UNIVERSIDAD CARLOS III DE MADRID

ESCUELA POLITÉCNICA SUPERIOR

GRADO EN INGENIERÍA INFORMÁTICA

Trabajo fin de grado

ESTIMACIÓN DE INTERVALOS DE PREDICCIÓN MEDIANTE COMPUTACIÓN EVOLUTIVA.

AUTOR: EDUARDO ARRESE JIMÉNEZ

TUTORA: RICARDO ALER MUR

COTUTOR: INÉS MARÍA GALVÁN LEÓN

23 de Septiembre de 2015

Título: *ESTIMACIÓN DE INTERVALOS DE PREDICCIÓN MEDIANTE COMPUTACIÓN EVOLUTIVA.*

Autor: *EDUARDO ARRESE JIMÉNEZ*

Tutora: *RICARDO ALER MUR*

Cotutor: *INÉS MARÍA GALVÁN LEÓN*

La defensa del presente Proyecto Fin de Carrera se realizó el día ; siendo calificada por el siguiente tribunal:

Presidente:

Secretario

Vocal

Habiendo obtenido la siguiente calificación:

Calificación:

Presidente **Secretario** **Vocal**

Resumen

Con la creciente necesidad de energía en el mundo moderno, cada vez es más importante la utilización de una energía limpia y segura.La energía en la que nos centraremos en este documento es la solar. Debido a las limitaciones de almacenamiento de este tipo de energía, es necesario el desarrollo de técnicas para poder predecir la electricidad generada. Se explicarán los métodos seguidos para la creación de modelos de predicción de intervalos basados en técnicas de inteligencia artificial, como son las redes de neuronas. Esta red contará 300 neuronas de entrada con los atributos atmosféricos sobre radiación solar y densidad de las nubes, y con dos salidas para representar ambos extremos del intervalo. Además, se hará uso de técnicas de optimización para los pesos de la red, para lo cual utilizaremos algoritmos de computación evolutiva como los genéticos o los de evolución diferencial. Para ello, también se detallara la metodología usada para obtener los cromosomas y función de fitness necesarias. Por último, estos modelos se compararán con un método de construcción de intervalos, la regresión cuantil.

Índice general

Lista de Figuras

Lista de Tablas

Capítulo 1

Introducción

1.1. Contexto histórico

Desde la llegada de la revolución industrial el ser humano se ha visto sometido a una creciente necesidad por una energía que mantenga en funcionamiento sus máquinas. A mediados del siglo XX y ante una mayor demanda se construyen las primeras centrales nucleares. Sin embargo, a raíz de una serie de accidentes que se han mantenido hasta la actualidad y debido a la dificultad de eliminar los desechos radiactivos generados, estas obligan al desarrollo de una forma de energía limpia y segura. Las energías renovables, como la fotovoltaica, eólica o hidráulica, pese a tener un coste económico mucho más elevado que el resto de energías, no ocasiona tantos riesgos y su impacto medioambiental resulta mínimo.

En el caso que nos ocupa, trataremos la producción de energía fotovoltaica. En 1954, A. D.M. Chapin, C.S. Fuller y G.L. Pearson, desarrollaron la primera célula fotovoltaica basada en el silicio. Siendo este primer panel de una efectividad del 6 %, pronto se llego hasta el 10 %[4], llegando en la actualidad a alcanzar cotas del 20 %.

1.1.1. Entorno socio-economico

En la actualidad, se calcula que este tipo energía proporciona el 1 % de la producción mundial de energía. Sin embargo, su utilización es prácticamente nueva y pese a este bajo porcentaje, su uso en los últimos 15 años se ha incrementado en más de 100 veces, siendo Alemania el país líder, seguido por España e Italia.

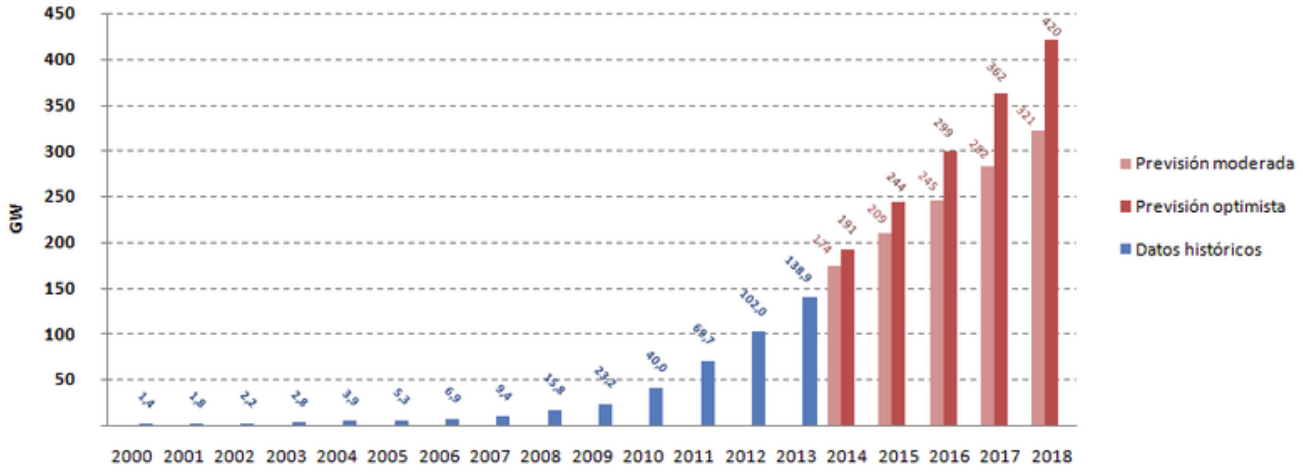

Figura 1.1: Evolución de la energía solar

Además, es reseñable los nuevos proyectos que se están llevando a cabo para impulsar este tipo de energía. Entre ellos la iniciativa más importante es la de la compañía SolarCity, fundada por los hermanos Peter y Lyndon Rive, que surgió como sugerencia de Elon Musk, empresario e ingeniero reconocido por la creación de PayPal o Tesla Motors. En solo 8 años desde su fundación ha pasado a ser la empresa instaladora de paneles solares más importante de Estados Unidos, no solo abriendo nuevas centrales fotovoltaicas, sino liderando el campo de la investigación de nuevos materiales y formas de almacenaje.

1.2. Motivación

Con la crisis actual de las energías no renovables se hace necesario encontrar una energía limpia, segura, pero sobretodo fiable. El reto no consiste en el desarrollo de la tecnología detrás del aprovechamiento de la energía solar, sino en la predicción a corto y medio plazo de la producción de energía. Esto es debido a que la energía solar, al igual que el resto de energías renovables, no puede ser almacenada, por lo tanto es necesario diseñar un sistema que permita aproximar cuanta energía puede garantizar una central fotovoltaica a lo largo del tiempo. Para esto, se han tenido en cuenta trabajos realizados anteriormente sobre el tema, siendo los sistemas de aprendizaje automático unos de los más extendidos. Dentro de esta categoría, es común encontrar enfoques como Redes de Neuronas Artificiales (RNAs) [10, 11], Máquinas de Soporte Vectorial (Support Vector Machines, SVMs)[15, 16] o Redes Bayesianas [3].

Por todo lo descrito anteriormente, la motivación de este trabajo consiste en proponer un nuevo sistema que sirva para mejorar la predicción de la producción fotovoltaica y, de este modo, potenciar el uso de este tipo de energía.

1.3. Objetivos

1.3.1. Objetivo Principal

Actualmente, los métodos de predicción clásicos en el que se aproxima el valor real, no tienen en cuenta la incertidumbre de dicho valor. Por ello, el objetivo principal de este trabajo fin de grado es el de construir un sistema de predicción de intervalos dadas unas variables meteorológicas previas con los que se pueda aproximar un margen en el que se puede asegurar que va a estar el valor predicho. Se creará una serie de modelos capaces de generar dos valores numéricos que servirán de límites para el intervalo, con el cual se valorará esta incertidumbre basándonos en términos contrapuestos, la **anchura** y el **grado de confianza(reliability)**.

1.3.2. Objetivo Secundario

Obtener mayor conocimiento sobre el funcionamiento de los algoritmos de computación evolutiva, así como de la regresión cuantil. Además, se estudiará como afectan los distintos parámetros de estos algoritmos a su actuación.

1.4. Contenido de la memoria

Introducción: En esta sección se dará una imagen de la evolución de la energía solar a través del tiempo, exponiendo las ideas más generales del proyecto y los problemas actuales que este trabajo intentará dar solución.

Contexto del trabajo: En esta parte se explicarán los conceptos técnicos de los que se ha hecho uso durante el trabajo. Se dará una visión general de las redes de neuronas, y en particular del perceptrón multicapa, de la computación evolutiva, con los algoritmos genéticos y el algoritmo de evolucion diferencial y la regresión cuantil.

Sistema desarrollado: Se explicará el proceso que se ha seguido para modelar el sistema. Empezando por el lenguaje elegido, los paquetes de código usados, así como la elección del cromosoma y la función de fitness.

Validación experimental: En este apartado se explicará brevemente los experimentos realizados, la metodología seguida y se tratarán los resultados obtenidos, dando una explicación de éstos.

Conclusiones y trabajos futuros: Se extraerán las conclusiones a las que se ha llegado a lo largo del desarrollo del trabajo y el conocimiento adquirido. Por último, se tratarán lineas de actuación futuras basadas en estas conclusiones.

Capítulo 2

Contexto del trabajo

En este capítulo se va a explicar de forma más especifica algunos conceptos de los que ya se ha hablado anteriormente en la introducción y que se utilizarán para desarrollar las pruebas. El objetivo es, por lo tanto, la profundización en las partes técnicas, así como en el contexto histórico, imprescindible para entender estos conceptos.

2.1. Redes de neuronas

Las Redes de Neuronas Artificiales(RNA), son un modelo computacional inspirado en la estructura cerebral. Suponen un intento de emular como el cerebro, a través de las redes de neuronas biológicas, procesa la información. En términos generales, la estructura de una RNA se constituye por, una serie de neuronas de entrada, con los datos iniciales, una o varias neuronas de salida y una serie de capas intermedias llamadas ocultas. La conexión entre neuronas suele tener asociado un peso, que junto al valor de la neurona, se les realizará una serie de operaciones que nos dará el valor de la nueva neurona. En esta sección se explicará el funcionamiento básico de una red neuronas, y de forma mas exhaustiva, del perceptrón multicapa, red usada en este estudio.

2.1.1. Contexto Histórico

El nacimiento de la primera RNA vino de la mano de Warren S. McCulloch, neurocirujano, y Walter Pitts, lógico, en el año 1943. Intentan crear un modelo de análisis lógico, aplicando lo que en ese momento ellos creían que era el comportamiento de una neurona. Un sistema binario

donde una neurona al recibir un estimulo, se activaría si este superaba un umbral.

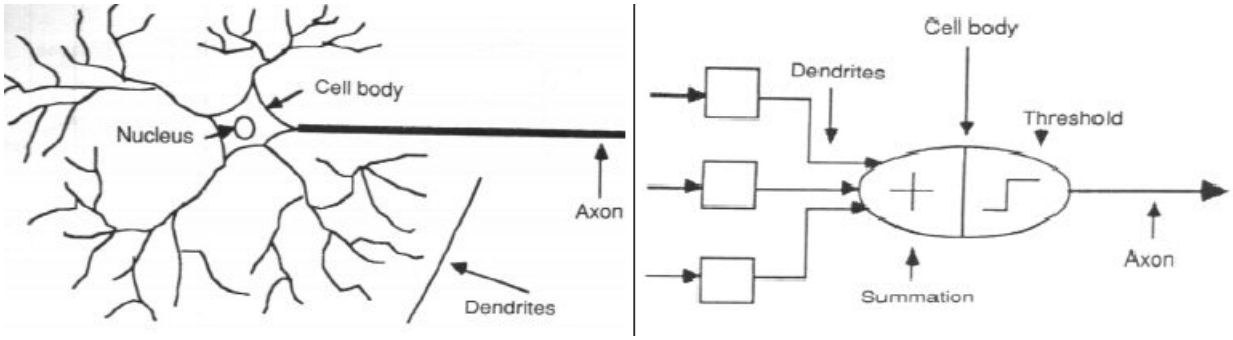

Figura 2.1: Comparación entre el modelo biológico y la red artificial

Como se puede ver en la figura 2.1, los datos de entrada del modelo, serían las señales dadas por otras neuronas, conectadas por las dendritas. Una vez llegan los estímulos a la neurona, estos se procesan, siendo representados por la operación que se puede apreciar en la imagen, se suman todas las entradas, y según un sistema binario, si el sumatorio es mayor que el umbral, la neurona dará 1, o 0 si no. En necesario aclarar, que en esta etapa, para los investigadores, las neuronas tenían un peso fijo, por lo que en principio, este primer modelo se llevó a cabo como un sistema de lógica simple, recreado en un circuito eléctrico. No fue hasta 1949 cuando Donald Hebb promovió la idea de que un canal neuronal se fortalecía cada vez que se usaba, dando puerta al desarrollo de las reglas de aprendizaje, con las cuales se podrían ajustar los pesos para mejorar la solución final. Esta idea, junto a las investigaciones anteriores, dieron pie al perceptrón simple, que, sin embargo, no pudo ser implementado dado las limitaciones computacionales de la época.

En 1959, dos investigadores de la universidad de Standford, Bernard Widrow y Marcian Hoff, diseñan la red ADALINE (ADAptative LINear Element), junto a la red MADALINE, fruto de la unión de varias redes ADALINE, usada en la época para encontrar patrones de radio. A diferencia del perceptrón simple, esta red tiene -1 y 1 como estado binarios, a diferencia del 0 y 1 del perceptrón simple, permitiendo mayor flexibilidad en el ajuste de los pesos. En 1962, Widrow y Hoff proponen una regla de aprendizaje en la que la variación del peso vendría dada, no por el estado obtenido, como en la regla del perceptrón:

$$\Delta w_{ij} = (\delta - y_{estado})x_i \tag{2.1}$$

donde δ es el estado deseado, donde y_{estado} representa el estado obtenido y x_i, el valor de la

neurona. Sino por el valor real obtenido en la salida:

$$\Delta w_{ij} = (\delta - y)x_i \tag{2.2}$$

donde y representa el valor real obtenido. Esta nueva regla se llamaría regla Widrow-Hoff, o regla Delta. Sin embargo, y pese a estos avances, el uso de las RNA cae en desuso, usándose MADALINE, solamente, para la eliminación de ruido en las transmisiones. Entre otros motivos, esto se debe a la imposibilidad de la comunidad académica de encontrar una red de neuronas multicapa supervisada y a que hasta ahora, se usaban funciones no diferenciables en todo su dominio, como la función escalonada [18].

A partir del año 1986, el concepto de RNA multicapa se une al algoritmo de retropropagación, derivado de la regla Delta, y de esta forma nace la primera red neuronal supervisada. Llamado de esta forma por ser capaz de transmitir el error a través de todas las capas de red, este algoritmo tenia una pega, serian necesarios muchos ciclos para optimizar los pesos [19].

En al actualidad, y con la mejora de la capacidad de cómputo, las RNA están presentes en muchos campos, tanto de investigación como financieros, con ejemplos como reconocimiento de caracteres de texto, concesión automatizada de créditos y financiación, coches automáticos o predicción de la bolsa. Además, a lo largo de este tiempo, han surgido más RNA, cada cual enfocado a una tarea distinta, como pueden ser los ya mencionados ADALINE y el perceptrón multicapa o los mapas autoorganizados de Kohonen y las de base radial [17]. Se espera, además, que ha medida que vaya avanzando al tecnología, surgan nuevas soluciones que mejoren la capacidad de las redes de neuronas o las amplíen. Entre estos avances se espera la integración de lógica difusa o la creación de hardware especifico, mejorando no solo la rapidez, sino el número de neuronas que pueden ser manejadas.

2.1.2. Estructura

Cuando nos referimos al término red dentro de las redes de neuronas artificiales, estamos hablando de las distintas partes que conforman esta y como están conectadas entre sí. De esta forma, es posible encontrarnos con modelos sencillos como un ADALINE, un perceptron multicapa, con capas ocultas, o mapas autoorganizados, donde las neuronas se disponen en una malla rectangular o hexagonal. Estas redes presentan rasgos comunes, cada neurona viene dada por un valor, uno numérico para las neuronas en un perceptron multicapa, la posición para una neurona en un mapa autoorganizado, y, además, cada unión entre neuronas tendrá asociado un peso, que condicionará su relación dentro de la red con sus neuronas vecinas.

Respecto a la estructura interna de una neurona, todas tienen un mismo patrón:

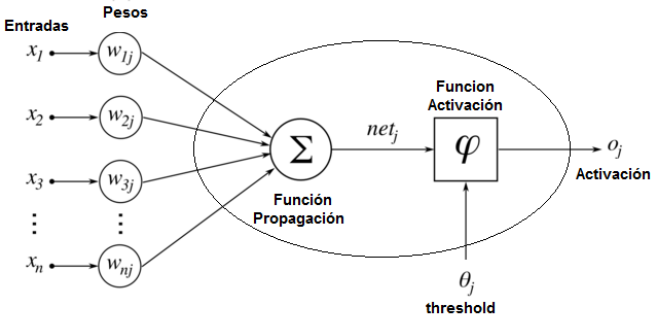

Figura 2.2: Estructura de una neurona artificial

La figura 2.2 nos muestra las distintas partes, a la neurona llegaran todos los valores desde las neuronas anteriores, recibiendo para ello una serie de valores, dados por la letra $x(x_1, x_2, x_3, ...x_n)$, y los pesos asociados a cada conexión, identificados por la letra $w(w_{1j}, w_{2j}, w_{3j}, ...w_{nj})$. Estos dos conjuntos de parámetros se multiplicarán entre ellos, y se les aplicará la función de propagación elegida, normalmente, el sumatorio, de la que obtendremos net_j:

$$net_j = \sum_{i=1}^{n} x_i w_{ij} \tag{2.3}$$

También es posible usar la distancia euclidea como función de propagación, aunque, solo para mapas autoorganizados o redes de base radial.

Sobre este valor se aplicará entonces la función de activación:

$$o_j = f(net_j) \tag{2.4}$$

donde o_j será la salida de la neurona.

En este punto es necesario hacer una pequeña reseña, se bien es posible usar cualquier función en esta parte, redes como el perceptrón multicapa necesitaran de una función derivable. Algunas de las funciones más usadas son:

- Función identidad:

$$f(x) = x - \theta, \tag{2.5}$$

  donde θ es un valor arbitrario. Esta función se usa normalmente con ADALINE, siendo fácil de implementar y rápido.

- Función escalón:

$$f(x) = \begin{cases} 0 & f(x) < \theta \\ 1 & \end{cases} \tag{2.6}$$

  donde θ es el umbral. Usada en el perceptrón simple, esta función solo puede dar dos valores, encendido o apagado.

- Función sigmoidal:

$$f(x) = \frac{1}{1 + e^{-x}} \tag{2.7}$$

- Función gausiana:

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}} \tag{2.8}$$

  Estas dos últimas funciones se usan en redes como el perceptrón multicapa, donde se necesitan funciones derivables.

2.1.3. Proceso de apredizaje

Aquello que hace que una RNA sea realmente útil es su capacidad para adaptarse, es posible decir que una red de neuronas aprende, es capaz de reconocer patrones, hacer estimaciones y agrupaciones, basándose en ejemplos a priori. Para representar esta adaptación una RNA irá transformando los pesos en las conexiones gracias a una regla de aprendizaje. Este proceso es cíclico, es decir, se repetirá hasta que los pesos en la red sean óptimos. Como esto normalmente es imposible, se aplican criterios de parada, siendo 3, los más importantes. Un número máximo de ciclos, tras los cuales, el aprendizaje se detendrá, una variación en el error cometido menor a un número arbitrario o los dos a la vez.

Elegir correctamente estos criterios de parada es fundamental, ya que puede llevar al sobreajuste, una situación en al que la red solo es capaz de ajustar los ejemplos para los que ha sido entrenada. Es necesario por ello probar la red entrenada con dos conjuntos de datos, los de entrenamiento y los de test. Estos dos conjuntos deben ser independientes y proporcionar en ambos casos una representación significativa de la población del problema.

Debemos distinguir entre dos tipos de entrenamiento, supervisado y no supervisado. En el primero se tiene en cuenta la salida deseada, que se compara con la obtenida para optimizar los pesos, la regla Delta generalizada o la regla regla Widrow-Hoff, basadas en el error cuadrático, son un ejemplo. En el segundo método no existe la salida deseada, y los pesos se ajustan mediante reglas, como las basadas en la distancia euclidea para las redes autoorganizadas. Es necesario añadir que, para el caso que nos ocupa, es posible valerse de algoritmos de optimización, como los genéticos y evolutivos, basados en mejoras de los individuos de una población, siendo estos individuos, una representación de los pesos de la red.

2.1.4. Perceptrón multicapa

A finales de los años 60, se llega a la conclusión de que las redes de neuronas simples como Adaline son incapaces de aproximar problemas no-lineales, como XOR, siendo necesario para ello una red con una serie de capas intermedias. En 1975 se propone el perceptrón multicapa, sin embargo, no es hasta 1986 cuando David Rumelharty y su equipo de investigación proponen un método de aprendizaje valido para optimizar los pesos de varias capas de neuronas, la regla Delta generalizada.

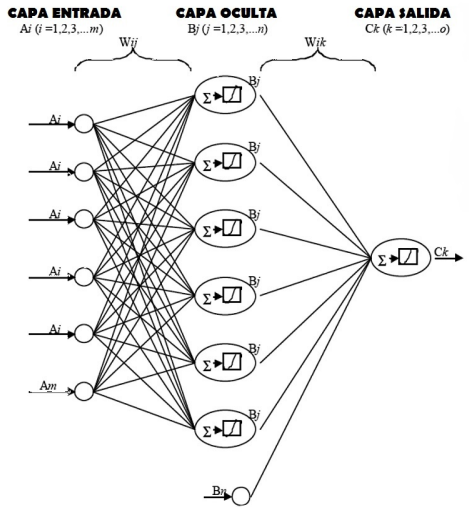

Figura 2.3: Ejemplo de estructura de una red de neuronas

Tal y como se puede ver en la figura 2.3, existen 3 partes. Cada neurona vendrá dada por un valor, tanto inicial o generado a partir de las neuronas anteriores, y conexiones con todas las neuronas de la siguiente capa, para cada una de las cuales se le asociará un peso. La primera es la capa de entrada, contiene los valores iniciales, dados por los atributos de cada ejemplo. La segunda sección representa las capas ocultas, en las que puede haber un número arbitrario de capas. Por último, la capa de salida, con un comportamiento similar a una capa oculta, del que se obtendrá el valor final.

A la hora de modelar esta estructura se debe tener en cuenta los siguientes aspectos:

- **Número de capas ocultas**: En una red multicapa puede existir un número arbitrario de capas ocultas, si bien, es posible obtener una red con una sola capa oculta que en al práctica resulte tan eficiente como una red con más capas ocultas.

- **Número de neuronas ocultas**: Hasta la fecha no existe una forma segura de determinar el número optimo de neuronas. Es por ello que para encontrar la mejor configuración se tenga que llevar una serie de pruebas de ensayo y error que la determinen. En algunos casos se han utilizado algoritmos genéticos para encontrar el número más adecuado [22].

Para este tipo de red procesará los datos de entrada, que se irán propagando por todas las neuronas de la red, siguiendo para ello el método anteriormente descrito, se sumarán los productos del valor de la neurona con el peso asociado, de todas las neuronas de la capa anterior y se le aplicará una función de activación:

$$o_j = f(\sum_{i=1}^{n} x_i w_{ij} + u_j) \tag{2.9}$$

donde u será el umbral de la neurona. Hay que destacar que para este tipo de red neuronal se usará una función derivable, como la sigmoidal. Este paso se repetirá hasta obtener el valor de las neuronas de la capa de salida.

Por último, una vez se ha obtenido la salida, es necesario realizar el aprendizaje. Para ello, se usará la regla Delta generalizada, basada en el error cuadrático. En este método, el error se irá propagando de atrás a adelante, optimizando los pesos que minimicen este error.

2.2. Computación Evolutiva

La computación evolutiva es una subfamilia de algoritmos de la inteligencia artificial basados en problemas de optimización, que a su vez, estará formada por tres campos, los algoritmos genéticos, la programación evolutiva y las estrategias evolutivas. En esta sección, se abordará su historia, su aparición y posterior evolución hasta la actualidad, se tratará en detalle el funcionamiento de los algoritmos genéticos, y por último, se explicará el método de evolución diferencial.

2.2.1. Contexto Histórico

Es posible encontrar estudios sobre computación evolutiva desde los años 50, gracias, entre otras, a las publicaciones de Bremermann o Friedberg. La idea principal partía de la aplicación de los principios adscritos por Charles Darwin sobre la adaptación y evolución de las especies. Por esto, este tipo de algoritmos siempre serán problemas de búsqueda heurística, ya que siempre es necesario elegir los mejores candidatos, al igual que en al evolución biológica. Sin embargo, su uso quedo relegado al campo teórico durante los siguientes 30 años, sobretodo, por la entonces falta de maquinas con el suficiente poder de cómputo.

Si bien durante los años setenta surge un gran número de publicaciones sobre el tema y es en este momento en el que surgen las tres familias que componen la computación evolutiva y que ya han sido enumeradas anteriormente. Los algoritmos genéticos son descritos por primera vez por Holland, labor que continuó Goldberg, en un intento de crear un modelo para procesos adaptativos, aunque evolucionó para convertirse en un optimizador de uso universal. A su vez, las

estrategias evolutivas fueron introducidas por Rechenberg y Schwefel con la idea de solucionar problemas de optimización, discretos y continuos. Por último, la programación evolutiva fue tratada por primera vez por Fogel y más tarde por Burgin, en la búsqueda de una autómata finito predictivo dependiendo de los datos de entrada. [1]

Estas tres ramas evolucionaron de forma independiente hasta que en los años 90 los distintos investigadores reunieron esfuerzos para crear un frente común en el cual pudiesen interactuar. Estos esfuerzos hicieron posible la creación en 1991 del "Parallel Problem Solving from Nature", una conferencia que se sigue celebrando en la actualidad, y que permite a investigadores del mundo entero intercambiar información sobre los últimos avances en el campo de la Computación evolutiva.

En la actualidad, este campo sigue creciendo y cada vez es más normal el uso de estas técnicas en el día a día, gracias, entre otras cosas, al crecimiento en la capacidad de cómputo y la mejora de los algoritmos. Entre sus aplicaciones se cuentan la optimización de redes de comunicación, teoría de juegos, ingeniería económica o los modelos macroeconómicos [9].

2.2.2. Algoritmos Genéticos

Posiblemente la rama de la computación evolutiva más conocida y más usada. La primera mención de éstos algoritmos se puede encontrar en los estudios realizados por Holland en 1962. Holland habría diseñado una serie de sistemas adaptables según las interacciones con el medio, para lo cual, este algoritmo se basaría en modelos competitivos de la supervivencia del más apto. De esta forma, Holland ideó un algoritmo en el que una serie de individuos, las soluciones a nuestro problema, compitieran entre ellos, siendo los ganadores los que darían lugar a la siguiente generación. A medidos de los 60, Holland y su equipo empezó a ver a esta población de soluciones posibles como una representación de los individuos de una especie, evolucionando a través del tiempo. Se creó de esta forma la analogía en la que estos individuos se codificaron en cromosomas, secuencias de unos y ceros, basándose para ello en la secuencia de ADN. Para poder obtener el mejor individuo, Holland incluyó un valor, el **fitness**, que haría posible la competición entre individuos. Por último, se llego a la conclusión que la herencia entre individuos no era más que la abstracción de operadores genéticos como la mutación, cruzamiento y la inversión, obteniendo así el método de generación de nuevos individuos [2].

Holland llegó a la conclusión de que existían tres características imprescindibles que distin-

guían a los algoritmos genéticos de otras técnicas de computación evolutiva:

- Las distintas soluciones se representarán mediante una serie de bits.
- Los individuos se seleccionaran mediante un método de selección de ruleta, basada en su valor fitness.
- El principal método de producción de nueva población será el cruzamiento

En la actualidad estas 3 características se mantienen pero se han añadido, además, otros métodos se selección y producción de los que se hablara màs adelante.

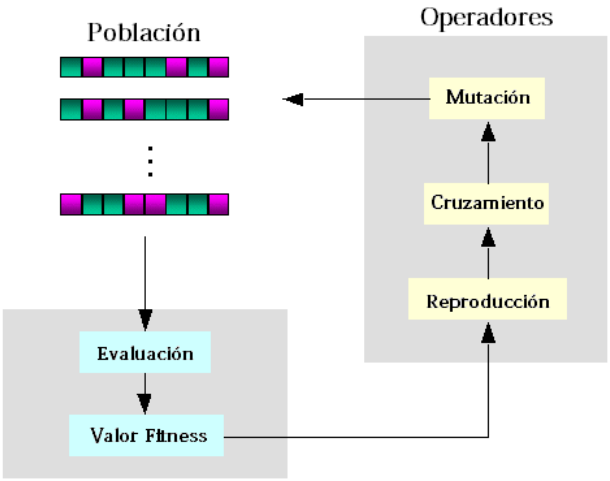

Figura 2.4: Esquema del proceso de un algoritmo genético

Podemos decir entonces que los algoritmos genéticos están formados por una serie de individuos a los que llamaremos fenotipos, es decir, la solución antes de ser codificada en los cromosomas, formados por una cadena binaria, siendo cada dígito un gen, en los que se codificarán los parámetros que servirán para medir la conveniencia del individuo, es decir, su fitness. Una vez codificados en estos cromosomas, se procederá a emular algunos de los comportamientos de la evolución biológica, para de esta forma producir una nueva población. Algunos de los operadores, como se puede ver en la figura 2.4 usadas son:

- **Selección:** Mediante esta técnica elegiremos a aquellos individuos que se vayan a ser cruzados en esta generación. Para ello se usara el valor del fitness y una serie de mecanismos:

- Selección de ruleta: según el valor de fitness obtenido para un cromosoma, se obtiene la probabilidad de cruce mediante la siguiente función:

$$p_n = \frac{f_i}{\sum_{j=1}^{N} f_j} \tag{2.10}$$

donde f_i es el valor de fitness del cromosoma i, N el número de cromosomas en la población y p_n la probabilidad de cruzamiento del cromosoma n-semo. De esta forma, obtendremos una serie de porcentajes que premiarán al individuo con mayor fitness, pero no impedirán que los peores salgan elegidos. Sin embargo, en casos extremos en los que solo unos pocos miembros de la población destaquen del resto, podría existir el riesgo de perdida de diversidad y una posible convergencia prematura, en la cual, todos los individuos serán copias de ellos mismos.

- Selección por torneo: se usará una pequeña muestra de individuos al azar que competirán entre ellos, siendo elegidos aquellos con mejor valor de fitness. Una de las bondades de este método es su rapidez, ya que solo se evalúa parte de la población y la reducción de la posibilidad de convergencia prematura..

- **Mutación:** En esta técnica se cambian un número al azar de bits en la cadena. Esta opción es importante ya que permite obtener individuos que de otra forma nunca se podrían generarse a partir de la población existente.

- **Cruzamiento:** Es el equivalente a la reproducción sexual biológica, en la cual, dos cromosomas se juntan para formar dos descendientes con características de ambos. Existen 6 modalidades de cruzamiento:

  - Recombinación en un punto: En esta modalidad se elige un punto en la cadena y se corta por ahí, intercambaindo el primer trozo del cromosoma A por el primer trozo del cromosoma B.

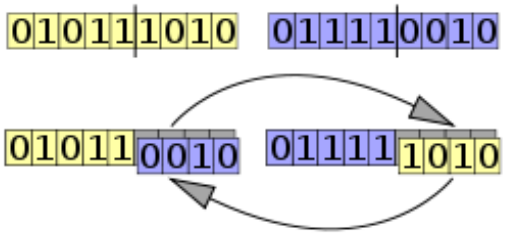

Figura 2.5: Recombinación en un punto

- Recombinación en dos puntos: Similar a la anterior, pero en esta ocasión se elegirán dos puntos, obteniendo 3 partes, se intercambiará la central del cromosoma A con el del cromosoma B.

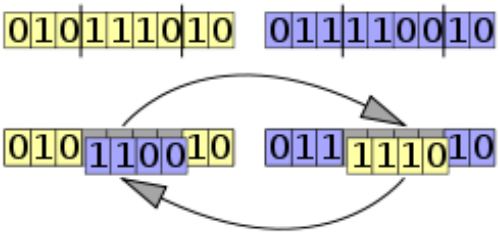

Figura 2.6: Recombinación en dos puntos

- Corte y empalme: En este caso se elige un punto de corte para cada cromosoma, ocasionando la creación de hijos con distinto número de genes.

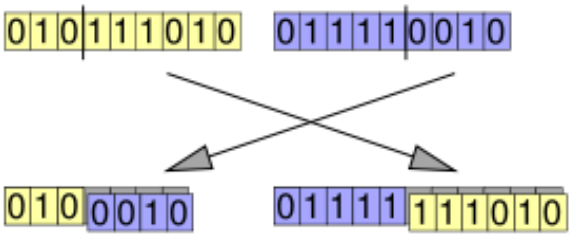

Figura 2.7: Corte y empalme

- Recombinación uniforme: En esta modalidad se valoran cada cada pareja de bits de forma individual. Si estos son diferentes, estos se intercambiarán según un porcentaje acordado a priori.

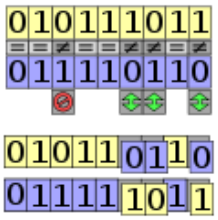

Figura 2.8: Recombinación uniforme

- Recombinación media: Similar al anterior, pero en este caso, se deberá contar el nú-

mero de parejas de bits diferentes, y se procederá a intercambiar la mitad de ellos.

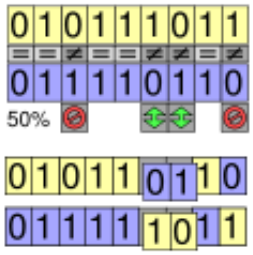

Figura 2.9: Recombinación media

- Recombinación de cromosomas ordenados: Este caso es especial ya que solo se usa cuando el cromosoma esta formado por genes ordenados. En esta modalidad se impedirá que existan duplicados en los individuos hijo. Es especialmente usado en problemas de optimización como el problema del viajero, en el que no pueden existir ciudades repetidas.

- **Reemplazo:** En esta parte se elegirán los mejores individuos según su valor de fitness obtenido. Es necesario acordar una población máxima con la que trabajará el algoritmo en al siguiente generación.

Es necesario recalcar que para obtener el valor fitness se usará una **función de fitness**. Posiblemente la elección más importante a la hora de diseñar nuestro algoritmo genético junto a la codificación del fenotipo, elegir una buena función de fitness permitirá al algoritmo acercarse al resultado deseado.

Como resumen, en el algoritmo genético ideado por Holland podemos apreciar los siguientes pasos, que se realizarán en cada generación y se repetirán hasta alcanzar un un individuo óptimo:

1. Inicialización: En este primer paso se generará la población inicial de forma aleatoria o generada previamente, pero manteniendo la diversidad.

2. Evaluación: Se obtiene el valor para cada individuo de la función de fitness y se ordena.

3. Selección: Según el criterio de selección se eligen a los individuos que se van a cruzar.

4. Cruzamiento y mutación: Se generarán dos nuevos individuos que mutarán y serán introducidos a la población.

5. Se repetirá el paso 3 y 4 hasta que se alcance el tamaño máximo de población.

6. Reemplazo: Se eligen a los mejores individuos que seguirán en la siguiente generación.

7. Se repetirá todo desde el paso 2 hasta que se alcance el número de ciclos estipulado o se llegue a un valor de fitness satisfactorio.

2.2.3. Evolución diferencial

Es un método de optimización, desarrollado por Rainer Storn y Kenneth Price [20], y basado en los principios de la computación evolutiva. El algoritmo cumple los siguientes requisitos:

- Capacidad para tratar con funciones de coste no lineales y no diferenciables.

- Debe ser paralelizable, de esta forma, podrá hacer uso de los ordenadores más potentes hasta la fecha.

- Se compone de las suficientes variables como para que resulte fácil de usar, Así mismo, estas variables deben ser fáciles de elegir.

- No tiene problemas de convergencia, este algoritmo es capaz de encontrar el mínimo global.

Su funcionamiento esta fuertemente influenciado por los algoritmos genéticos, sin embargo, este tiene ciertas diferencias que lo hacen diferente. Seguramente la más importante es el uso de vectores de prueba (cromosomas), un vector resultado de la diferencia de pesos de dos vectores padre y la posterior suma con otro vector en la población. Además, los vectores están formados por cadenas de números reales, a diferencia de las cadenas de bits en los algoritmos genéticos.

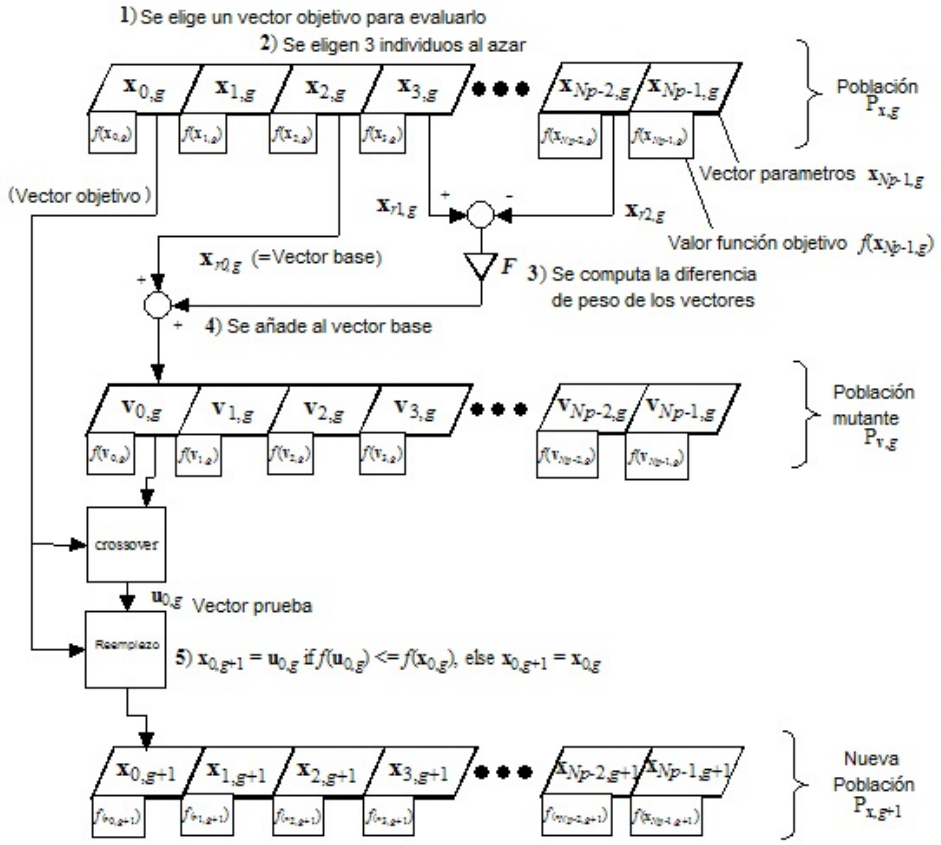

Figura 2.10: Esquema de la evolución diferencial

Tras observar la figura 2.10, podemos ver las similitudes con un algoritmo genético, siendo los pasos a seguir los mismos, exceptuando por la selección, que en este algoritmo es un paso innecesario como se explica a continuación:

1. Inicialización: Se crean los vectores iniciales, que tal y como detallan Storn y Price, deberán cubrir todo el espectro de parámetros posibles.

2. Mutación: Se generará un vector de prueba para cada individuo de la población. Para esto, se elegirá cada vez 3 vectores de forma aleatoria de entre los miembros de la población. El vector de prueba será la diferencia de peso entre dos de esos vectores, sumado al tercer vector.

3. Cruzamiento: Este vector de prueba generado para cada individuo será sometido a cruzamiento con su vector objetivo, es decir, el vector que se esta evaluando.

4. Reemplazo: Se obtendrá el fitness tanto del vector objetivo (x_i, G, donde G es la generación

actual) como del nuevo vector prueba $(u_i, G+1)$. De aquí obtendremos que:

$$x_i, G+1 = min((x_i, G), (u_i, G+1)) \tag{2.11}$$

siendo $x_i, G+1$ el nuevo valor para el vector objetivo i.

5. Se repetirá todo desde el paso 2 hasta que se alcance el número de ciclos estipulado o se llegue a un valor de fitness satisfactorio.

Como se puede ver, el paso de selección no es necesario ya que el candidato se crea a partir los miembros de la propia población y el vector objetivo siendo evaluado.

2.3. Regresión Cuantil

Definimos un cuantil como un punto dentro de una función de distribución de una variable aleatoria. Para una serie de q cuantiles, estos dividirán una serie de datos ordenados en q subconjuntos, siendo cada uno de estos cuantiles el valor que servirán de borde entre cada uno de estos subconjuntos Es decir, dentro de una distribución, un cuantil de orden p con $(0 < p < 1)$, dividirá los valores en dos grupos, aquellos por debajo de p y aquellos por encima.

Desarrollado por Roger Koenker [5], la regresión cuantil es un método por el cual se puede predecir un cuantil cualesquiera dadas una serie de variables. Dado un valor p, la regresión cuantil predecirá el valor x en una distribución, de esta forma, para un cuantil con valor $p = 0{,}5$, el resultado sería la mediana, es decir, el valor que dividiese en dos la distribución. A diferencia del método de los mínimos cuadrados que estima una función según la variación de la media, los resultados obtenidos con este método serán más robustos ante valores atípicos, y permite estimar cualquier cuantil, permitiendo así un estudio de un punto en cualquier lugar de la distribución.

2.4. Estimación de intervalos

En la estimación de la energía generada por una central fotoeléctrica, se debe tener en cuanta una de las limitaciones de este tipo de energía, no se puede almacenar. Debe existir un método estadístico para predecir la energía para que, de esta forma, se pueda realizar un plan en el cual se optimice la energía en el sistema. Cunado se predice este valor con un método clásico de regresión, la incertidumbre, el error cometido, es demasiado grande como para que una empresa pueda usar este valor con seguridad.

Por este motivo algunos autores han señalado el uso de intervalos de predicción para paliar el problema de la incertidumbre [12, 25]. Este tipo de intervalos vienen dados por parámetros que permiten medir la incertidumbre, e incluso, ajustarla. Además, al poder ajustar con mucha seguridad el limite inferior y superior en los que sabemos que va estar valor a predecir, sólo se necesitaría sumar al límite inferior del intervalo la energía almacenada que se va a necesitar para estar seguro de que va a suplir a la red eléctrica.

Los dos parámetros que definen un buen intervalo de predicción son la cobertura y la anchura del intervalo. El objetivo principal es, por supuesto, una cobertura del 100 %, es decir, que todas los valores se encuentren dentro del intervalo, mientras se mantiene la menor anchura posible. Ambos parámetros resultan, a veces, incompatibles entre si, ya que resulta muy sencillo que el valor se encuentra dentro del intervalo si la anchura ocupa todo el dominio, y al revés, si la anchura es demasiado pequeña podemos encontrarnos con una cobertura demasiado baja.

2.5. Lenguaje R

R es un lenguaje por procedimientos orientado a la computación estadística. Fuertemente basado en el lenguaje S, que fue diseñado por John Chambers en Bell Laboratories, pero bajo una licencia gratuita dentro del proyecto GNU. Es usado principalmente en el campo de la minería de datos y la estadística por su capacidad de procesar grandes cantidades de datos, la creación de gráficos y su extensibilidad, ya que es posible crear paquetes que aumenten sus capacidades. Gracias a esto, R se ha convertido en los últimos años en el software estadístico más usado, como muestra la figura 2.11.

Entre sus ventajas se encuentran el soporte de aritmética de matrices, es decir, es un lenguaje orientado a trabajar con matrices, con rendimientos equiparables a Matlab. Es un lenguaje escrito en C++ y Fortran, por lo que tiene una fuerte estructura orientada a objetos con funciones genéricas, algo poco común en los lenguajes estadísticos. Esta característica hace posible que un mismo método se comporte de forma diferente según el objeto que se le pase por argumentos. Además, cuenta con integración con Latex a través de la función **Sweave**[7], siendo posible generar documentos con gráficas y tablas actualizadas cada vez que los datos de análisis cambien o se actualicen.

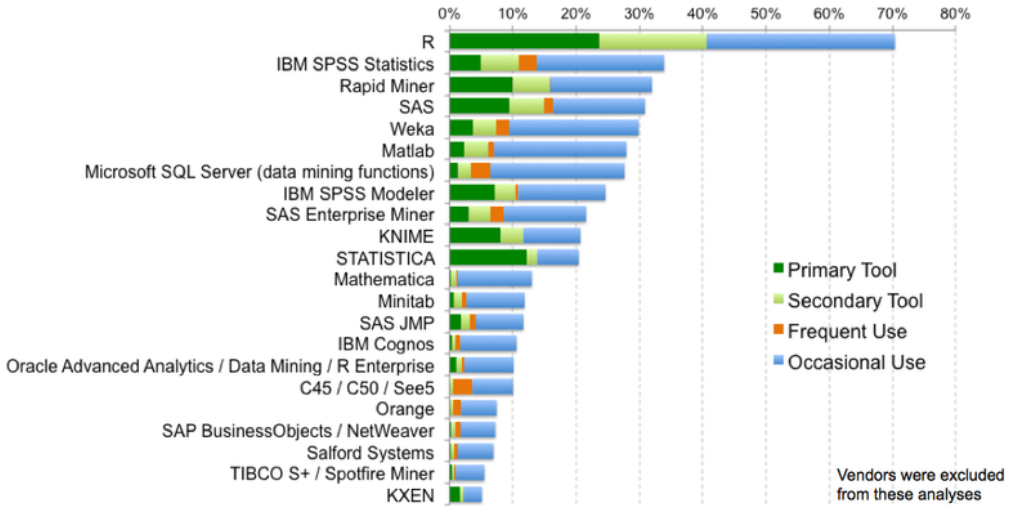

Figura 2.11: Porcentaje de uso de lenguajes de computación estadística

3 motivos primaron en la elección de este lenguaje sobre los otros:

- Es un lenguaje gratuito, mejorado por la comunidad del proyecto GNU de forma altruista, al igual que los paquetes que lo componen.
- La capacidad para añadir librerías o paquetes lo convierte en un lenguaje de las magnitudes de otros más pesados como puede ser MatLab.
- Tiene integración con Latex a través de Sweave. Esto ha permitido la creación de partes de la memoria, como gráficas y tablas, en tiempo real.

Para la realización del código se utilizará la versión 3.2.1 y además se usará RStudio, un IDE con una interfaz más moderna y fácil de usar, que como característica más importante, permite la ejecución de Sweave de forma nativa y el gestionado de paquetes.

Por último, se usarán los siguientes paquetes para la implementación de la red de neuronas, los algoritmos genéticos, el algoritmo de evolución diferencial y la regresión cuantil.

2.5.1. Paquete nnet

El paquete "nnet"[23] implementa una red de neuronas multicapa, con una cada oculta. Este paquete permite la elección del número de capas ocultas, de criterios de parada como el número de iteraciones máximo o el umbral de menor error a batir. Este paquete ya incluye la técnica de la regla Delta generalizada que actualizará los pesos. Para la generación del modelo

se usará la función "nnet", que tomará como parámetros, los datos de entrenamiento, el número de iteraciones máxima y el número de neuronas ocultas. Una vez generado el modelo se usará la función "predict"que tomará como argumentos el conjunto de datos de test y el modelo generado anteriormente.

2.5.2. Paquete GA

El paquete "GA"[14] constituye una herramienta sencilla y muy completa con los aspectos más generales de los algoritmos genéticos. Permite fácilmente cambiar los parámetros de la mutación, selección, cruzamiento, los criterios de parada y otros. La función más importante del paquete será "ga", que producirá un objeto conteniendo los datos más importantes del modelo optimizado, entre ellos, el cromosoma ganador. Esta función tomara una serie de parámetros para su funcionamiento:

- **Tipo:** Definirá el tipo de valores que compondrán los cromosomas, "binary", para valores binarios, real value", para valores reales y "permutation", para listas ordenadas.
- **Función:** La función de fitness para la selección de candidatos.
- **Min y Max:** Valores máximos y mínimos que pueden alcanzar los parámetros en los cromosomas.
- **Population:** Generará una población inicial dado un cromosoma inicial.
- **maxIter:** El número de iteraciones máximas.

2.5.3. Paquete deOptim

El paquete deOptim [13] contendrá todo lo necesario para construir un algoritmo de evolución diferencial. Será necesario usar la función "DEoptim"que creara un objeto con el valor de los mejores vectores en cada generación y el mejor vector. Esta función necesitará los siguientes atributos,

- **Función:** La función que evaluará el peso de cada vector.
- **Lower y Upper:** Valores máximos y mínimos que pueden alcanzar los parámetros en los vectores.

- **Control:** Una serie de valores no necesarios ya que existen por defecto, sin embargo, se cambiarán los valores de iterMax, para definir la generación máxima, NP, para definir el tamaño de la población, e initialPop para definir la población inicial.

2.5.4. Paquete quantreg

El paquete quantreg [6] es una herramienta con la implementación integra de la regresion cuantil, programada por Roger Koenker, el principal instigador de la regresión cuantil en el siglo XX, y creador del método actual para estimar cuantiles. Se usarán las funciones rqz "predict", la primera generará el modelo para los atributos de entrenamiento, para lo cual se elegirán dos valores para τ, correspondientes a los cuantiles a predecir, el primero servirá para predecir el margen inferior del intervalo y el segundo el superior. Una vez construido el modelo, se usará "predict", con los atributos de test como parámetro.

Capítulo 3

Sistema desarrollado

En este capítulo se describirán todas las consideraciones tomadas antes de hacer la experimentación y de escoger los conjuntos de datos. Se explicará el método para obtener el intervalo de confianza, los cromosomas y la función de fitness elegida y el porque de las decisiones tomadas para su modelado ya que una buena elección de éstos parámetros asegura el buen funcionamiento de los algoritmos evolutivos. Además, se hará un repaso del código implementado, con una reseña hacia los paquetes usados para incluir algoritmos genéticos, redes de neuronas, algoritmos de evolución diferencial y la predicción de cuantiles con la regresión cuantil.

3.1. Introducción

El fin del sistema desarrollado es de construir un modelo capaz de generar intervalos de confianza para un conjunto de datos. Para la creación de este sistema de predicción de intervalos contaremos con una perceptrón multicapa, con dos salidas, que corresponderán a los límites inferior y superior del intervalo. Esta red se entrenará usando técnicas de computación evolutiva, un algoritmo genético y un algoritmo de evolución diferencial, para lo cual será necesario diseñar una función de fitness que evalúe los candidatos posibles, los cromosomas.

3.2. Cromosoma

Posiblemente la elección de la estructura del cromosoma sea una de las elecciones más importantes a la hora de diseñar un buen un buen algoritmo evolutivo. Este cromosoma tendrá una longitud de 9092 genes, dada por los pesos de la red, 301 pesos de cada neurona de entrada a

las 30 neuronas ocultas y 31 neuronas ocultas, contando las neuronas umbrales, por el número de salidas, tal y comos e muestra en el figura 3.1. Estos 9092 pesos se codificarán como valores reales, con valores comprendidos entre -1 y 1. Por último, y para darle mayor flexibilidad al algoritmo, se toma como máximo y mínimo para cada gen, 100 y -100, respectivamente, aunque en las pruebas realizadas ningún gen esté fuera del primer rango.

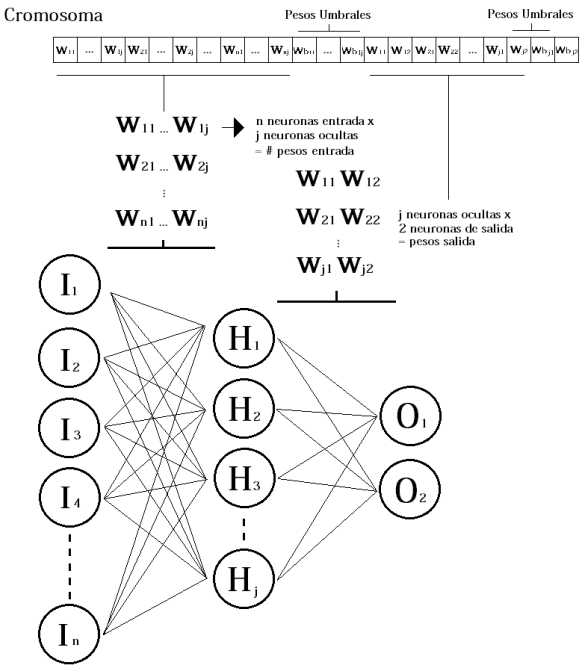

Figura 3.1: Codificación cromosoma

Se adopta esta medida ya que es necesario codificar los pesos de una red de neuronas como valores reales para que esta se ajuste correctamente. Aunque es posible codificar cada valor real, en nuestro caso coma flotante, en una string de bits, usando 32 bits para cada valor, si seguimos la notación dada por IEEE para precision simple, dejando un cromosoma con una cadena de 290944 bits de longitud. Sin embargo esta medida es contraproducente ya que para un algoritmo evolutivo, los cromosomas más grandes suponen un mayor tiempo de cómputo y perdida de fiabilidad, además, en varios experimentos, el uso de valores reales frente a cadenas binarias en redes neuronales da mejores resultados [24].

3.3. Función de Fitness

Otro aspecto imprescindible para que un algoritmo evolutivo sea realmente eficiente es la elección de una buena función de fitness. En el caso que nos ocupa nos basaremos en el acercamiento llevado acabo por Can Wan et al. en el artículo "Direct interval forecasting of wind power"[25]. En este estudio se trata de obtener un intervalo de confianza utilizando una **Extreme Learning Machine**, una red neuronal de una sola capa oculta usada para clasificación o regresión. Su particularidad radica en que los pesos de las conexiones entre las neuronas de entrada y la capa oculta se eligen aleatoriamente y no se actualizan, mientras que los pesos restantes se obtienen de forma analítica.

En el estudio estos últimos pesos son optimizados en base a dos criterios de evaluación de los intervalos de predicción, la anchura y la media del error de cobertura (**ACE**, Average Coverage Errors). La primera mide la diferencia entre el límite superior del intervalo y el inferior, cuanto más pequeña, será un intervalo más preciso. El segundo mide la diferencia entre el número de mediciones que se encuentran dentro del intervalo de predicción (**PICP**, Prediction Interval Coverage Probability)), y el valor nominal de la cobertura que se quiere conseguir(**PINC**, Prediction Interval Nominal Confidence). El PICP vendrá dado por la función:

$$PICP = \frac{1}{N_s} \sum_{i=1}^{N_s} k_i \tag{3.1}$$

donde N_s representa el número de valores de salida y k_i el valor condicionado a si la salida i se encuentra dentro del intervalo de predicción, que puede tomar 1 si lo esta o 0 si no. Mientras que el PINC será un porcentaje dado por:

$$PINC = (100 - \alpha)\,\% \tag{3.2}$$

donde α será un valor entre 0 y 100 que indicará que porcentaje de los valores pueden estar fuera del intervalo, de esta forma, podemos eliminar las salidas irregulares. Tal y como dice el artículo, PICP y PINC tenderán a igualarse, si el valor de PICP es menor que PINC querrá decir que ha habido un menor número de aciertos de los esperados, mientras que si es mayor, querrá decir que ha habido demasiados aciertos, empeorando otros aspectos del intervalo, como la anchura.

En el artículo se usa un objetivo F, dado por la suma de la anchura y el ACE, multiplicados ambos por unos coeficientes que dependerán de a cual de los dos parámetros se le quiere dar más importancia. Nuestra función de fitness seguirá la misma premisa.

Ambos métodos usarán una red de neuronas con una sola capa oculta que genere los intervalos de predicción, pero en el nuestro se usará un algoritmo evolutivo para optimizar los pesos. En la función de fitness se diseñara la red que nos de los limites inferior y superior de los intervalos para cada valor, con los cuales se averiguara la anchura, como la media de la diferencia de todos los pares de predicciones y el ACE, calculado tal y como se ha explicado antes. Usaremos entonces un coeficiente entre 0 y 1 para otorgar importancia a uno de los dos parámetros. Como se quiere que el resultado de la función de fitness se encuentre en un rango entre 0 y 1, el coeficiente para la anchura será igual a este coeficiente - 1.

La función de fitness será la siguiente:

$$fitness = (\tau * anchura) + (\tau - 1) * (min(0, ACE)) \tag{3.3}$$

donde τ corresponderá al coeficiente.

Cabe destacar por último, que como se puede ver en la función 3.3, ACE será 0 si este es positivo. Esto es debido al cálculo de ACE (PICP - PINC) en el cual si el número de aciertos es mayor del esperado puede resultar positivo. De permitir este valor se reduciría el valor de la función, lo cual no es deseable, por lo que es necesario penalizar este comportamiento. En la práctica, el valor óptimo para ACE se alcanzará cuando PICP = PINC.

3.4. Implementación

Se realizarán dos programas, por un lado, el primero servirá para hacernos una idea del error al predecir el valor de la energía producida y para decidir el número de neuronas ocultas para el segundo programa. Aquella configuración que obtenga el menor error absoluto para el mismo número de iteraciones será la elegida. En el segundo, se implementarán tres métodos para generar los intervalos de confianza. Para los dos primeros se modelará, de nuevo, una red de neuronas, pero esta vez, solo será necesario la parte de predicción de la salida, es decir, no se actualizarán los pesos ya que estos serán optimizados por el algoritmo genético, en el primer método, y por el algoritmo de evolución diferencial para el segundo. Esta red generará dos salidas,

los limites inferior y superior de cada intervalo, que luego servirán para la función de fitness. El tercer método, la regresión cuantil solo necesitará una llamada al paquete correspondiente con los valores de τ deseados.

En primer programa se usará el paquete "nnet"para construir el perceptron multicapa. El primer paso será el de normalizar los datos de entrada y salida para el rango [0, 1], para ello se usara la siguiente fórmula:

$$XNor_{ij} = \frac{X_{ij} - min(X_j)}{max(X_j) - min(X_j)} \tag{3.4}$$

donde $XNor_i$ sería el valor normalizado, X_j el vector j de la matriz de datos de entrada o salida, y X_{ij} un valor dentro del vector.

Para automatizar el proceso usaremos la función çaret", que incorpora este método de pre-proceso de datos, entre otros. Una vez se han normalizado los 4 conjuntos de datos, salida y entrada de los valores de entrenamiento y test, se pasa a entrenar la red, para esto se optimizarán los pesos y se calculará el error tanto de entrenamiento como de test en cada ciclo, para llevar esto a cabo, se usara un bucle for, en el que cada ciclo sea una iteración, de esta forma es posible graficar el error cometido en cada ciclo. De esta forma será posible observar la 1) la evolución de los errores, 2) si existe sobre-entrenamiento.

En el segundo programa, se comenzará creando la función de fitness, como ya se ha explicado antes, vendrá dada por el resultado de una red neuronal y el valor del PICP y la anchura. Para crear la red se seguirán los siguientes pasos:

1. Se crean 3 matrices, la primera, de dimensión 301 x 1, contendrá el número de entradas más el umbral, los otros dos, de dimensiones 301 x 30 y 31 x 2, contendrán todos los pesos de la red, aquellos desde las neuronas de entrada a las ocultas y desde las ocultas a las de salida.

2. Se obtienen los valores de las neuronas ocultas, para ello se sigue el método de sumatorio del peso de la conexiones por el valor de las neuronas de entrada y se le somete a la función sigmoidal. Como ya se ha dicho, R es un lenguaje que trabaja mejor con matrices, por lo tanto podemos ordenar estas para que el proceso sumatorio del producto de las conexiones por los valores de las neuronas sea igual al **producto matricial**. Una vez obtenemos la nueva matriz, le aplicamos la función sigmoidal.

3. Se vuelve a repetir el paso anterior para obtener los valores de las neuronas de salida.

El código resultante se puede ver en la siguiente sección:

Listing 3.1: Implementación del perceptrón multicapa

```
#Se aniaden los valores de entrada y el umbral
inputs <- data.matrix(train_file)
inputs <- cbind(inputs, c(1))

#Se crean las matrices para los pesos y se rellenan. Como estos
#se almacenan en una lista entre iteraciones, se cogen los 9020
#primeros y los 62 ultimos
inputWeights <- matrix(par[1:capaentrada],ncol=ocultas)
hiddenWeights <- matrix(par[(capaentrada+1):length(par)],ncol=salidas)

#Se realiza el producto matricial y la funcion sigmoidal, tanto
#para sacar la matriz de valores para la capa oculta como para
#la matriz de salida.
hiddenValues <- sigmoid(inputs%*%inputWeights2)
hiddenValues <- cbind(hiddenValues, c(1))
outputs <- sigmoid(hiddenValues%*%hiddenWeights2)
```

De esta forma se obtienen los limites inferior y superior del intervalo de confianza para los pesos dados que permitirán producir la anchura y el ACE. Para este primer valor se calcula la diferencia entre los pares de valores obtenidos y se realiza la media. En cuanto al segundo valor, se utilizará la ventaja de R para el cálculo matricial, se crearán dos matrices nuevas, la primera vendrá dada por la diferencia entre las salidas reales y los valores para el rango inferior, para el segundo, se calculará lo mismo pero para el límite superior del intervalo. Se procederá a multiplicar ambos vectores entre sí. Para aquellos valores que sean positivos significará que la salida asociada no estaba dentro del rango y se incluirán en un nuevo vector. Para obtener el PICP se dividirá la longitud de este último vector obtenido entre la longitud del vector de salida. Una vez obtenido el PICP se le restará el PINC para obtener el ACE. Se procederá entonces a

obtener el valor de fitness, como ya se ha especificado en el anterior apartado.

Una muestra del código usado se puede ver a continuación:

Listing 3.2: Funcion de fitness

```
lowrange <- train_out-outputs[,2]
upperrange <- train_out-outputs[,1]
r=lowrange*upperrange
in_range <- r[r<0]

PICP <- length(in_range)/nrow(train_file)
width <- mean(abs(outputs[,1]-outputs[,2]))
ACE <- PICP - PINC

fitness_function <- (abs(coeff*width)) + ((coeff-1)*(min(0,ACE)))
return(fitness_function)
```

A continuación se pasará a implementar los dos métodos de optimización de los pesos de la red, el algoritmo genético y el algoritmo de evolución diferencial. Se usarán los paquetes antes descritos.

Para comprobar la fiabilidad de los modelos generados por los dos algoritmos se pasarán los datos de test y de entrenamiento por la red de neuronas, para los cuales se creará una gráfica con las salidas reales de ambos conjuntos y sus intervalos de confianza, así como una tabla con los valores para la función de fitness, de la anchura, y el ACE, así como al desviación típica, usada para evitar una gran disparidad entre las anchuras.

Por último, se implementará la regresión cuantil, para la cual usaremos el paquete "quantreg", en el que solo necesitaremos definir 3 variables, el conjunto de datos de entrenamiento, el conjunto de test y el valor de τ, en nuestro caso, un vector con dos valores que corresponderán a los limites inferior y superior. Para generar los pares de valores para el intervalo de confianza se usará la función rq", que generará el modelo, con el conjunto de entrenamiento y los valores de τ como parámetros, tras la cual, se usara la función "predict", con los valores de test. Una vez obtenidos estos pares de valores, se comprobará su validez usando la función de fitness del apartado 3.2, y se crearán las gráficas y tablas de la misma forma que para los anteriores algoritmos.

Capítulo 4

Validación Experimental

En esta sección se detallarán los datos usados, así como las variables con las que se ha querido hacer la experimentación, número de neuronas ocultas, coeficientes, número de ciclos y grado de confianza. Por último, se hará un análisis de los resultados obtenidos para cada modelo y un análisis de éstos en su conjunto.

4.1. Descripción de los datos

Para la realización de las pruebas se ha usado un conjunto de datos de entrenamiento y test proveniente de la competición para la predicción de energía solar, organizado por la .American Metereological Society". Originalmente esta competición estaba enfocada en la predicción de energía para cada día especificado por la dirección de la competición, sin embargo, estos datos serán usados para la estimación de intervalos.

Los datos se componen de 16 mediciones tomadas cada 5 minutos, en 16 estaciones meteorológicas dentro de la red de Oklahoma. Estas mediciones corresponden a 16 puntos repartidos en una cuadricula dentro del territorio donde va a estimar un intervalo. Sin embargo, para el desarrollo de nuestro modelo solo se usarán las 4 mediciones más cercanas ya que de usar todas las mediciones el tamaño de nuestra red de neuronas sería demasiado grande como para ser un modelo eficiente.

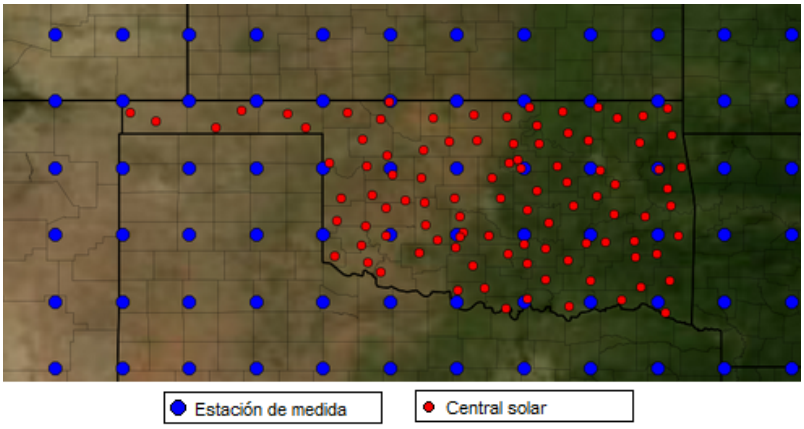

Figura 4.1: Ubicación de estaciones de medida y las centrales solares

Cada medición estará compuesta de 15 parámetros, repetidos todos 5 veces. Es posible ver un resumen de éstos parámetros en la tabla 4.1

Parámetro	Descripción	Unidad
$apcp\_sfc$	Precipitaciones acumuladas en la superficie las últimas 3 horas.	kgm^{-2}
$dlwrf\_sfc$	Media de radiación de onda larga recibida en la superficie.	Wm^{-2}
$dswrf\_sfc$	Media de radiación de onda corta recibida en la superficie.	Wm^{-2}
$pres\_msl$	Presión del aire a nivel del mar.	Pa
$pwat\_eatm$	Agua precipitable sobre la atmósfera.	kgm^{-2}
$spfh\_2m$	Humedad especifica a 2 metros por encima del suelo.	$kgkg^{-1}$
$tcdc\_eatm$	Porcentaje total de nubes en toda la atmósfera.	%
$tcolc\_eatm$	Condensación total en toda la atmósfera.	kgm^{-2}
$tmax\_2m$	Temperatura máxima de las últimas 3 horas a 2 metros sobre el suelo.	K
$tmin\_2m$	Temperatura mínima de las últimas 3 horas a 2 metros sobre el suelo.	K
tmp_2m	Temperatura actual a 2 metros sobre el suelo.	K
tmp_sfc	Temperatura de la superficie.	K
$ulwrf\_sfc$	Radiación de onda larga reflejada en la superficie.	Wm^{-2}
$ulwrf\_tatm$	Radiación de onda larga reflejada en lo alto de la atmósfera.	Wm^{-2}
$uswrf\_sfc$	Radiación de onda corta reflejada en la superficie.	Wm^{-2}

Tabla 4.1: Parámetros en el conjunto de datos

Estos datos están repartidos en dos sets, uno de entrenamiento con 4380 entradas y otro de para llevar a cabo la validación de las predicciones realizadas, con 733 entradas. Desde el amanecer hasta el anochecer, como estos conjuntos representan mediciones tomadas cada 5 minutos, se aprecia la serie temporal cuando representamos la energía producida.

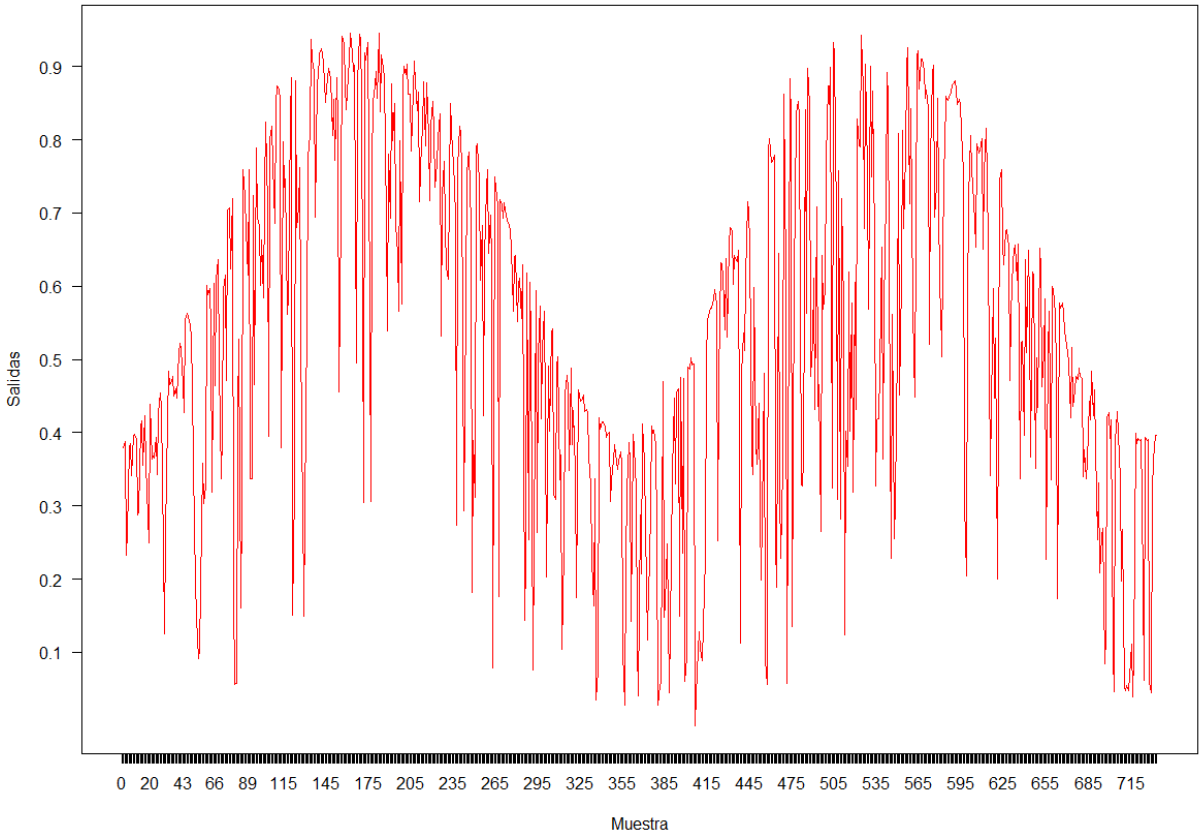

Figura 4.2: Histórico de salidas reales para el conjunto de test

Como se aprecia en la figura 4.2 las mediciones coinciden con una posible medición en tiempo real, obtenemos mediciones más bajas cuando amanece pero van aumentando según avanza el día, teniendo un pico a su mitad. La oscilación presentada a lo largo de la medición está motivada por las condiciones atmosféricas, un aumento en la densidad de las nubes puede suponer un descenso en la energía producida.

4.2. Parámetros usados

En este apartado detallaremos aquellos parámetros que han servido para hacer la experimentación. Estos se explicarán por partes, la primera, para dictaminar el número de neuronas ocultas y las iteraciones máximas que se usarán en la siguiente parte de la experimentación, en la cual se valorarán dos factores, los coeficientes que afectan a la función de fitness y el valor del PINC.

Para la primera fase se comenzará por fijar el número de iteraciones óptimo, que se irá incrementando hasta encontrar la mejor opción. Tras lo cual se realizarán 4 pruebas con distintas configuraciones de la capa oculta, con 5, 10, 20 y 30 neuronas.

En la segunda fase de la experimentación se tomarán los siguientes valores para los parámetros:

- **PINC:** variará según al porcentaje de valores que se desee dejar fuera del intervalo, las salidas más atípicas. Se valorarán los siguientes porcentajes: 1 %, 5 %, 10 %.
- **Coeficiente τ:** dará más peso a una de las dos medidas de la función de fitness dentro del rango [0..1]. Multiplicará a ambos, pero en el primer valor por τ y en el segundo por $\tau - 1$. Se experimentará con 3 posibilidades: 0.2, 0.3, 0.4. Este último parámetro no se usará en el método de regresión cuantil ya que este genera modelos fijos, no basados en heurística como los algoritmos evolutivos.

Por último, cabe señalar que para el resto de parámetros se ha usado:

- **Algoritmo genético:** Como se está tratando con un problema de optimización de una serie de valores reales el tipo elegido será real value", con 3000 iteraciones y unos valores máximos y mínimos para los parámetros de los cromosomas de 100 y -100, respectivamente, para permitir una mayor libertad en los pesos de al red. Se han escogido los siguientes valores por defecto para los siguientes parámetros:
  - Mutación: 0.8
  - Cruzamiento: 0.1
  - Población máxima: 50
- **Evolución diferencial:** Para el valor máximo y mínimo de la población de usaran 100 y -100, respectivamente, como en el anterior. Para los parámetros en control se han vuelto

a usar 3000 iteraciones. Además, se ha utilizado el valor por defecto de los siguientes parámetros:

- Coeficiente de diferencia de peso: Asigna un peso a cada uno de los vectores a ser restados, por defecto, 0.8 en el rango [0..2]
- Cruzamiento: 0.5
- Población máxima: 50, aunque se recomienda que para este tipo de algoritmo una población de al menos 10 veces el número de parámetros en el vector, sin embargo, como hay 9092 parámetros el tiempo de cómputo para cada generación sería demasiado grande.

4.3. Resultados

En esta sección se analizarán los resultados obtenidos, analizando cada uno de los métodos usados por separado y por último conjuntamente.

4.3.1. Configuración de la red de neuronas

En esta primera parte de la experimentación se obtendrá la configuración óptima para la red de neuronas en la función de fitness.

Para ello, se lanzará el entrenamiento de la red dentro de un bucle, por cada ciclo, se entrenará a la red una iteración y se guardará el error cometido, tanto en entrenamiento como en test, y los pesos generados. Tras esto, se volverá a entrenar la red una iteración con los nuevos pesos. Este proceso se detendrá cuando no haya una disminución del error significativa o cuando haya sobreentrenamiento.

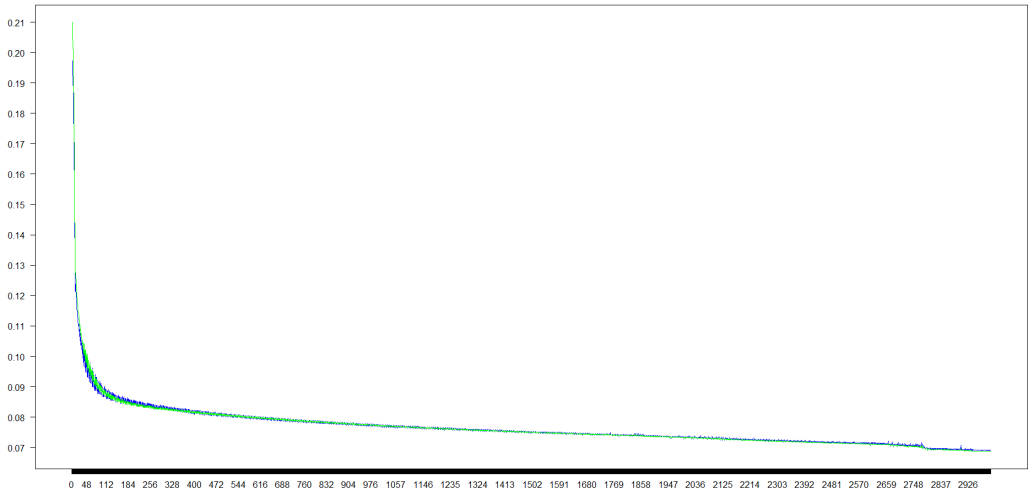

Figura 4.3: Evolución error entrenamiento y test

Una vez se haya fijado este parámetro se realizarán 3 pruebas con cada una de las configuraciones. En la tabla 4.2 se pueden ver los resultados, expresados en términos del error medio cometido en las 3 pruebas, tanto para el conjunto de test como de entrenamiento:

# neu	Entrenamiento	Test
5	0.07193163	0.07156912
10	0.0708727	0.06970933
20	0.0694258	0.06941554
30	0.0691006	0.06899346

Tabla 4.2: Error Medio

Tal y como se ve, el menor error que se consigue es para una configuración de 30 neuronas ocultas. Se descartó seguir aumentando el número de neuronas ya que como se puede apreciar en la tabla 4.2, la diferencia de error entre una configuración con 20 neuronas y 30, es muy pequeña, mientras que el tiempo de computo crece demasiado como para que merezca al pena.

4.3.2. Algoritmo Genético

En este apartado se analizarán los resultados del algoritmo genético, para el cual en total se habrá realizado 9 pruebas diferentes, repetidas 3 veces, en las que se obtendrán el valor de la

función de fitness del mejor cromosoma, el valor de la anchura obtenido, del PICP y del ACE y de la desviación estándar.

En esta fase de la experimentación se realizarán 3 pruebas con cada coeficiente para cada valor de PINC. La tabla 4.3 muestra los resultados de la media de las tres ejecuciones.

Coef. τ	PINC	Conjunto	Anchura	PICP	ACE	Desv. Estan.
0.2	0.01	Entren	0.5814336	0.9824443	-0.0075556	0.111571
		Test	0.585649456	0.977848794	-0.012151206	0.11278
	0.05	Entren	0.581433631	0.982444386	-0.007555614	0.11157
		Test	0.504615393	0.945429741	-0.004570259	0.09611
	0.1	Entren	0.474835931	0.9	0	0.140929
		Test	0.476991	0.8981355	-0.0018644	0.141262
0.3	0.01	Entren	0.4779281	0.956316	-0.0336834	0.10966
		Test	0.4847294	0.950886	-0.0391132	0.113591
	0.05	Entren	0.4716963	0.948782	-0.0012176	0.12448
		Test	0.4818790	0.946793	-0.003206	0.127913
	0.1	Entren	0.4697156	0.9191780	0.0191780	0.09343
		Test	0.4742661	0.923146	0.0231468	0.096066
0.4	0.01	Entren	0.399732	0.916666	-0.0733333	0.089327
		Test	0.4081049	0.9131423	-0.0768576	0.094627
	0.05	Entren	0.4142856	0.9090563	-0.0409437	0.107029
		Test	0.4210882	0.9045026	-0.0454974	0.108017
	0.1	Entren	0.3878198	0.9	0	0.098184
		Test	0.3913495	0.8963165	-0.003683	0.1004839

Tabla 4.3: Resultados para el algoritmo genético

Como cabía de esperar, al aumentar el coeficiente que mejora el valor de la anchura frente al grado de confianza, esta disminuye considerablemente, obteniendo a cambio peores tasas para PINCs más bajos. De esta forma, podemos observar que para un PINC de 0.01, obtenemos un error de 0.0077 para un coeficiente de 0.2, pero sube hasta un error de 0.073 para un coeficiente de 0.4, más del 7 % de las salidas de test y entrenamiento mal predichas. Sin embargo, la anchura

baja casi un 33 % para el coeficiente más alto.

Este fenómeno se puede observar fácilmente si representamos los dos intervalos. En la figura 4.4, con dos gráficas de test, se puede apreciar como los intervalos en la segunda gráfica se ajustan mejor, sin embargo, también hay mayor número de puntos inferiores o superiores por encima de los valores reales esperados.

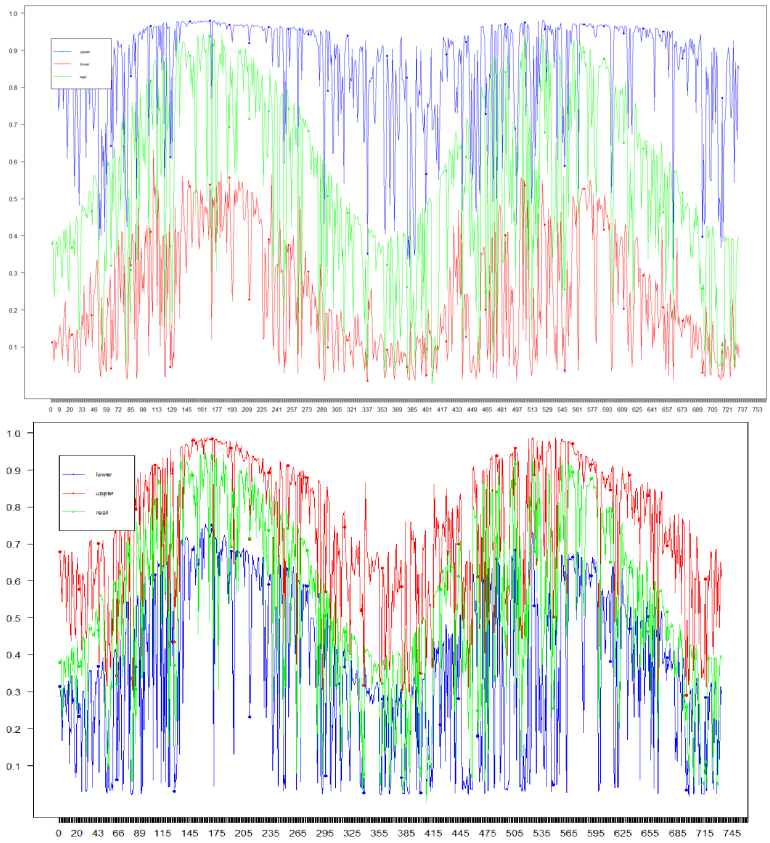

Figura 4.4: Comparación intervalos GA

Es remarcable observar que pese no se aprecia una disminución significativa en la anchura cuando aumenta el PICP, aunque en los pasos previos a la experimentación se esperaba lo contrario. Por último, el valor de la desviación típica nos permite ver que los intervalos producidos siguen una pauta en cuanto a anchura, y son regulares, es decir, se han producido intervalos con similar anchura para cada prueba de la experimentación.

4.3.3. Evolución diferencial

Para este algoritmo se ha realizado la misma experimentación que en el anterior, sin embargo, al ser un algoritmo diferente, especialmente preparado para usar valores reales, se espera una mayor optimización.

Los resultados obtenidos se pueden ver en la tabla 4.4:

Coef. τ	PINC	Conjunto	Anchura	PICP	ACE	Desv. Estan.
0.2	0.01	Entren	0.578382284	0.979452055	-0.010547945	0.110073173
		Test	0.58683024	0.983628922	-0.006371078	0.114886237
	0.05	Entren	0.385415895	0.923744292	-0.026255708	0.115374542
		Test	0.39656102	0.929968167	-0.020031833	0.121762303
	0.1	Entren	0.384960382	0.899467275	-0.000532725	0.109201983
		Test	0.394222054	0.915416098	0.015416098	0.115255641
0.3	0.01	Entren	0.475319362	0.957534247	-0.032465753	0.104433535
		Test	0.485361354	0.967257844	-0.022742156	0.107875222
	0.05	Entren	0.470241986	0.94414003	-0.00585997	0.114745012
		Test	0.479317867	0.954524784	0.004524784	0.120967052
	0.1	Entren	0.429914177	0.899771689	-0.000228311	0.104670951
		Test	0.441175197	0.917235107	0.017235107	0.109492641
0.4	0.01	Entren	0.397401006	0.915144597	-0.074855403	0.090420725
		Test	0.408999543	0.934060937	-0.055939063	0.095498845
	0.05	Entren	0.385556616	0.910273973	-0.039726027	0.090755802
		Test	0.393925473	0.92633015	-0.02366985	0.096924004
	0.1	Entren	0.375081352	0.89870624	-0.00129376	0.099493692
		Test	0.384920122	0.914961346	0.014961346	0.108702788

Tabla 4.4: Resultados para el algoritmo de evolución diferencial

En este caso resulta algo sorprendente ya que para PINCs de 0.05 y 0.1, se consiguen anchuras cercanas a 0.39, tanto para coeficientes de 0.2 y de 0.4, lo que dificulta establecer una correlación entre el coeficiente y la anchura y el ACE obtenidos. Como se puede ver en la figura 4.5, la diferencia entre las gráficas con los intervalos generados para los datos de test, la primera con

coeficiente 0.4 y 0.1 de PINC y la otra con coeficiente 0.2 y 0.05 de PINC es mínima:

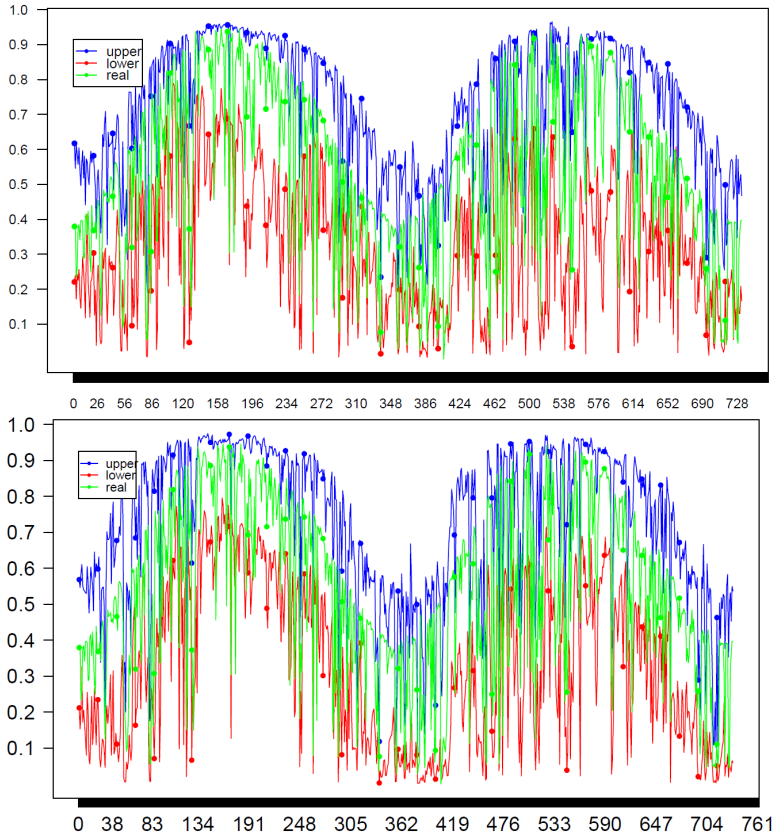

Figura 4.5: Comparación intervalos Evolución Diferencial

De nuevo, la desviación típica vuelve a estar dentro de los rangos normales y no hay valores atípicos. Se observa una disminución significativa de la anchura para un coeficiente de 0.2, si bien esto puede deberse a un mal entrenamiento en la red para el primer valor del PINC, ya que esto no se observa para el resto de coeficientes.

4.3.4. Regresión Cuantil

En esta sección se analizarán los resultados de la regresión cuantil, así como la experimentación realizada. En este método solo es posible es variar un parámetro, los cuartiles a predecir, que corresponderán con los límites inferior y superior del intervalo.

Los parámetros corresponderán con los valores de PINC, es decir, se discriminará el 1 %, el 5 % y el 10 % de la población. Para representar esto se predecirán los cuartiles en los pares de puntos (0.005, 0.995), (0.025, 0.975) y (0.05, 0.95). Para estos valores se han obtenido los siguientes resultados:

PINC	Conjunto	Anchura	PICP	ACE	Desv. Estan.
0.01	Entren.	0.351597	0.938584	-0.051415	0.146246
	Test	0.356493	0.894952	-0.095047	0.162838
0.05	Entren.	0.315009	0.913013	-0.036986	0.135206
	Test	0.318196	0.881309	-0.068690	0.146751
0.1	Entren.	0.260782	0.8773973	-0.022602	0.116815
	Test	0.259974	0.837653	-0.062346	0.121423

Tabla 4.5: Resultados para regresión cuantil

En la tabla 4.5 se puede ver como el error cometido se hace más grande cuanto mayor es el valor de PINC, sin embargo, la anchura disminuye de forma constante. Es necesario remarcar que el error se mantiene constante, es decir, la diferencia entre el PICP y el PINC se mantiene, por lo que podemos decir que este método de predicción siempre cometerá un fallo de entre un 3 % y un 7 %. Por ejemplo, es posible encontrarse con valores en los limites del intervalo que claramente sobresalen del valor real, tal y como puede verse en la gráfica 4.6.

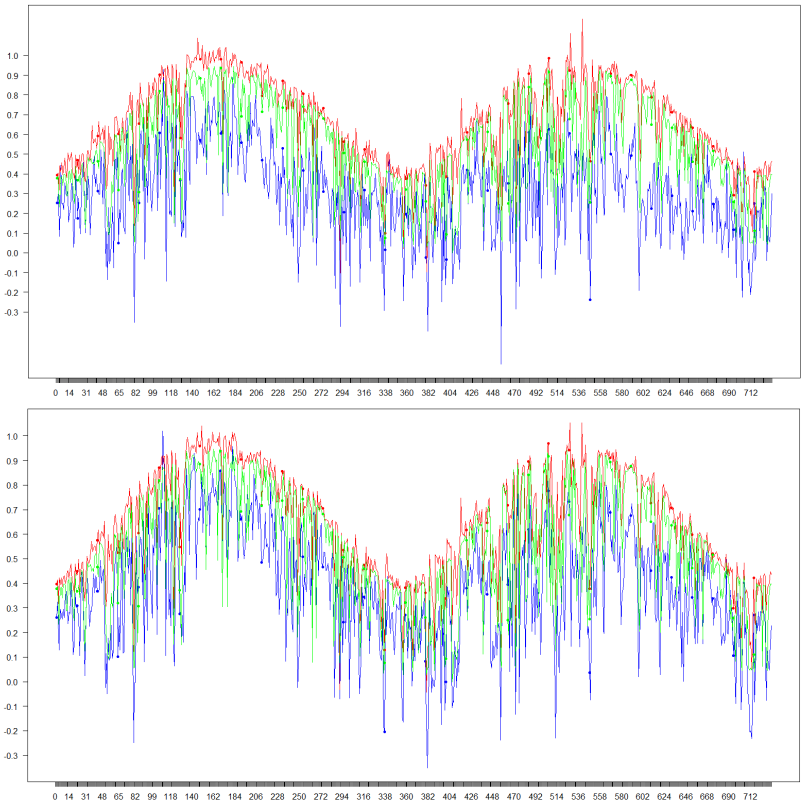

Figura 4.6: Comparación intervalos regresión cuantil

4.3.5. Análisis

A continuación se analizarán los resultados en su conjunto, para ello, utilizaremos una serie de gráficas ordenadas por el PINC en el que representaremos en el eje vertical el 1 - PICP y en el horizontal la anchura. Cuanto más se acerque este último valor al PINC, mejor ACE se habrá conseguido.

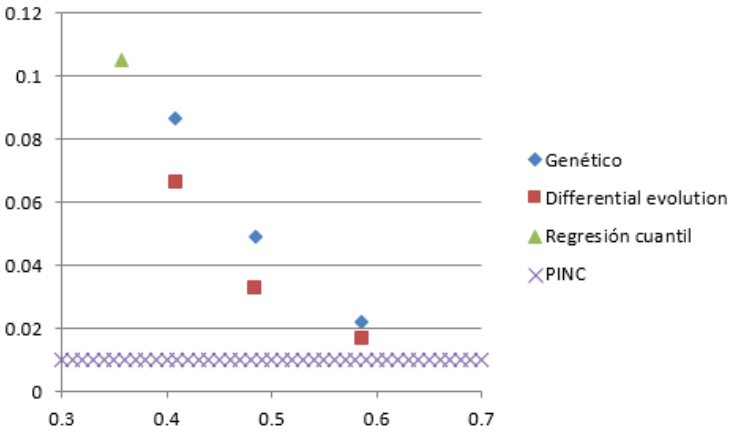

Figura 4.7: Comparación resultados PINC 0.01

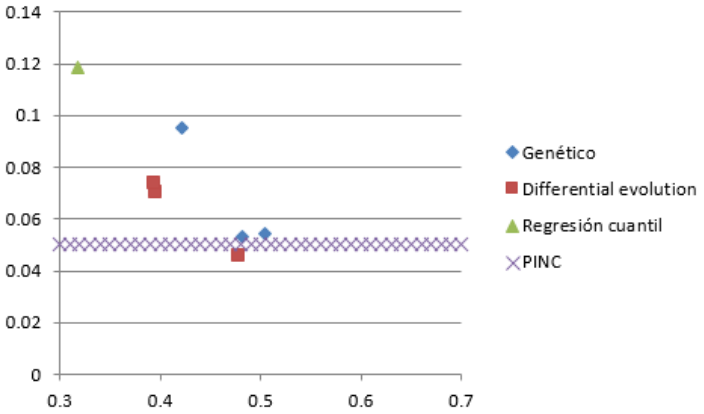

Figura 4.8: Comparación resultados PINC 0.05

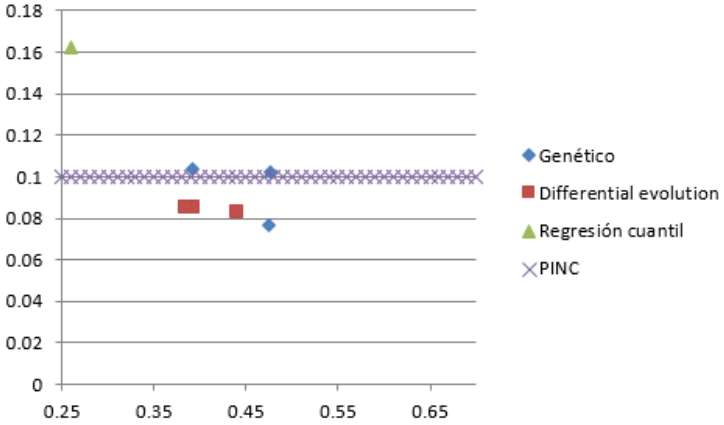

Figura 4.9: Comparación resultados PINC 0.1

A primera vista se puede observar que el menor error cometido se asocia a los algoritmos de evolución diferencial. Es fácil de apreciar esto, sobretodo, en la figura 4.7, donde para todos los coeficientes, el algoritmo obtiene mejor PICP mientras se mantiene la anchura respecto al genético. Ya que estos coeficientes ajustan la importancia de los atributos en la función de fitness, cuanto más pequeño sea el coeficiente, más mejorará el PICP. Sin embargo, no podemos fijarnos en esta medida como un indicativo del error, para esto, debemos fijarnos en el ACE. Cuanto mayor el coeficiente, peor resultados obtendremos con PINC mayores. Esto se puede ver si comparamos la diferencia entre el ACE para un coeficiente de 0.4 y de 0.2, frente a un error del 7 % en el primero, este baja a una media del 1 % e incluso, en casos como algunos resultados de las figuras 4.8 y 4.9, el valor del PICP es menor que el del PINC, por lo que el ACE será 0. Podemos deducir por lo tanto, que cuando se quiera optimizar el ACE frente a la anchura, siempre será mejor usar valores de PINC altos y coeficientes que den menos importancia a la anchura.

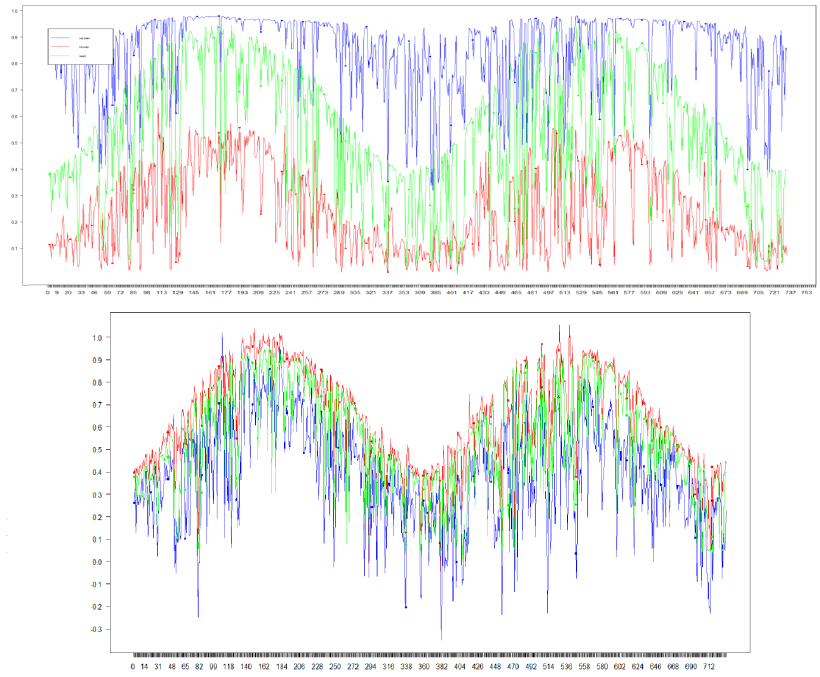

Figura 4.10: Comparación algoritmo evolutivo(1) y regresión cuantil(2)

Si lo que queremos es reducir la anchura, la regresión cuantil es el mejor método, en las figuras 4.7, 4.8, 4.9 esto se observa perfectamente. Con cualquier PINC que se elija la anchura será menor que la generada por los otros dos métodos, por ejemplo en la figura 4.10 se comparan

el resultado obtenido por un algoritmo evolutivo y otro por la regresión cuantil, en el cual se aprecia un intervalo más ajustado. Sin embargo, pese a dar esta mejora, también apreciamos en las gráficas que de todos los modelos, es aquel con el PICP más alejado del PINC, con los valores de ACE más altos, llegando a tener un error del 16 % en el peor de los casos. Por todo esto, a la hora de crear modelos en los que se quiera premiar la anchura, la mejor solución es usar esta técnica.

Por último, también es necesario tener en cuenta un modelo en el que se mantenga un equilibrio entre los dos factores. La regresión cuantil produce errores de predicción muy altos aunque mantenga una anchura más ajustada, y , además, tiene valores muy altos en la desviación típica, que puede suponer que se produzcan desviaciones mayores en los intervalos. Por esto, es necesario elegir entre los dos algoritmos de computación evolutiva. Para este caso, el algoritmo de evolución diferencial parece ser lo mejor, tal y como se aprecia, sobretodo, en la figura 4.7. Podríamos añadir incluso que este último algoritmo da mejores resultados de forma global que el genético una vez analizados todos los errores y anchuras, sin embargo, esta mejora es tan pequeña que no es posible asegurarlo.

Capítulo 5

Conclusiones y futuros trabajos

En esta sección se hará un análisis del problema, viendo si se han satisfecho los objetivos iniciales a través de los resultados obtenidos. Además, se propondrán futuras lineas de trabajo para extender la predicción de intervalos de predicción a otras técnicas, mientras se mejoran las actuales.

5.1. Conclusiones

Como ya se ha dicho, en nuestra sociedad, el uso de energía se hace imprescindible, la vida moderna tal y como la concebimos no sería posible sin ella. Sin embargo, este uso indiscriminado de la energía, agota los recursos del planeta y lo contamina. Es por ello que se presenta necesario el desarrollo de nuevas energías limpias y renovables, como la solar, la eólica o la mareomotriz. En este proyecto nos hemos centrado en la energía solar, un valor en desarrollo y que puede tener un rol muy importante en un futuro cercano. No obstante, este tipo de energía presenta un grave problema, no se puede controlar su producción, y al no poder almacenarse para su uso posterior no es posible garantizar una cobertura energética fiable.

Para reducir esta incertidumbre es necesario construir un modelo que permita medirla. Para este problema se ha optado por la predicción de intervalos de confianza, para los que dados su anchura, permitirá saber la incertidumbre de la medida. A intervalos más amplios, mayor incertidumbre tendrá la medición. Por ello, en este proyecto se realizará un sistema que minimice esta anchura y mientras que garantiza que los valores reales se encuentran en el intervalo. Usando un perceptrón multicapa se obtendrán el limite superior e inferior del intervalo y se optimizarán

sus pesos mediante un algoritmo genético y un algoritmo de evolución diferencial. Además, compararemos ambos sistemas con un algoritmo de regresión cuantil, en que prediremos el valor de los cuantiles que correspondan a los limites del intervalo.

Tras la realización de estas pruebas se ha llegado a las siguientes conclusiones. Tanto el algoritmo genético como el de evolución diferencial obtienen muy buenos resultados a la hora de aproximar el ACE al PINC, lo cual indica que en la mayoría de los casos, todos los valores de salida se encontraban dentro del intervalo predicho. Sin embargo, estos algoritmos presentan una pega, la anchura que obtenida para PINCs pequeños es muy amplia, lo cual supone un alto nivel de incertidumbre en la medición, y puede no ser útil si se necesitase un modelo más exigente.

Por otro lado, se puede ver que la regresión cuantil consigue los mejores valores de anchura, consiguiendo anchuras del 30 % de la distribución total, sobre los otros algoritmos que se quedan en el 40 %. Como pega a esta última técnica, hay que decir que su ACE es muy elevado, esto supone un número moderado de datos fuera del intervalo. Como este hecho se mantiene para los tres valores del PINC sobre los que se ha experimentado, podemos decir que el uso de esta técnica no es aconsejable.

Por último, hay que remarcar los resultados del algoritmo de evolución diferencial para valores de PINC de 0.1. Estos presentan la anchura menos elevada con un ACE próximo a cero. Si bien la anchura no pasa del 36 % es una mejora considerable frente a los peores resultados con un 56 % y bien podrían ser usados en una prueba real.

Entre todo lo aprendido en este proyecto, lo más destacable es todo aquello relacionado con la computación evolutiva. Antes de empezar este trabajo, jamas hubiese podido imaginar que este campo fuese tan útil y tuviese un campo de uso tan amplio. Además, son en su conjunto, algoritmos muy sencillos, fáciles de implementar, en los que lo lo más importante radica en la elección de un buen cromosoma y una función de fitness, junto a la elección de los parámetros de selección, cruzamiento y mutación.

5.2. Futuros Trabajos

Una de los primeras medidas a tomar, sería la inclusión de modelos numéricos de predicción atmosférica entre los métodos a comparar. Estos han resultado ser muy eficientes a la hora de predecir parámetros dependientes del clima [8, 21], usando para ello ecuaciones de la mecánica de fluidos. Además, se podría incluir algoritmos de optimización multiobjetivo, que eliminaría

la existencia de los coeficientes, ya que de esta forma se optimizarían tanto al anchura como el ACE.

Otra de las medidas a tomar pasaría por la codificación de los cromosomas en strings binarias para el algoritmo genético, pero, usando una codificación propia, en al que, por ejemplo, solo se representen números entre el 0 y el 1 con 6 dígitos decimales. De esta forma, suponiendo que que se siga una codificación del tipo $(\textbf{numero real x } 10^6)_2$ solo se necesitarían 20 bits por cada parámetro, 2^{20}, hasta 1048576 números, obteniendo cromosomas de 181840 bits de longitud.

Por último, sería necesario la inclusión de pruebas cambiando los parámetros de cruzamiento, selección y mutación, así como el coeficiente de la diferencia de peso en el algoritmo de evolución diferencial. Pese a que los valores por defecto son los más usados, una variación de éstos podría resultar en una mejora en la producción de individuos.

APÉNDICES

APÉNDICE A

PRESUPUESTO DEL PROYECTO

En este apéndice se presentan justificados los costes globales de la realización de este Proyecto Fin de Carrera. Estos costes provienen de gastos de personal, pero también de gastos materiales y se pueden ver en las siguientes tablas: A.1 y A.2.

Fase 1	*Documentación*	50 *horas*
Fase 2	*Implementación del software*	80 *horas*
Fase 3	*Experimentación*	144 *horas*
Fase 4	*Redacción de la memoria del proyecto*	86 *horas*

Tabla A.1: Fases del Proyecto

En la Tabla A.1 se muestran las fases del proyecto y el tiempo aproximado para cada una de ellas. Así pues, para la documentación se usaron 50 horas repartidas a lo largo del proyecto, 80 horas fueron a parar a la implementación del software y el testeo del código, 144, 6 días, fue el tiempo durante el cual se generaron los modelos de predicción para cada prueba, y el resto, en al realización de la memoria. Entonces, teniendo en cuanta el sueldo medio de un informático en España, los gastos de personal ascienden a 4725€

Ordenador de gama media	900 €
Alquiler de un local (250 €/mes)	1500 €
Gastos varios	700 €

Tabla A.2: Costes de material

En la Tabla A.2 se pueden ver los costes asociados a material, repartidos entre un ordenador donde implementar el código y llevar a cabo la experimentación, el alquiler de un lugar de trabajo, y gastos tales como material fungible, llamadas telefónicas, electricidad, Internet, etc. El gasto total es de 3100 €.

Tras ver las tablas anteriores, el presupuesto total es el siguiente:

Concepto	**Importe**
Costes personal	4725 €
Costes material	3100 €
Base imponible	7825 €
I.V.A. (16 %)	1225 €
TOTAL	9077 €

Tabla A.3: Presupuesto

APÉNDICE B

PLANIFICACIÓN

En este apéndice se hará un resumen de las tareas realizadas, con su desglose de tiempo tomado para cada actividad.

B.1. Fases

Este apartado servirá para detallar cada una de las fases de las que ha constado el proyecto.Se ha dividido en 6 fases, que se detallan a continuación.

- **Documentación:** En esta fase se ha hecho un estudio de las técnicas más empleadas en el campo de la predicción de intervalos. Se han recogido artículos sobre otros proyectos parecidos, y se ha documentado sobre las tecnologías necesarias para la implementación.
- **Diseño:** Una vez se ha terminado de analizar estos aspectos, se ha pasado el diseño del sistema, teniendo en cuenta el estado del arte en las nuevas técnicas y tecnologías.
- **Implementación:** Esta fase se compone de la codificación de la solución escogida en el diseño. Esta fase se ha divido en hitos para reducir su complejidad. Primero se ha implementado el primer programa que nos dará la estructura de la red, en el segundo hito se ha implementado la red de neuronas que generara los intervalos y la función de fitness, después, se han codificado todos los métodos con los que se va a experimentar y por último la lógica para la normalización de los datos.
- **Experimentación:** En esta fase se ha llevado a cabo al experimentación. Debido al número de pruebas y al tiempo de cada una de ellas esta fase ha ocupado mucho tiempo.

- **Análisis de resultados:** Aquí se han recogido los resultados y se han creado las gráficas y tablas.
- **Realización e la memoria:** En esta última fase se ha realizado la memoria, con la explicación del estudio realizado, los resultados recogidos y las conclusiones a las que se ha llegado.

B.2. Diagrama de Gantt

A continuación se muestra en el siguiente diagrama de Gantt de la figura B.1, el desglose de días trabajados en el proyecto. Cabe destacar que la media de trabajo diario es de 2 horas y media, lo que multiplicado por el numero de días trabajados, da un tiempo total de 397 horas.

Id	Modo de tarea	Nombre de tarea	Duración	Comienzo	Fin	Predecesoras
1		Documentación	51 días	dom 01/03/15	vie 08/05/15	
2		Diseño	5 días	mar 05/05/15	lun 11/05/15	
3		Implementación - Hito 1	2 días	mar 12/05/15	mié 13/05/15	2
4		Implementación-Hito 2	14 días	mié 13/05/15	lun 01/06/15	
5		Implementación - Hito 3	3 días	lun 01/06/15	mié 03/06/15	
6		Implementación - Hito 4	2 días	jue 04/06/15	vie 05/06/15	
7		Experimentación	30 días	lun 08/06/15	vie 17/07/15	6
8		Analisis de los resultados	4 días	lun 20/07/15	jue 23/07/15	7
9		Realización de la memoria	48 días	jue 23/07/15	lun 28/09/15	8

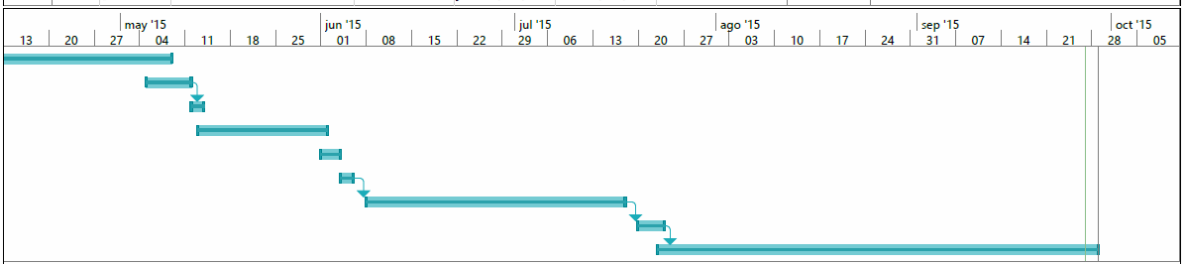

Figura B.1: Diagrama de Gantt

APÉNDICE C

Marco regulador

Este trabajo de fin de grado ha sido realizado tomando en cuenta el ámbito legal que a éste pudiera afectar. Todos los programas que se han usado son gratuitos y de software libre, exceptuando Office Project, para el cual se ha usado la licencia gratuita que provee la universidad. Tanto el lenguaje R como el compilador están dentro del proyecto GNU, que los hace gratuitos, así como Latex y TexMaker, usados para escribir la memoria.

Todas las librerías usadas son de uso público ya que han sido publicadas para R, y por lo tanto, tienen la misma licencia. El código realizado tiene licencia creative commons y puede ser usado para futuros trabajos sin restricción alguna.

Por último, los datos usados son de dominio público. Han sido publicados por la American Meteorological Society y se pueden encontrar en la siguiente página web: https://www.kaggle.com/c/ams-2014-solar-energy-prediction-contest/data

APÉNDICE D

Resumen Ingles

D.1. Abstract

As the modern world's need for energy keeps growing, using a clean source of energy is more important every day. In this project we will focus on solar energy. Due to the limitations of this kind of energy in regard of the way it is stored, it is necessary to develop technics to predict the generated electricity. It will be explained the methods for the creation of models of prediction for intervals based on artificial intelligence technics, such as neural networks. This network will have 300 inputs neurons that will specify the atmospheric attributes like solar radiation or cloud density, and two output neurons that will represent the extremes of the interval. Also, we will use optimization technics for the weights of the network, for which we will use evolutionary computation algorithms, such as genetics algorithms and differential evolution algorithms. For this, it will be detailed the methodology used for the construction of the chromosomes and the fitness function. At least, these models will be compared with an interval predictor method, the quantile regression.

D.2. Introduction

D.2.1. Historical context

Since the beginning of the industrial revolution, man has been subjected to a growing need for energy to keep their machines working. In 1950, and because of a greater demand of energy, the first nuclear plant is built. However, following a series of accidents and due to the difficulty

to dispose the radioactive wastes generated, a new source of clean and secure energy is created. Renewable energies, such as photovoltaic or hydraulic, despite having an economic cost much higher than the rest of energies, they do not cause so many risks and its environmental impact is minimum.

In this project, we will talk about the production of photovoltaic energy. In 1954, A.D.M. Chapin, C.S. Fuller and G.L. Pearson, develop the first photovoltaic cell based on silicon. The first cell had an effectivity of the 6 %, but, soon, it reached the 10 %, nowadays, there are cells capable of reaching the 20 %.

D.2.2. Social-economical environment

Nowadays, it is calculated that this kind of energy provides the 1 % of the world's production. However, the use of this energy is relatively new and, even though this low percentage, its use in the last 15 years has increased more than a 100 times, being Germany its leader, followed by Spain and Italy.

Also, is remarkable how new projects are starting to appear to boost this kind of energy. The most important is the initiative of the company Solar City, founded by the brothers Peter and Lyndon Rive, which emerged as a suggestion from Elon Musk, entrepreneur and engineer, recognized for the creation of Tesla Motors and PayPal. In just 8 years from its foundation, it has become the biggest photoelectrical based company in the United States, opening new solar farm and also, leading the field of investigation in new material and storage methods.

D.3. Motivation

With the nowadays crisis, of the non-renewable energies, it is necessary to find a new clean, secure and, moreover, reliable source of energy. The develop if the technology behind the solar energy is not the challenge, but the prediction on a short and medium term of the energy production. This is due to the fact that solar energy cannot be stored, like the other renewable energies, and so, it is necessary to find a system that allows how much energy can guarantee a photovoltaic central. For this, projects done previously have been taken into account, been machine learning the most extended. Inside this category Artificial Neural Networks, Support Vector Machines and Bayesian Networks are common.

For all that has been described previously, the motivation in this project consist in the

proposition of a new system that allows an improvement on the prediction of the photovoltaic production.

D.4. Objectives

D.4.1. Principal objective

Nowadays, classical methods for the prediction of real values do not take into account the uncertainty of that prediction. Because of that, the main objective of this project is the construction of a system capable of predict intervals given some weather variables, in which it will be ensured that the value is inside. We will created a series of models able to generate 2 numeric values that will be the limits of the interval that will be validated with two term, the width of the interval and the reliability.

D.4.2. Second objective

To acquire new knowledge about the evolutionary computation algorithms and the quantile regression. Also, it will be studied how the different parameters affect the algorithms performance.

D.5. Conclusions

After the experimentation, we will reach the next conclusions. Either the genetic algorithm, neither the differential evolution algorithm seems to obtain good result at predicting an interval with a good width, however, they obtain incredible results while predicting the ACE.

On the other hand, the quantile regression shows spectacular width, in the range of 25 % and 30 % of the total distribution, but, the ACE values are far behind the evolutionary algorithms.

At least, using the differential evolution algorithms for higher values of the PICP seems to give the best values, having widths of about a 36 % and an ACE of 0.

Bibliografía

[1] T. Back, U. Hammel y H.-P. Schwefel. "Evolutionary computation: comments on the history and current state". En: *Evolutionary Computation, IEEE Transactions on* 1.1 (1997), págs. 3-17.

[2] Thomas Back, David B Fogel y Zbigniew Michalewicz. *Handbook of evolutionary computation*. IOP Publishing Ltd., 1997.

[3] Antonio Bracale y col. "A Bayesian method for short-term probabilistic forecasting of photovoltaic generation in smart grid operation and control". En: *Energies* 6.2 (2013), págs. 733-747.

[4] Adolf Goetzberger y Christopher Hebling. "Photovoltaic materials, past, present, future". En: *Solar Energy Materials and Solar Cells* 62.1–2 (2000), págs. 4-6.

[5] Roger Koenker. *Quantile regression*. 38. Cambridge university press, 2005.

[6] Roger Koenker y Maintainer Roger Koenker. "Package 'quantreg'". En: (2015).

[7] Friedrich Leisch. "Sweave: Dynamic generation of statistical reports using literate data analysis". En: *Compstat*. Springer. 2002, págs. 575-580.

[8] Andrew C Lorenc. "Analysis methods for numerical weather prediction". En: *Royal Meteorological Society, Quarterly Journal* 112 (1986), págs. 1177-1194.

[9] Carlos A. Coello Coello Luis Vicente Santana Quintero. "Una introducción a la Computación Evolutiva y alguna de sus aplicaciones en Economía y Finanzas". En: *Revista de métodos cuantitativos para la economía y la empresa* 2 (2006), págs. 3-26.

[10] Adel Mellit y Alessandro Massi Pavan. "A 24-h forecast of solar irradiance using artificial neural network: Application for performance prediction of a grid-connected {PV} plant at Trieste, Italy". En: *Solar Energy* 84.5 (2010), págs. 807-821.

[11] M. Mohandes, S. Rehman y T.O. Halawani. "Estimation of global solar radiation using artificial neural networks". En: *Renewable Energy* 14.1–4 (1998). 6th Arab International Solar Energy Conference: Bringing Solar Energy into the Daylight, págs. 179-184. ISSN: 0960-1481.

[12] H Morita y col. "Interval prediction of annual maximum demand using grey dynamic model". En: *International Journal of Electrical Power & Energy Systems* 18.7 (1996), págs. 409-413.

[13] Katharine Mullen y col. "DEoptim: An R package for global optimization by differential evolution". En: *Journal of Statistical Software* 40.6 (2011), págs. 1-26.

[14] Luca Scrucca. "GA: A Package for Genetic Algorithms in R". En: *Journal of Statistical Software* 53.4 (2013), págs. 1-37. URL: http://www.jstatsoft.org/v53/i04/.

[15] N. Sharma y col. "Predicting solar generation from weather forecasts using machine learning". En: *Smart Grid Communications (SmartGridComm), 2011 IEEE International Conference on*. Oct. de 2011, págs. 528-533.

[16] Jie Shi y col. "Forecasting Power Output of Photovoltaic Systems Based on Weather Classification and Support Vector Machines". En: *Industry Applications, IEEE Transactions on* 48.3 (mayo de 2012), págs. 1064-1069.

[17] The University of Stanford. *Applications of neural networks*. URL: http://cs.stanford.edu/people/eroberts/courses/soco/projects/neural-networks/Applications/index.html (visitado 12-08-2015).

[18] The University of Stanford. *Neural Networks, History: The 1940's to the 1970's*. URL: http://cs.stanford.edu/people/eroberts/courses/soco/projects/neural-networks/History/history1.html (visitado 12-08-2015).

[19] The University of Stanford. *Neural Networks, History: The 1980's to the present*. URL: http://cs.stanford.edu/people/eroberts/courses/soco/projects/neural-networks/History/history2.html (visitado 12-08-2015).

[20] Rainer Storn y Kenneth Price. "Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces". En: *Journal of global optimization* 11.4 (1997), págs. 341-359.

[21] Hilding Sundqvist, Erik Berge y Jón Egill Kristjánsson. "Condensation and cloud parameterization studies with a mesoscale numerical weather prediction model". En: *Monthly Weather Review* 117.8 (1989), págs. 1641-1657.

[22] Jinn-Tsong Tsai, Jyh-Horng Chou y Tung-Kuan Liu. "Tuning the structure and parameters of a neural network by using hybrid Taguchi-genetic algorithm". En: 17 (2006), págs. 69-80.

[23] W. N. Venables y B. D. Ripley. *Modern Applied Statistics with S*. Fourth. ISBN 0-387-95457-0. New York: Springer, 2002. URL: http://www.stats.ox.ac.uk/pub/MASS4.

[24] E Vonk, Lakhmi C Jain y Ray P Johnson. *Automatic generation of neural network architecture using evolutionary computation*. Vol. 14. World Scientific, 1997, págs. 84-85.

[25] Can Wan, Zhao Xu y Pierre Pinson. "Direct interval forecasting of wind power". En: *Power Systems, IEEE Transactions on* 28.4 (2013), págs. 4877-4878.