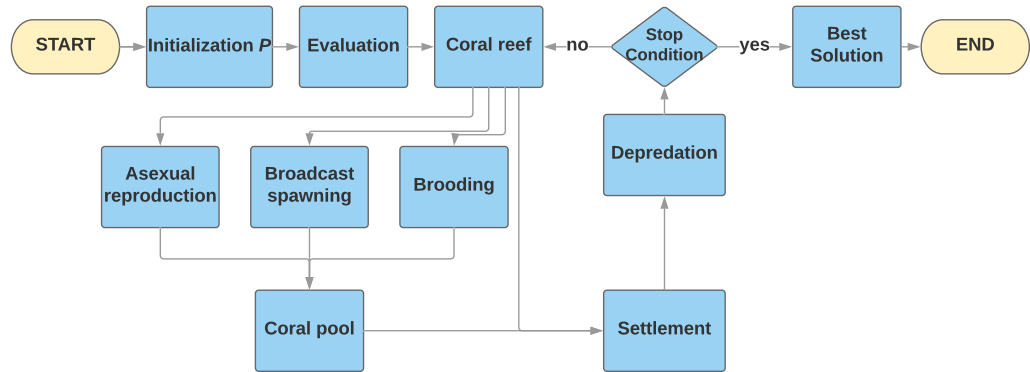


Figura 4.7: Diagrama de flujo CRO, FUENTE: Durán Rosal [2019].

Cabe mencionar que, a pesar de ser un algoritmo reciente, es un área de investigación activa que ha mostrado buenos resultados en muchos problemas de optimización y se sigue mejorando Duran-Rosal et al. [2018].

## 4.4. ELM con algoritmos evolutivos

### 4.4.1. GA-ELM

El uso de algoritmos genéticos aplicados a modelos neuronales, en este caso *Extreme Learning Machine*, es un paradigma que se ha usado en multitud de trabajos de investigación y en sectores muy variados, como puede ser: Purnomo et al. [2015], Albadr et al. [2020], Alencar et al. [2016].

En el artículo Albadr et al. [2020] se propone utilizar un OGA-ELM *Optimised Genetic Algorithm-Extreme Learning Machine* con tres criterios de selección: **random**, **K-tournament**, **roulette wheel** para detectar el COVID-19 utilizando imágenes de rayos X. Siguiendo los principios de selección, operadores de cruce y mutación de los algoritmos genéticos, junto con las ventajas propias de ELM, que son:

#### 4.4. ELM con algoritmos evolutivos

---

1. Capacidad para evitar el sobreajuste en su versión regularizada o aplicando técnicas de validación cruzada.
2. Usabilidad en clasificación binaria y multiclase.
3. Capacidad de funcionar como una máquina de vectores de soporte basada en kernel con una estructura de red neuronal.

Los resultados de dicho algoritmo muestran una precisión del 100 % detectando el virus del COVID-19 y bajo un tiempo de cómputo rápido, resolviendo la problemática de las Tomografías Computarizadas de Tórax.

Otro de los artículos mencionados [Purnomo et al. \[2015\]](#) se centra en el estudio de la cultivación de microalgas para aliviar el valor de concentración de CO<sub>2</sub>. Para este estudio, de nuevo, aplican un algoritmo genético combinado con ELM y validación cruzada para evitar los problemas de sobreajuste, consiguiendo resultados experimentales de *r-value*, *RMSE* y *RAE* superiores a otros algoritmos de regresión como:

- MLP (*Multilayer Perceptron*).
- LR (*Linear Regression*).
- RBF (*Radial basis Function*).
- IR (*Inference in Regression*).
- LMS (*Least Mean Squares*).
- ELM básico.
- Gau.
- M5Rules.

El último de los artículos mencionados es el presentado en [Alencar et al. \[2016\]](#), donde se analiza la técnica de poda de neuronas de la

## 4. Antecedentes

---

capa oculta en ELM mediante el uso de algoritmos genéticos. Debido a la problemática de la mala generalización que se presenta en ELM cuando el número de neuronas en la capa oculta aumenta, se propone un enfoque GA-ELM, que selecciona un subconjunto de neuronas de la capa oculta para optimizar una función de aptitud multiobjetivo, la cual define un equilibrio entre la precisión y el número de neuronas podadas.

A modo de resumen, el uso de algoritmos genéticos combinados con ELM es una tendencia en aumento en la investigación y en diferentes sectores. Los algoritmos genéticos proporcionan una herramienta para optimizar los parámetros de ELM, mejorando su rendimiento en tareas específicas y ayudando a solucionar problemas de sobreajuste y mala generalización. Esta combinación permite obtener resultados precisos y eficientes en tiempo de cómputo en una gran variedad de tareas.

### 4.4.2. PSO-ELM

En el artículo de investigación de [Kumar et al. \[2022\]](#) se diseña un enfoque de red neuronal híbrida basado en inteligencia de enjambre de dos etapas para la selección óptima de características y la optimización conjunta de los parámetros aprendibles de las redes neuronales con el fin de predecir el precio de cierre de tres índices bursátiles principales para la predicción a múltiples horizontes.

Aunque las RNAs *feed-forward* (FFNN) tienen la capacidad de lidiar con datos complejos y no lineales inciertos, el mayor desafío en las RNAs es definir su arquitectura.

Para superar los problemas de mínimos locales y sobreajuste de datos que pueden presentarse en algoritmos de retropropagación tradicionales, se combinan técnicas de búsqueda automática de característica y otros parámetros del modelo, basadas en inteligencia de enjambre combinando PSO y su versión discreta DPSO junto con algoritmo de Levenberg-Marquardt (LM).

La codificación de las distintas potenciales soluciones del algoritmo

#### 4.4. ELM con algoritmos evolutivos

se hace por medio de partículas con la siguientes propiedades:

- Selección de características (*feature-selection*).
- Determinación de nodos en capa oculta.
- Pesos y sesgos evolutivos iniciales.

En la Figura 4.8 se muestra la representación de una de las soluciones del problema, y se puede observar con claridad el uso de DPSO para hacer una búsqueda de optimización a través del subconjunto de características y neuronas ocultas del modelo, mientras que PSO se encarga del ajuste de los pesos y los sesgos de las neuronas.

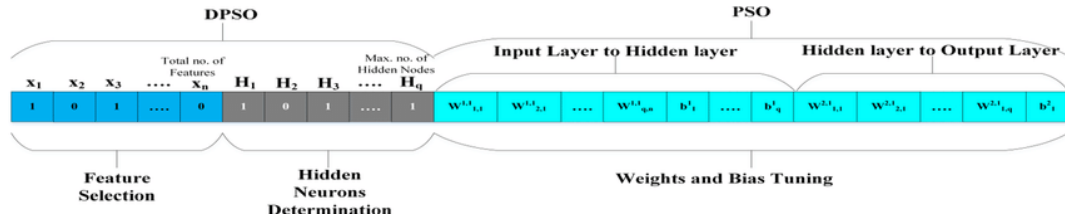


Figura 4.8: Representación de la solución, FUENTE: Kennedy and Eberhart [1995].

Por último en la Figura 4.9 se puede ver la arquitectura al completo del modelo neuronal híbrido de ELM con el proceso iterativo de entrenamiento, conectando las predicciones hechas por el modelo con la capa de entrada y procediendo a realidad el *tuning* de los parámetros.

#### 4. Antecedentes

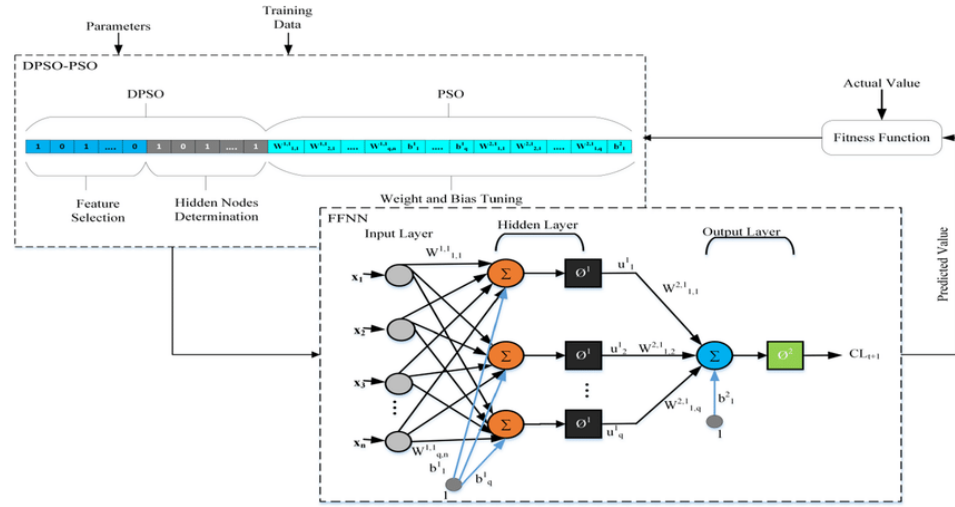


Figura 4.9: Proceso iterativo DPSO-PSO-FFNN para la optimización conjunta de la topología de red y los parámetros de FFNN, FUENTE: Kennedy and Eberhart [1995].







## 5 Restricciones

A continuación, se expondrán todas aquellas restricciones existentes en el ámbito del diseño y que condicionan la elección de una u otra alternativa a la hora de elegir entre distintas opciones en fases posteriores del Trabajo Fin de Grado. Este apartado se dividirá en dos partes bien diferenciadas que presentarán los dos tipos de factores a tener en cuenta:

- **Factores dato:** Estas restricciones forman parte del entorno de desarrollo del TFG y, por tanto, son inmutables.
- **Factores estratégicos:** Son todas aquellas opciones cuya elección parten del interés del alumno a la hora de realizar el trabajo y no son influenciadas por ningún agente externo. Un ejemplo claro sería la propia redacción de este manual técnico, el cual se está llevando a cabo mediante un sistema de composición de textos  $\text{\LaTeX}$ , debido a su amplia capacidad de plasmar en el documento un lenguaje matemático (complejo en fórmulas).

## 5.1. Factores Dato

Los factores datos comprenden un grupo de restricciones cuyo cumplimiento es obligatorio y debe permanecer constante en toda la ejecución del TFG, a expensas de ninguna excepción y/o necesidad de justificación:

- **Restricciones Humanas:** El desarrollo de este TFG es responsabilidad única del alumno David Pineda Peña del *Grado en Ingeniería Informática y Tecnologías Virtuales*, y el tutor Antonio Manuel Durán Rosal, perteneciente al Dpto. de Métodos Cuantitativos y a la Escuela Superior de Ingeniería de la Universidad Loyola Andalucía.
- **Restricciones Económicas:** Dado que el proyecto es un TFG, el alumno deberá hacer uso de software libre y librerías públicas en todo momento. En caso de necesitar utilizar algún software de carácter privado, éste deberá estar permitido fuera del ámbito profesional y carecer de coste monetario.
- **Restricciones Temporales:** La entrega acordada por ambas partes de este TFG se realizará en la convocatoria especial de Febrero de 2023, sin opción a extenderse más allá de dicha fecha.
- **Restricciones de Hardware:** El alumno proyectista deberá utilizar sus propios equipos informáticos para la realización del TFG, y ejecutar las pruebas, tanto en su equipo local, como en los servidores *cloud* de Google Research.
- **Restricciones de Software:** Utilización de Google Colab y Visual Studio Code como entorno de ejecución para el lenguaje de programación Python.

### 5.2. Factores Estratégicos

Los factores estratégicos son aquellas limitaciones dictadas por el alumno o el tutor del TFG:

Por un lado, el lenguaje de programación multiparadigma Python se ha escogido debido a los siguientes motivos:

- Amplia capacidad de desarrollo y conocimiento por parte del alumno, acerca de este lenguaje y sus librerías, gracias al uso extenso que se ha hecho a lo largo de la carrera.
- Gran velocidad de cálculo y alta funcionalidad con transformaciones y operaciones matriciales.
- Existencia de una gran documentación.
- Software libre con librerías preinstaladas.
- Orientación a objetos y fácil manejo de variables.

Una vez más, la documentación de este manual está siendo realizada en L<sup>A</sup>T<sub>E</sub>X, gracias a su manejo de documentos y estructura, que permite segmentar las partes de este trabajo en archivos *.tex* independientes. Junto con su capacidad de plasmar lenguaje matemático de aspecto académico.





## 6 Recursos

A continuación se exponen los distintos recursos humanos, de *hardware* y de *software* con los que se ha realizado el presente Trabajo Fin de Grado.

### 6.1. Recursos Humanos

El equipo de desarrollo lo constituyen:

- **David Pineda Peña:** Alumno del Grado de *Ingeniería Informática y Tecnologías Virtuales* de la Escuela Superior de Ingeniería en la Universidad Loyola Andalucía. Será el encargado del estudio, diseño, implementación y documentación del TFG.
- **Antonio M. Durán Rosal:** Profesor Adjunto del Departamento de Métodos Cuantitativos de la Universidad Loyola Andalucía e investigador del Grupo de Investigación MICA de la propia Universidad. Es el encargado de dar soporte técnico y teórico para las tareas de análisis, diseño y programación, así como la dirección y supervisión de la consecución de los objetivos del TFG.

## 6.2. Recursos Hardware

Los recursos hardware utilizados son propiedad personal del alumno y consta de un PC de sobremesa con las siguientes características:

- **Intel Core i5-10400F 2.90 GHz** (64 bits y 6 núcleos de proceso).
- **16 GB DDR4 3200Mhz** de memoria RAM.
- Disco duro **SSD** de 240GB.
- Disco duro **HDD** de 1TB.
- Tarjeta gráfica **GeForce RTX 3060 Ti**.

Para el desarrollo de las pruebas, se ha hecho uso del entorno de Google Colab. Dicho entorno, se ejecuta por completo en la nube y proporciona un acceso limitado a 12 horas continuadas a la GPU **NVIDIA Tesla K80** con el plan gratuito. Las características de la unidad gráfica mencionada son:

- **4992 núcleos** de NVIDIA CUDA.
- Hasta **2,91** teraflops de rendimiento en operaciones de precisión doble.
- Hasta **8,73** teraflops de rendimiento en operaciones de precisión simple.
- **24 GB** de memoria GDDR5.

## 6.3. Recursos Software

### 6.3.1. Sistemas Operativos

- *Microsoft Windows 10 Pro*

### 6.3.2. Lenguajes y Herramientas de Programación

- *Python 3.9.12*, como lenguaje de programación.
- *Visual Studio Code 1.74.2*, como IDE para la implementación del código.
- *Anaconda Distribution 22.11.1*, como gestor de paquetes, librerías y entornos de Python.

### 6.3.3. Herramientas de Documentación

- *Overleaf*: Entorno de desarrollo para la elaboración de una documentación en  $\text{\LaTeX}$ .
- *Notion*: Software de gestión de proyectos para la organización y decisión de la documentación final del TFG.





## Parte II

# Análisis y Especificación de Requisitos





## 7 Descripción General del Problema

En el siguiente capítulo se procede a estructurar y disgregar de forma concisa el problema planteado. Se explicará al detalle el algoritmo desarrollado en este Trabajo Fin de Grado, empezando por la representación de las soluciones al problema y las distintas fases de evolución, hasta finalizar con el pseudocódigo de dicho algoritmo para tener un visión global de todo lo explicado durante el capítulo.

### 7.1. Algoritmo Propuesto

En líneas generales se pretende desarrollar un algoritmo CRO-ELM, que combine el modelo de red neuronal SLFN de Extreme Learning Machine, con el algoritmo de *Coral Reef Optimization*, dotando a la red neuronal ELM de una fase de entrenamiento en su fase inicial. Con esta propuesta se busca conseguir una considerable mejora en las predicciones del modelo, a cambio un incremento computacional asumible.

Combinar algoritmos evolutivos con ELM solventa problemas de generalización en la red y trae consigo una mejora en el rendimiento de la misma, tal y como se ha comentado en el Capítulo 4.

### 7.1.1. Representación de los Corales

En el algoritmo, cada una de las posibles soluciones al problema se les denomina **corales** y van a ser objetos de una clase **Larvae**, cuyo diagrama de clases será detallado en el Capítulo 11.

Estos corales, al englobarse dentro de un algoritmo evolutivo van a estar incluidos en una población de corales conocida como **Reef** (arrecife). Del mismo modo, al tratarse de las posibles soluciones de nuestra red neuronal van a estar formados por los siguientes elementos:

1. **Selección de características:** Se trata de un vector de enteros de la misma dimensión que las características de los datos de entradas, y cuya representación concluye si dicha característica va a tener una conexión con la capa oculta. Esto permitirá hacer una futura *feature selection* de nuestros datos de entrada optimizando los algoritmos.
2. **Pesos:** Cada conexión de las neuronas de entrada con las neuronas de la capa oculta, llevan asociado un peso  $w$  que define cómo de relevante es dicha conexión en la red neuronal. La representación de estos pesos viene dada por una matriz de reales en el intervalo  $\mathbf{U}[-1, 1]$ .
3. **Bias:** Es un valor asociado a cada neurona de la capa oculta que se utiliza para ajustar la salida de la neurona independientemente de la entrada. El bias se utiliza para desplazar la función de activación hacia arriba o hacia abajo, lo que permite a la red neuronal adaptarse mejor a los datos de entrada. Su representación es un vector de reales en el intervalo  $\mathbf{U}[-1, 1]$ .

Estos tres elementos son la base fundamental de este TFG ya que, como se ha detallado previamente, el entrenamiento del modelo y la optimización que se va a realizar sobre la capa de entrada es el ajuste y validación de todos ellos.

## 7. Descripción General del Problema

---

### 7.1.2. Generación del Arrecife

El arrecife inicial del algoritmo es generado de forma aleatoria, asignando un porcentaje  $\rho_0$  de los espacios del arrecife para ser ocupados por corales y dejando el resto de espacios libres.

De esta forma, el algoritmo tendrá que computar y evaluar solamente aquellos espacios en los que haya corales. Para generar estos corales iniciales se asignan valores aleatorios a la parte de selección de características, pesos y bias de cada coral. Los valores asignados deben estar dentro del rango especificado en la representación de los corales, en este caso  $\mathbf{U}[-1, 1]$  para los pesos y el bias, y un vector de enteros para el para la representación de la selección de las características.

Es importante mencionar que el valor de  $\rho_0$  es un parámetro importante que debe ser ajustado cuidadosamente, ya que un valor demasiado alto puede generar una sobrecarga en el cálculo del algoritmo, y por otro lado un valor demasiado bajo puede limitar la diversidad de la población inicial y dificultar la evolución del arrecife.

### 7.1.3. Evaluación

La evaluación del rendimiento de los corales se lleva a cabo mediante lo que se conoce como la **función de salud** del propio coral. Dicha función de salud representa la función objetivo del problema.

Para el caso del algoritmo CRO-ELM desarrollado en este TFG, ésta función objetivo será la propia función del modelo ELM que se ha detallado en el Capítulo 4.2.

### 7.1.4. Reproducción

En el algoritmo CRO clásico y en nuestro TFG se llevan a cabo tres tipos de reproducción, tal y como ocurre en el mundo real con los arrecifes de coral. Esos tipos de reproducción son:

### Reproducción Asexual

Una vez se hace una evaluación de la población actual de corales yacientes en el arrecife, se lleva a cabo un proceso de reproducción asexual o fragmentación.

Dada la función de salud de cada coral, se realiza un ordenamiento de todo el arrecife y se duplica una fracción  $\mathbf{F}_a$  de aquellos con mejor rendimiento. Dicha fracción  $\mathbf{F}_a$  es un parámetro del algoritmo CRO el cual tiene relevante importancia en el desarrollo del mismo y se debe validar.

Una vez se han duplicado todos los corales, por cada uno de ellos se obtienen lo que se conoce como *larva*, la cual se almacena dentro de una piscina de larvas y que posteriormente supondrá la clave evolutiva del algoritmo.

### Reproducción Externa, *Broadcast spawning*

Además de la reproducción asexual por fragmentación de los corales, también se cuenta con el enfoque de reproducción sexual, y en este apartado, externa. De nuevo se sigue un parámetro  $\mathbf{F}_b$  con el que se selecciona uniformemente una fracción aleatoria de los corales existentes para ser *broadcast spawners* (corales reproductores).

Cuando ya se han seleccionado a todos los corales candidatos para reproducirse, se procede a seleccionarlos por parejas y se lleva a cabo un cruce de las mismas, obteniendo una larva hija con carga genética de ambos padres. Todas las larvas descendientes van a parar a la misma piscina de larvas que se ha comentado anteriormente.

Es importante mencionar que, a diferencia del algoritmo genético, en el algoritmo CRO y en nuestro TFG, los candidatos padres desarrollan un único hijo. Por último, los corales escogidos para cruzarse solo pueden ser escogidos una vez solamente.

## 7. Descripción General del Problema

---

### Brooding

Como tercera, y última técnica de reproducción, se cuenta con la reproducción sexual interna. En esta fase se lleva a cabo la formación de una larva hija por medio de una variación aleatoria de un coral incubador o *brooding-reproductive coral*.

De manera trivial, los corales incubadores son todos aquellos que no han formado parte en la fase de *broadcast* (reproducción sexual externa) y se denotan como:  $\mathbf{1} - \mathbf{F_b}$ .

Se finaliza la fase de reproducción del algoritmo, incluyendo en la piscina de larvas a todas aquellas nuevas hijas generadas durante esta parte.

### 7.1.5. Asentamiento de la larva

Una vez se han generado las larvas a través de los procesos de reproducción, se procede al asentamiento de las mismas. En este proceso, cada larva es evaluada según su propia función de salud (rendimiento), y teniendo en cuenta varios factores como: la disponibilidad de espacio y la competencia con otras larvas, da lugar el asentamiento de las mismas.

En una primera instancia de este proceso se genera un número, el cual designa el agujero del arrecife en el que va a intentar asentarse la larva candidata.

Llegados a este punto pueden darse los siguientes tres escenarios:

1. **Sin competencia:** En este caso la larva no se encuentra con ningún coral antecedente en el hueco designado y se asienta de forma directa.
2. **Competencia favorable:** En esta situación, el espacio ya estaba ocupado por un coral anterior pero cuyo rendimiento es menor que el de la larva que está intentando asentarse, resultando así en una victoria para la larva y reemplazando al coral, que



termina por descartarse.

3. **Competencia desfavorable:** Igual que en el anterior, la larva encuentra competencia en el espacio designado para asentarse, pero en este caso el coral asentado tiene un mejor rendimiento. Llegados a este punto, en vez de descartar la larva candidata, se opta por darle una serie de intentos a la larva, definidos como un entero en las propiedades de la Clase Larva, para asentarse en algún espacio antes de ser descartada finalmente.

Es importante mencionar que el proceso de asentamiento es crucial para el desarrollo del algoritmo ya que garantiza que solo los individuos más aptos se quedan en el arrecife y contribuyen a su evolución.

### 7.1.6. Depredación

En el algoritmo CRO, también se tiene en cuenta el factor de depredación. Esto se refleja en la eliminación de una fracción  $\mathbf{F}_d$  de los corales con peor rendimiento en su función de salud.

La depredación simula la presión natural ejercida por organismos que se alimentan de corales. Al eliminar a los individuos menos aptos, se asegura que solo los corales más resistentes y adaptados sobreviven en el arrecife. Y, además, los huecos libres permiten el asentamiento de nuevas larvas en la siguiente iteración del algoritmo favoreciendo la diversidad de las soluciones.

Es importante mencionar que el valor de  $\mathbf{F}_d$  es un parámetro que se debe ajustar cuidadosamente, ya que un valor demasiado alto puede resultar en una eliminación excesiva de individuos, mientras que un valor demasiado bajo puede permitir que los individuos menos aptos se reproduzcan y se establezcan en el arrecife.

## 7. Descripción General del Problema

---

### 7.1.7. Criterio de Parada

El principal criterio de parada resulta en que se alcancen un número de generaciones establecido por el usuario de la aplicación.

### 7.1.8. Pseudocódigo del Algoritmo

---

**Algorithm 1:** Algoritmo CRO-ELM.

---

**Entrada:** Conjunto de datos de entrada y parámetros válidos para el algoritmo CRO  
**Salida:** Predicciones optimizadas del modelo  
Generación de un arrecife de coral aleatorio aplicando la configuración de entrada  
Evaluación los corales existentes en el arrecife inicial **for** *cada iteración del algoritmo* **do**  
    Reproducción asexual (budding)  
    Procesos de reproducción sexual (*broadcast spawning* y *brooding*)  
    Asentamiento de las larvas  
    Evaluar la nueva población no computada en el arrecife de coral  
    Proceso de depredación  
    Incrementar iteración  
**end**  
Entrenar ELM con los atributos del mejor coral del arrecife  
Obtener predicciones sobre el conjunto de test

---





## 8 Especificación de Requisitos

En este capítulo se procede a hacer un estudio más amplio de la aplicación desarrollada durante este Trabajo Fin de Grado.

### 8.1. Introducción

La finalidad del TFG consiste en desarrollar un sistema capaz de ejecutar el modelo de Extreme Learning Machine, con los algoritmos bioinspirados ya mencionados, para el entrenamiento de la redes neuronales de entrada, haciendo un ajuste de los pesos y una selección de características (*feature-selection*). Finalmente, el sistema también dará la opción de visualizar el entrenamiento de manera gráfica o exportar los datos generados durante su ejecución. Cabe destacar que el usuario final será un investigador con conocimientos de programación y conceptos de *Machine Learning*, dado que éste será el encargado de escoger los *datasets* de entrenamiento e introducir los datos sobre los hiperparámetros propios de los diferentes algoritmos.

Las motivaciones para realizar este TFG y los problemas principales que se pretenden resolver son los siguientes:

- La implementación de tres algoritmos bioinspirados aplicados al

modelo ELM para el ajuste de los pesos de la capa de entrada y la selección de características de esta.

- Desarrollo y despliegue de un *dashboard* interactivo, que facilite la introducción de los datos del problema y dote al usuario de una visualización gráfica del proceso de ejecución, además de la posibilidad de exportar los resultados finales en formato texto *.xlsx*.

## 8. Especificación de Requisitos

---

### 8.2. Participantes en el TFG

Los participantes que han colaborado en el desarrollo del presente TFG son detallados en las tablas 8.1, 8.2.

Participante	David Pineda Peña
<b>Organización</b>	Universidad Loyola Andalucía
<b>Rol</b>	Desarrollador
<b>Es desarrollador</b>	Si
<b>Es cliente</b>	No
<b>Es usuario</b>	No
<b>Comentarios</b>	Alumno de Grado en Ingeniería Informática y Tecnologías Virtuales de la Universidad Loyola Andalucía en el campus de Sevilla. Será el encargado de desarrollar la aplicación y la documentación

Tabla 8.1: Participante David Pineda Peña.

Participante	Antonio Manuel Durán Rosal
<b>Organización</b>	Universidad Loyola Andalucía
<b>Rol</b>	Tutor
<b>Es desarrollador</b>	No
<b>Es cliente</b>	Si
<b>Es usuario</b>	Si
<b>Comentarios</b>	Profesor del Dpto. de Métodos Cuantitativos de la Universidad Loyola Andalucía. Será el encargado de dar soporte para consultas de diseño y la programación de la aplicación, así como la dirección y supervisión del trabajo realizado.

Tabla 8.2: Participante Antonio Manuel Durán Rosal.

## 8.3. Catálogo de Objetivos del Sistema

En las siguientes tablas quedan reflejados los objetivos planteados durante el estudio inicial del TFG y que se buscan satisfacer tras el desarrollo de la aplicación. Las tablas serán nombradas bajo el siguiente estándar **OBJ-XXX**.

<b>OBJ-001</b>	Desarrollo de una interfaz gráfica
<b>Versión</b>	1.0
<b>Autores</b>	David Pineda Peña
<b>Fuentes</b>	Antonio Manuel Durán Rosal
<b>Descripción</b>	Ejecutar los algoritmos bioinspirados mediante una interfaz gráfica sencilla y con opción a ser configurada por el usuario de la aplicación.
<b>Subobjetivos</b>	OBJ-002 Desarrollo de algoritmos bioinspirados en el modelo ELM OBJ-003 Opciones de configuración OBJ-004 Manejo de datos de entrada OBJ-005 Almacenamiento de resultados
<b>Comentarios</b>	Ésta interfaz será desarrollada mediante el lenguaje de programación Python y haciendo uso de la librería Panel

Tabla 8.3: Objetivo perteneciente a la implementación de la interfaz gráfica.

En la Figura 8.1 se muestra la distribución a seguir en el diseño de la interfaz gráfica del presente TFG.

## 8. Especificación de Requisitos

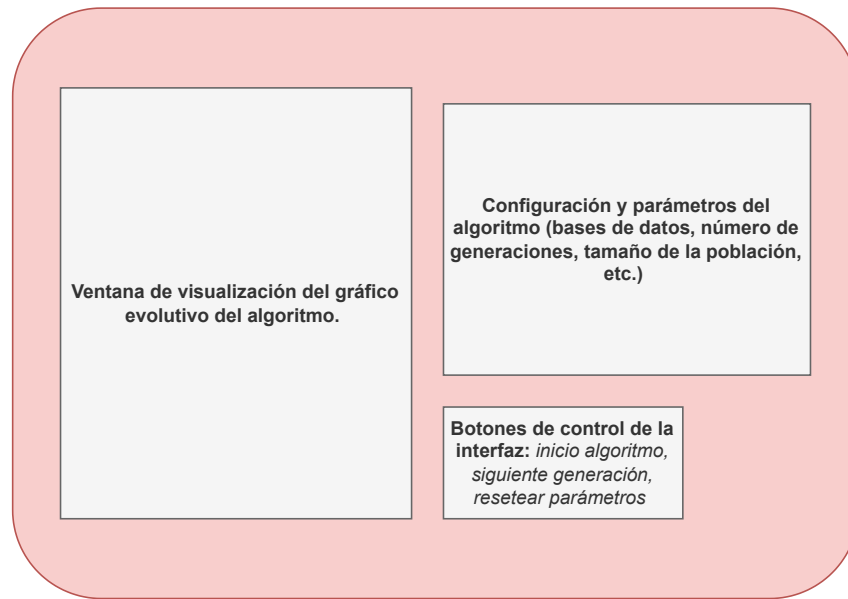


Figura 8.1: Esquema del OBJ-001 relacionado con la interfaz gráfica.

<b>OBJ-002</b>	Desarrollo de algoritmos bioinspirados en el modelo de <i>ELM</i>
<b>Versión</b>	1.0
<b>Autores</b>	David Pineda Peña
<b>Fuentes</b>	Antonio Manuel Durán Rosal
<b>Descripción</b>	Implementar los algoritmos bioinspirados para la optimización de la red ELM.
<b>Subobjetivos</b>	OBJ-003 Opciones de configuración OBJ-004 Manejo de datos de entrada OBJ-005 Almacenamiento de resultados OBJ-006 Implementación del modelo ELM OBJ-007 Implementación del algoritmo GA OBJ-008 Implementación del algoritmo PSO OBJ-009 Implementación del algoritmo CRO
<b>Comentarios</b>	Se ha subdivido este objetivo en 6 puesto que engloba las implementaciones de los diferentes algoritmos bioinspirados y la construcción de la red neuronal

Tabla 8.4: Objetivo perteneciente a la implementación de los algoritmos bioinspirados en el modelo ELM.



### 8.3. Catálogo de Objetivos del Sistema

<b>OBJ-003</b>	Opciones de configuración
<b>Versión</b>	1.0
<b>Autores</b>	David Pineda Peña
<b>Fuentes</b>	Antonio Manuel Durán Rosal
<b>Descripción</b>	Definir las distintos datos del sistema configurables por el usuario y que determinarán la ejecución del sistema y los resultados obtenidos.
<b>Subobjetivos</b>	Ninguno
<b>Comentarios</b>	El usuario podrá configurar: la base de datos del experimento, parámetros relativos a la población, número de iteraciones, hiperparámetros propios de la red neuronal y el algoritmo metaheurístico seleccionado

Tabla 8.5: Objetivo perteneciente a las opciones de configuración.

<b>OBJ-004</b>	Entrada de datos
<b>Versión</b>	1.0
<b>Autores</b>	David Pineda Peña
<b>Fuentes</b>	Antonio Manuel Durán Rosal
<b>Descripción</b>	El usuario podrá introducir los datos de entrada del problema a través de ficheros.
<b>Subobjetivos</b>	Ninguno
<b>Comentarios</b>	Los archivos de texto deberán existir bajo la extensión de nombre <i>.csv</i>

Tabla 8.6: Objetivo perteneciente a la entrada de datos.

<b>OBJ-005</b>	Almacenamiento de resultados
<b>Versión</b>	1.0
<b>Autores</b>	David Pineda Peña
<b>Fuentes</b>	Antonio Manuel Durán Rosal
<b>Descripción</b>	Almacenar las salidas de los experimentos en archivos de extensión <i>.xlsx</i> .
<b>Subobjetivos</b>	OBJ-010 Representación gráfica de los resultados
<b>Comentarios</b>	Se llevará a cabo mediante el botón de exportación que aparece en la interfaz gráfica

Tabla 8.7: Objetivo perteneciente al almacenamiento de resultados.

## 8. Especificación de Requisitos

---

<b>OBJ-006</b>	Implementación del modelo ELM
<b>Versión</b>	1.0
<b>Autores</b>	David Pineda Peña
<b>Fuentes</b>	Antonio Manuel Durán Rosal
<b>Descripción</b>	Implementar la versión regularizada de la red neuronal Extreme Learning Machine.
<b>Subobjetivos</b>	Ninguno
<b>Comentarios</b>	Se desarrollará siguiendo los principios descritos en [Huang et al., 2006]

Tabla 8.8: Objetivo perteneciente a la implementación del modelo ELM.

<b>OBJ-007</b>	Implementación del algoritmo GA
<b>Versión</b>	1.0
<b>Autores</b>	David Pineda Peña
<b>Fuentes</b>	Antonio Manuel Durán Rosal
<b>Descripción</b>	Implementar el algoritmo genético.
<b>Subobjetivos</b>	Ninguno
<b>Comentarios</b>	Se desarrollará siguiendo los principios descritos en [Holland, 1992]

Tabla 8.9: Objetivo perteneciente a la implementación del algoritmo genético.

<b>OBJ-008</b>	Implementación del algoritmo PSO
<b>Versión</b>	1.0
<b>Autores</b>	David Pineda Peña
<b>Fuentes</b>	Antonio Manuel Durán Rosal
<b>Descripción</b>	Implementar el algoritmo basado en optimización de enjambre de partículas PSO.
<b>Subobjetivos</b>	Ninguno
<b>Comentarios</b>	Se desarrollará siguiendo los principios descritos en [Kennedy and Eberhart, 1995]

Tabla 8.10: Objetivo perteneciente a implementación del algoritmo PSO.

### 8.3. Catálogo de Objetivos del Sistema

---

<b>OBJ-009</b>	Implementación del algoritmo CRO
<b>Versión</b>	1.0
<b>Autores</b>	David Pineda Peña
<b>Fuentes</b>	Antonio Manuel Durán Rosal
<b>Descripción</b>	Implementar el algoritmo basado en la reproducción de los corales CRO
<b>Subobjetivos</b>	Ninguno
<b>Comentarios</b>	Se desarrollará siguiendo los principios descritos en [Salcedo-Sanz et al., 2014a]

Tabla 8.11: Objetivo perteneciente a implementación del algoritmo CRO.

<b>OBJ-010</b>	Representación gráfica de los resultados
<b>Versión</b>	1.0
<b>Autores</b>	David Pineda Peña
<b>Fuentes</b>	Antonio Manuel Durán Rosal
<b>Descripción</b>	Representar en formato de gráfico la evolución de los algoritmos durante los experimentos.
<b>Subobjetivos</b>	Ninguno
<b>Comentarios</b>	Ninguno

Tabla 8.12: Objetivo perteneciente a la representación gráfica de los resultados.

### 8.4. Catálogo de Requisitos del Sistema

#### 8.4.1. Requisitos de la Información

##### Requisitos de Almacenamiento de la Información

Representados bajo la nomenclatura **IRQ-XXX**. En cuanto a los requisitos de información se identifican los siguientes:

<b>IRQ-001</b>	Resultados de los experimentos
<b>Versión</b>	1.0
<b>Autores</b>	David Pineda Peña
<b>Fuentes</b>	Antonio Manuel Durán Rosal
<b>Dependencias</b>	OBJ-001 Desarrollo de una interfaz gráfica OBJ-005 Almacenamiento de resultados OBJ-010 Representación gráfica de los resultados
<b>Descripción</b>	Representar y dar al usuario como opción la exportación a hoja de cálculo los resultados de la ejecución.
<b>Comentarios</b>	En el requisito no funcional <i>NFRQ-003</i> se especificarán los formatos de salida para el almacenamiento de los diagramas gráficos

Tabla 8.13: Requisito de Información perteneciente a la representación de los experimentos.

## 8.4. Catálogo de Requisitos del Sistema

---

<b>IRQ-002</b>	Configuración de los experimentos
<b>Versión</b>	1.0
<b>Autores</b>	David Pineda Peña
<b>Fuentes</b>	Antonio Manuel Durán Rosal
<b>Dependen- cias</b>	OBJ-003 Opciones de configuración OBJ-005 Almacenamiento de resultados
<b>Descripción</b>	Almacenar la configuración de cada experimento en el fichero <i>.xlsx</i> creado al exportar los resultados
<b>Comentarios</b>	Los valores de configuración que se guardarán son aquellos referentes a la población y los hiperparámetros relativos a la red neuronal ELM y al algoritmo bioinspirado seleccionado

Tabla 8.14: Requisito de Información perteneciente a la configuración de los experimentos.

## 8. Especificación de Requisitos

### 8.4.2. Requisitos Funcionales

Para los requisitos funcionales del sistema, con la nomenclatura **FRQ-XXX** se registran los siguientes:

<b>FRQ-001</b>	Aplicar configuración de entrada
<b>Versión</b>	1.0
<b>Autores</b>	David Pineda Peña
<b>Fuentes</b>	Antonio Manuel Durán Rosal
<b>Objetivos Asociados</b>	OBJ-003 Opciones de configuración OBJ-004 Entrada de datos
<b>Requisitos Asociados</b>	Ninguno
<b>Descripción</b>	Utilizar la configuración de cada experimento proporcionada por el usuario asignando los valores introducidos a los parámetros de la ejecución.
<b>Comentarios</b>	Ninguno

Tabla 8.15: Requisito funcional perteneciente a aplicar la configuración de entrada.

<b>FRQ-002</b>	Ejecutar algoritmo
<b>Versión</b>	1.0
<b>Autores</b>	David Pineda Peña
<b>Fuentes</b>	Antonio Manuel Durán Rosal
<b>Objetivos Asociados</b>	OBJ-002 Desarrollo de algoritmos bioinspirados en el modelo de ELM OBJ-004 Entrada de datos OBJ-005 Almacenamiento de resultados OBJ-006 Implementación del modelo ELM OBJ-007 Implementación del algoritmo genético OBJ-008 Implementación del algoritmo PSO OBJ-009 Implementación del algoritmo CRO
<b>Requisitos Asociados</b>	Ninguno
<b>Descripción</b>	Se dará comienzo a la ejecución de la aplicación utilizando el algoritmo seleccionado.
<b>Comentarios</b>	Ninguno

Tabla 8.16: Requisito funcional perteneciente a la ejecutar el algoritmo.

## 8.4. Catálogo de Requisitos del Sistema

<b>FRQ-003</b>	Almacenar los resultados de los algoritmos
<b>Versión</b>	1.0
<b>Autores</b>	David Pineda Peña
<b>Fuentes</b>	Antonio Manuel Durán Rosal
<b>Objetivos Asociados</b>	OBJ-005 Almacenamiento de resultados
<b>Requisitos Asociados</b>	Ninguno
<b>Descripción</b>	Una vez finalizada la ejecución y si el usuario lo decide se almacenará en un fichero de datos <i>.xlsx</i> los resultados del experimento.
<b>Comentarios</b>	Este almacenamiento de los resultados solo se realizará si el usuario activa el botón de exportación que se incluye en la interfaz gráfica. En caso contrario no se producirá ningún fichero

Tabla 8.17: Requisito funcional perteneciente a almacenar los resultados de los algoritmos.

<b>FRQ-004</b>	Representación gráfica de rendimiento del algoritmo
<b>Versión</b>	1.0
<b>Autores</b>	David Pineda Peña
<b>Fuentes</b>	Antonio Manuel Durán Rosal
<b>Objetivos Asociados</b>	OBJ-002 Desarrollo de algoritmos bioinspirados en el modelo de ELM OBJ-010 Representación gráfica de los resultados
<b>Requisitos Asociados</b>	Ninguno
<b>Descripción</b>	En el gráfico de visualización que se muestra en la interfaz gráfica se verá la evolución del <i>fitness</i> del algoritmo seleccionado, teniendo en cuenta la mejor solución de la generación actual
<b>Comentarios</b>	Ninguno

Tabla 8.18: Requisito funcional perteneciente a la representación gráfica de rendimiento del algoritmo.

## 8. Especificación de Requisitos

---

### 8.4.3. Requisitos no Funcionales

Por último se describen los requisitos no funcionales que se han impuesto en el desarrollo del sistema, siguiendo la nomenclatura **NFR-XXX**:

<b>NFR-001</b>	Lenguaje de programación Python
<b>Versión</b>	1.0
<b>Autores</b>	David Pineda Peña
<b>Fuentes</b>	Antonio Manuel Durán Rosal
<b>Dependencias</b>	OBJ-002 Desarrollo de algoritmos bioinspirados en el modelo de ELM OBJ-006 Implementación del modelo ELM OBJ-007 Implementación del algoritmo genético OBJ-008 Implementación del algoritmo PSO OBJ-009 Implementación del algoritmo CRO
<b>Descripción</b>	El uso del lenguaje de programación Python se debe a que es un lenguaje de alto nivel con gran potencia de cálculo con librerías de operaciones matriciales avanzadas y al avanzado conocimiento del proyecto sobre el lenguaje.
<b>Comentarios</b>	Ninguno

Tabla 8.19: Requisito no funcional relativo al lenguaje de programación Python.



## 8.4. Catálogo de Requisitos del Sistema

<b>NFR-002</b>	Usabilidad
<b>Versión</b>	1.0
<b>Autores</b>	David Pineda Peña
<b>Fuentes</b>	Antonio Manuel Durán Rosal
<b>Dependencias</b>	OBJ-001 Desarrollo de una interfaz gráfica  OBJ-002 Desarrollo de algoritmos bioinspirados en el modelo <i>ELM</i> OBJ-003 Opciones de configuración
<b>Descripción</b>	Deberá desarrollarse un sistema fácil de usar, con apenas curva de aprendizaje y dotado con una interfaz interactiva que no exija al usuario de grandes conocimientos para poder usarla, o en su defecto, que requiera poco tiempo para familiarizarse con ella.
<b>Comentarios</b>	Con el objetivo de facilitar el uso se proporcionará de un manual de usuario detallado y con guías de uso sobre la entrada de parámetros y salida de datos. La usabilidad será en todo momento objeto de interés en el desarrollo de la interfaz de usuario

Tabla 8.20: Requisito no funcional perteneciente a la usabilidad.

<b>NFR-003</b>	Formatos de almacenamiento para los resultados
<b>Versión</b>	1.0
<b>Autores</b>	David Pineda Peña
<b>Fuentes</b>	Antonio Manuel Durán Rosal
<b>Dependencias</b>	OBJ-004 Opciones de configuración  OBJ-005 Almacenamiento de resultados
<b>Descripción</b>	Los resultados de las ejecuciones serán guardados una vez el usuario haya pulsado el botón de exportación, y se llevará a cabo mediante la creación de un fichero <i>.xlsx</i> con los resultados y la configuración de parámetros e hiperparámetros de dicho experimento.
<b>Comentarios</b>	Ninguno

Tabla 8.21: Requisito no funcional perteneciente a los formatos de almacenamiento para los resultados.

## 8. Especificación de Requisitos

---

<b>NFR-004</b>	Fiabilidad y optimización
<b>Versión</b>	1.0
<b>Autores</b>	David Pineda Peña
<b>Fuentes</b>	Antonio Manuel Durán Rosal
<b>Dependencias</b>	OBJ-001 Desarrollo de una interfaz gráfica  OBJ-002 Desarrollo de algoritmos bioinspirados en el modelo ELM
<b>Descripción</b>	Se debe garantizar el uso y manejo de todo tipo de errores y excepciones posibles de aparecer tanto en el uso de la interfaz gráfica como en la ejecución de los algoritmos. En caso de algún error en el sistema, aparecerá una notificación descriptiva del error para el entendimiento del usuario. Adicionalmente, los órdenes de complejidad de los algoritmos y el sistema en sí deben estar optimizados en cuestión de tiempo de ejecución.
<b>Comentarios</b>	Ninguno

Tabla 8.22: Requisito no funcional perteneciente a la fiabilidad y optimización.

<b>NFR-005</b>	Formatos válidos para la entrada de texto
<b>Versión</b>	1.0
<b>Autores</b>	David Pineda Peña
<b>Fuentes</b>	Antonio Manuel Durán Rosal
<b>Dependencias</b>	OBJ-004 Entrada de datos  OBJ-005 Almacenamiento de resultados
<b>Descripción</b>	Para la entrada se hará uso de la extensión de archivo <i>.csv</i> . Mientras que el posterior almacenamiento de resultados se hará bajo la extensión de archivo <i>.xlsx</i> .
<b>Comentarios</b>	Ambos formatos de archivo son ampliamente usados y reconocidos en el sector sobre el que se está trabajando

Tabla 8.23: Requisito no funcional perteneciente a los formatos válidos para la entrada de texto.

## 8.5. Matriz de Trazabilidad

Tras detallar los objetivos y los distintos requisitos de información, funcionales y no funcionales del sistema, se presenta la matriz de trazabilidad 8.24 para así tener una visión global más clara y resumida acerca de cómo se relacionan todos ellos.

TRM-001	OBJ-001	OBJ-002	OBJ-003	OBJ-004	OBJ-005	OBJ-006	OBJ-007	OBJ-008	OBJ-009	OBJ-010
IRQ-001	X				X					X
IRQ-002			X		X					
FRQ-001			X	X						
FRQ-002		X		X	X	X	X	X	X	
FRQ-003					X					
FRQ-004		X								X
NFR-001		X				X	X	X	X	
NFR-002	X	X	X							
NFR-003				X	X					
NFR-004	X	X								
NFR-005				X	X					

Tabla 8.24: Matriz de trazabilidad para relacionar objetivos del TFG.

### 8.6. Formato de Entrada/Salida de Datos

En la sección anterior se ha hecho referencia a que la aplicación desarrollada tendrá un manejo de entrada y almacenamiento de datos, mediante ficheros *.csv* y *.xlsx*. A continuación, se detalla la estructura que tendrán estos ficheros.

#### 8.6.1. Formato de Entrada de Datos

En la Figura 8.2 se puede ver lo que sería un fichero *.csv*. Los datos están ordenados de tal forma que, en cada fila se tendría lo que sería patrón de la base de datos, y por columnas están las características de entrada de dicho patrón, a excepción de la última que representa la clase que el algoritmo va a tratar de predecir.

## 8.6. Formato de Entrada/Salida de Datos

	A	B	C	D	E
1	14.23	1.71	2.43	15.6	127.0
2	13.2	1.78	2.14	11.2	100.0
3	13.16	2.36	2.67	18.6	101.0
4	14.37	1.95	2.5	16.8	113.0
5	13.24	2.59	2.87	21.0	118.0
6	14.2	1.76	2.45	15.2	112.0
7	14.39	1.87	2.45	14.6	96.0
8	14.06	2.15	2.61	17.6	121.0
9	14.83	1.64	2.17	14.0	97.0
10	13.86	1.35	2.27	16.0	98.0
11	14.1	2.16	2.3	18.0	105.0
12	14.12	1.48	2.32	16.8	95.0
13	13.75	1.73	2.41	16.0	89.0
14	14.75	1.73	2.39	11.4	91.0
15	14.38	1.87	2.38	12.0	102.0
16	13.63	1.81	2.7	17.2	112.0
17	14.3	1.92	2.72	20.0	120.0
18	13.83	1.57	2.62	20.0	115.0
19	14.19	1.59	2.48	16.5	108.0
20	13.64	3.1	2.56	15.2	116.0
21	14.06	1.63	2.28	16.0	126.0

Figura 8.2: Formato de fichero .csv de Entrada de Datos.

### 8.6.2. Formato de Salida de Datos

El formato de salida de datos se corresponde con un *.xlsx* tal como se presenta en la Figura 8.3.

Dicho fichero de salida contiene la información sobre la configuración y resultados del experimento realizado mediante la aplicación:

- Base de datos seleccionada.
- Número de generaciones para el algoritmo bioinspirado.

## 8. Especificación de Requisitos

---

- Tamaño de la población.
- *Fitness* de los algoritmos bioinspirados.
- Tiempo de ejecución de los algoritmos.
- Hiperparámetros del modelo neuronal tales como.
  - Número de nodos en capa oculta.
  - Parámetro de complejidad.
- Hiperparámetros propios de cada algoritmo bioinspirados

## 8.6. Formato de Entrada/Salida de Datos

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W
Dataset	Generations	Population size	GA	PSO	CRO	GA time	PSO time	CRO time	D	C	Crossover prob	Mutation prob	w max	w min	c1	c2	rho0	eta	f broadcast	f asexual	f depredation	depredation probability
1 adult.csv	20	100	81.97451	75.87901	82.08199	40.16451	19.767	45.35704	10	0.1	0.65	0.15	1	0.1	2	2	0.6	2	0.6	0.2	0.2	0.01
2 balance-scale.csv	20	100	68	44	71.2	1.366999	0.630998	1.416031	10	0.1	0.65	0.15	1	0.1	2	2	0.6	2	0.6	0.2	0.2	0.01
3 balloons-a.csv	20	100	100	0	100	0.740032	0.339969	0.749032	10	0.1	0.65	0.15	1	0.1	2	2	0.6	2	0.6	0.2	0.2	0.01
4 balloons-b.csv	20	100	100	25	100	0.739002	0.349028	0.75297	10	0.1	0.65	0.15	1	0.1	2	2	0.6	2	0.6	0.2	0.2	0.01
5 balloons-c.csv	20	100	100	50	100	0.707031	0.341971	0.753	10	0.1	0.65	0.15	1	0.1	2	2	0.6	2	0.6	0.2	0.2	0.01
6 balloons-d.csv	20	100	100	75	75	0.722003	0.361001	0.755971	10	0.1	0.65	0.15	1	0.1	2	2	0.6	2	0.6	0.2	0.2	0.01
7 banknote-auth	20	100	97.81818	53.81818	97.81818	1.845969	0.778	1.853001	10	0.1	0.65	0.15	1	0.1	2	2	0.6	2	0.6	0.2	0.2	0.01
8 blood-transfus	20	100	75.33333	75.33333	75.33333	1.365998	0.614002	1.421999	10	0.1	0.65	0.15	1	0.1	2	2	0.6	2	0.6	0.2	0.2	0.01
9 breast-cancer-v	20	100	96.49123	62.2807	95.61404	1.219001	1.181001	1.366998	10	0.1	0.65	0.15	1	0.1	2	2	0.6	2	0.6	0.2	0.2	0.01
10 breast-cancer-v	20	100	79.48718	82.05128	87.17949	0.83403	0.886002	0.841968	10	0.1	0.65	0.15	1	0.1	2	2	0.6	2	0.6	0.2	0.2	0.01

Figura 8.3: Vista del fichero desde una hoja de cálculo.

## 9 Herramientas Utilizadas

A lo largo de este capítulo se detallan las principales tecnologías con las cuales se ha desarrollado este Trabajo Fin de Grado tras hacer un análisis de las alternativas posibles.

### 9.1. Lenguaje de Programación Python

Python es un lenguaje de programación de alto nivel y de propósito general. Fue creado a principios de los 90 por **Guido van Rossum** en el Instituto Nacional de Investigación y Matemática Aplicada (**CNRI**) en los Países Bajos.

Uno de los principales atractivos de Python es su sintaxis legible y concisa, lo que facilita la escritura y la lectura del código. Además, cuenta con una gran cantidad de bibliotecas y frameworks que amplían sus capacidades y lo hacen adecuado para una amplia variedad de tareas, desde el procesamiento de



Figura 9.1: Logo Python.



## 9.1. Lenguaje de Programación Python

---

texto y datos hasta la creación de aplicaciones web y de escritorio.

Otro punto a favor de Python es su comunidad activa y en constante crecimiento. Esto ha llevado a la creación de una amplia variedad de recursos y documentación en línea, así como a una amplia adopción por parte de empresas y organizaciones de todo el mundo.

### 9.1.1. Historia

Python se originó en los Países Bajos a principios de los 90. Su creador, Guido van Rossum, trabajaba en el Instituto Nacional de Investigación y Matemática Aplicada (CNRI) en ese momento y quería crear un lenguaje de programación que fuera fácil de usar y accesible para principiantes. El nombre del lenguaje, Python, se inspiró en los *Monty Python*, un grupo de humoristas británicos.

La primera versión de Python, la **0.9.0**, fue lanzada en febrero de 1991. En esa época, Python se consideraba un lenguaje de programación experimental y se utilizaba principalmente para tareas de sistemas, como la escritura de *scripts* de *shell*. A medida que Python maduró, se convirtió en un lenguaje de programación de propósito general y comenzó a ser utilizado en una amplia variedad de tareas.

En los años 2000, **Python 2.0** fue lanzado con una serie de nuevas características, como el soporte para clases y módulos de nivel superior. Python 2 se convirtió rápidamente en la versión dominante de Python y se utilizó ampliamente en la industria.

Para el 2008, se lanzó **Python 3.0**, que era una actualización significativa del lenguaje con una serie de cambios de diseño que buscaban mejorar la legibilidad y coherencia del código. Aunque Python 3 fue diseñado para ser compatible con el código escrito en Python 2, algunos cambios en la sintaxis y las bibliotecas hicieron que fuera necesario modificar código existente para que funcionara con Python 3. Esto ha llevado a que muchos usuarios continúen utilizando Python 2, a pesar de que la versión 2 ya no recibe actualizaciones oficiales desde enero

## 9. Herramientas Utilizadas

---

de 2020.

En la actualidad, Python sigue siendo uno de los lenguajes de programación más populares y ampliamente utilizados en el mundo. Se utiliza en una amplia variedad de tareas, desde el procesamiento de texto, pasando por la creación de aplicaciones web y de escritorio, y principalmente se utiliza en el sector de la IA y *Machine Learning*.

### 9.1.2. Características Principales

Dentro de la multitud de características principales que se atribuyen al lenguaje de programación Python, destacan las siguientes:

- **Amplia variedad de bibliotecas y frameworks:** Python cuenta con una gran cantidad de bibliotecas y frameworks que amplían sus capacidades y lo hacen adecuado para una amplia variedad de tareas. Por ejemplo, hay bibliotecas para el procesamiento de texto y datos, el aprendizaje automático y el desarrollo de aplicaciones web. Esto hace que Python sea una opción muy versátil y capaz de manejar una amplia variedad de problemas y necesidades.
- **Sintaxis legible y concisa:** Una de las características más atractivas de Python es su sintaxis legible y concisa, que hace que sea fácil de leer y escribir. Esto se debe en parte a la utilización de indentación para definir bloques de código en lugar de utilizar caracteres como llaves o puntos y comas. Esto hace que el código Python sea muy legible y fácil de seguir, incluso para aquellos que no están familiarizados con el lenguaje.
- **Tipado dinámico y conteo de referencias:** Python es un lenguaje de tipado dinámico, lo que significa que no es necesario declarar el tipo de datos de una variable. Esto significa que es posible asignar cualquier tipo de datos a una variable, como un número, una cadena de texto o una lista. Además, Python utiliza

## 9.1. Lenguaje de Programación Python

---

un sistema de conteo de referencias para gestionar la asignación y liberación de memoria de las variables. Esto significa que Python lleva un registro de cuántas referencias a una variable existen en el código y libera la memoria de la variable cuando ya no es necesaria.

- **Lenguaje de alto nivel:** Python es un lenguaje de alto nivel, lo que significa que está diseñado para ser fácil de usar y entender para los seres humanos. Esto lo hace ideal para principiantes y para aquellos que no tienen una formación en informática. Aunque es un lenguaje de alto nivel, también es posible utilizarlo para tareas más avanzadas y complejas.
- **Amplia adopción:** Python es ampliamente utilizado en la industria y en la academia y es utilizado por empresas y organizaciones de todo el mundo. Algunos ejemplos de empresas que utilizan Python incluyen Google, Netflix e Instagram. Esto se debe en parte a sus capacidades versátiles y a la gran cantidad de bibliotecas y *frameworks* disponibles.
- **Comunidad activa y en constante crecimiento:** Python tiene una comunidad activa y en constante crecimiento, con un gran número de usuarios y desarrolladores que contribuyen al lenguaje y a sus bibliotecas. Esto ha llevado a la creación de una amplia variedad de recursos y documentación en línea, así como a una amplia adopción por parte de empresas y organizaciones de todo el mundo.
- **Enlace dinámico de métodos:** Python utiliza el enlace dinámico de métodos, lo que significa que los métodos y funciones son resueltos en tiempo de ejecución. Esto significa que es posible modificar el comportamiento de una función o método en tiempo de ejecución, sin tener que recompilar el código. Esto hace que Python sea muy flexible y permite a los desarrolladores escribir código que se adapte a diferentes entornos y necesidades.

## 9. Herramientas Utilizadas

---

- **Facilidad de extensión:** Python es un lenguaje fácil de extender, lo que significa que es posible escribir código en Python que llame a bibliotecas o funciones escritas en otros lenguajes, como C o C++. Esto hace que Python sea ideal para la integración con otras bibliotecas y aplicaciones.

### 9.2. Librería Panel

Panel es una biblioteca de Python que permite a los desarrolladores crear aplicaciones de análisis de datos **interactivas** y **visuales**. Se utiliza ampliamente en el mundo empresarial y académico y ha sido adoptada por una amplia gama de empresas y organizaciones.

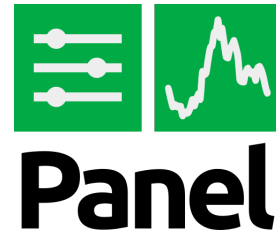


Figura 9.2: Logo Panel.

La biblioteca de visualización Python Panel se basa en **Matplotlib** y **Bokeh**, y permite a los desarrolladores incorporar gráficos y visualizaciones realizados con estas bibliotecas en las aplicaciones de Panel. Además, ofrece herramientas para desarrollar interfaces de usuario personalizadas, incluyendo paneles, pestañas y menús editables, así como componentes como botones y campos de texto.

#### 9.2.1. Historia

La librería Panel fue creada por **HoloViz**, un equipo de desarrollo de herramientas de visualización y análisis de datos en Python. La primera versión de Panel fue lanzada en marzo de **2018** y desde entonces ha sido ampliamente adoptada por la comunidad de Python.

Desde su lanzamiento, Panel ha recibido varias actualizaciones y mejoras, incluyendo la adición de nuevas características y la corrección de errores. En julio de 2018, se lanzó la versión 0.4 de Panel, que

incluyó la capacidad de crear aplicaciones web con Panel y la integración con Bokeh. En marzo de 2019, se lanzó la versión 0.6, que incluyó soporte para **gráficos 3D** y la integración con la librería **HoloViews**.

En junio de 2019, Panel fue aceptada como proyecto oficial de **NumFOCUS**, una organización sin ánimo de lucro que apoya proyectos de código abierto en el campo de la ciencia de datos. Esto significó que Panel recibió el respaldo financiero y de recursos necesarios para continuar su desarrollo y mejora.

En agosto de 2019, se lanzó la versión 0.8 de Panel, que incluyó mejoras en la integración con Bokeh y la adición de una API de widgets personalizables. En noviembre de 2019, se lanzó la versión 0.9, que incluyó la integración con Matplotlib y la adición de una API de servidor de aplicaciones.

En los últimos años, Panel ha seguido evolucionando y recibiendo actualizaciones y mejoras. Actualmente, se encuentra en la versión 1.1 y cuenta con una amplia base de usuarios y una comunidad activa de desarrolladores. Se espera que Panel siga creciendo y evolucionando en el futuro, proporcionando herramientas poderosas para el análisis de datos y la visualización en Python.

### 9.2.2. Características Principales

Entre las características típicas que se atribuyen a la librería Panel destacan:

- **Interfaces de usuario personalizables:** Permite a los desarrolladores crear interfaces de usuario personalizadas con facilidad.
- **Integración con otras librerías de visualización:** Está diseñado para integrarse con otras librerías de visualización de Python, como Matplotlib y Bokeh.
- **Personalización de widgets:** Proporciona una *API* de widgets personalizables que permite a los desarrolladores crear sus

## 9. Herramientas Utilizadas

---

propios widgets personalizados.

- Arquitectura lista para *plugins*.
- **Servidor de aplicaciones:** Proporciona una API de servidor de aplicaciones que permite a los desarrolladores crear aplicaciones web con Panel.
- **Documentación y soporte:** Cuenta con una amplia documentación y una comunidad activa de usuarios y desarrolladores que brindan soporte y ayuda.



# Parte III

## Diseño del Sistema







## 10 Arquitectura del Sistema

Se procede a desarrollar cómo se estructura modularmente el sistema y detallar las relaciones de control de los módulos y subsistemas.

### 10.1. Módulos

La aplicación a desarrollar va a estar dividida en dos grandes módulos:

- **Interfaz Gráfica de Usuario (GUI):** Con la finalidad de presentar al usuario, de una manera sencilla e intuitiva el software desarrollado, se ofrece una interfaz gráfica. Dicha interfaz de usuario será la encargada de gestionar todos los parámetros de configuración de los experimentos dados por el usuario, quien será el encargado de escoger la base de datos objetivo entre las distintas disponibles, o el tipo de algoritmo a ejecutar, junto con los propios hiperparámetros del propio algoritmo. Cabe mencionar que el módulo de la Interfaz Gráfica permite la opción de ejecutar el algoritmo por completo, o ejecutarlo generación por generación al mismo tiempo que visualiza en un gráfico la evolución del programa. Una vez finalizado, también se encargará de mostrar los resultados por pantalla y permitirá descargar en

formato *.xlsx* los resultados del experimento.

- **Algoritmos bioinspirados:** En este módulo se incluyen los diferentes algoritmos bioinspirados que se han referenciado en el Capítulo 4.

A continuación, en las Tablas 10.1 y 10.2 se especifica cada módulo y su correspondiente funcionalidad.

<b>Módulo</b>	Interfaz gráfica, <i>GUI</i>
<b>Descripción</b>	Se implementa una interfaz gráfica intuitiva y sencilla para facilitar al usuario la interacción y la visualización de resultados del módulo de los algoritmos
<b>Funcionalidad</b>	La funcionalidad de este módulo es permitir al usuario configurar y lanzar los experimentos del módulo de los algoritmos

Tabla 10.1: Módulo implementado de la interfaz gráfica.

<b>Módulo</b>	Algoritmos biosinpirados
<b>Descripción</b>	Se implementan los distintos algoritmos bioinspirados con su aplicación en el modelo de Extreme Learning Machine desarrollados en el Trabajo Fin de Grado
<b>Funcionalidad</b>	La funcionalidad de este módulo es la ejecución de los algoritmos evolutivos para el entrenamiento de redes ELM

Tabla 10.2: Módulo implementado de algoritmos bioinspirados.

### 10.1.1. Relación Algoritmo-Usuario

En el diagrama de la Figura 10.1 se ilustra de manera gráfica la relación entre el usuario y el módulo **Algoritmos de aprendizaje supervisado** junto con la funcionalidad del mismo.

## 10. Arquitectura del Sistema

---

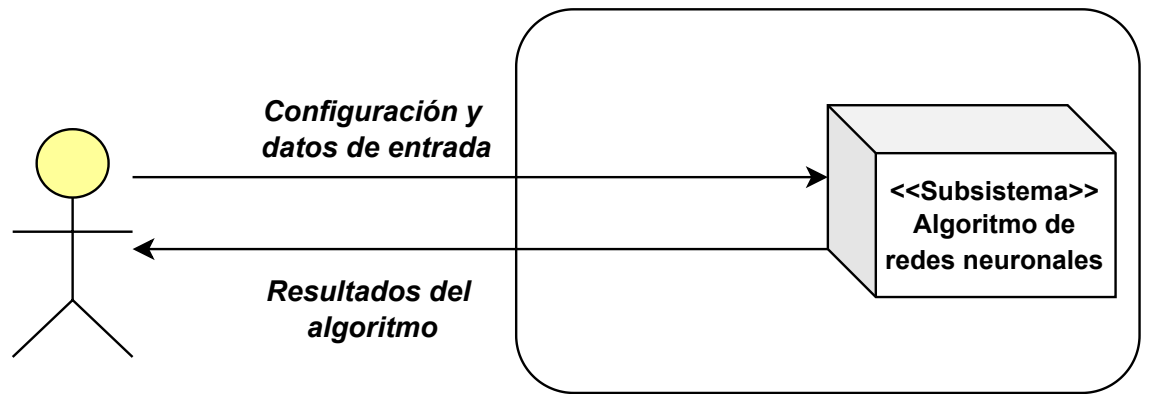


Figura 10.1: Diagrama de interacción Algoritmo-Usuario.

### 10.1.2. Relación Interfaz Gráfica-Usuario

En el diagrama de la Figura 10.2 se ilustra de manera gráfica de la relación entre el usuario y el módulo *Interfaz Gráfica de Usuario* junto con la funcionalidad del mismo.

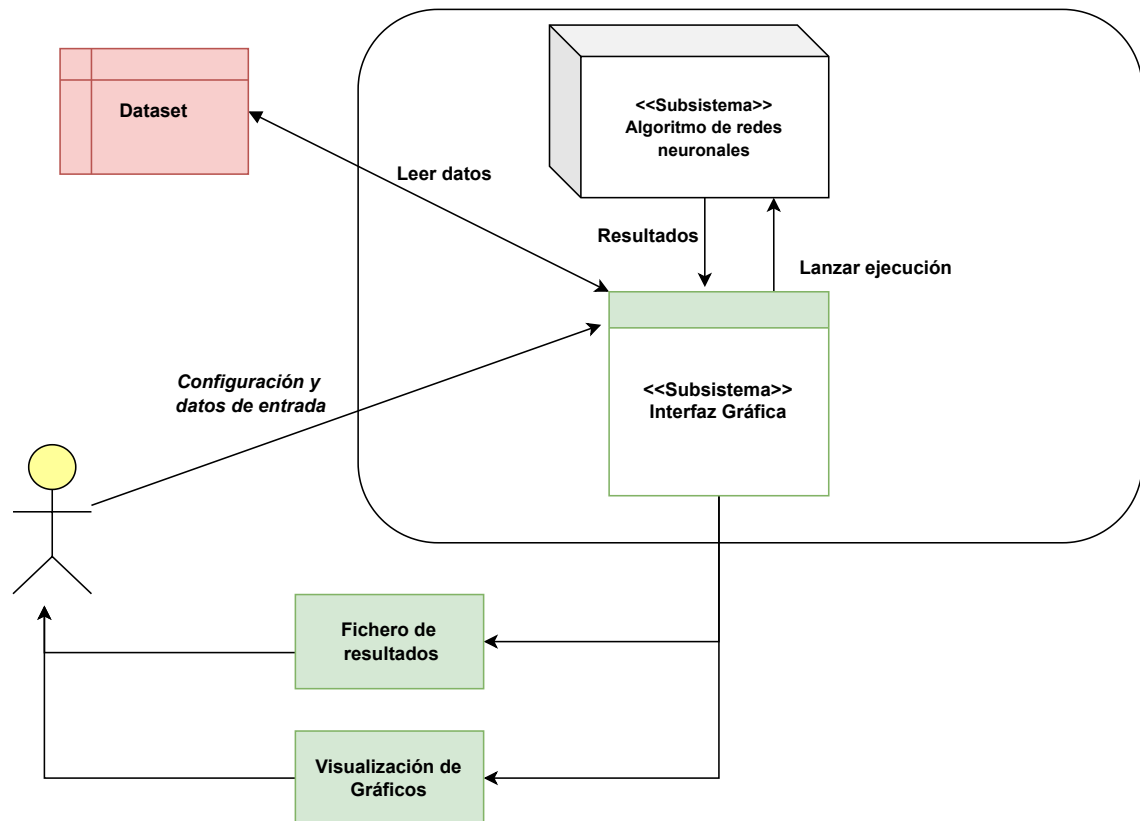


Figura 10.2: Diagrama de interacción Interfaz Gráfica-Usuario.



## 11 Diagramas de Clases

El siguiente capítulo tiene por objetivo seguir detallando el sistema de este Trabajo Fin de Grado. Dado que la implementación del código está realizado mediante el lenguaje de alto nivel **Python**, y bajo el paradigma **enfoque orientado a objetos**, se procede a especificar mediante diagramas de clases todas aquellas clases que han resultado del diseño de la aplicación.

Con el objetivo de tener una comprensión más asequible se ha dividido los diagramas de las distintas clases que componen el sistema.

### 11.1. Diagramas de Clases

Los diagramas de clases se utilizan para comprender cómo se relacionan entre sí los distintos componentes de un sistema de software y para visualizar la estructura del sistema, así como los atributos y métodos propios de una clase en particular, o las relaciones de: herencia, agregación o asociación, con el resto de clases.

Para todos los diagramas de clases que se van a exponer a continuación, se hará uso de la notación que se muestra en la Tabla 11.1.

A continuación, se muestran los diagramas de las clases implementados en el TFG.

## 11.2. Diagrama de Clases del Sistema Completo

Clase: nombre		
Descripción de la clase		
Datos		
variable1	tipo variable	descripción de la variable
...	...	
Métodos		
método1	tipo de método	descripción del método
...	...	

Tabla 11.1: Ejemplo general para la especificación de clases.

## 11.2. Diagrama de Clases del Sistema Completo

En la Figura 11.1 se puede ver el diagrama de clases del sistema completo de una forma resumida, en las secciones siguientes se mostrarán con más detalle dichos diagramas. A primera vista se concluye que existen seis clases: tres clases *Individual*, *Particle* y *Larvae* para describir a las posibles soluciones al problema, de manera singular, y tres clases *Population*, *Swarm* y *Reef* para el conjunto de las soluciones actuales de la generación.

A continuación, se describen en profundidad cada una de las clases que se han creado en el sistema.

## 11. Diagramas de Clases

---

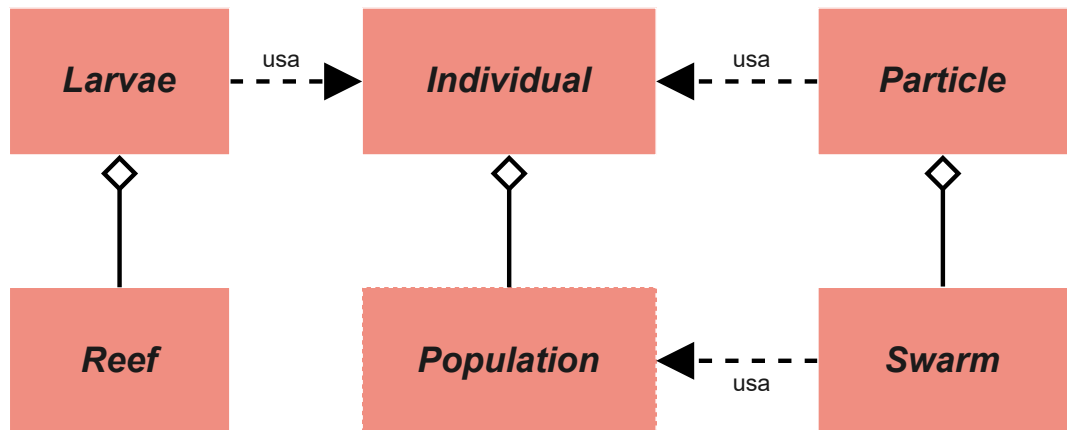


Figura 11.1: Diagrama de clases del sistema completo.



## 11.3. Clase Individual

En este apartado se procede a describir la clase '*Individual*', que va a representar una posible solución al problema del Genetic Algorithm. En la Figura 11.2 se muestra el diagrama de clases. En la Tabla 11.2 se especifica de forma concisa dicha clase.

Clase: Individual		
Esta clase contendrá los atributos propios de la solución candidata.		
Datos		
- _chromosome	array	Guarda la selección de características de la solución.
- _weights	array	Guarda los valores de los pesos de las neuronas de entradas de la solución.
- _bias	array	Guarda los valores de los sesgos de las neuronas ocultas de la solución.
- _fitness	float	Esta variable describe el rendimiento de la propia solución.
- _needs_update	bool	Esta variable describe si se ha evaluado al individuo o necesita actualizar su fitness.
Métodos		
+ Individual		Método constructor de la clase Individual.
+ get_chromosome	array	Función que devuelve el vector de características del individuo.
+ get_weights	array	Función que devuelve la matriz de pesos del individuo.
+ get_bias	array	Función que devuelve el vector de sesgos del individuo.
+ get_fitness	float	Función que devuelve el fitness del individuo.
+ get_needs_update	float	Función que devuelve si el individuo necesita actualizar su fitness o no.
+ set_chromosome	void	Función que asigna y actualiza el vector de características del individuo.

## 11. Diagramas de Clases

---

+ set_weights	void	Función que asigna y actualiza la matriz de pesos del individuo.
+ set_bias	void	Función que asigna y actualiza el vector de sesgos del individuo.
+ set_fitness	void	Función que asigna y actualiza el fitness del individuo.
+ set_needs_update	void	Función que asigna y actualiza el dato lógico referente al cálculo del fitness del individuo.
+ create_individual	Individual	Función que devuelve un objeto de la clase Individual con atributos: cromosoma, pesos y sesgos generados aleatoriamente, y fitness infinito.

Tabla 11.2: Especificación de la clase Individual.

<b><i>Individual</i></b>
<ul style="list-style-type: none"><li>- _chromosome: array</li><li>- _weights: array</li><li>- _bias: array</li><li>- _fitness: float</li><li>- _needs_update: bool</li></ul>
<ul style="list-style-type: none"><li>+ Individual()</li><li>+ get_chromosome(): array</li><li>+ get_weights(): array</li><li>+ get_bias(): array</li><li>+ get_fitness(): float</li><li>+ get_needs_update(): bool</li><li>+ set_chromosome(): void</li><li>+ set_weights(): void</li><li>+ set_bias(): void</li><li>+ set_fitness(): void</li><li>+ set_needs_update(): void</li><li>+ create_individual(): Individual</li></ul>

Figura 11.2: Diagrama de clases de la clase Individual.

### 11.4. Clase Particle

En este apartado se procede a describir la clase '*Particle*' y que va a representar una posible solución al problema basado en enjambres, Particle Swarm Optimization. En la Figura 11.3 se muestra el diagrama de clase. En la Tabla 11.3 se especifica de forma concisa dicha clase.

Clase: Particle		
Esta clase contendrá los atributos propios de la solución candidata.		
Datos		
- _chromosome	array	Guarda la selección de características de la solución.
- _weights	array	Guarda los valores de los pesos de las neuronas de entradas de la solución.
- _bias	array	Guarda los valores de los sesgos de las neuronas ocultas de la solución.
- _fitness	float	Esta variable describe el rendimiento de la propia solución.
- _needs_update	bool	Esta variable describe si se ha evaluado al individuo o necesita actualizar su fitness.
- _best_fitness	float	Esta variable guarda el mejor rendimiento histórico de la propia solución.
- _best_weights_position	array	Guarda la posición de los mejores pesos históricos de la partícula.
- _best_bias_position	array	Guarda la posición de los mejores sesgos históricos de la partícula.
- _best_chromosome_posit	array	Guarda la posición de la mejor selección de características de la partícula.
- _weights_velocity	array	Guarda la velocidad de la partícula en el espacio de pesos.
- _bias_velocity	array	Guarda la velocidad de la partícula en el espacio de sesgos.
- _chromosome_velocity	array	Guarda la velocidad de la partícula en el espacio de cromosomas.

## 11.4. Clase Particle

Métodos		
+ Partícula		Método constructor de la clase Partícula.
+ get_chromosome	array	Función que devuelve el vector de características del individuo.
+ get_weights	array	Función que devuelve la matriz de pesos del individuo.
+ get_bias	array	Función que devuelve la el vector de sesgos del individuo.
+ get_fitness	float	Función que devuelve el fitness del individuo.
+ get_needs_update	float	Función que devuelve si el individuo necesita actualizar su fitness o no.
+ get_best_fitness	float	Función que devuelve el mejor fitness de la partícula hasta el momento.
+ get_best_weights_positic	array	Función que devuelve la posición de los mejores pesos de la partícula hasta el momento.
+ get_best_bias_position	array	Función que devuelve la posición de los mejores pesos de la partícula hasta el momento.
+ get_best_chromosome_p	array	Función que devuelve la posición del mejor cromosoma de la partícula hasta el momento.
+ get_weights_velocity	array	Función que devuelve la velocidad de la partícula en el espacio de pesos.
+ get_bias_velocity	array	Función que devuelve la velocidad de la partícula en el espacio de sesgos.
+ get_chromosome_velocit	array	Función que devuelve la velocidad de la partícula en el espacio de cromosomas.
+ set_weights	void	Función que asigna y actualiza la matriz de pesos del individuo.
+ set_bias	void	Función que asigna y actualiza la el vector de sesgos del individuo.
+ set_chromosome	void	Función que asigna y actualiza el vector de características del individuo.
+ set_fitness	void	Función que asigna y actualiza el fitness del individuo.

## 11. Diagramas de Clases

---

+ set_needs_update	void	Función que asigna y actualiza el dato lógico referente al cálculo del fitness del individuo.
+ set_best_fitness	void	Función que asigna y actualiza el mejor fitness de la partícula hasta el momento.
+ set_best_weights_positio	void	Función que asigna y actualiza la posición de los mejores pesos de la partícula hasta el momento.
+ set_best_bias_position	void	Función que asigna y actualiza la posición de los mejores sesgos de la partícula hasta el momento.
+ set_best_chromosome_p	void	Función que asigna y actualiza la posición del mejor cromosoma de la partícula hasta el momento.
+ set_weights_velocity	void	Función que asigna y actualiza la velocidad de la partícula en el espacio de pesos.
+ set_bias_velocity	void	Función que asigna y actualiza la velocidad de la partícula en el espacio de sesgos.
+ set_chromosome_velocity	void	Función que asigna y actualiza la velocidad de la partícula en el espacio de cromosomas.
+ create_particle	Particle	Función que devuelve un objeto de la clase Particle con atributos: cromosoma, pesos y sesgos generados aleatoriamente, fitness infinito y velocidades iniciales a 0.

Tabla 11.3: Especificación de la clase Particle.

<b>Particle</b>
<ul style="list-style-type: none"> <li>- _chromosome: array</li> <li>- _weights: array</li> <li>- _bias: array</li> <li>- _fitness: float</li> <li>- _needs_update: bool</li> <li>- _best_fitness: float</li> <li>- _best_weights_position: array</li> <li>- _best_bias_position: array</li> <li>- _best_chromosome_position: array</li> <li>- _weights_velocity: array</li> <li>- _bias_velocity: array</li> <li>- _chromosome_velocity: array</li> </ul>
<ul style="list-style-type: none"> <li>+ Particle()</li> <li>+ get_chromosome(): array</li> <li>+ get_weights(): array</li> <li>+ get_bias(): array</li> <li>+ get_fitness(): float</li> <li>+ get_needs_update(): bool</li> <li>+ get_best_fitness(): float</li> <li>+ get_best_weights_position(): array</li> <li>+ get_best_bias_position(): array</li> <li>+ get_best_chromosome_position(): array</li> <li>+ get_weights_velocity(): array</li> <li>+ get_bias_velocity(): array</li> <li>+ get_chromosome_velocity(): array</li> <li>+ set_chromosome(): void</li> <li>+ set_weights(): void</li> <li>+ set_bias(): void</li> <li>+ set_fitness(): void</li> <li>+ set_needs_update(): void</li> <li>+ set_best_fitness(): void</li> <li>+ set_best_weights_position(): void</li> <li>+ set_best_bias_position(): void</li> <li>+ set_best_chromosome_position(): void</li> <li>+ set_weights_velocity(): void</li> <li>+ set_bias_velocity(): void</li> <li>+ set_chromosome_velocity(): void</li> <li>+ create_particle(): Particle</li> </ul>

Figura 11.3: Diagrama de clases de la clase Particle.

### 11.5. Clase Larvae

En este apartado se procede a describir la clase '*Larvae*' y que va a representar una posible solución al problema basado en reproducción de arrecifes, Coral Reef Optimization. En la Figura 11.4 se muestra el diagrama de clases. En la Tabla 11.4 se especifica de forma concisa dicha clase.

Clase: Larvae		
Esta clase contendrá los atributos propios de la solución candidata.		
Datos		
- _chromosome	array	Guarda la selección de características de la solución.
- _weights	array	Guarda los valores de los pesos de las neuronas de entradas de la solución.
- _bias	array	Guarda los valores de los sesgos de las neuronas ocultas de la solución.
- _fitness	float	Esta variable describe el rendimiento de la propia solución.
- _needs_update	bool	Esta variable describe si se ha evaluado al individuo o necesita actualizar su fitness.
- _attempts	int	Guarda el número de intentos restantes de la larva para asentarse antes de ser depredada.
Métodos		
+ Larvae		Método constructor de la clase Larvae.
+ get_chromosome	array	Función que devuelve el vector de características del individuo.
+ get_weights	array	Función que devuelve la matriz de pesos del individuo.
+ get_bias	array	Función que devuelve la el vector de sesgos del individuo.
+ get_fitness	float	Función que devuelve el fitness del individuo.



## 11.5. Clase Larvae

+	float	Función que devuelve si el individuo necesita actualizar su fitness o no.
+ get_attempts	int	Función que devuelve el número de intentos restantes de la larva para asentarse en el arrecife.
+ set_weights	void	Función que asigna y actualiza la matriz de pesos del individuo.
+ set_bias	void	Función que asigna y actualiza la el vector de sesgos del individuo.
+	void	Función que asigna y actualiza el vector de características del individuo.
+ set_fitness	void	Función que asigna y actualiza el fitness del individuo.
+	void	Función que asigna y actualiza el dato lógico referente al cálculo del fitness del individuo.
+ set_attempts	void	Función que asigna y actualiza el número de intentos de asentamiento en el arrecife.
+ create_larvae	Larvae	Función que devuelve un objeto de la clase Larvae con atributos: cromosoma, pesos y sesgos generados aleatoriamente, fitness infinito y el número de intentos de asentamiento introducidos por el usuario

Tabla 11.4: Especificación de la clase Larvae.

## 11. Diagramas de Clases

---

<b><i>Larvae</i></b>
<ul style="list-style-type: none"><li>- <code>_chromosome</code>: array</li><li>- <code>_weights</code>: array</li><li>- <code>_bias</code>: array</li><li>- <code>_fitness</code>: float</li><li>- <code>_needs_update</code>: bool</li><li>- <code>_attempts</code>: int</li></ul>
<ul style="list-style-type: none"><li>+ <code>Larvae()</code></li><li>+ <code>get_chromosome()</code>: array</li><li>+ <code>get_weights()</code>: array</li><li>+ <code>get_bias()</code>: array</li><li>+ <code>get_fitness()</code>: float</li><li>+ <code>get_needs_update()</code>: bool</li><li>+ <code>get_attempts()</code>: int</li><li>+ <code>set_chromosome()</code>: void</li><li>+ <code>set_weights()</code>: void</li><li>+ <code>set_bias()</code>: void</li><li>+ <code>set_fitness()</code>: void</li><li>+ <code>set_needs_update()</code>: void</li><li>+ <code>set_attempts()</code>: void</li><li>+ <code>create_larvae()</code>: Larvae</li></ul>

Figura 11.4: Diagrama de clases de la clase Larvae.

## 11.6. Clase Population

En este apartado se procede a describir la clase '*Population*' y que va a representar al conjunto de individuos del GA. En la Figura 11.5 se muestra el diagrama de clases. En la Tabla 11.5 se especifica de forma concisa dicha clase.

Clase: Population		
Esta clase contendrá los atributos propios del conjunto de individuos.		
Datos		
- _size	int	Guarda el valor del tamaño de la población.
- _genes_list	array	Guarda los individuos presentes en la población actual.
- _best_gene	Any	Guarda la mejor solución con mayor rendimiento del general de la población.
Métodos		
+ Population		Método constructor de la clase Population.
+ get_size	array	Función que devuelve valor del tamaño de la población.
+ get_genes_list	array	Función que devuelve la lista de individuos en la población.
+ get_best_gene	Any	Función que devuelve el mejor individuo.
+ set_genes_list	void	Función que asigna y actualiza la lista de individuos de la población.
+ set_best_gene	void	Función que asigna y actualiza el mejor individuo en la población.
+ crea- te_population	Population	Método estático que devuelve un objeto de la clase Population con una población de individuos generados aleatoriamente y el tamaño dado.

## 11. Diagramas de Clases

---

+		void	Método de la clase que dado un individuo y su posición, introduce al individuo en dicha posición dentro de la lista de soluciones de la población.
+	in-	void	Método de la clase que actualiza el atributo de mejor individuo y la posición 0 de la lista de genes a partir de un individuo dado.
	sert_best_gene		

Tabla 11.5: Especificación de la clase Population.

<b><i>Population</i></b>
- _size: int - _genes_list: array - _best_gene: Any
+ Population() + get_size(): int + get_genes_list(): array + get_best_gene(): Any + set_genes_list(): void + set_best_gene: void + create_population(): Population + add_gene_to_list_at_index(): void + insert_best_gene(): void

Figura 11.5: Diagrama de clases de la clase Population.

## 11.7. Clase Swarm

En este apartado se procede a describir la clase '*Swarm*' que va a representar al enjambre de partículas del algoritmo PSO. En la Figura 11.6 se muestra el diagrama de clases. En la Tabla 11.6 se especifica de forma concisa dicha clase.

Clase: Swarm		
Esta clase contendrá los atributos propios del enjambre de partículas.		
Datos		
- _size	int	Guarda el valor del tamaño de la población.
- _genes_list	array	Guarda las partículas actuales en el enjambre.
- _best_gene	Any	Guarda la partícula con mejor rendimiento del enjambre.
- _global_best_fitness	float	Guarda la mejor solución con mayor rendimiento del general de la población.
- _global_best_weight	float	Guarda la mejor solución con mayor rendimiento del general de la población.
- _global_best_bias	float	Guarda la mejor solución con mayor rendimiento del general de la población.
- _global_best_chron	float	Guarda la mejor solución con mayor rendimiento del general de la población.
Métodos		
+ Swarm		Método constructor de la clase Swarm.
+ get_size	int	Función que devuelve el valor de tamaño del enjambre.
+ get_genes_list	array	Función que devuelve la lista de partículas actuales en el enjambre.
+ get_best_gene	Any	Función que devuelve la mejor partícula.
+ get_global_best_fit	float	Función que devuelve el mejor fitness del enjambre.

## 11. Diagramas de Clases

---

+	array	Función que devuelve la mejor matriz de pesos del enjambre.
+	array	Función que devuelve el mejor vector de sesgos del enjambre
+	array	Función que devuelve el mejor vector de características del enjambre.
+	void	Función que asigna y actualiza la lista de partículas en el enjambre.
+	void	Función que asigna y actualiza la mejor partícula.
+	void	Función que actualiza y asigna el mejor fitness del enjambre.
+	void	Función que actualiza y asigna la mejor matriz de pesos del enjambre.
+	void	Función que actualiza y asigna el mejor vector de sesgos del enjambre
+	void	Función que actualiza y asigna el mejor vector de características del enjambre.
+	Swarm	Método estático que devuelve un objeto de la clase Swarm con un enjambre de partículas generadas aleatoriamente y el tamaño dado.

Tabla 11.6: Especificación de la clase Swarm.

<b>Swarm</b>
<ul style="list-style-type: none"><li>- _size: int</li><li>- _genes_list: array</li><li>- _best_gene: Particle</li><li>- _global_best_fitness: float</li><li>- _global_best_weights: array</li><li>- _global_best_bias: array</li><li>- _global_best_chromosome: array</li></ul>
<ul style="list-style-type: none"><li>+ Swarm():</li><li>+ get_size(): int</li><li>+ get_genes_list(): array</li><li>+ get_best_gene(): Particle</li><li>+ get_global_best_fitness(): float</li><li>+ get_global_best_weights(): array</li><li>+ get_global_best_bias(): array</li><li>+ get_global_best_chromosome(): array</li><li>+ set_genes_list(): void</li><li>+ set_best_gene: void</li><li>+ set_global_best_fitness(): void</li><li>+ set_global_best_weights(): void</li><li>+ set_global_best_bias(): void</li><li>+ set_global_best_chromosome(): void</li><li>+ create_swarm(): Swarm</li></ul>

Figura 11.6: Diagrama de clases de la clase Swarm.

## 11. Diagramas de Clases

### 11.8. Clase Reef

En este apartado se procede a describir la clase '*Reef*' que va a representar al arrecife de corales en nuestro algoritmo CRO. En la Figura 11.7 se muestra el diagrama de clases. En la Tabla 11.7 se especifica de forma concisa dicha clase.

Clase: Reef		
Esta clase contendrá los atributos propios del arrecife.		
Datos		
- _size	int	Guarda el valor del tamaño del arrecife.
- _free_occupied_ratio	float	Guarda el valor de la fracción de colonias ocupadas en el arrecife por corales.
- _corals_list	array	Guarda los corales actuales en el arrecife.
- _best_coral	Any	Guarda el coral con mejor rendimiento del arrecife.
- _sorted_indexes	array	Guarda una lista ordenada de los índices de los corales en la población.
- _larvae_attempts	float	Guarda el número máximo de intentos que tiene cada larva para asentarse antes de ser depredadas.
Métodos		
+ Reef		Método constructor de la clase Reef.
+ get_size	int	Función que devuelve el valor de tamaño del arrecife.
+ get_free_occupied_ratio	float	Función que devuelve el valor de la fracción de colonias ocupadas en el arrecife.
+ get_corals_list	array	Función que devuelve la lista de corales actuales en el arrecife.
+ get_best_coral	Any	Función que devuelve el mejor coral del arrecife.
+ get_sorted_indexes	array	Función que devuelve la lista ordenada de índices de corales en la población.



## 11.8. Clase Reef

+		float	Función que devuelve el valor de intentos de asentamiento que tiene cada larva.
+	get_larvae_attempts		
+	set_corals_list	array	Función que asigna y actualiza la lista de corales actuales en el arrecife.
+	set_best_coral	Any	Función que asigna y actualiza el mejor coral del arrecife
+	create_reef	Reef	Método estático que devuelve un objeto de la clase Reef con un arrecife con corales generados aleatoriamente en su fracción de colonias ocupadas.
+	insert_new_larvae_in	in-void	Método de la clase que dada una larva y una posición, introduce en la posición dada de la lista de corales a dicha larva.
+	remove_coral_from_hole	remo-void	Método de la clase que dada una posición, elimina de la lista de corales al coral que se encuentre en esa posición.
+	sort_by_fitness	void	Método de la clase que ordena la lista de índices de los corales según el fitness de los corales existentes en la lista de corales.

Tabla 11.7: Especificación de la clase Reef.

## 11. Diagramas de Clases

---

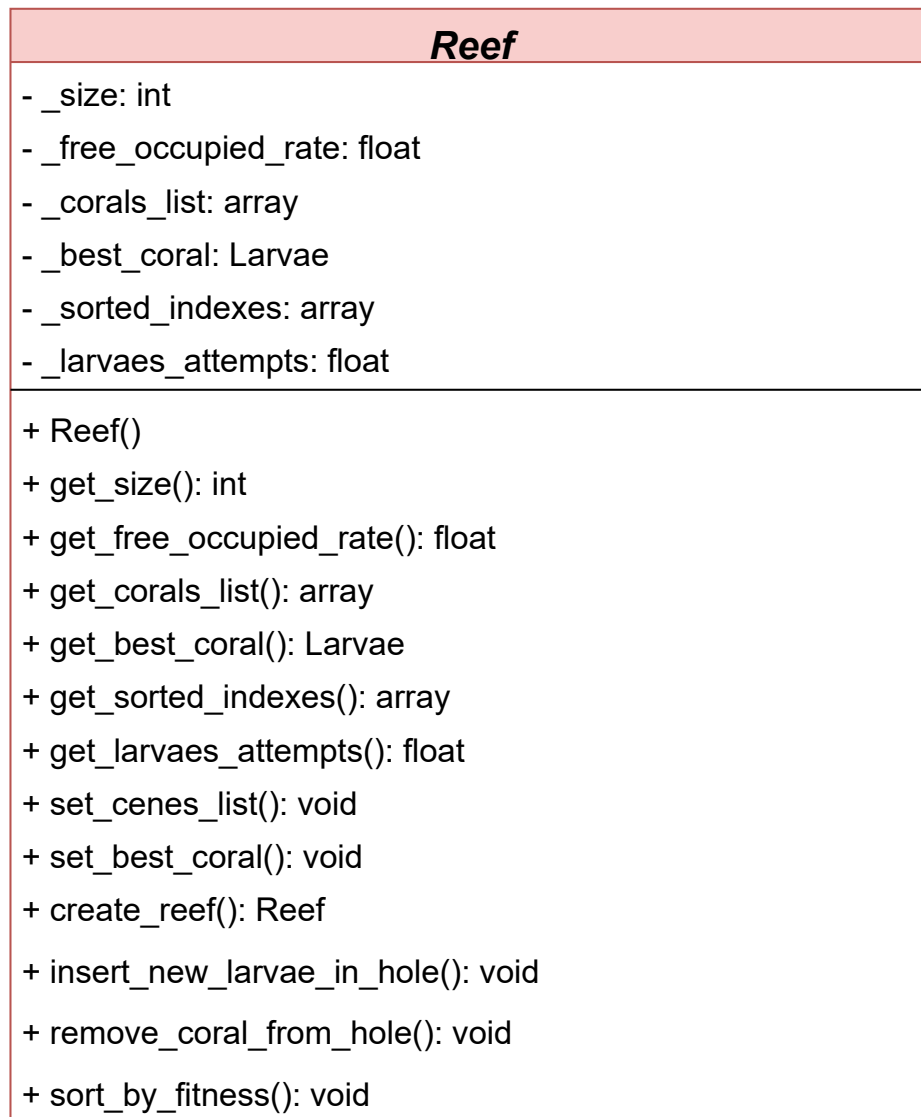


Figura 11.7: Diagrama de clases de la clase Reef.





## 12 Diseño de la Interfaz Gráfica

En el siguiente capítulo se expondrán las opciones, configuraciones y eventos que tendrá la interfaz gráfica de presente Trabajo Fin de Grado. Para el prototipo de la interfaz se hará uso de *diagrams.net*, que es un software de código abierto para dibujo gráfico multiplataforma gratuito.

En la Sección 8.3 se adelantaron los objetivos generales y varias de las características con las que contaría la interfaz de usuario. De nuevo, se muestra la Figura 12.1 con el esquema que ha servido de base para la implementación de la interfaz.

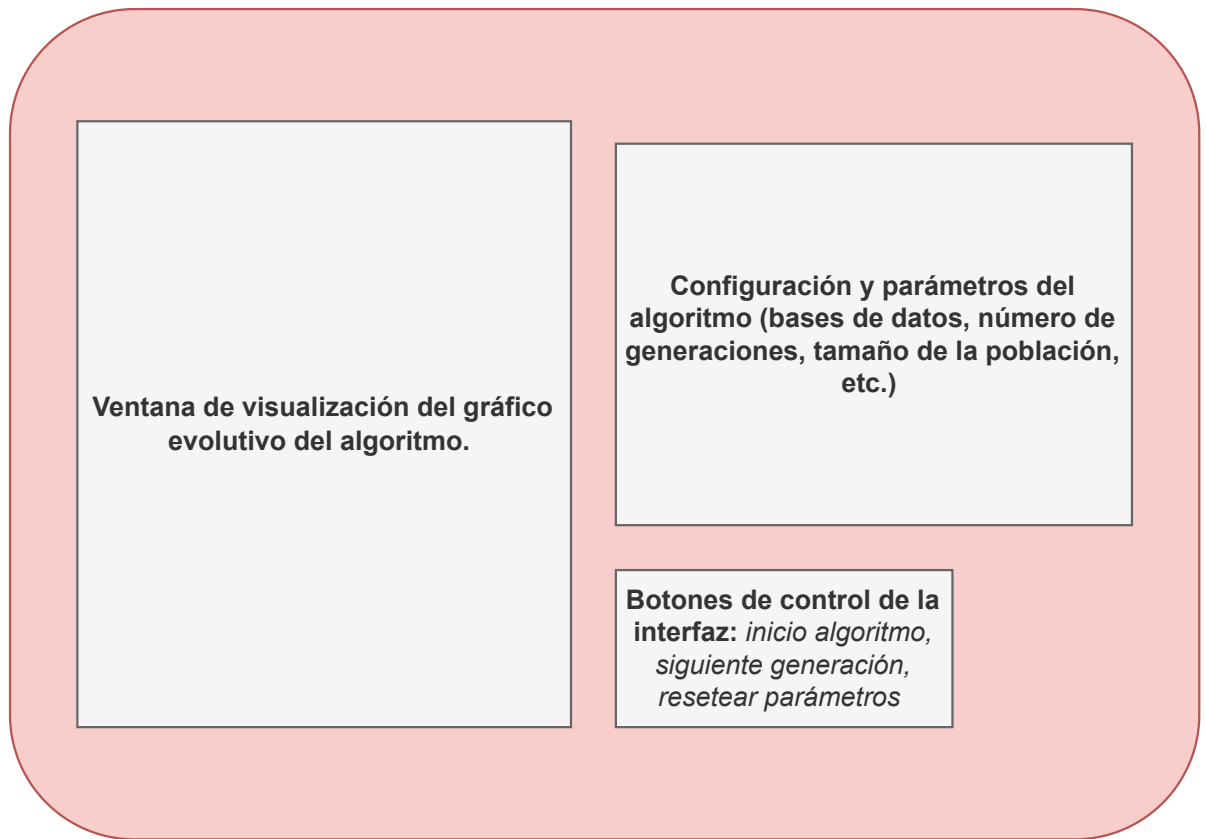


Figura 12.1: Secciones de la interfaz gráfica.

### 12.1. Estructura de la Interfaz Gráfica

En esta sección se muestra el prototipo diseñado de la interfaz gráfica, la cual tendrá un estructura semejante a la de la Figura 12.2.

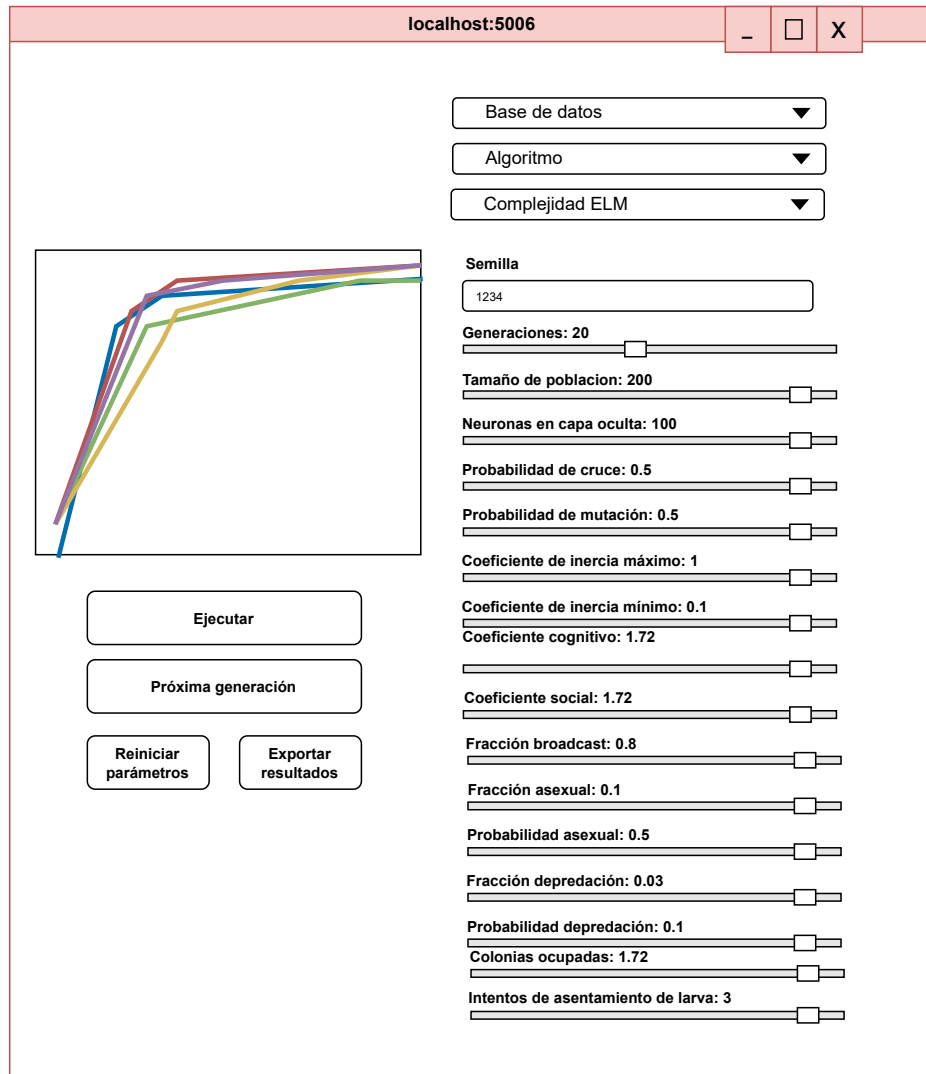


Figura 12.2: Estructura de la interfaz de usuario.

## 12.1. Estructura de la Interfaz Gráfica

---

Como se puede observar, a grandes niveles se puede dividir la interfaz gráfica en tres grandes secciones:

- **Parámetros del algoritmo.** Esta sección ocupa toda la mitad derecha de la ventana.
- **Gráfico Interactivo de rendimiento de la población.** Esta sección ocupa la esquina superior izquierda de la ventana.
- **Botones de eventos.** Esta sección ocupa la esquina izquierda inferior de la ventana.

### 12.1.1. Sección de Parámetros del Algoritmo

Aquí se engloban todos los datos y parámetros que el usuario puede modificar para obtener distintos resultados en los experimentos. Es importante remarcar que dependiendo del algoritmo seleccionado, los parámetros que aparecen para modificar son distintos dado que se requieren de distintos hiperparámetros para la ejecución de cada uno. Estos elementos configurables son:

- **Bases de datos:** Consiste en un desplegable en el que se puede escoger la base de datos de prueba entre una lista de ficheros *.csv*.
- **Algoritmo:** Consiste en un desplegable para seleccionar el algoritmo bioinspirado encargado de realizar el entrenamiento de las redes ELM.. Los algoritmos posibles son:
  - GA-ELM.
  - PSO-ELM.
  - CRO-ELM.
- **Semilla:** Consiste en una entrada de texto numérica para determinar el valor de la semilla del generador de números aleatorios.

## 12. Diseño de la Interfaz Gráfica

---

- **Complejidad:** Consiste en un desplegable para determinar el valor del hiperparámetro **C** del modelo.
- **Probabilidad de mutación:** Opción para fijar la probabilidad con la que cada individuo va a ser mutado.
- **Número de generaciones:** Consiste en un control deslizante para escoger el número de generaciones del algoritmo bioinspirado.
- **Tamaño de la población:** Consiste en un control deslizante para escoger el número de individuos dentro de la población.
- **Neuronas en capa oculta:** Consiste en un control deslizante para escoger el valor del hiperparámetro **D** del modelo.
- **Probabilidad de cruce:** Consiste en un control deslizante para escoger el valor de la probabilidad de cruce. Este parámetro solo es visible si se escoge el algoritmo GA-ELM.
- **Probabilidad de mutación:** Consiste en un control deslizante para escoger el valor de la probabilidad de mutación. Este parámetro solo es visible si se selecciona el algoritmo GA-ELM.
- **Coeficiente de inercia máximo  $\omega_{max}$ :** Consiste en un control deslizante para escoger el valor máximo del coeficiente de inercia de las partículas. Este parámetro solo es visible si se elige el algoritmo PSO-ELM.
- **Coeficiente de inercia mínimo  $\omega_{min}$ :** Consiste en un control deslizante para escoger el valor máximo del coeficiente de inercia de las partículas. Este parámetro solo es visible si se selecciona el algoritmo PSO-ELM.
- **Coeficiente cognitivo  $c_1$ :** Consiste en un control deslizante para escoger el valor de coeficiente cognitivo de las partículas. Este parámetro solo es visible si se escoge el algoritmo PSO-ELM.



## 12.1. Estructura de la Interfaz Gráfica

---

- **Coefficiente social  $c_2$** : Consiste en un control deslizante para escoger el valor de coeficiente cognitivo de las partículas. Este parámetro solo es visible si se selecciona el algoritmo PSO-ELM.
- **Fracción broadcast  $F_b$** : Consiste en un control deslizante para escoger el valor de fracción del arrecife que va a llevar a cabo reproducción externa. Este parámetro solo es visible si se escoge el algoritmo CRO-ELM.
- **Fracción asexual  $F_a$** : Consiste en un control deslizante para escoger el valor de fracción del arrecife que va a llevar a cabo reproducción asexual en caso de que se cumpla su probabilidad. Este parámetro solo es visible si se elige el algoritmo CRO-ELM.
- **Fracción depredación  $F_d$** : Consiste en un control deslizante para escoger el valor de fracción del arrecife que va a ser eliminada del arrecife si se cumple su probabilidad. Este parámetro solo es visible si se escoge el algoritmo CRO-ELM.
- **Probabilidad asexual  $P_a$** : Consiste en un control deslizante para escoger el valor de probabilidad para que los corales se reproduzcan asexualmente. Este parámetro solo es visible si se elige el algoritmo CRO-ELM.
- **Probabilidad de depredación  $P_d$** : Consiste en un control deslizante para escoger el valor de probabilidad por el que se va a determinar si una larva es depredada o sigue viva. Este parámetro solo es visible si se selecciona el algoritmo CRO-ELM.
- **Tasa de colonias ocupadas  $\rho_0$** : Consiste en un control deslizante para escoger el valor de la tasa de ocupación del arrecife. Este parámetro solo es visible si se escoge el algoritmo CRO-ELM.
- **Intentos de asentamiento de la larva  $\eta$** : Consiste en un control deslizante para escoger el número de intentos de asentamiento previos a la eliminación de la larva. Este parámetro solo es visible si se elige el algoritmo CRO-ELM.

## 12. Diseño de la Interfaz Gráfica

---

### 12.1.2. Sección de Gráfico Interactivo

En la parte superior izquierda de la GUI se encuentra un gráfico interactivo que describe la evolución del rendimiento del algoritmo durante las generaciones.

En el eje Y aparece representado el valor de fitness de los individuos, en el intervalo  $[0,1]$ , mientras que en el eje X se marca el número de generaciones. De esta manera, la lectura que se debe realizar del gráfico es cómo mejoran los individuos a lo largo de la fase de entrenamiento, a medida que pasan las generaciones.

### 12.1.3. Sección de Eventos

Esta sección se encuentra inmediatamente debajo del gráfico interactivo, y consta de los botones que llevan a cabo las diferentes funcionalidades de la interfaz gráfica:

- **Ejecutar algoritmo:** Botón para ejecutar el algoritmo de manera completa con las configuraciones dadas en la Sección 12.1.1.
- **Siguiente generación:** Botón para ejecutar una generación del algoritmo con las configuraciones establecidas.
- **Reiniciar parámetros:** Botón para ajustar todos los parámetros a su valor por defecto.
- **Exportar resultados:** Botón para exportar los datos de salida del experimento a un fichero *.xlsx*.

## 12.2. Diagrama de Secuencia de la Interfaz

Para entender el flujo de información y la interacción usuario-sistema-algoritmo de una forma general, incluimos un diagrama de

## 12.2. Diagrama de Secuencia de la Interfaz

secuencia en la Figura 12.3.

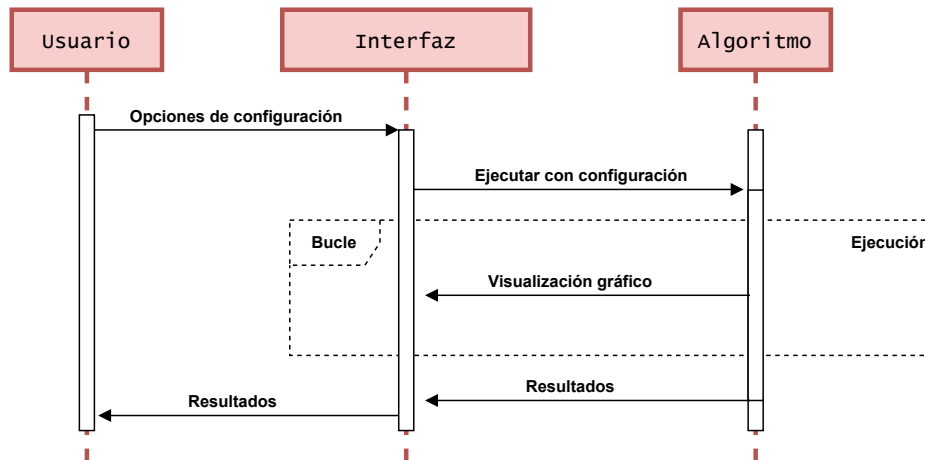


Figura 12.3: Diagrama de secuencia de la interacción usuario-sistema-algoritmo.

Como se observa en la Figura 12.3, el diagrama cumple con los siguientes pasos:

1. El usuario introduce por medio de la interfaz la configuración de los parámetros del experimento.
2. Tras presionarse el botón de 'Ejecutar algoritmo' se lleva a cabo la ejecución de la aplicación a través de todas las generaciones y se va mostrando de manera cíclica el gráfico de rendimiento del algoritmo. En caso de presionar el botón de 'Siguiete generación', solo se avanza en una unidad el número de generación y se actualiza el gráfico con los resultados de dicha iteración.
3. Una vez finaliza el bucle algorítmico y se han visualizado todos los resultados de manera gráfica, se muestra un resumen de los resultados del experimento en la interfaz.
4. En el caso final de que el usuario presione el botón de 'Exportar resultados', la herramienta descargará los archivos en formato hoja de cálculo para que el usuario lo pueda abrir y compartir.

## Parte IV

## Pruebas





## 13 Pruebas

En el siguiente capítulo se describe el conjunto de pruebas aplicadas en el sistema que aseguran la funcionalidad y calidad del código generalmente, no solo del algoritmo si no de la funcionalidad de los métodos.

Existen multitud de prácticas dentro de la Ingeniería del Software, y un gran campo de esta rama son las pruebas. En cualquier desarrollo software donde se pretenda contar con cierto sello de calidad y asegurar el correcto funcionamiento del mismo al cliente y al usuario final, debe contar con una fase de pruebas.

Supone tanta importancia este apartado, que existen muchos enfoques los cuales basan el desarrollo en las pruebas, ya que si se lleva a cabo con éxito esta fase, se podrán detectar con mucha antelación los errores, y probar el sistema de forma iterativa e incremental, sin romper con el código que se haya escrito anteriormente. Estos últimos ejemplos, desembocarían en un retraso del desarrollo y un elevado coste al proyecto.

Debido a que en este trabajo se han desarrollado varios algoritmos y clases, cada una implementando métodos propios, se debe asegurar que todo funciona de manera individual, y posteriormente, una vez se haya integrado todo el sistema.

Teniendo claro la gran influencia de las pruebas en un proyecto software, se procede a mostrar los tipos de pruebas que se han llevado a cabo y su aplicación en el presente sistema en particular.

### 13.1. Estrategia de Pruebas

El objetivo es acabar con un sistema de pruebas completo, que abarque todos los componentes del Trabajo Fin de Grado y sea capaz de validar todas las partes del proceso explicado en la Sección 7.1.

Este sistema no solo debe validar el correcto funcionamiento de todas las partes del código, sino que debe detectar los errores que se producen durante el desarrollo, antes de dar con la implementación final de cada parte.

Para tal fin se han llevado a cabo una series de pruebas que reciben el nombre de *pruebas de caja negra* y *pruebas de caja blanca*. En cada mejora del código, se ejecutaban dichas pruebas y en el caso negativo de que éstas reportasen algún tipo de error, se volvía a hacer un repaso únicamente de aquellas partes que estuviesen afectadas por la última mejora hasta dar con la respuesta al problema.

A continuación, se detallan el conjunto de casos de pruebas que se han realizado.

### 13.2. Pruebas Unitarias

Esta sección de pruebas unitarias hace referencia a aquellas pruebas de software que se encargan de asegurar que cada unidad o componente de un sistema funciona correctamente. Cada unidad se refiere a una porción individual de código, ya sea una función o un método de una clase.

La idea detrás de las pruebas unitarias consiste en declarar una serie

## 13. Pruebas

---

de pruebas automatizadas para cada unidad de código, con el objetivo de detectar errores en una fase temprana del proceso de desarrollo. Esto permite a los desarrolladores detectar y corregir problemas antes de que el sistema se integre y se despliegue, ahorrando tiempo y esfuerzo en el proceso de pruebas y depuración.

Al comienzo del capítulo se ha adelantado que se han llevado a cabo dos tipos de pruebas: las pruebas estructurales o *pruebas de caja blanca* y las pruebas funcionales o *pruebas de caja negra*.

### 13.2.1. Pruebas de Caja Blanca

Este tipo de pruebas se enfocan en el código interno de una unidad o componente del sistema. Se utilizan para asegurar que el código está funcionando correctamente y cumpliendo con los requisitos especificados.

En las pruebas de caja blanca, se examina el comportamiento de cada unidad de código en detalle, utilizando técnicas como el análisis estructural del código, el análisis de flujo de datos y la verificación de las condiciones lógicas. Esto permite identificar posibles problemas de rendimiento, problemas de corrección o brechas de seguridad en el código.

Una de las ventajas de las pruebas de caja blanca es que permiten detectar problemas en el código que son difíciles de captar a simple vista. También pueden ayudar a garantizar que el código esté cumpliendo con las especificaciones y los estándares de calidad.

Con los resultados de estas pruebas ha sido posible detectar y solucionar errores a lo largo de la fase de codificación, especialmente durante el desarrollo de los diferentes métodos de selección y operadores genéticos de los algoritmos bioinspirados, ya que son comunes a todos los algoritmos desarrollados y era de vital importancia su correcto funcionamiento.

De este modo, a medida que se han ido implementando las distintas



rutinas de ejecución del sistema, se iba comprobando tanto la eficiencia como la estructura para asegurar que no se producía ningún tipo de error.

Para realizar estas pruebas de caja blanca se ha tenido en cuenta lo siguiente:

1. Se ha verificado que en cada módulo se realizaban las operaciones genéticas y posteriormente se almacenaba dicha información genética en las estructuras internas de los objetos de manera correcta.
2. Se ha asegurado la ejecución de cada una de las sentencias que formaban parte del módulo depurando la ejecución completa.
3. Se ha verificado que el flujo de información para todas las ramas posibles del módulo se realiza correctamente.
4. Se han estudiado los límites de los bucles y realizado pruebas en ellos.

### 13.2.2. Pruebas de Caja Negra

Las pruebas de caja negra son una técnica de pruebas de software que se enfocan en el comportamiento externo de una unidad o componente del sistema, sin tener en cuenta su código interno. El objetivo principal de estas pruebas es asegurar que el sistema cumple con los requisitos funcionales especificados.

En las pruebas de caja negra, se proporciona una entrada específica al sistema y se observa su comportamiento y salida sin preocuparse de cómo está codificado el módulo por dentro. Principalmente se utiliza este tipo de pruebas para probar el sistema desde una perspectiva del usuario final, asegurando que todo está funcionando correctamente y cumpliendo con los requisitos especificados.

Una de las ventajas de las pruebas de caja negra es que son fáciles de implementar y no requieren conocimiento detallado del código interno

## 13. Pruebas

---

del sistema.

Cada uno de los requisitos especificados en el Capítulo 8 va a estar cubierto por una prueba de caja negra que corrobore su éxito en la implementación.

Las estrategias de caja negra realizadas han sido:

1. Se estudió los valores óptimos de los hiperparámetros de entrada para cada algoritmo bioinspirado y de la red Extreme Learning Machine, con especial énfasis en las condiciones límites.
2. Se llevaron a cabo programas de prueba con una menor carga de datos para validar el caso de prueba, el funcionamiento del módulo y la impresión de resultados.
3. Se validaron las respuestas del módulo ante ciertos datos de entrada y condiciones, para los cuales se conocía la salida esperada.

Como estrategia seguida cabe destacar la prueba del algoritmo con distintas bases de datos y distintas configuraciones, comprobando que los datos sean correctos en todo momento. Además se han probado todas las funciones relacionadas con la lectura y escritura de ficheros, la correcta representación de las gráficas, etc.

Con los resultados de las pruebas de caja negra en nuestro TFG se ha podido reproducir y dar cobertura a los siguientes errores:

- Al llevar a cabo la actualización de las posiciones y velocidades del algoritmo PSO. El problema no llegaba a ningún tipo de solución con buen *fitness* debido a que los coeficientes de inercia y cognitivo, tenían valores por encima de su umbral de rendimiento. Esto hizo que todas las partículas diesen saltos muy grandes entre cada iteración de la búsqueda y no convergiese. La solución que se adoptó fue imponer un rango de valores  $[0,1)$  para el coeficiente de inercia, además de una función de amortiguación para dicho parámetro.

- Uno de los problemas más grave y cuya solución ha ahorrado mucho tiempo en el TFG es en la validación de la carga de los datasets al algoritmo. Entre las bases de datos iniciales, varias de ellas tenían un formato incorrecto y provocaban una excepción que interrumpía el sistema por completo y echaba a perder todos los resultados computados hasta el momento. Finalmente se optó por eliminar aquellos ficheros .csv cuyos datos no se podían extraer correctamente.

## 13.3. Pruebas de Integración

Tras haber completado las pruebas unitarias y antes de seguir con las pruebas del sistema y de aceptación, se han realizar las pruebas de integración.

Las pruebas de integración se utilizan para verificar la interacción y la comunicación entre diferentes módulos o componentes de una aplicación. Éstas tienen como objetivo asegurar que los diferentes componentes de nuestra aplicación funcionan correctamente juntos, y que no hay problemas de compatibilidad o conflictos entre ellos.

Especialmente estas pruebas cobran importancia en proyectos de alta magnitud como el abordado ya que se disponen muchos módulos interconectados.

En el caso particular de este TFG, las pruebas se han hecho de forma ascendente, probando a integrar uno a uno los módulos que habían sido probados y validados individualmente en las pruebas unitarias, y observando que la integración no provoca un fallo en la ejecución. Primero se comenzó con los módulos de más bajo nivel y se fue escalando hasta tener integrado el sistema al completo.

Así estas pruebas se han realizado de la siguiente forma:

1. Se ha comprobado en cada paso incremental que la interacción

## 13. Pruebas

---

entre el modulo añadido con el conjunto de módulos ya probados funciona correctamente.

2. Se ha comprobado que el flujo de información se realiza correctamente entre el nuevo módulo y los diferentes módulos ya integrados.
3. Se ha comprobado que la ejecución de todas las ramas posibles del conjunto de módulos del sistema hasta el punto alcanzado sea correcta.

Como el desarrollo de una interfaz gráfica que ejecuta los diferentes algoritmos del sistema y tiene un flujo de datos de configuración para la entrada, y otros datos de resultado como salida, se ha considerado oportuno validar mediante una prueba de integración el correcto funcionamiento e interacción de la misma.

### 13.4. Pruebas del Sistema

Llegados a este punto, se presenta un sistema funcional, integrado y robusto ante los errores. Es por ello que una vez se han completado todas las pruebas anteriores se deben realizar pruebas sobre sistema.

Estas pruebas son las últimas que se realizan antes de presentarle el sistema al cliente o usuario final y tienen como objetivo asegurar que el sistema cumpla con los requisitos funcionales y no funcionales especificados.

Las pruebas del sistema se realizan en un entorno de pruebas controlado que se asemeje lo máximo posible al entorno de producción. Esto permite evaluar el comportamiento que tendrá el sistema en condiciones similares a las que se utilizará en la vida real.

Algunos de los tipos de pruebas que se ejecutan sobre el sistema son:

- ***Pruebas de rendimiento:*** evalúan el rendimiento del sistema en términos de velocidad, escalabilidad y capacidad.
- ***Pruebas de seguridad:*** evalúan la capacidad del sistema para protegerse contra amenazas externas y para cumplir con los estándares de seguridad.
- ***Pruebas de recuperación ante desastres:*** evalúan la capacidad del sistema para recuperarse de fallos o problemas inesperados.

En el próximo Capítulo 14 se mostrará el desarrollo de estas pruebas, analizando por un lado que el sistema cumple con todos los objetivos planteados en la Sección 8.3, y por otro lado, que se han superado dichas pruebas.

## 13.5. Pruebas de Aceptación

Por último, con las pruebas de aceptación se vuelve a verificar que el sistema cumple con las necesidades y expectativas del cliente o usuario final. Estas pruebas se realizan en un entorno de pruebas que se asemeja al entorno de producción, al igual que hacíamos con las del sistema, pero con la gran diferencia de que en estas existe una participación activa del cliente o usuario final.

Existen varios tipos de pruebas de aceptación, aunque las principales son:

- ***Pruebas de aceptación del usuario (UAT):*** se llevan a cabo con la participación de los usuarios finales para evaluar si el sistema cumple con los requisitos especificados y si está listo para su lanzamiento, además de evaluar diferentes puntos del sistemas, tales como su facilidad de uso o fiabilidad.

### 13. Pruebas

---

- ***Pruebas de aceptación operacional (OAT)***: son realizadas por el equipo de operaciones y se enfocan en la capacidad del sistema para operar en un entorno de producción y pueden incluir pruebas de rendimiento, seguridad, disponibilidad y escalabilidad. Además, también se evalúa la capacidad del sistema para integrarse con otras aplicaciones y sistemas existentes en el entorno de producción.

Es relevante mencionar que estas pruebas son realizadas con posterioridad al desarrollo, debido a que no se considera como presentable un sistema que no haya pasado exitosamente el apartado de pruebas de integración.

Por muy objetivo que se pretenda ser y haber tenido en cuenta todos los *edge-cases* del sistema, la realidad no es otra que lo contrario. Al haber formado parte del diseño y desarrollo del sistema, y habiendo estado implicado desde el inicio del mismo, no se puede simular auténticamente la experiencia y los comportamientos que adoptará un usuario que se encuentre por primera vez frente a nuestro conjunto de módulos.

Es por eso que aún habiendo realizado tantas pruebas y de distintos tipos, en esta etapa surgen errores jamás antes vistos y que se han corregir antes de publicar nada.





## 14 Resultados Experimentales

En el siguiente capítulo se pretende poner a prueba el sistema, en términos de rendimiento, y realizar un estudio comparativo de los diferentes algoritmos bioinspirados aplicados al entrenamiento de redes *Extreme Learning Machine*, frente a la propuesta de algoritmo de este Trabajo Fin de Grado.

La necesidad de estas pruebas es obvia por varias razones:

1. Se busca verificar que la aplicación funciona a nivel general y no sufre de fallos ante la entrada de distintas configuraciones de parámetros.
2. Se busca validar el rendimiento del algoritmo propuesto, en pos de determinar si supone una mejora a lo ya conocido.
3. Se busca conocer aquellas bases de datos en las que nuestro algoritmo tiene una alta capacidad predictiva.



### 14.1. Condiciones de los Experimentos

Las pruebas experimentales se han llevado a cabo empleando bases de datos cuyas instancias aparecen etiquetadas y son de uso público. A su vez, se ha hecho uso de unos parámetros óptimos [Duran-Rosal et al. \[2018\]](#), [Salcedo-Sanz et al. \[2014b\]](#) para cada tipo de algoritmo.

#### 14.1.1. Bases de datos

Las bases de datos empleadas en los experimentos se muestran en la Tabla 14.1. En ella aparecen las bases de datos ordenadas de forma alfabética. Se incluye la forma (número de patrones, número de características), el número de clases y la distribución de los patrones en cada una de las clases. Como se puede observar, se ha testado el algoritmo en bases de datos que van desde las 2 hasta las 9 clases, y en las que existe un perfecto balanceo o están totalmente desbalanceadas. El propósito es comprobar la efectividad del algoritmo propuesto en bases de datos de distinta naturaleza.

## 14. Resultados Experimentales

Bases de datos			
Dataset	Forma	Clases	Distribución
breast-cancer-wisconsin-diagnostic	(569, 31)	2	(357, 212)
breast-cancer-wisconsin-prognostic	(194, 33)	2	(148, 46)
breast-cancer	(286, 40)	2	(218, 68)
chess-king-rook-vs-king-pawn	(3196, 39)	2	(1527, 1669)
cnae-9	(1080, 857)	9	(120, ..., 120)
electrical-grid	(10000, 13)	2	(3620, 6380)
fertility	(100, 10)	2	(88, 12)
haberman-survival	(306, 4)	2	(225, 81)
hayes-roth	(132, 5)	3	(51, 51, 30)
hepatitis	(80, 20)	2	(13, 67)
horse-colic	(300, 122)	2	(191, 109)
indian-liver-patient	(579, 11)	2	(414, 165)
lung-cancer	(32, 147)	3	(9, 13, 10)
mammographic-mass	(878, 15)	2	(461, 417)
monks-problems-2	(432, 7)	2	(290, 142)
mushroom	(8124, 112)	2	(4208, 3916)
ozone-level-detection-one	(1848, 73)	2	(1791, 57)
planning-relax	(182, 13)	2	(130, 52)
qsar-biodegradation	(1055, 42)	2	(699, 356)
soybean-small	(47, 46)	4	(10, 10, 10, 17)
spambase	(4601, 58)	2	(2788, 1813)
spect-heart	(80, 23)	2	(40, 40)
statlog-project-german-credit	(1000, 60)	2	(700, 300)
thoracic-surgery	(470, 28)	2	(400, 70)
thyroid-disease-allhyper	(2800, 28)	4	(8, 7, 62, 2723)
thyroid-disease-allrep	(2800, 28)	4	(2713, 23, 29, 35)
thyroid-disease-dis	(2028, 29)	2	(39, 1989)
thyroid-disease-sick-euthyroid	(3163, 21)	2	(2870, 293)
tic-tac-toe-endgame	(958, 28)	2	(332, 626)
weight-lifting-exercises	(4024, 55)	5	(1365, 901, 112, 276, 1370)

Tabla 14.1: Bases de datos empleadas en los experimentos.

### 14.1.2. Parámetros de los Experimentos

Los parámetros utilizados en los experimentos han sido previamente justificados y validados [Duran-Rosal et al. \[2018\]](#), [Salcedo-Sanz et al. \[2014b\]](#). De esta forma, todos los algoritmos tienen la misma ventaja competitiva. Concretamente, la configuración de los parámetros es la siguiente:

#### General

- Número de generaciones: 100.
- Tamaño de la población: 200
- Neuronas en capa oculta: 50.
- Complejidad: 0.1.

#### Algoritmo Genético

- Probabilidad de cruce: 0.8.
- Probabilidad de mutación: 0.2.

#### Particle Swarm Optimization

- Coeficiente de inercia máximo  $\omega_{max}$ : 0.72.
- Coeficiente de inercia mínimo  $\omega_{min}$ : 0.1.
- Coeficiente cognitivo  $c_1$ : 1.49.
- Coeficiente social  $c_2$ : 1.49.

#### Coral Reef Optimization

- Fracción broadcast  $F_b$ : 0.98.

## 14. Resultados Experimentales

---

- Fracción asexual  $F_a$ : 0.05.
- Fracción depredación  $F_d$ : 0.05.
- Probabilidad asexual  $P_a$ : 0.001.
- Probabilidad de depredación  $P_d$ : 0.01.
- Tasa de corales ocupados  $\rho_0$ : 0.8.
- Intentos de asentamiento de la larva  $\eta$ : 3.

Es importante destacar que, debido a que se tratan de algoritmos estocásticos, se han llevado a cabo 30 ejecuciones distintas cambiando el número de semilla del generador aleatorio de números en cada una de ellas. Esto se ha hecho con el fin de tener una instanciación inicial distinta de los pesos, sesgos y la características seleccionadas para cada experimento. Finalmente, se ha tenido en cuenta el promedio y la desviación típica del total de resultados a la hora de representarlos en la Tabla 14.2.

### 14.2. Resultados

A continuación, se muestra el promedio de los resultados experimentales obtenidos en las pruebas. Inicialmente, la Tabla 14.2 presenta la media junto a la desviación típica del rendimiento de los algoritmos bioinspirados mencionados en el Capítulo 4, y de la propuesta de algoritmo CRO-ELM detallada en el Capítulo 7. Seguido a esto, se recogen los promedios de los tiempos de ejecución de cada algoritmo en la Tabla 14.3, de tal manera que no solo se tenga una visión en cuanto al rendimiento, sino también en el coste computacional de los mismos.

## 14.2. Resultados

Dataset	GA-ELM	PSO-ELM	CRO-ELM
breast-cancer-wisconsin-diagnostic	0,9658 <sub>0,0050</sub>	0,9693 <sub>0,0126</sub>	<b>0,9737</b> <sub>0,0041</sub>
breast-cancer-wisconsin-prognostic	<b>0,8205</b> <sub>0,0000</sub>	0,7974 <sub>0,0255</sub>	<b>0,8205</b> <sub>0,0000</sub>
breast-cancer	0,6931 <sub>0,0136</sub>	0,7034 <sub>0,0158</sub>	<b>0,7052</b> <sub>0,0055</sub>
chess-king-rook-vs-king-pawn	0,9270 <sub>0,0056</sub>	0,9283 <sub>0,0138</sub>	<b>0,9391</b> <sub>0,0074</sub>
cnae-9	0,2329 <sub>0,0341</sub>	0,2630 <sub>0,0406</sub>	<b>0,2991</b> <sub>0,0319</sub>
electrical-grid	0,8380 <sub>0,0049</sub>	0,8542 <sub>0,0173</sub>	<b>0,8568</b> <sub>0,0040</sub>
fertility	<b>0,9000</b> <sub>0,0000</sub>	0,8850 <sub>0,0242</sub>	<b>0,9000</b> <sub>0,0000</sub>
haberman-survival	<b>0,7097</b> <sub>0,0000</sub>	0,6903 <sub>0,0198</sub>	<b>0,7097</b> <sub>0,0000</sub>
hayes-roth	0,5407 <sub>0,0358</sub>	0,5889 <sub>0,0443</sub>	<b>0,6296</b> <sub>0,0000</sub>
hepatitis	0,8500 <sub>0,0323</sub>	<b>0,8688</b> <sub>0,0198</sub>	0,8625 <sub>0,0264</sub>
horse-colic	0,7717 <sub>0,0236</sub>	0,7500 <sub>0,0236</sub>	<b>0,7783</b> <sub>0,0249</sub>
indian-liver-patient	<b>0,6293</b> <sub>0,0000</sub>	0,6267 <sub>0,0058</sub>	<b>0,6293</b> <sub>0,0000</sub>
lung-cancer	0,4857 <sub>0,0999</sub>	0,4429 <sub>0,1251</sub>	<b>0,5714</b> <sub>0,0952</sub>
mammographic-mass	0,7722 <sub>0,0057</sub>	<b>0,7756</b> <sub>0,0077</sub>	0,7710 <sub>0,0071</sub>
monks-problems-2	<b>0,7356</b> <sub>0,0000</sub>	0,6586 <sub>0,0402</sub>	<b>0,7356</b> <sub>0,0000</sub>
mushroom	0,9948 <sub>0,0035</sub>	0,9955 <sub>0,0029</sub>	<b>0,9996</b> <sub>0,0003</sub>
ozone-level-detection-one	<b>0,9676</b> <sub>0,0000</sub>	<b>0,9676</b> <sub>0,0000</sub>	<b>0,9676</b> <sub>0,0000</sub>
planning-relax	<b>0,7027</b> <sub>0,0000</sub>	<b>0,7027</b> <sub>0,0000</sub>	<b>0,7027</b> <sub>0,0000</sub>
qsar-biodegradation	0,8270 <sub>0,0121</sub>	<b>0,8441</b> <sub>0,0298</sub>	<b>0,8441</b> <sub>0,0115</sub>
soybean-small	0,8300 <sub>0,0823</sub>	0,8300 <sub>0,1059</sub>	<b>0,8600</b> <sub>0,0699</sub>
spambase	0,8375 <sub>0,0093</sub>	<b>0,8736</b> <sub>0,0108</sub>	0,8647 <sub>0,0049</sub>
spect-heart	0,7188 <sub>0,0329</sub>	0,7000 <sub>0,0574</sub>	<b>0,7438</b> <sub>0,0198</sub>
statlog-project-german-credit	0,7710 <sub>0,0129</sub>	0,7410 <sub>0,0115</sub>	<b>0,7845</b> <sub>0,0192</sub>
thoracic-surgery	<b>0,7979</b> <sub>0,0000</sub>	0,7947 <sub>0,0072</sub>	<b>0,7979</b> <sub>0,0000</sub>
thyroid-disease-allhyper	<b>0,9725</b> <sub>0,0009</sub>	0,9704 <sub>0,0017</sub>	0,9707 <sub>0,0029</sub>
thyroid-disease-allrep	0,9193 <sub>0,0146</sub>	0,9371 <sub>0,0230</sub>	<b>0,9445</b> <sub>0,0162</sub>
thyroid-disease-dis	<b>0,9852</b> <sub>0,0000</sub>	<b>0,9852</b> <sub>0,0000</sub>	<b>0,9852</b> <sub>0,0000</sub>
thyroid-disease-sick-euthyroid	<b>0,8863</b> <sub>0,0000</sub>	<b>0,8863</b> <sub>0,0000</sub>	<b>0,8863</b> <sub>0,0000</sub>
tic-tac-toe-endgame	0,8589 <sub>0,0378</sub>	0,7974 <sub>0,0525</sub>	<b>0,9599</b> <sub>0,0110</sub>
weight-lifting-exercises	0,8043 <sub>0,0506</sub>	0,8617 <sub>0,0363</sub>	<b>0,9080</b> <sub>0,0137</sub>

Tabla 14.2: Rendimiento promedio de los algoritmos.

## 14. Resultados Experimentales

Dataset	GA-ELM	PSO-ELM	CRO-ELM
breast-cancer-wisconsin-diagnostic	27,3296 <sub>0000,2439</sub>	16,0133 <sub>0000,2845</sub>	12,3737 <sub>0000,1247</sub>
breast-cancer-wisconsin-prognostic	21,3296 <sub>0000,1413</sub>	12,2356 <sub>0000,1840</sub>	10,3784 <sub>0000,1748</sub>
breast-cancer	23,3598 <sub>0000,2047</sub>	13,5874 <sub>0000,2333</sub>	11,0346 <sub>0000,1516</sub>
chess-king-rook-vs-king-pawn	68,1471 <sub>0000,5525</sub>	40,7561 <sub>0000,5352</sub>	26,0958 <sub>0000,6212</sub>
cnae-9	66,1826 <sub>0000,3788</sub>	41,6031 <sub>0000,2957</sub>	25,4759 <sub>0000,1766</sub>
electrical-grid	296,6884 <sub>0003,8222</sub>	184,0644 <sub>0002,7552</sub>	98,6881 <sub>0001,3852</sub>
fertility	17,2443 <sub>0000,2559</sub>	10,2933 <sub>0000,4289</sub>	9,2640 <sub>0000,1515</sub>
haberman-survival	19,0008 <sub>0000,5676</sub>	11,8977 <sub>0000,3624</sub>	9,5326 <sub>0000,3014</sub>
hayes-roth	15,5063 <sub>0000,4157</sub>	10,3489 <sub>0000,4332</sub>	8,7024 <sub>0000,2878</sub>
hepatitis	18,2268 <sub>0000,4535</sub>	10,3817 <sub>0000,3949</sub>	9,4478 <sub>0000,2846</sub>
horse-colic	26,0951 <sub>0000,7734</sub>	15,0198 <sub>0000,3854</sub>	12,0389 <sub>0000,3440</sub>
indian-liver-patient	25,3028 <sub>0000,8001</sub>	14,8874 <sub>0000,3712</sub>	11,7258 <sub>0000,3094</sub>
lung-cancer	16,0903 <sub>0000,4833</sub>	10,0379 <sub>0000,3117</sub>	8,7605 <sub>0000,2798</sub>
mammographic-mass	29,4966 <sub>0000,3985</sub>	18,5489 <sub>0000,2005</sub>	13,4654 <sub>0000,2051</sub>
monks-problems-2	22,0073 <sub>0000,2047</sub>	14,1444 <sub>0000,1782</sub>	10,1517 <sub>0000,1005</sub>
mushroom	267,8066 <sub>0004,5350</sub>	164,4840 <sub>0003,6891</sub>	89,0202 <sub>0002,6266</sub>
ozone-level-detection-one	47,3430 <sub>0002,4498</sub>	27,9459 <sub>0001,5743</sub>	18,7133 <sub>0001,0447</sub>
planning-relax	19,5253 <sub>0000,5445</sub>	11,0263 <sub>0000,3401</sub>	9,5621 <sub>0000,4490</sub>
qsar-biodegradation	34,6893 <sub>0000,4057</sub>	20,5047 <sub>0000,2979</sub>	14,4552 <sub>0000,1994</sub>
soybean-small	16,1887 <sub>0000,2655</sub>	9,2630 <sub>0000,2900</sub>	8,6786 <sub>0000,1337</sub>
spambase	157,0992 <sub>0002,4873</sub>	96,3633 <sub>0001,5659</sub>	53,7077 <sub>0000,8291</sub>
spect-heart	16,6429 <sub>0000,3031</sub>	10,0089 <sub>0000,4580</sub>	8,7772 <sub>0000,2243</sub>
statlog-project-german-credit	35,2920 <sub>0000,3651</sub>	20,9668 <sub>0000,2257</sub>	14,8222 <sub>0000,1415</sub>
thoracic-surgery	25,5156 <sub>0000,7865</sub>	14,8837 <sub>0000,5513</sub>	11,8063 <sub>0000,3880</sub>
thyroid-disease-allhyper	59,0635 <sub>0002,3039</sub>	34,7211 <sub>0001,4996</sub>	23,3355 <sub>0000,7585</sub>
thyroid-disease-allrep	59,2679 <sub>0002,3146</sub>	35,6873 <sub>0001,2178</sub>	23,4439 <sub>0000,8329</sub>
thyroid-disease-dis	48,6903 <sub>0002,3929</sub>	28,9238 <sub>0001,2975</sub>	19,3495 <sub>0000,7846</sub>
thyroid-disease-sick-euthyroid	63,1164 <sub>0002,0663</sub>	38,8995 <sub>0001,5128</sub>	24,0579 <sub>0000,8376</sub>
tic-tac-toe-endgame	32,9974 <sub>0001,2020</sub>	19,6124 <sub>0000,9103</sub>	14,2504 <sub>0000,5794</sub>
weight-lifting-exercises	78,6155 <sub>0003,7468</sub>	47,9179 <sub>0002,3926</sub>	28,8857 <sub>0001,0173</sub>

Tabla 14.3: Tiempos de ejecución promedio (en segundos) de los algoritmos.

## 14.3. Discusión

Finalizados los experimentos, y una vez se ha completado la Tabla 14.2 según la métrica de evaluación de *Correct Classification Rate*, y la Tabla 14.3 según los tiempos de ejecución de cada algoritmo expresados en segundos, se procede a hacer una disección de los resultados obtenidos.

Si se hace la comparativa de rendimientos en la Tabla 14.2, se puede observar como en 26 de las 30 bases de datos empleadas, el algoritmo propuesto (CRO-ELM) es superior a los demás, en 3 ocasiones es el segundo mejor algoritmo, y sólo en 1 ocasión reporta un CCR inferior con respecto a los algoritmos del estado del arte. No obstante, merece la pena destacar que en dicha base de datos el rendimiento es prácticamente el mismo, encontrando diferencias en el tercer decimal. Además, en este caso en concreto, se podría mejorar si se incrementa el número de iteraciones en el algoritmo ya que si se tiene en cuenta la Tabla de 14.3, se ve que para la configuración de población y generación utilizadas en los experimentos, el algoritmo de CRO-ELM ha tardado un promedio de 13,4654 segundos, un tiempo menor al resto de algoritmos bastante considerable.

En este sentido, profundizando en los tiempos de ejecución obtenidos en la Tabla 14.3, se observa como el algoritmo CRO-ELM supera a los demás, siendo PSO-ELM el segundo y GA-ELM el último, en todos los casos. Esta diferencia es tan clara que los tiempos se reducen a un tercio con respecto al GA-ELM y a un poco más de la mitad con respecto al PSO-ELM. Este hecho era de esperar ya que, el algoritmo de CRO es una mejora del clásico GA, puesto que además de realizar una fase de reproducción asexual, lleva a cabo una fase de depredación que deja huecos vacíos en la población. Esto resulta en un menor número de individuos por generación que evaluar y, por tanto, un coste computacional mucho más bajo.

Este análisis nos puede hacer concluir que si establecemos un criterio de parada basándonos en el tiempo de ejecución como puede ser el número de evaluaciones máximas que cada algoritmo debe realizar, los resultados en tiempo estarían más cercanos a los otros dos algoritmos, sin embargo, el rendimiento en la métrica CCR sería mucho mejor.

# Parte V

## Conclusiones







## 15 Conclusiones del Autor

En el siguiente capítulo se hace un repaso de aquellos objetivos que se plantearon en el inicio de este Trabajo Fin de Grado y que una vez finalizada la aplicación, pasadas todas las pruebas y extraídos los resultados experimentales se procede a concretar los requisitos satisfechos.

### 15.1. Objetivos de Formación Alcanzados

En esta sección se agrupan todas aquellas competencias adquiridas por el alumno en la realización del TFG:

1. El alumno ha realizado un estudio en profundidad del lenguaje Python y del paradigma enfoque orientado a objetos.
2. El alumno ha adquirido una base amplia y sólida de las principales librerías de Python referentes al manejo de datos y cálculo matemático y estadístico.
3. El alumno ha diseñado y desarrollado una interfaz gráfica interactiva para la configuración y ejecución de experimentos.

## 15.2. Objetivos Operacionales Alcanzados

---

4. El alumno ha expandido sus conocimientos acerca de los algoritmos del estado del arte, y también sobre modelos neuronales para clasificación y regresión.
5. El alumno ha llevado a cabo la planificación, investigación, diseño, implementación y documentación de un proyecto software al completo, validando la eficacia y calidad de la propuesta, obteniendo como resultado el presente TFG.
6. El alumno ha obtenido conocimientos de la herramienta  $\text{\LaTeX}$  y adquirido conocimiento sobre la redacción de artículos de alta calidad tipográfica y carga matemática.

## 15.2. Objetivos Operacionales Alcanzados

En referencia a los objetivos operacionales alcanzados en el desarrollo de la aplicación se concluye:

1. Se ha implementado con éxito un algoritmo de aprendizaje automático basado en el modelo neuronal *Extreme Learning Machine*, con un algoritmo bioinspirado basado en la reproducción de los arrecifes de coral que realiza una selección de características y ajuste de los pesos y sesgos de las conexiones neuronales en la entrada del modelo.
2. De igual forma, se han implementado correctamente otros dos algoritmos del estado del arte, algoritmo genético y algoritmo *Particle Swarm Optimization*, adaptándolos en la arquitectura del sistema para poner a prueba sus rendimiento y hacer una comparación con el algoritmo propuesto *Coral Reef Optimization* para el entrenamiento de redes ELM.
3. Se ha conseguido implementar una interfaz gráfica de usuario que permite modificar las configuraciones de parámetros y seleccionar el algoritmo del experimento, facilitando el uso de la herramienta implementada al usuario final de la aplicación.

## 15. Conclusiones del Autor

---

4. Se ha incluido una mapa dinámico para la visualización del rendimiento de los individuos, con el fin de tener una visión más clara de la optimización que realizan los algoritmos bioinspirados durante la fase de entrenamiento.
5. Se han añadido botones que permiten al usuario tener un manejo completo de la aplicación, pudiendo restablecer automáticamente las configuraciones a su valor por defecto, avanzar generación a generación el algoritmo, y también exportar los resultados a un fichero *.xlsx*.
6. En cuanto a los resultados, el algoritmo implementado CRO-ELM, ha mejorado a los dos algoritmos del estado del arte tanto en rendimiento como en coste computacional. Esto puede dar por concluida la investigación planteada al inicio de este TFG, así como la posibilidad de seguir profundizando en estos temas en TFGs futuros.

Como resultado de la agregación de los objetivos mencionados, tanto formales como operacionales, el alumno proyectista estima que se ha logrado cumplir con todos los objetivos propuestos al inicio del TFG.





## 16 Futuras Mejoras

En el siguiente capítulo se detallan las posibles futuras mejoras a llevar a cabo, en caso de si se continuase con el desarrollo de la aplicación y de la propuesta del algoritmo. Cabe mencionar que las propuestas de mejora, incluyen tanto a la interfaz gráfica desarrollada, como a la ejecución y rendimiento de la propuesta de algoritmo.

### 16.1. Mejoras de la propuesta

#### 16.1.1. Librerías de Optimización

Una de las primeras propuestas y que tendría un gran impacto sería el hecho de utilizar una librería de optimización, tales como **Keras** o **Tensorflow**, para realizar los cálculos y protitpado de la red de una manera más eficiente incluso aprovechando la potencia de las tarjetas gráficas actuales.

#### 16.1.2. Técnicas de Regularización

Otra de las grandes mejoras aplicables al modelo neuronal *Extreme Learning Machine* consiste en utilizar una técnica de regularización

diferente. Actualmente en el modelo se usa una regularización de tipo *Ridge* (L2) [Hilt and Seegrist \[1977\]](#), es por eso que se propone implementar una regularización *LASSO* (*Least Absolute Shrinkage and Selection Operator*) (L1) [Tibshirani \[1996\]](#) ya que reduce a cero los coeficientes de las características menos importantes, y por tanto disminuye la complejidad de la estructura del modelo. Además, esta regularización *LASSO* ha reportado en varios experimentos y estudios, resultados favorables frente a la regularización *Ridge* como es el caso de [Várkonyi and Buza \[2019\]](#).

### 16.1.3. Función de Activación

En cuanto a las funciones de activación empleadas por el modelo en el cálculo de la matriz  $H$  de la capa oculta, se propone hacer uso de una función de activación *TanhRe*, producto de la combinación de las funciones de activación *ReLU* y tangente hiperbólica, cuya definición y validación de resultados aparecen en [Ratnawati et al. \[2020\]](#). Dicha función sigue la expresión:

$$f(x) = \begin{cases} x_i & \text{if } x > 0, \\ \tanh(x) & \text{if } x \leq 0. \end{cases} \quad (16.1)$$

### 16.1.4. Métricas de evaluación

Una de las posibles mejoras claras en este Trabajo Fin de Grado es la implementación de distintas métricas de evaluación. En la actual aplicación se utiliza el *Correct Classified Rate*, pero se pueden incluir otras métricas y que el usuario final elija con cual optimizar. Además, esto supondría la inclusión de los problemas de regresión mediante métricas propias para dichos problemas.

### 16.2. Mejoras de la interfaz gráfica

#### 16.2.1. Despliegue Servidor

La interfaz gráfica que se entrega con el TFG, se despliega de manera local, en un servidor *localhost*. Es por ello que se propone como mejora realizar un despliegue en la nube, ya sea Microsoft Azure o Google Cloud Platform (GCP). Ambos servidores cloud están soportados y bien integrados con la librería de Panel, sobre la que se ha desarrollado la herramienta de la aplicación.

#### 16.2.2. *Multithreading*

Otra de las ventajas que aporta el lenguaje Python, es la concurrencia que permite el propio lenguaje a ejecutar varios hilos. De esta manera se podría implementar un sistema multihilo que ejecuten los tres algoritmos bioinspirados implementados, evitando al usuario de elegir el algoritmo de optimización empleado en el experimento, y obteniendo resultados de todos, con una sola ejecución. Esto, además, supondría una ventaja a la hora de poder ejecutar el algoritmo en la nube.







## BIBLIOGRAFÍA

- M. A. A. Albadr, S. Tiun, M. Ayob, F. T. Al-Dhief, K. Omar, and F. A. Hamzah. Optimised genetic algorithm-extreme learning machine approach for automatic covid-19 detection. *PloS one*, 15(12): e0242899, 2020.
- A. S. Alencar, A. R. R. Neto, and J. P. P. Gomes. A new pruning method for extreme learning machines via genetic algorithms. *Applied Soft Computing*, 44:101–107, 2016.
- G. Cybenko, D. P. O’Leary, and J. Rissanen. *The Mathematics of Information Coding, Extraction and Distribution*, volume 107. Springer Science & Business Media, 2012.
- A. M. Durán Rosal. Time series data mining: preprocessing, analysis, segmentation and prediction. applications. 2019.
- A. M. Duran-Rosal, P. A. Gutierrez, S. Salcedo-Sanz, and C. Hervás-Martínez. A statistically-driven coral reef optimization algorithm for optimal size reduction of time series. *Applied Soft Computing*, 63:139–153, 2018.
- A. GALIPIENSO, M. ISABEL, M. A. Cazorla Quevedo, O. Colomina Pardo, F. Escolano Ruiz, and M. A. LOZANO ORTEGA. *Inteligencia artificial: modelos, técnicas y áreas de aplicación*. Ediciones Paraninfo, SA, 2003.
- D. E. Golberg. Genetic algorithms in search, optimization, and machine learning. *Addion wesley*, 1989(102):36, 1989.

- D. E. Hilt and D. W. Seegrist. *Ridge, a computer program for calculating ridge regression estimates*. Department of Agriculture, Forest Service, Northeastern Forest Experiment . . . , 1977.
- J. Holland. Adaptation in natural and artificial systems. *Ann Arbor*, 1975.
- J. H. Holland. *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence*. MIT press, 1992.
- G.-B. Huang, Q.-Y. Zhu, and C.-K. Siew. Extreme learning machine: theory and applications. *Neurocomputing*, 70(1-3):489–501, 2006.
- J. Kennedy and R. Eberhart. Particle swarm optimization. In *Proceedings of ICNN'95-international conference on neural networks*, volume 4, pages 1942–1948. IEEE, 1995.
- S. Kullback and R. A. Leibler. On information and sufficiency. *The annals of mathematical statistics*, 22(1):79–86, 1951.
- G. Kumar, U. P. Singh, and S. Jain. Swarm intelligence based hybrid neural network approach for stock price forecasting. *Computational Economics*, 60(3):991–1039, 2022.
- J. L. McClelland, D. E. Rumelhart, P. R. Group, et al. *Parallel Distributed Processing, Volume 2: Explorations in the Microstructure of Cognition: Psychological and Biological Models*, volume 2. MIT press, 1987.
- W. W. P. McCulloch. «a logical calculus of ideas immanent in nervous activity». *Bulletin of Mathematical Biophysics* 5, pages 115–133, 1943.
- E. H. Moore. On the reciprocal of the general algebraic matrix. *Bull. Am. Math. Soc.*, 26:394–395, 1920.
- D. M. J. Purnomo, S. C. Purbarani, A. Wibisono, D. Hendrayanti, A. Bowolaksono, P. Mursanto, D. H. Ramdhan, and W. Jatmiko. Genetic algorithm optimization for extreme learning machine based microalgal growth forecasting of chlamydomonas sp. In *2015 International Conference on Advanced Computer Science and Information Systems (ICACSIS)*, pages 243–248, 2015. doi: 10.1109/ICACSIS.2015.7415189.

## BIBLIOGRAFÍA

---

- D. E. Ratnawati, Marjono, Widodo, and S. Anam. Comparison of activation function on extreme learning machine (elm) performance for classifying the active compound. In *AIP Conference Proceedings*, volume 2264, page 140001. AIP Publishing LLC, 2020.
- D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533–536, 1986.
- S. Salcedo-Sanz, J. Del Ser, I. Landa-Torres, S. Gil-López, and J. Portilla-Figueras. The coral reefs optimization algorithm: a novel metaheuristic for efficiently solving optimization problems. *The Scientific World Journal*, 2014, 2014a.
- S. Salcedo-Sanz, J. E. Sanchez-Garcia, J. A. Portilla-Figueras, S. Jimenez-Fernandez, and A. M. Ahmadzadeh. A coral-reef optimization algorithm for the optimal service distribution problem in mobile radio access networks. *Transactions on Emerging Telecommunications Technologies*, 25(11):1057–1069, 2014b.
- R. Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society: Series B (Methodological)*, 58(1):267–288, 1996.
- D. T. Várkonyi and K. Buza. Extreme learning machines with regularization for the classification of gene expression data. In *ITAT*, pages 99–103, 2019.
- Wikipedia. Función rampa — Wikipedia, the free encyclopedia. <http://es.wikipedia.org/w/index.php?title=Funci%C3%B3n%20rampa&oldid=137803832>, 2023a. [Online; accessed 31-January-2023].
- Wikipedia. Función sigmoide — Wikipedia, the free encyclopedia. <http://es.wikipedia.org/w/index.php?title=Funci%C3%B3n%20sigmoide&oldid=147179398>, 2023b. [Online; accessed 30-January-2023].
- Wikipedia. Tangente hiperbólica — Wikipedia, the free encyclopedia. <http://es.wikipedia.org/w/index.php?title=Tangente%20hiperb%C3%B3lica&oldid=134732053>, 2023c. [Online; accessed 30-January-2023].
- Wikipedia. Perceptrón — Wikipedia, the free encyclopedia. <http://es.wikipedia.org/w/index.php?title=Perceptr%C3%B3n>

## BIBLIOGRAFÍA

---

[C3%B3n&oldid=148351059](#), 2023d. [Online; accessed 15-January-2023].

# Parte VI

## Apéndices





# A Manual de Usuario

Este documento supone de un manual para el usuario sobre la aplicación hecha en en lenguaje de programación Python acerca del *Desarrollo de Algoritmos Bioinspirados para el entrenamiento de Redes Neuronales Extreme Learning Machine*. Ésta aplicación a su vez, constituye el Trabajo de Fin de Grado del alumno David Pineda Peña.

Iniciamos esta guía de la aplicación describiendo el producto software resultante del TFG.

## A.1. Descripción del Producto

El producto incluye tres funcionalidades que se consideran principales:

- Ofrece al usuario una interfaz gráfica interactiva y de fácil uso para realizar experimentos mediante los diferentes algoritmos que se incluyen y la posibilidad de configurar sus parámetros.
- Permite la exportación y almacenamiento de los resultados experimentales en ficheros de hoja de cálculo para el análisis del rendimiento predictivo del sistema.



## A.2. Requisitos del Sistema

---

- Muestra en forma de mapa dinámico los resultados que va alcanzando, en su fase de entrenamiento, el algoritmo elegido durante su ejecución.

Se analizan cada uno de estos puntos por separado.

Por resumir de manera concisa la función global de la aplicación, se puede concluir que mediante el uso de una interfaz gráfica el usuario se pueden lanzar experimentos con la configuración que se deseen y ver en tiempo real el rendimiento de dicho experimento, a la vez que cuenta con la posibilidad de exportar los resultados una vez termine su ejecución.

Dicho sistema cuenta con las siguientes características principales:

- Basado y desarrollado en el lenguaje de programación Python.
- Dashboard interactivo simple y estructurado.
- Lanzamiento de experimentos de libre configuración al deseo del usuario.
- Generación de gráficos en tiempo real de ejecución.
- Creación de ficheros a partir de los resultados obtenidos.

## A.2. Requisitos del Sistema

Para la ejecución en *Google Colaboratory* no se requiere de ningún hardware o software en específico, solamente se necesita contar con una **cuenta Google** y tener espacio libre en **Google Drive**. Esto se debe a que es una herramienta de navegador desarrollada por el equipo de Google, y sirve como editor de código Python sin necesidad de configuración extra.

El único inconveniente de esta herramienta es que para el plan gratuito de Google Colab, es necesario tener una actividad constante con

## A. Manual de Usuario

---

la ventana del navegador si se desea mantener el entorno de ejecución activo.

Para ello, cuenta con la opción de escritorio, donde se recomienda hacer uso de la distribución libre y de código abierto *Anaconda*. Ésta distribución ya cuenta con la amplia mayoría de librerías avanzadas de Python y R, además de muchos *IDEs* (Entornos de desarrollo integrado) para la ejecución de código.

Los requisitos mínimos para poder instalar y hacer uso de Anaconda son:

### A.2.1. Requisitos Mínimos

- Memoria: 4GB de RAM.
- Almacenamiento: 5 GB de espacio en disco duro para descargar e instalar.

### A.2.2. Requisitos Recomendados

Mencionados los requisitos mínimos, para poder hacer un uso completo de Anaconda e instalar librerías adicionales o crear *environments*, que son entornos aislados de Anaconda donde se pueden instalar librerías y paquetes específicos para un proyecto, se recomienda tener:

- Procesador Intel o AMD x86-64 con más de 4 núcleos.
- 8GB de RAM.
- 20 GB de espacio en disco duro para descargar e instalar la distribución y crear entornos.

### A.2.3. Requisitos Software

- Sistema operativo: Windows 8 o posterior, macOS 10.13+ de 64 bits o Linux, incluidos Ubuntu, RedHat, CentOS 7+ y otros.

- Arquitectura del sistema: Windows- 64-bit x86, MacOS- 64-bit x86 y M1, Linux- 64-bit x86, 64-bit aarch64 (AWS Graviton2), 64-bit Power8/Power9, s390x (Linux en IBM Z y LinuxONE).

## A.3. Instalación del Software

La aplicación desarrollada está disponible en multitud de formatos. Primero, se cuenta con un formato físico de **memoria USB** en el que se encuentra el archivo *.ipynb* (*Jupyter Notebook*) disponible para cargar en Google Colab o en cualquier entorno de desarrollo que soporte dicha extensión.

De igual manera se puede descargar desde el repositorio público de [GitHub](#) del alumno.

Tal y como se ha comentado en el apartado A.2, si se opta por la opción de utilizar Google Colab no será necesaria ninguna instalación software previa. Por el contrario, si se opta por el uso de la distribución libre de Anaconda, será necesario la instalación de la misma.

### A.3.1. Instalación Anaconda Distribution

Si se usa la arquitectura de sistema x86, ya sea Windows, macOS o Linux, se dispone de la opción simple de instalación gráfica. Para ello, se debe ir a la [página del instalador](#) de Anaconda Distribution y descargar dicho instalador.

Una vez descargado el ejecutable, se abre y sólo se tendrán que seguir los pasos en la ventana gráfica de instalación.

Para aquellos que prefieran hacer una instalación desde la terminal en sistemas macOS, Linux u otros, encontramos una guía detallada con recomendaciones dependiendo del uso que se le vaya a dar en el siguiente enlace <https://docs.anaconda.com/anaconda/install/>.

### A.4. Desinstalación del Software

Si se desea desinstalar la aplicación solo hará falta eliminar el directorio donde se encuentre la misma, o donde se haya clonado el repositorio de GitHub.

Adicionalmente, si se quiere hacer una desinstalación de la herramienta de Anaconda se tendrá que hacer de la siguiente forma:

#### A.4.1. Desinstalación Anaconda

Para desinstalar Anaconda del equipo, es tan fácil como, en caso de que se use un sistema operativo Windows, abrir el *Anaconda Prompt* que traerá integrado la propia herramienta, o en caso de macOS y Linux abrir la terminal, y escribir la siguiente línea:

```
conda install anaconda-clean
```

Una vez se dispone del paquete *anaconda-clean*, se puede ejecutar cualquiera de los dos siguientes comandos:

*anaconda-clean*, si se quiere confirmar cada archivo y directorio que se va a eliminar.

*anaconda-clean --yes*, si se quiere borrar todo directamente.

### A.5. Ejecución del Software

Con el fin de ejecutar el software una vez ha sido instalado en el equipo o cargado en Google Colab, se deben seguir los siguientes pasos:

Abrir el archivo ***GUI.ipynb*** el cual ejecutará y desplegará la interfaz gráfica de usuario en un servidor *localhost* en el **puerto 5006**.

## A.6. Módulo Interfaz Gráfica

Ahora que el servidor *localhost* ha sido desplegado con la interfaz gráfica, ya se dispone de acceso a interactuar con ella y configurar los parámetros de los experimentos. En la Figura A.1 se puede observar de manera general la apariencia que tiene la interfaz gráfica y sus diferentes partes, aunque dependiendo del algoritmo que se seleccione los parámetros visibles cambian, tal y como se ha detallado en la Sección 12.1.1 del Capítulo 12:

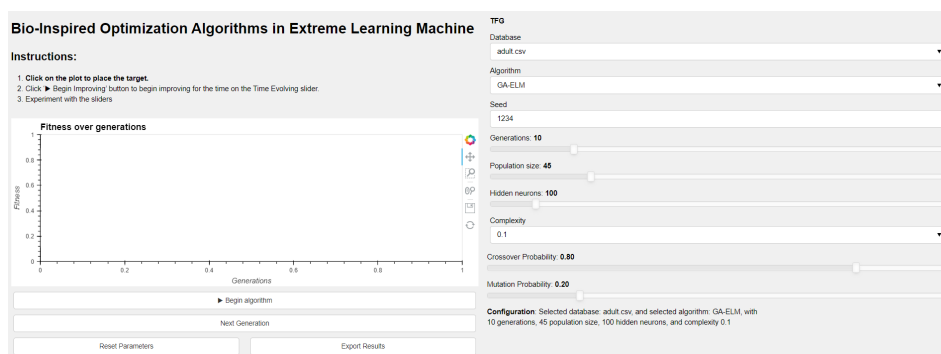


Figura A.1: Apariencia de la interfaz gráfica.

Las partes principales que componen dicha interfaz gráfica son:

- **Parámetros del algoritmo.** Esta sección ocupa toda la mitad derecha de la ventana y es la encargada de recoger los datos del experimento elegidos por el usuario. Se compone de:
  - **Bases de datos:** Consiste en un desplegable en el que se puede escoger la base de datos de prueba entre una lista de ficheros *.csv*.
  - **Algoritmo:** Consiste en un desplegable para seleccionar el algoritmo bioinspirado encargado de realizar el entrenamiento de las redes neuronales del modelos. Los algoritmos posibles son:
    - GA-ELM.

- PSO-ELM.
- CRO-ELM.
- **Semilla:** Consiste en una entrada de texto numérica para determinar el valor de la semilla del generador de números aleatorios.
- **Complejidad:** Consiste en un desplegable para determinar el valor del hiperparámetro **C** del modelo.
- **Probabilidad de mutación:** Opción para fijar la probabilidad con la que cada individuo va a ser mutado.
- **Número de generaciones:** Consiste en un control deslizante para escoger el número de generaciones del algoritmo bioinspirado.
- **Tamaño de la población:** Consiste en un control deslizante para escoger el número de individuos dentro de la población.
- **Neuronas en capa oculta:** Consiste en un control deslizante para escoger el valor del hiperparámetro **D** del modelo.
- **Probabilidad de cruce:** Consiste en un control deslizante para escoger el valor de la probabilidad de cruce. Este parámetro solo es visible si se escoge el algoritmo GA-ELM.
- **Probabilidad de mutación:** Consiste en un control deslizante para escoger el valor de la probabilidad de mutación. Este parámetro solo es visible si se escoge el algoritmo GA-ELM.
- **Coeficiente de inercia máximo  $\omega_{max}$ :** Consiste en un control deslizante para escoger el valor máximo del coeficiente de inercia de las partículas. Este parámetro solo es visible si se escoge el algoritmo PSO-ELM.
- **Coeficiente de inercia mínimo  $\omega_{min}$ :** Consiste en un control deslizante para escoger el valor máximo del coefi-

ciente de inercia de las partículas. Este parámetro solo es visible si se selecciona el algoritmo PSO-ELM.

- **Coefficiente cognitivo  $c_1$** : Consiste en un control deslizante para escoger el valor de coeficiente cognitivo de las partículas. Este parámetro solo es visible si se elige el algoritmo PSO-ELM.
- **Coefficiente social  $c_2$** : Consiste en un control deslizante para escoger el valor de coeficiente cognitivo de las partículas. Este parámetro solo es visible si se escoge el algoritmo PSO-ELM.
- **Fracción broadcast  $F_b$** : Consiste en un control deslizante para escoger el valor de fracción del arrecife que va a llevar a cabo reproducción externa. Este parámetro solo es visible si se selecciona el algoritmo CRO-ELM.
- **Fracción asexual  $F_a$** : Consiste en un control deslizante para escoger el valor de fracción del arrecife que va a llevar a cabo reproducción asexual en caso de que se cumpla su probabilidad. Este parámetro solo es visible si se elige el algoritmo CRO-ELM.
- **Fracción depredación  $F_d$** : Consiste en un control deslizante para escoger el valor de fracción del arrecife que va a ser eliminada del arrecife si se cumple su probabilidad. Este parámetro solo es visible si se elige el algoritmo CRO-ELM.
- **Probabilidad asexual  $P_a$** : Consiste en un control deslizante para escoger el valor de probabilidad para que los corales se reproduzcan asexualmente. Este parámetro solo es visible si se escoge el algoritmo CRO-ELM.
- **Probabilidad de depredación  $P_d$** : Consiste en un control deslizante para escoger el valor de probabilidad por el que se va a determinar si una larva es depredada o sigue viva. Este parámetro solo es visible si se elige el algoritmo CRO-ELM.

- **Tasa de colonias ocupadas  $\rho_0$ :** Consiste en un control deslizante para escoger el valor de la tasa de ocupación del arrecife. Este parámetro solo es visible si se selecciona el algoritmo CRO-ELM.
- **Intentos de asentamiento de la larva  $\eta$ :** Consiste en un control deslizante para escoger el número de intentos de asentamiento previos a la eliminación de la larva. Este parámetro solo es visible si se escoge el algoritmo CRO-ELM.
- **Gráfico Interactivo de rendimiento de la población.** Esta sección ocupa la esquina superior izquierda de la ventana y se encarga de mostrar un mapa dinámico sobre la evolución del fitness de las soluciones candidatas de la población a lo largo de las generaciones.
- **Botones de eventos.** Esta sección ocupa la esquina inferior izquierda de la ventana y recoge todas las acciones que puede llevar a cabo la interfaz gráfica. Entre ellas encontramos:
  - **Ejecutar algoritmo:** Botón para ejecutar el algoritmo de manera completa con las configuraciones dadas en la sección 12.1.1.
  - **Siguiente generación:** Botón para ejecutar una generación del algoritmo con las configuraciones dadas.
  - **Reiniciar parámetros:** Botón para ajustar todos los parámetros a su valor por defecto.
  - **Exportar resultados:** Botón para exportar los datos de salida del experimento a un fichero *.xlsx*.



