UNIVERSIDAD POLITÉCNICA DE MADRID

ESCUELA TÉCNICA SUPERIOR
DE INGENIEROS DE TELECOMUNICACIÓN

PROYECTO FIN DE CARRERA

DISEÑO E IMPLEMENTACIÓN DE ALGORITMOS INTELIGENTES PARA LA GESTIÓN ACTIVA DE LA DEMANDA ELÉCTRICA EN EL SECTOR RESIDENCIAL

Eduardo Matallanas de Ávila

2010

UNIVERSIDAD POLITÉCNICA DE MADRID

ESCUELA TÉCNICA SUPERIOR
DE INGENIEROS DE TELECOMUNICACIÓN

Reunido el tribunal examinador en el da de la fecha, constituido por

Presidente:	Dr D.	Félix Monasterio-Huelin Maciá
Vocal:	Dra D^a.	Estefanía Caamaño Martín
Secretario:	Dr D.	Álvaro Gutiérrez Martín
Suplente:	Dr D.	Francisco Javier Jiménez Leube

para juzgar el Proyecto Fin de Carrera titulado:

DISEÑO E IMPLEMENTACIÓN DE ALGORITMOS INTELIGENTES PARA LA GESTIÓN ACTIVA DE LA DEMANDA ELÉCTRICA EN EL SECTOR RESIDENCIAL

del alumno D. Eduardo Matallanas de Ávila
dirigido por D. Álvaro Gutiérrez Martín

Acuerdan otorgar la calificación de: _____

Y, para que conste, se extiende firmada por los componentes del tribunal, la presente diligencia

Madrid, a _____ de _____ de _____

El Presidente	La Vocal	El Secretario
Fdo: _____	Fdo: _____	Fdo: _____

UNIVERSIDAD POLITÉCNICA DE MADRID

ESCUELA TÉCNICA SUPERIOR
DE INGENIEROS DE TELECOMUNICACIÓN

DEPARTAMENTO DE TECNOLOGÍAS ESPECIALES
APLICADAS A LA TELECOMUNICACIÓN

PROYECTO FIN DE CARRERA

DISEÑO E IMPLEMENTACIÓN DE ALGORITMOS INTELIGENTES PARA LA GESTIÓN ACTIVA DE LA DEMANDA ELÉCTRICA EN EL SECTOR RESIDENCIAL

Autor:
Eduardo Matallanas de Ávila

Director:
Álvaro Gutiérrez Martín

2010

Resumen

El presente proyecto desarrolla un sistema de control encargado de la Gestión de la Demanda Eléctrica en un prototipo de vivienda solar. La vivienda incorpora una fuente de generación fotovoltaica, baterías, una conexión a la red eléctrica y un sistema domótico para controlar los electrodomésticos. El sistema de control desarrollado es un sistema distribuido que incorpora dos capas: una de planificación y otra de coordinación, ambas implementadas mediante redes neuronales sintonizadas con algoritmos genéticos. Además, el sistema de control permite aumentar la eficiencia energética de la vivienda planificando las tareas demandadas por el usuario, gracias a un aprovechamiento inteligente de la energía solar.

PALABRAS CLAVE: Gestión de la Demanda Eléctrica, Generación Distribuida Fotovoltaica, Sistema de Control, Red Neuronal, Algoritmo Genético.

Agradecimientos

Como bien dice mi padre: “es de bien nacido ser agradecido”, por eso en estas breves líneas quiero agradecer a todos aquellos que me han apoyado durante mi paso universitario.

Querría empezar con mi familia, gracias a mis padres y mis hermanos, Paco, Libo, Javi y Jorge, porque siempre me han apoyado y cuando mayor era la adversidad siempre me han dado ánimos para continuar. Gracias también porque con su esfuerzo yo he podido seguir estudiando.

Gracias a todos mis amigos, Adri, Emilio, Rober, Robi, Guille, Víctor, Víctor Satán, Pablito, Alberto, Juan y muchos más, porque aunque siempre les he tenido que dejar un poco apartados cuando de estudios se trataba, han comprendido muy bien mi situación y siempre me han apoyado. Aunque cambiara las noches de salir con ellos, por los libros y las ecuaciones de Maxwell. Gracias por todas aquellas risas que nos hemos echado durante estos largos, pero a la vez cortos cinco años.

Gracias a todos mis amigos de la universidad, Rafa, Ángel, Garion, Luisja, Pizarrín, Víctor, David, Carlos, Luismi y un largo etcétera, porque aunque ha sido breve me lo he pasado muy bien con todos vosotros yendo a clases unipersonales, al grupo que acaba siempre en 2 y por todas las risas que nos hemos echado dentro y fuera de clase.

Gracias también a Manuel Castillo, mi compañero de fatigas en el labo, ha sido quien me ha acompañado durante todo el año en el labo. Día tras día allí encerrados ambos, cada uno trabajando en lo suyo, pero las charlas y risas compartidas han aligerado la carga.

También quiero dar las gracias a Álvaro Gutiérrez Martín, mi tutor, por echarme una mano siempre, aunque estuviera muy ocupado, y también por todo el apoyo que me ha dado. Gracias también a Félix Monasterio-Huelin Maciá quien gracias a su manera de pensar me abrió la cabeza y empecé a ver las cosas de otra forma, gracias también porque con sus charlas animadas se ha podido llevar mejor la carga de trabajo. Ambos han sido una inspiración para mi y sus consejos, charlas y ayudas

me han servido mucho durante este año. También decir que gracias a ellos tomé la decisión de seguir estudiando y dedicarme a la investigación.

Por último dar las gracias a una persona muy especial para mi con la que he compartido todo este año, ha tenido que soportar mucho mis agobios y nervios ante los trabajos y exámenes. Siempre ha estado a mi lado durante los momentos difíciles y me ha animado más cuando lo necesitaba. Por todo ello muchísimas gracias Marta.

Sé que dejo a mucha gente sin mencionar pero estas líneas son breves para todo lo que debiera expresar. Pero para todos aquellos que han estado presentes en mi evolución durante la carrera y para todos aquellos que me conocen, gracias.

Muchas gracias a todos.

Índice general

Resumen iv

Agradecimientos v

Índice General vii

Índice de Figuras ix

Índice de Tablas xi

Lista de acrónimos xii

1. Introducción y Objetivos 1
- 1.1. Motivación . . . 1
- 1.2. Estado del arte . . . 2
  - 1.2.1. La demanda eléctrica en España . . . 3
  - 1.2.2. Generación fotovoltaica distribuida . . . 5
  - 1.2.3. Sistemas de control . . . 7
  - 1.2.4. GeDELOS-FV . . . 8
  - 1.2.5. Magicbox . . . 9
- 1.3. Objetivos del proyecto . . . 11
- 1.4. Organización del proyecto . . . 12

2. Redes Neuronales y Algoritmos Genéticos 15
- 2.1. Redes neuronales . . . 15
  - 2.1.1. Introducción . . . 16
    - 2.1.1.1. Modelo biológico . . . 17
    - 2.1.1.2. Modelo de neurona artificial . . . 19
    - 2.1.1.3. Ventajas de las redes neuronales . . . 24
  - 2.1.2. Tipos de redes . . . 25
- 2.2. Algoritmos Genéticos . . . 31

2.2.1. Introducción 31
2.2.2. Operadores Genéticos 33

3. Simulador GedelosSim **39**
3.1. Estructura 39
3.2. Elementos 43
3.2.1. Ficheros de entrada 44
3.2.2. Electrodomésticos 48
3.2.3. Objetos de la casa 50
3.2.4. Sensores 52
3.2.5. Controladores 53
3.3. Ejecución 54

4. Controlador Neuronal: Implementación y Resultados **57**
4.1. Introducción 57
4.2. Software 59
4.3. El Planificador 67
4.3.1. Introducción 67
4.3.2. Arquitectura de la red 71
4.3.3. El algoritmo genético 74
4.3.4. Evolución 77
4.3.5. Análisis de la evolución 81
4.4. El Coordinador 84
4.4.1. Introducción 84
4.4.2. Arquitectura de la red 87
4.4.3. El algoritmo genético 89
4.4.4. Evolución 92
4.4.5. Análisis de la evolución 95
4.5. Sistema Completo 97
4.5.1. Postevaluación 99

5. Conclusiones y líneas futuras **103**
5.1. Conclusiones 103
5.2. Líneas futuras 105
5.3. Contribuciones 105

Bibliografía **107**

Índice de figuras

1.1. Crecimiento interanual de la demanda eléctrica (Imaz, 2007). 3
1.2. Estructura de GeDELOS-FV. 9
1.3. Imagen de la fachada sur de “MagicBox”. 10

2.1. Diagrama de bloques del sistema nervioso. 17
2.2. Estructura de una neurona (Martín and Sanz, 2001). 18
2.3. Jerarquía neuronal (Martín and Sanz, 2001). 19
2.4. Modelo de neurona artificial genérico (Martín and Sanz, 2001). 20
2.5. Modelo de neurona estándar (Martín and Sanz, 2001). 23
2.6. Red neuronal monocapa. 26
2.7. Red neuronal multicapa. 27
2.8. Red neuronal recurrente. 27

3.1. Estructura Magicbox. 41
3.2. Estructura simulador. 43
3.3. Perfil Fotovoltaico de potencia prevista. 48
3.4. Estructura de la librería de los electrodomésticos. 49

4.1. Arquitectura del controlador. 59
4.2. Ejemplo de restricción temporal. 67
4.3. Ejemplo de perfil fotovoltaico predicho. 68
4.4. Estructura del planificador. 69
4.5. Ejemplo de salida de planificación. 69
4.6. Ejemplo de planificación. 71
4.7. Ejemplo de perfil fotovoltaico predicho. 71
4.8. Arquitectura del planificador. 73
4.9. Ejemplo de fitness (a) datos de entrada y (b) función de fitness. 77
4.10. Resultados de la evolución: (a) 10 perfiles, (b) 30 perfiles, (c) 50 perfiles, (d) 90 perfiles, (e) 142 perfiles y (f) variación del intervalo temporal. . 80

4.11. Resultado de ejecución: (a) intervalos de 4 horas y (b) intervalo temporal único. 83
4.12. Perfiles con cambios bruscos. 84
4.13. Tareas coordinadas. 84
4.14. Ejemplo de coordinación. 86
4.15. Arquitectura del coordinador. 88
4.16. Función de fitness (a) comparación entre salidas y (b) comparación entre entrada y salida. 92
4.17. Ejemplo de entradas para 3 tareas distintas. 93
4.18. Resultados de la evolución: (a) 200 generaciones y (b) 10000 generaciones. 94
4.19. Resultados de la ejecución: (a) y (c) entradas al coordinador y (b) y (d) salidas del coordinador. 96
4.20. Ejemplo sistema completo: (a) entrada, (b) planificación y (c) coordinación. 99
4.21. Ejemplo sistema completo caso peor: (a) entrada, (b) planificación y (c) coordinación. 101
4.22. Ejemplo sistema completo caso mejor: (a) entrada, (b) planificación y (c) coordinación. 102

Índice de tablas

2.1. Funciones de Activación habituales (Martín and Sanz, 2001). 22
2.2. Clasificación de las redes neuronales por tipo de aprendizaje y arquitectura. 29

3.1. Ejemplo de fichero de parámetros. 46

Lista de Acrónimos

IES: Instituto de Energía Solar.

GD: Generación Distribuida.

GDE: Gestión de la Demanda Eléctrica.

GeDELOS-FV: Gestión de la Demanda ELéctrica dOméstica con tecnología Solar FotoVoltaica.

gedelosSim: simulador para el proyecto GeDELOS-FV.

Capítulo 1

Introducción y Objetivos

1.1. Motivación

El presente Proyecto Final de Carrera está motivado por la realización de un sistema de control para la Gestión Activa de la Demanda Eléctrica mediante el uso de Algoritmos Inteligentes en un entorno residencial con Generación Solar Fotovoltaica. El sistema de control se ha realizado en un entorno virtualizado que caracteriza una vivienda dotada de un sistema domótico que permite controlar los diferentes electrodomésticos de los que dispone una vivienda corriente.

El objetivo de este sistema es conseguir la máxima eficiencia energética, mediante la maximización del autoconsumo en detrimento del consumo de la red eléctrica. El sistema se basará en unas redes neuronales sintonizadas mediante algoritmos genéticos. Las redes neuronales poseen características de optimización, capacidad de generalización, adaptabilidad, herencia de la información que procesan, tolerancia a fallos, bajo consumo, etc. (Haykin, 2009). Por todas estas cualidades, las redes neuronales encajan perfectamente con el concepto de maximización del autoconsumo. Además, utilizaremos los algoritmos genéticos para sintonizar la red neuronal. Los algoritmos genéticos son algoritmos de búsqueda probabilística u optimización que transforman iterativamente un conjunto de objetos matemáticos llamado población, cada uno con un valor de coste (fitness) asociado, en una nueva población de descendientes usando operaciones genéticas naturales tales como crossover (reproducción

sexual) y mutación (Golberg, 2006).

Se pretende que el sistema sea capaz de planificar las tareas del hogar del día siguiente, teniendo en cuenta las preferencias del usuario (Castillo et al., 2009) y el perfil fotovoltaico previsto de generación del día siguiente (Caamaño-Martín et al., 2010), coordinándolas de tal manera que se maximice el autoconsumo. Así pues, el sistema actúa de forma casi transparente al usuario encargándose de la puesta en marcha de las tareas del hogar en los momentos de mayor producción energética, lo que conlleva un aumento de la eficiencia energética y un ahorro económico por parte del usuario.

1.2. Estado del arte

Hoy en día la electricidad es un recurso imprescindible para desempeñar cualquier tarea, ya sea en el hogar o en la oficina. Cada vez es más necesario el uso de aparatos eléctricos que nos simplifican las tareas y debido a las crecientes necesidades de la población, es necesario generar mayores cantidades de energía (S.G.P.E., 2007). Sin embargo, los recursos disponibles son limitados. La solución para cubrir las necesidades energéticas de la población no pasa únicamente por cambiar los métodos de generación eléctrica o mejorar los métodos de transporte (R.E.E., 2006), sino también por el usuario final, quien con un consumo eficiente podrá reducir su consumo energético (Pérez et al., 2005). De este modo, es necesario realizar una Gestión Activa de la Demanda Eléctrica, mediante la cual se minimice el consumo del usuario y de este modo optimice la energía consumida por éste (Pérez et al., 2005; Tecnalia, 2007).

Por otro lado, en los últimos años estamos viendo un aumento en el uso de sistemas energéticos distribuidos como pueden ser las instalaciones fotovoltaicas con las cuales cualquier usuario puede generar su propia energía para uso personal y/o entregarla a la red (Degner et al., 2006). Estos sistemas hacen más complicada la estructura de la red y aumentan la necesidad de su controlabilidad (Caamaño-Martín et al., 2009b).

La Gestión de la Demanda Eléctrica (Pérez et al., 2005) se perfila como una de las principales líneas de desarrollo a la hora de mejorar la eficiencia energética (CCEE,

2002). En las siguientes Secciones veremos detalladamente el entorno actual donde se enmarca este Proyecto Fin de Carrera, el cual se encuentra dentro de un proyecto de I+D de plan nacional, GeDELOS-FV (Caamaño-Martín et al., 2009a).

1.2.1. La demanda eléctrica en España

El análisis de la evolución de la demanda de energía en España en los últimos 20 años revela crecimientos continuos desde 1993 (ver Figura 1.1), con tasas de crecimiento promedio anual en el periodo 1996-2006 del 5 % (S.G.P.E., 2007), notablemente superiores al crecimiento del PIB (3,8 % en el mismo periodo) e indicativas de una baja intensidad energética de nuestra economía.

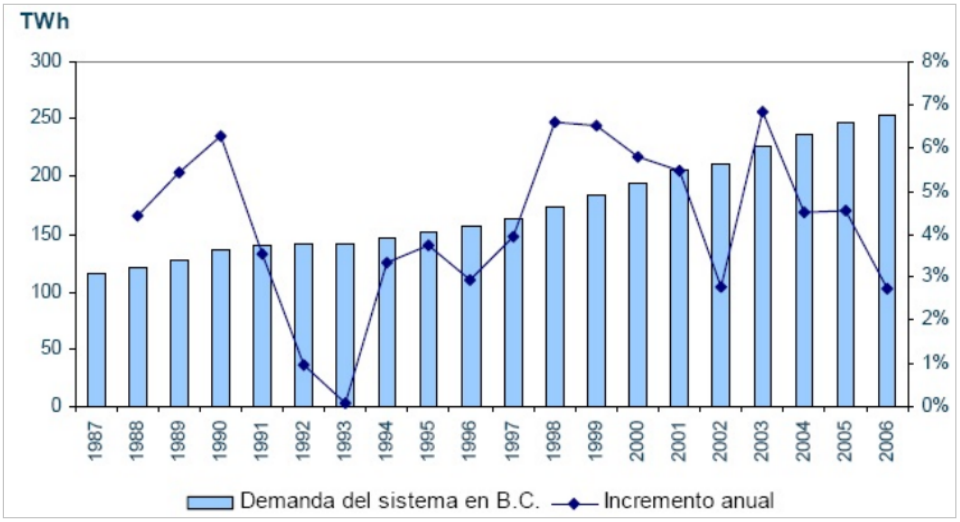


Figura 1.1: Crecimiento interanual de la demanda eléctrica (Imaz, 2007).

Por otra parte, del análisis de la evolución anual de la demanda en el sistema eléctrico peninsular, se puede constatar cómo los distintos usos eléctricos determinan las variaciones mensuales de consumo, siendo máximos durante los meses de diciembre y enero en invierno (puntas de invierno), y junio y julio en verano (puntas de verano) (Tecnalia, 2007). A parte de la evolución de la demanda eléctrica durante el año, hay que tener en cuenta la evolución de la demanda diaria, que presenta

fuertes oscilaciones dependientes de la temperatura ambiente, la actividad laboral y la evolución de la actividad económica (R.E.E., 1998). Las variaciones del perfil de carga diario, aún siendo en principio aleatorias, pueden ser explicadas en gran parte por modelos basados en los patrones de consumo de los principales sectores de consumidores (D.R.E.D., 2007).

En lo que respecta a la cobertura de la demanda en el sistema peninsular, cabe destacar la contribución de la generación acogida al denominado "Régimen Especial" (producción procedente de energías renovables y cogeneración), lo que confirma el crecimiento continuado experimentado por este tipo de generación en los últimos años (R.E.E., 2006).

A diferencia de otros productos energéticos, la electricidad no es almacenable a gran escala, por lo que el suministro requiere un equilibrio instantáneo entre la generación y la demanda. Como consecuencia, las infraestructuras de generación y red necesarias se dimensionan para asegurar el suministro eléctrico en las horas de máxima demanda. Sin embargo, las elevadas tasas de crecimiento del consumo en los últimos años han generado situaciones comprometidas para el sistema eléctrico. Ello obliga a incrementar cada año la capacidad del sistema para poder atender a la demanda en todo momento, aunque una parte importante de esta generación sólo se utiliza unas pocas horas, con el consiguiente coste económico, social y medioambiental (Imaz, 2007). Las consecuencias de este hecho, se traducen en variaciones importantes del precio de la electricidad, que en un mismo día puede variar en un factor de 5:1 entre horas de mayor y menor demanda (Tecnalia, 2007).

Por otra parte, el sistema eléctrico español se enfrenta a importantes desequilibrios regionales derivados de un modelo de generación fuertemente centralizado (R.E.E., 2006). Esto implica que la electricidad se genera a grandes distancias de los puntos de consumo. Así, en zonas donde existe mucha demanda y poca generación pueden existir problemas de congestión de la red debidos a la saturación de las líneas de transporte y distribución. Esto conlleva riesgos para la seguridad y calidad del consumo, además de las pérdidas de energía que provoca el transporte de electricidad (en torno al 9 %

en España) (Valentí, 2007).

De las diversas alternativas existentes para evitar los riesgos que pueden causar estos desequilibrios, hasta ahora la más empleada por los gestores de la red eléctrica han sido trabajos de mejora en el sistema eléctrico. Otras medidas identificadas, que guardan una estrecha relación con el presente Proyecto son:

- **La incorporación de Generación Distribuida cerca de los lugares de consumo** que, además de reducir la congestión de las líneas, permita disminuir las pérdidas eléctricas en el sistema debidas al transporte y distribución.

- **Una gestión eficaz de la Demanda Eléctrica** que permita disminuir los grandes picos de consumo eléctrico, de forma que los recursos del sistema eléctrico se utilicen con mayor eficiencia. En este sentido, una de las características más determinantes de las redes eléctricas del futuro será la capacidad de los consumidores de desempeñar un papel activo en la cadena del suministro eléctrico.

La demanda eléctrica anual de los hogares españoles ha experimentado un aumento sustancial en las dos últimas décadas, pasando de una tasa de crecimiento anual promedio del 1,3 % en el periodo 1989-1996 a un 5,7 % en el periodo 1998-2006 (D.R.E.D., 2007) como resultado del aumento de la población, un mayor equipamiento de los hogares, la escasa proporción en la adquisición de electrodomésticos eficientes y los bajos precios de la electricidad.

1.2.2. Generación fotovoltaica distribuida

Históricamente, los sistemas de suministro eléctrico se han diseñado siguiendo un esquema de integración vertical, sin embargo en los últimos años estamos asistiendo al crecimiento de la Generación Distribuida (GD, pequeños generadores dispersos por las redes eléctricas) para satisfacer las necesidades energéticas de un mundo cada vez más interconectado (Degner et al., 2006). Las ventajas de la Generación Distribuida pueden verse desde diferentes perspectivas:

- **Perspectiva estratégica:** ante la elevada dependencia de muchos países de combustibles fósiles para la generación eléctrica y el envejecimiento de las infraestructuras de transporte y distribución.

- **Perspectiva económica:** debido a la liberalización de los mercados eléctricos y nuevos desarrollos tecnológicos.

- **Perspectiva medioambiental:** debido a las emisiones de gases de efecto invernadero y otros contaminantes derivados del uso de combustibles fósiles y su influencia en el cambio climático.

Los beneficios que producen la GD con Energía Fotovoltaica alcanzan un mayor valor si la generación tiene lugar en las zonas donde se produce la demanda eléctrica y especialmente en áreas urbanizadas (Linder, 2007), donde se aprovecha la capacidad de integración arquitectónica que tienen los módulos fotovoltaicos. Los principales beneficios (Groppi, 2002) son:

- **En términos instantáneos:** reducción de pérdidas de transporte y distribución, mejoras en la calidad y continuidad del suministro en horas de alta demanda eléctrica y reducción de impactos medioambientales.

- **A medio y largo plazo:** reducción de la capacidad adicional necesaria para atender las puntas de consumo y distribución de inversiones futuras necesarias para aumentar la capacidad de las redes, sobre todo en baja tensión.

Los sistemas fotovoltaicos integrados en viviendas pueden ser de varios tipos:

- **Sistemas aislados:** son propios de lugares remotos a los que no llega la red de suministro eléctrico. Constan de un generador fotovoltaico, una batería, un regulador de carga y mecanismo de protección. En el caso de que se quieran emplear cargas en AC se incluye un inversor. Es habitual complementar la instalación con un pequeño generador diésel.

- **Sistemas con conexión a red:** es el tipo de sistema más habitual en viviendas en España y resto de Europa. La red eléctrica de suministro es de fácil acceso y la instalación fotovoltaica se conecta directamente a ella. El sistema fotovoltaico esta compuesto por el generador, inversor de conexión a red y elementos de medida y protección.

- **Sistemas híbridos:** están conectados a la red de suministro eléctrico al mismo tiempo que constan de algún elemento que permite almacenar energía, generalmente baterías.

De especial interés para la gestión de la demanda son los sistemas híbridos. Combinan las ventajas de la GD con la posibilidad de funcionar en modo autónomo, bien debido a un fallo de red, bien como decisión del usuario. Dentro de los sistemas híbridos es posible encontrar dos topologías distintas: sistemas de acoplo en AC y sistemas de acoplo en DC. Los sistemas de acoplo en AC son más eficientes energéticamente, su diseño es más modular y escalable y permiten la conexión de mayor potencia fotovoltaica. Este último será el utilizado en este Proyecto Fin de Carrera.

1.2.3. Sistemas de control

La Ingeniería de Control tiene una larga historia (Bennett, 1979), actualmente está presente en muchos campos tanto de la ciencia como de la ingeniería permitiendo mejorar la monitorización y el control de los sistemas donde es utilizada.

La convergencia del control, la comunicación y la computación (Graham and Kumar, 2003) se plantea en la actualidad como una integración de las tecnologías desarrolladas en cada campo específico, en el que se toma en consideración la integración con el entorno (Vasilakos and Pedriyzc, 2006). Esta convergencia señala la importancia de las arquitecturas frente a los algoritmos concretos de resolución de tareas, cuestión que ya había sido señalada desde los comienzos de las teorías de máquinas inteligentes (Valavanis and Saridis, 1992).

Sin embargo desde la perspectiva de la Gestión de la Demanda Eléctrica se encuentra en sus comienzos, no por falta de soluciones tecnológicas, sino por la ausencia casi total en la definición de los problemas. En cuestión de desarrollos domóticos, la situación es desoladora por cuanto que se observa una miríada de soluciones *ad-hoc* carentes de metodología. En general, los recientes estudios domóticos (Romero, 2004) toman en consideración la totalidad de la vivienda, pero casi siempre exclusivamente en términos de comunicación de datos, dejando de lado tanto los aspectos de control como los de las fuentes de energía.

Sin embargo, en recientes trabajos (Liang and Du, 2008) se presentan sistemas de control inteligente (controladores neuronales borrosos y otras técnicas provenientes de la IA) resolviendo de manera integral el problema de la calefacción, ventilación y aire acondicionado, tomando en consideración el problema de reducción de la potencia eléctrica instantánea consumida.

1.2.4. GeDELOS-FV

El presente Proyecto Fin de Carrera, se encuentra enmarcado dentro de un proyecto de plan Nacional, GeDELOS-FV (Caamaño-Martín et al., 2009a) (Gestión de la Demanda ELéctrica dOméstica con tecnología Solar Fotovoltaica) que pretende explorar las posibilidades de gestionar la demanda eléctrica doméstica mediante tecnologías de generación distribuida y sistemas de control, donde el sistema es capaz de modificar la curva de demanda en función de condicionantes de tipo interno y externo. El proyecto implementa un sistema de control de cargas de la vivienda teniendo en cuenta información propia de la casa y la proveniente de fuentes externas, consiguiendo con todo ello una optimización en el consumo eléctrico de la vivienda sin que exista una pérdida de funcionalidad.

Además del sistema de control, la vivienda dispone de un sistema fotovoltaico encargado de los siguientes objetivos: i) supervisión del funcionamiento del sistema fotovoltaico, ii) estimación de la producción eléctrica y iii) control del funcionamiento de la instalación fotovoltaica (Porro, 2010).

En la Figura 1.2 se muestra un esquema de la estructura completa de GeDELOS-FV. El proyecto se ha desarrollado en la vivienda "MagicBox", la cual dispone de todo el equipamiento necesario, para el control y monitorización.

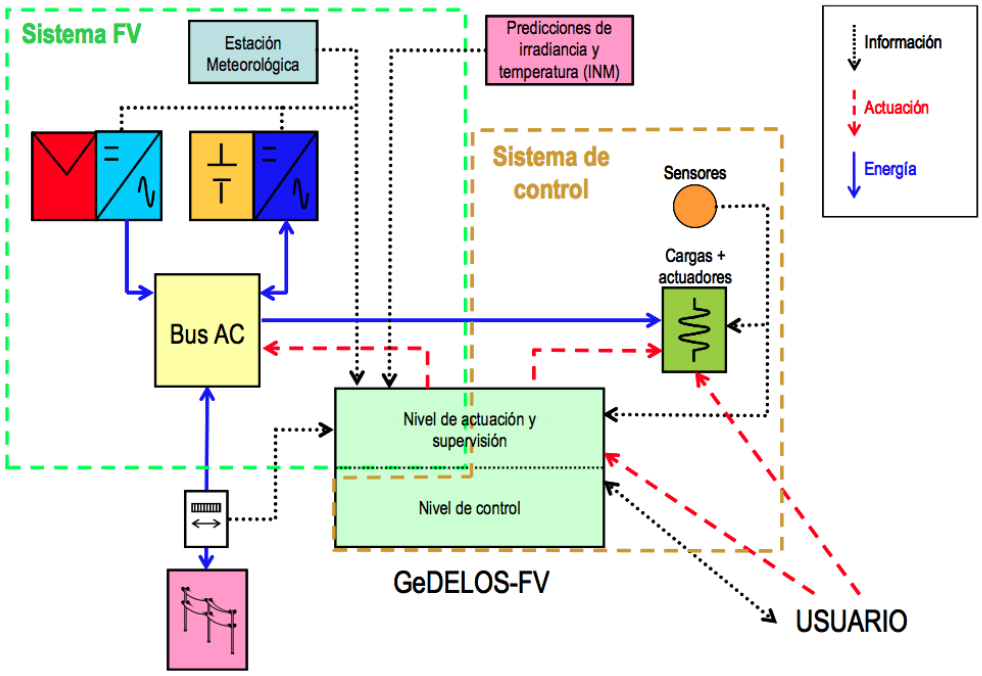


Figura 1.2: Estructura de GeDELOS-FV.

1.2.5. Magicbox

En otoño del 2005 se celebró en Washington D.C. la segunda edición del concurso internacional Solar Decathlon (Moon et al., 2005) patrocinado por el Departamento de Energía de los Estados Unidos. Solar Decathlon es un concurso dirigido a universidades y persigue difundir la posibilidad de conciliar las buenas prácticas arquitectónicas con un uso racional de la energía, a través del aprovechamiento de la energía solar tanto pasiva como activamente y del uso de tecnologías eficientes actualmente disponibles.

El prototipo "MagicBox" (ver Figura 1.3) constituye una propuesta novedosa

que combina la aplicación de principios de diseño bioclimáticos con la integración de tecnologías solares para la producción de electricidad y agua caliente sanitaria (solar fotovoltaica y solar térmica, respectivamente). El uso de las tecnologías de la información y las comunicaciones para el mantenimiento de las variables de confort constituye el tercer elemento innovador del prototipo del que disponemos. Asimismo, cuenta con un sistema de acondicionamiento novedoso basado en el calor latente y el cambio de estado de sustancias (denominadas "geles de cambio de estado") que se encuentran confinadas bajo el suelo de la casa, las cuales se cargan y descargan de energía mediante corrientes de aire inducidas.

Figura 1.3: Imagen de la fachada sur de "MagicBox".

De especial interés para el presente proyecto es el sistema fotovoltaico diseñado. Este incorpora la tecnología más avanzada de microgeneración, denominada "acoplamiento en AC", que garantiza el máximo aprovechamiento de la energía solar (generadores fotovoltaicos con distintas inclinaciones para aplanar la curva anual

de generación e inversores con seguimiento automático de la máxima potencia del generador), al tiempo que posibilita la incorporación de otras tecnologías de generación auxiliares, en este caso acumuladores electroquímicos y la conexión a red tanto para obtener energía como para verter en ella el excedente. La potencia instalada en la casa es de 8,3 kWp de generación fotovoltaica, producida por 77 módulos los cuales ocupan $63\,m^2$ de cubierta y $9\,m^2$ de la fachada sur y una capacidad en las baterías de 90 kWh.

En el interior, la vivienda está equipada con una lavadora, secadora, lavavajillas, horno, frigorífico/congelador y campana de extracción todos ellos de bajo consumo y dotados de un sistema de control domótico conectados mediante una red PLC (Castillo, 2009). Gracias a una pasarela, podemos comunicarnos y obtener tanto información de los electrodomésticos, como enviar órdenes básicas de funcionamiento. Aparte de estos electrodomésticos controlables la vivienda incorpora diferentes cargas que no están dotadas de este sistema como los ordenadores, luces, calefacción, aire acondicionado y vitrocerámica.

1.3. Objetivos del proyecto

El objetivo fundamental del proyecto es la realización de un controlador neuronal para la gestión activa de la demanda. Dicho controlador está dividido en dos grandes bloques:

- **Planificador Distribuido Neuronal.** La planificación de las distintas cargas se realiza de forma independiente para cada una de ellas, es decir, se hace una planificación distribuida y modular. Así pues tenemos un planificador por cada uno de los electrodomésticos. Las entradas del planificador se componen del perfil fotovoltaico de generación predicho para el día siguiente (Porro, 2010) y las preferencias horarias del usuario en cuanto a la realización de las tareas (Castillo, 2009). De esta forma se intenta optimizar la curva de la demanda respetando los requisitos del usuario.

- **Coordinador Neuronal.** Sus entradas se componen de las salidas de los planificadores y la duración de cada una de las tareas que el usuario quiere llevar a cabo. El coordinador tratará de ajustar dichas tareas para que no se solapen en el tiempo y se respete lo más posible su similitud con las salidas planificadas.

Para la consecución de los objetivos se ha dividido el proyecto en los siguientes subobjetivos:

- Estudio y elección de la red neuronal.
- Desarrollo y análisis de una arquitectura neuronal para satisfacer la optimización del autoconsumo.
- Estudio de algoritmos genéticos.
- Análisis de la estructuración del cromosoma.
- Estudio de los algoritmos en función de parámetros de escalabilidad y convergencia.
- Estudio de la implementación conjunta de ambos algoritmos.
- Implementación de dichos algoritmos en una vivienda simulada para la comprobación del funcionamiento de los mismos.

En resumen, con el desarrollo de un controlador neuronal se persigue aplicar técnicas más sofisticadas de control para resolver el problema de la eficiencia energética acercándonos a los objetivos del proyecto GEDELOS-FV. Con esta primera versión del sistema de control, se perfila la arquitectura del sistema y se obtienen los primeros resultados respecto a la eficiencia energética buscada.

1.4. Organización del proyecto

En el Capítulo 1 se ha descrito la motivación y estado del arte del Proyecto Final de Carrera dentro de un proyecto del plan nacional más amplio (GeDELOS-FV). Igualmente, se han definido los objetivos a llevar a cabo en el Proyecto Final de Carrera.

En el Capítulo 2 se introducen las redes neuronales y los algoritmos genéticos. Además también se describen los diferentes tipos de redes neuronales que se han estudiado y se exponen los diferentes operadores genéticos.

En el Capítulo 3 se desarrolla la estructura del simulador, la implementación de cada uno de los electrodomésticos, la inclusión de los perfiles energéticos, el usuario, etc. En definitiva se expone el entorno virtualizado en el que se han desarrollado los estudios.

En el Capítulo 4 se desarrollan los controladores neuronales que se han desarrollado. Y se describen los resultados que se han obtenido así como su implementación. También se describe el entrenamiento llevado a cabo con el algoritmo genético en cada una de las partes del controlador.

Finalmente, el Capítulo 5 muestra las conclusiones y líneas futuras.

Capítulo 2

Redes Neuronales y Algoritmos Genéticos

A lo largo de este Capítulo describiremos los algoritmos inteligentes, provenientes de la inteligencia artificial, que se han utilizado para el desarrollo de este Proyecto Fin de Carrera.

En la Sección 2.1 se introducen las redes las neuronales, desde el modelo en el que se inspiran hasta como se realiza su implementación artificial. También se presentarán distintos tipos de redes neuronales existentes en la literatura.

En la Sección 2.2 se describen los algoritmos genéticos así como los distintos operadores genéticos. Nos centraremos en los operadores más utilizados, aunque también se expondrán los algoritmos genéticos menos corrientes y más sofisticados.

2.1. Redes neuronales

En esta Sección introduciremos el concepto de las redes neuronales, así como para qué sirven y justificación en este Proyecto Fin de Carrera.

El trabajo con redes neuronales artificiales, o comúnmente denominadas redes neuronales, ha sido motivado por el hecho de que el cerebro humano procesa la información de forma completamente distinta a la de un ordenador convencional. El cerebro es un ordenador altamente complejo, no lineal y que procesa la información en paralelo. Tiene la habilidad de organizar las partes que constituyen su estructura,

conocidas como neuronas, para realizar determinados cálculos (reconocimiento de patrones o control motriz) mucho más rápido que el ordenador más potente existente hoy en día (Haykin, 2009).

2.1.1. Introducción

Una red neuronal artificial trata de modelar la forma en que el cerebro humano realiza determinadas tareas o funciones. Las redes neuronales están divididas en pequeñas partes que se encargan de procesar la información denominadas neuronas o unidades de proceso (Haykin, 2009). Pero podemos recoger todo lo dicho anteriormente en la siguiente definición:

> **Una red neuronal** es un procesador paralelo masivamente distribuido formado por unidades simples de procesamiento que tienen una propensión natural para almacenar conocimiento experimental y hacerlo disponible para su uso. Se asemeja al cerebro por dos motivos:
>
> - El conocimiento es adquirido por la red de su entorno a través de un proceso de aprendizaje.
> - Las conexiones interneuronales, conocidas como pesos sinápticos, se usan para almacenar el conocimiento adquirido (Haykin, 2009).

El procedimiento usado en el proceso de aprendizaje se denomina algoritmo de aprendizaje. En este Proyecto Fin de Carrera no se han utilizado algoritmos de aprendizaje, sino que hemos hecho uso de algoritmos genéticos para sintonizar las redes neuronales (ver Sección 2.2). El algoritmo genético al igual que el algoritmo de aprendizaje se encarga de modificar los pesos sinápticos de la red neuronal para acercarse al objetivo deseado. La forma en que se modifican los pesos sinápticos se describirá más adelante dentro de este Capítulo cuando se explique el algoritmo genético en la Sección 2.2.

2.1.1.1. Modelo biológico

Para comprender mejor el funcionamiento de las redes neuronales artificiales es conveniente obtener ciertas nociones del modelo en el que está inspirado. Sin embargo, es importante tener en cuenta que las redes neuronales artificiales se encuentran únicamente inspiradas en las biológicas, y su funcionamiento dista bastante del comportamiento real (no totalmente conocido) de las mismas. El sistema nervioso humano puede interpretarse como un sistema formado por tres bloques (ver Figura 2.1) (Haykin, 2009):

- **Receptores:** son los encargados de transformar los estímulos externos captados por las terminaciones nerviosas del cuerpo en impulsos eléctricos que pueda entender el cerebro.
- **Cerebro** *(red neuronal):* es la parte central del sistema nervioso, la cual recibe continuamente información, la procesa y toma las decisiones apropiadas. En la Figura 2.1 existe una doble conexión con el resto de elementos. Hay una transmisión de la información hacia adelante desde los receptores a los actuadores y otra en sentido contrario, lo que indica una realimentación en el sistema nervioso.
- **Actuadores:** convierte los impulsos eléctricos generados en el cerebro en respuestas del cuerpo humano.

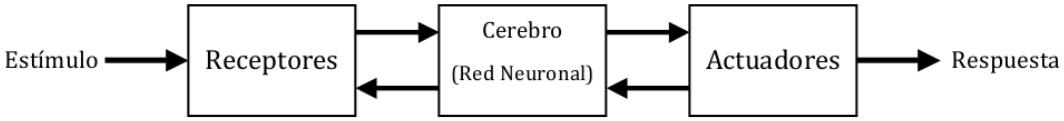


Figura 2.1: Diagrama de bloques del sistema nervioso.

El cerebro a su vez está compuesto por unidades mucho más pequeñas, las neuronas. Un cerebro puede contener aproximadamente 10 mil millones de neuronas

y 60 trillones de conexiones o sinapsis (Purves, 2006). Aunque las neuronas pueden presentar múltiples formas, muchas suelen tener un aspecto similar con un cuerpo celular o soma, del que surge un árbol de ramificaciones (árbol dentrítico) formado por las dentritas y del que también sale una fibra tubular denominada axón (ver Figura 2.2). Las funciones típicas de cada una de las partes de las neuronas son (Bear et al., 2007):

- **Dentrita** es el canal principal de entrada de información. Aunque pueden llevarse a cabo pequeñas operaciones en los árboles dentríticos que resultan de la asociación de dentritas.

- **Soma** es el cuerpo de la neurona y donde se procesa principalmente la información. Éste puede recibir directamente la información sin pasar por la dentrita.

- **Axón** es el canal de salida.

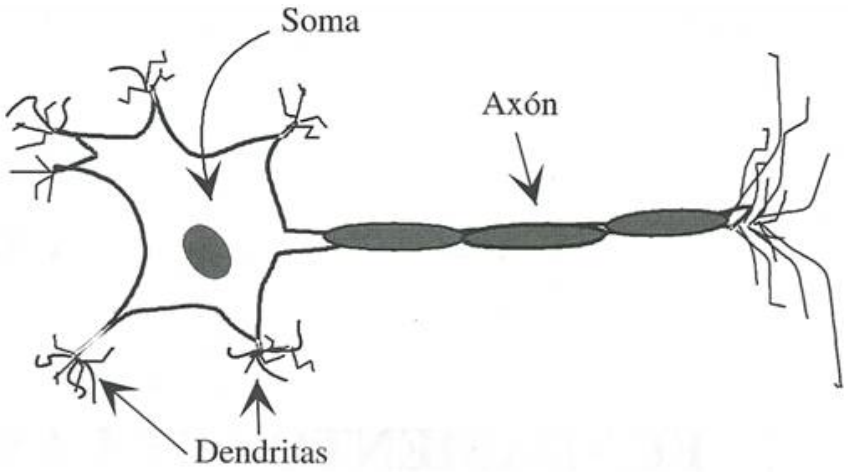


Figura 2.2: Estructura de una neurona (Martín and Sanz, 2001).

Las interacciones entre neuronas se lleva a cabo mediante la sinapsis. En la sinapsis las neuronas pueden ser: neuronas que envían las señales, presinápticas, o

las que envían la información, neuronas postsinápticas. Las señales nerviosas pueden transmitirse eléctricamente o químicamente. La transmisión química se da fuera de la neurona y consiste en el intercambio de neurotransmisores. La eléctrica prevalece en el interior de la neurona y hace uso de descargas que se producen en el cuerpo y se propagan por el axón (Bear et al., 2007).

Como hemos visto, las neuronas biológicas se agrupan en conjuntos organizadas en capas formando un sistema con funcionalidad propia. El conjunto de todos estos subsistemas da lugar a un sistema global. Las redes neuronales artificiales se agrupan de forma similar para obtener esta estructura. Partimos de que en el sistema artificial la unidad de procesamiento más pequeña será la neurona artificial, que se agrupará en capas, varias capas formarán una red neuronal y una red neuronal (o varias de ellas) junto con las entradas y salidas y algún módulo convencional adicional, constituirán el sistema global de proceso (ver Figura 2.3) (Martín and Sanz, 2001).

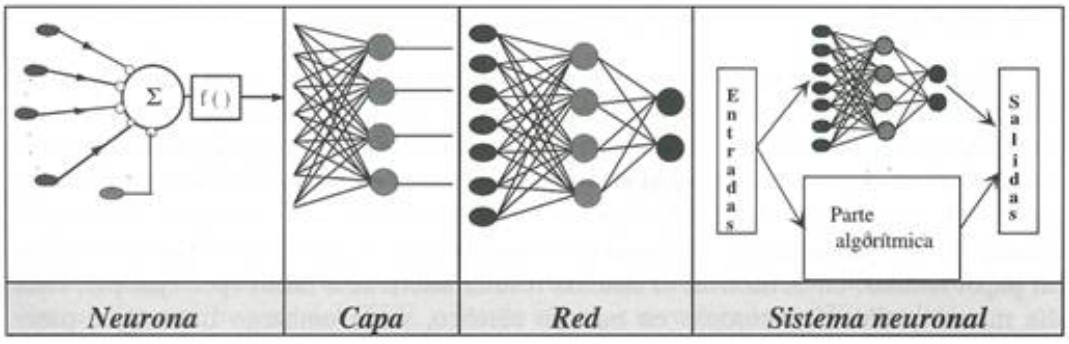


Figura 2.3: Jerarquía neuronal (Martín and Sanz, 2001).

2.1.1.2. Modelo de neurona artificial

Una vez conocido el modelo bioinspirado de las redes neuronales artificiales es necesario saber como se modelan éstas. Como ya se ha dicho con anterioridad una neurona es la unidad de proceso fundamental que recibe información del exterior, de una o varias neuronas presinápticas, y proporciona una única respuesta (Martín and

Sanz, 2001).

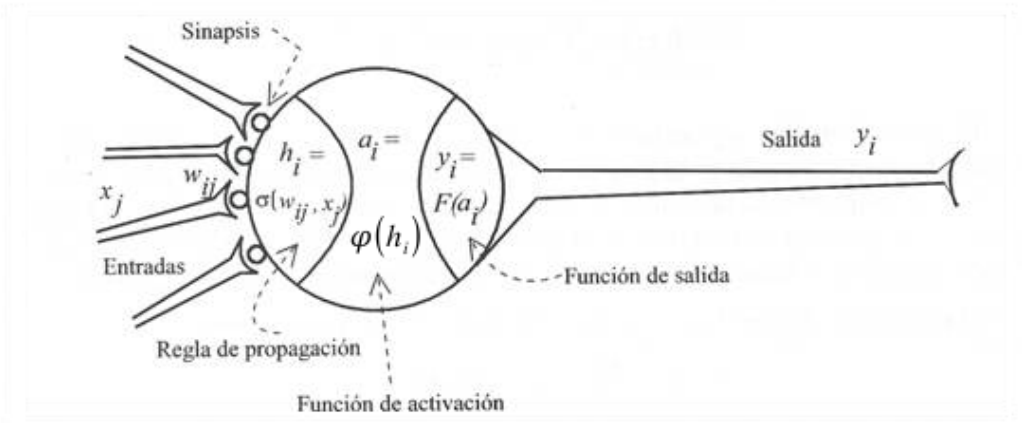


Figura 2.4: Modelo de neurona artificial genérico (Martín and Sanz, 2001).

Los elementos que componen la neurona de la Figura 2.4 son (Haykin, 2009; Martín and Sanz, 2001):

- **Entradas ($x_j(t)$).** Es el conjunto de salidas de las neuronas presinápticas. Tanto las entradas como las salidas pueden ser binarias o continuas dependiendo de la aplicación y del modelo.

- **Pesos sinápticos (w_{ij}).** Representan la intensidad de la interacción entre la neurona presináptica j y la neurona postsináptica i.

- **Regla de propagación ($\sigma(w_{ij}, x_j(t))$).** Permite obtener, junto con las entradas y los pesos, el valor del potencial postsináptico $h_i(t) = \sigma(w_{ij}, x_j(t))$.

  La regla de propagación más habitual es de tipo lineal y consiste en una suma ponderada de las entradas con los pesos sinápticos

  $$h_i(t) = \sum_j w_{ij} \cdot x_j(t) \tag{2.1}$$

  De esta forma dada una entrada positiva, si el peso es negativo se tenderá a inhibir la neurona postsináptica, mientras que si es positivo se tenderá a excitarla.

También existen otras funciones de propagación, por ejemplo la de la distancia euclídea

$$h_i^2(t) = \sum_j (x_j(t) - w_{ij})^2 \tag{2.2}$$

Esta regla opera de manera distinta a la anterior ya que representa la distancia entre las entradas y los pesos. Cuando la diferencia de las entradas y los pesos es pequeña se tenderá a inhibir la neurona mientras que si la diferencia es elevada estaremos excitándola.

- **Función de activación ($\varphi_i(a_i(t-1), h_i(t))$).** Proporciona el estado de activación actual ($a_i(t)$) de la neurona en función del estado de activación anterior ($a_i(t-1)$) y del potencial postsináptico actual ($h_i(t)$), es decir,

$$a_i(t) = \varphi_i(a_i(t-1), h_i(t)). \tag{2.3}$$

Aunque en muchos casos se considera que el estado actual de la neurona no depende del estado anterior, por lo que:

$$a_i(t) = \varphi_i(h_i(t)). \tag{2.4}$$

La función de activación $\varphi(\cdot)$ se suele considerar determinista, monótona creciente y continua. Las funciones de activación más usadas se recogen en la Tabla 2.1.

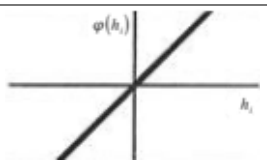
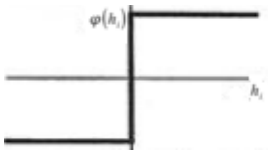
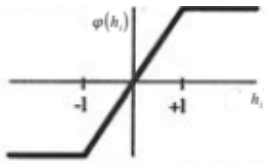
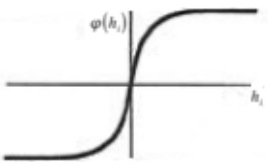
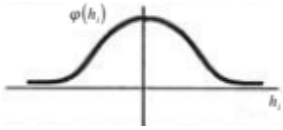
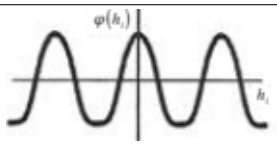
	Función	Rango	Gráfica
Identidad	$a_i(t) = h_i(t)$	$(-\infty, +\infty)$	
Escalón	$a_i(t) = signo(h_i(t))$ $a_i(t) = H((h_i(t))$	$[-1, +1]$ $[0, 1]$	
Lineal a tramos	$x = h_i(t)$ $a_i(t) = \begin{cases} -1 & \text{si } x < -l, \\ x & \text{si } -l \geq x \leq +l, \\ 1 & \text{si } x > +l. \end{cases}$	$[-1, +1]$	
Sigmoidea	$a_i(t) = \frac{1}{1+e^{-h_i(t)}}$ $a_i(t) = tgh(h_i(t))$	$[0, +1]$ $[-1, +1]$	
Gaussiana	$a_i(t) = A \cdot e^{-B \cdot h_i(t)}$	$[0, +1]$	
Sinusoidal	$a_i(t) = A \cdot sen(\omega \cdot h_i(t) + \phi)$	$[-1, +1]$	

Tabla 2.1: Funciones de Activación habituales (Martín and Sanz, 2001).

- **Función de salida ($F_i(a_i(t))$).** Proporciona la salida actual ($y_i(t) = F_i(a_i(t))$) en función de su estado actual de activación. En la mayoría de los casos suele usarse como función de salida la identidad de tal modo que la salida es la función de activación, $y_i(t) = F_i(a_i(t)) = a_i(t)$.

Finalmente la operación que lleva a cabo la neurona se expresa del siguiente modo:

$$y_i(t) = F_i(\varphi_i[a_i(t-1), \sigma_i(w_{ij}, x_j(t))]) \tag{2.5}$$

Este modelo se inspira del modelo biológico en el sentido de que trata de juntar varias entradas y proporcionar una respuesta que se propagará hacia adelante.

En este Proyecto Fin de Carrera se ha tratado con un modelo neuronal estándar que consiste en (ver Figura 2.5): el conjunto de entradas $x_j(t)$ y pesos sinápticos w_{ij}, la regla de propagación de la suma ponderada $h_i(t) = \sigma_i(w_{ij}, x_j) = \sum w_{ij} \cdot x_j(t)$, la función de activación que consiste en una sigmoide que sólo depende de su potencial postsináptico actual $a_i(t) = \varphi_i(h_i(t)) = \frac{1}{1+e^{-h_i(t)}}$ y la función de salida identidad $y_i(t) = F(a_i(t)) = a_i(t)$. También se ha añadido un sesgo (θ_i) denominado en inglés *bias*. Este parámetro sirve como ajuste en el cálculo de la neurona ya que introduce un grado más de libertad. θ_i tiene el efecto de aumentar o reducir la función de activación dependiendo de que su valor sea positivo o negativo respectivamente (Haykin, 2009). Por lo tanto, la operación completa que realizan las neuronas, es:

$$y_i(t) = \frac{1}{1 + e^{-(\sum w_{ij} \cdot x_j(t) + \theta_i)}} \tag{2.6}$$

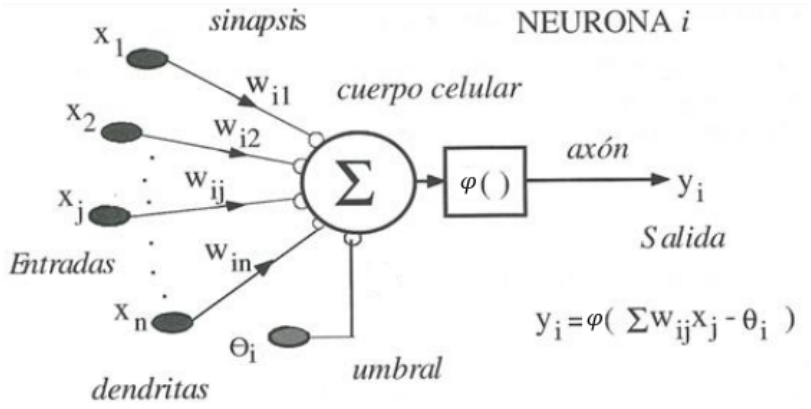


Figura 2.5: Modelo de neurona estándar (Martín and Sanz, 2001).

2.1.1.3. Ventajas de las redes neuronales

Después de haber descrito los modelos neuronales que se han utilizado, es necesario exponer las ventajas que justifiquen su uso. Las redes neuronales extraen su poder de computación de su estructura distribuida y de la capacidad de aprender (evolucionar en nuestro caso) y de generalización. La generalización consiste en obtener salidas razonables para entradas que no se encontraron durante el entrenamiento. Estas propiedades hace que las redes sean capaces de encontrar solución a problemas complejos. Además las redes neuronales ofrecen las siguientes propiedades y capacidades (Haykin, 2009):

- **No linealidad.** Las neuronas pueden ser lineales o no lineales y de este modo las redes podrán ser lineales o no. Además, este tipo de no linealidad es distribuida a través de toda de la red. La propiedad de no linealidad es importante debido a que se pueden plantear problemas no lineales a resolver con redes neuronales.

- **Mapa de entrada-salida.** Las redes neuronales son capaces de aprender mediante un aprendizaje supervisado, que implica la modificación de pesos sinápticos mediante una serie de ejemplos de entrenamiento. Cada ejemplo consiste en una señal de entrada a la que le corresponde una salida deseada, entonces la red ajusta sus pesos para que la salida se parezca lo más posible a la deseada. Luego se presentan todos los ejemplos de entrenamiento y se escogen en orden aleatorio. Así es como la red aprende de los ejemplos construyendo un mapa de entrada-salida del problema de antemano. De este modo cuando a la red se le presente una entrada distinta de las del entrenamiento pero parecida en su forma, la red debe ser capaz de dar una respuesta válida que resuelva el problema.

- **Adaptabilidad.** Las redes neuronales tienen una capacidad intrínseca de adaptar sus pesos sinápticos a cambios que ocurren a su alrededor. Una red neuronal entrenada para resolver un problema específico puede volverse a entrenar para operar en diferentes condiciones. Además cuando una red

neuronal está operando en un ambiente no estacionario, la propia red neuronal es capaz de modificar sus pesos sinápticos en tiempo real, gracias a algoritmos de aprendizaje, como la regla de Hebb (Haykin, 2009).

- **Confianza en la respuesta** en un clasificador de patrones. Una red neuronal puede ser diseñada para dar información no sólo en la selección de un patrón en concreto, sino también de la confianza depositada en la elección.

- **Información contextual.** En una red neuronal el conocimiento está representado por cada una de las neuronas y de sus funciones de activación. Cada neurona dentro de la red neuronal se ve afectada por la actividad global del resto de neuronas.

- **Tolerancia a fallos.** Una red neuronal es capaz de seguir en funcionamiento a pesar de que la sinapsis entre neuronas esté dañada, esto se debe a la naturaleza distribuida de la información almacenada en la red.

- **Uniformidad de análisis y diseño.** Existe uniformidad a la hora de usar las redes neuronales en los distintos ámbitos de aplicación, ya que usan la misma notación, la neurona es la parte común a todas las redes, los algoritmos de aprendizaje son compartidos por diferentes aplicaciones, etc.

- **Analogía neurobiológica.** El diseño de las redes neuronales está inspirado en el cerebro humano, puede observarse en el procesamiento distribuido y paralelo de la información, además de la capacidad de tolerancia a fallos, su rapidez y poder de resolución de problemas. Por este motivo las redes neuronales son un instrumento utilizado para la resolución de problemas complejos.

2.1.2. Tipos de redes

Una vez explicado los principios de funcionamiento de las redes neuronales y las razones de su uso, en esta Sección se clasificarán de manera muy general los distintos tipos de redes neuronales. Las redes neuronales se pueden clasificar de

muchas maneras, una de ellas y la más sencilla es según la arquitectura de red (Haykin, 2009):

- **Redes monocapa** son aquellas en las que las neuronas se organizan por capas. Este tipo de redes están formadas por una capa de entrada de nodos que envían su información a la capa de neuronas de salida donde se procesa la información de la entrada (ver Figura 2.6). El flujo de información va en una sola dirección, y este recorre la red hacia adelante.

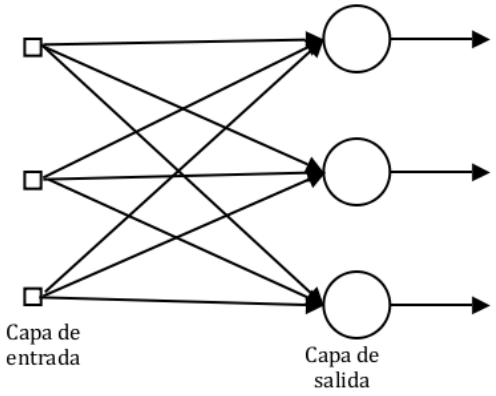


Figura 2.6: Red neuronal monocapa.

- **Redes multicapa.** Este tipo de arquitectura es una generalización del tipo anterior y tiene más de una capa de procesamiento de la información entre la capa de entrada y la de salida. Las neuronas de las capas intermedias se denominan neuronas ocultas y a la capa se la denomina capa oculta. Es oculta ya que esta parte de la red neuronal no es accesible directamente desde el exterior. Al añadir una capa oculta la red es capaz de obtener estadísticos de mayor orden de sus entradas. Este tipo de red puede estar total o parcialmente conectada (ver Figura 2.7).

- **Redes recurrentes.** Este tipo de red se diferencia de las anteriores en la existencia de lazos de realimentación en la red. Estos lazos pueden ser entre

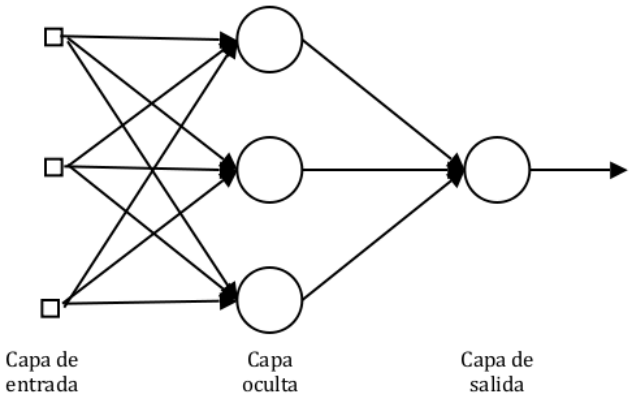


Figura 2.7: Red neuronal multicapa.

neuronas de diferentes capas, neuronas de la misma capa o, entre una misma neurona. Esta estructura la hace especialmente adecuada para estudiar la dinámica de los sistemas no lineales (ver Figura 2.8).

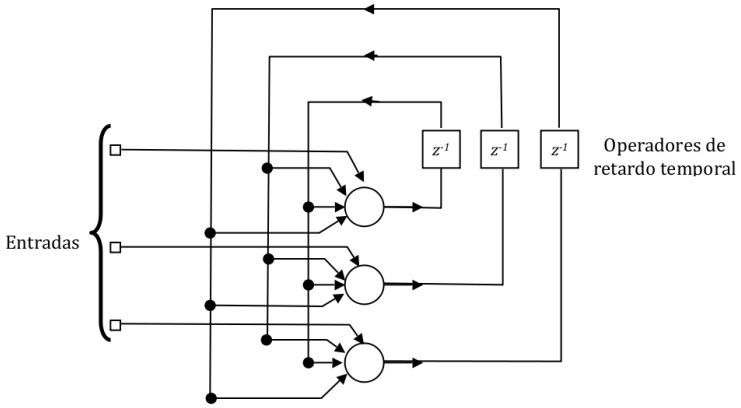


Figura 2.8: Red neuronal recurrente.

Pero también se pueden clasificar las redes neuronales por el tipo de aprendizaje. El aprendizaje es un proceso por el cual se ajustan los parámetros libres de la red a través de un proceso de estimulación por el entrono que rodea la red (Martín and Sanz, 2001). Los tipos de aprendizaje más comunes son:

- **Aprendizaje supervisado.** Se presenta a la red un conjunto de patrones, junto con la salida deseada u objetivo, e iterativamente ésta ajusta sus pesos hasta que la salida tiende al objetivo, para ello utiliza información del error que comete en cada paso.

- **Aprendizaje no supervisado o autoorganizado.** En este tipo de aprendizaje se presenta a la red multitud de patrones sin adjuntar la salida que se desea. La red, por medio de la regla de aprendizaje, estima la función densidad de probabilidad de los patrones de entrada, a partir de lo cual puede reconocer regularidades en el conjunto de entrada, extraer rasgos, o agrupar patrones según su similitud.

- **Aprendizaje híbrido.** Es una combinación de los dos tipos de aprendizajes anteriores, el supervisado y el no supervisado, los cuales tienen lugar en distintas capas de neuronales.

- **Aprendizaje reforzado.** En este aprendizaje se emplea información sobre el error cometido, como en el aprendizaje supervisado, pero en este caso sólo existe una señal de error, que representa un índice global del rendimiento de la red (sólo se le indica a la red lo bien o lo mal que lo está haciendo). Además no se suministra explícitamente la salida deseada, como en el caso no supervisado.

Una vez indicados los principales criterios de clasificación, en la Tabla 2.2 se recoge la clasificación de las principales redes neuronales atendiendo a su arquitectura y al algoritmo de aprendizaje.

De entre todas las redes anteriores cabe destacar (Haykin, 2009; Martín and Sanz, 2001):

- **Perceptrón Simple.** Es un clasificador, asigna a un vector de N valores un valor binario, usando una transformación no lineal. Así cada vector pertenece a una de las particiones que crea el perceptrón.

Modelos de redes neuronales artificiales

Supervisado		**No supervisado**	
Realimentados	*Unidireccionales*	*Realimentados*	*Unidireccionales*
BSB	Perceptrón	ART	LAM y OLAM
Fuzzy Cog. Map	Adalina/Madalina	Hopfield	Mapas de Kohonen
BP through time	Perceptrón Multicapa	BAM	Neocognitrón
	Back Propagation (BP)		Redes PCA
	Time-delay NN		
	CMAC		
	Correlación en cascada		
	Máquina de Boltzman		
	LVQ		
	GRNN		
	Support Vector Machine		

Híbrido	**Reforzado**
RBF	Premio-castigo asociativo
Contrapropagación	Crítico adaptativo

Tabla 2.2: Clasificación de las redes neuronales por tipo de aprendizaje y arquitectura.

- **Perceptrón Multicapa.** Se trata de una variante del perceptrón simple al que se le han añadido capas ocultas y por lo tanto, es capaz de resolver problemas que no son linealmente separables.

- **Adalina.** Su nombre proviene de ADAptative Linear Neuron, su modelo de neurona es similar a la del perceptrón pero de respuesta lineal, cuyas entradas pueden ser continuas.

- **Back Propagation.** Se trata de un algoritmo de aprendizaje supervisado que se usa para entrenar redes neuronales artificiales. El algoritmo consiste en minimizar un error por medio de descenso de gradiente, por lo que la parte esencial del algoritmo es el cálculo de las derivadas parciales de dicho error con respecto a los parámetros de la red neuronal.

- **Redes de Hopfield.** Es una forma de red neuronal artificial recurrente creada por John Hopfield (Hopfield, 1982). Las redes de Hopfield se usan como sistemas

de memoria asociativa con unidades binarias. Están diseñadas para converger a un mínimo local, pero la convergencia a uno de los patrones almacenados no está garantizada.

- **Máquina de Boltzman.** Pueden considerarse como la contrapartida estocástica y generativa de las redes de Hopfield. Son capaces de aprender mediante representaciones internas y de representar y resolver complicados problemas combinatorios.

- **Mapas autoorganizados.** Los mapas autoorganizados o SOM (Self-Organizing Map), también llamados redes de Kohonen son un tipo de red neuronal no supervisada, competitiva, distribuida de forma regular en una rejilla de, normalmente, dos dimensiones, cuyo fin es descubrir la estructura subyacente de los datos introducidos en ella. A lo largo del entrenamiento de la red, los vectores de datos son introducidos en cada neurona y se comparan con el vector de peso característico de cada neurona. La neurona que presenta menor diferencia entre su vector de peso y el vector de datos es la neurona ganadora (o BMU) y ella y sus vecinas verán modificados sus vectores de pesos.

- **ART (Adaptative Resonance Theory).** Es un modelo de red neuronal artificial que basa su funcionamiento en la manera en que el cerebro procesa información y que describe una serie de modelos de redes neuronales que utilizando métodos de aprendizaje supervisado y no supervisado abordan problemas tales como el reconocimiento y la predicción de patrones.

Éstas son unas de las redes más usadas pero existen muchos otros tipos de redes. De entre todas estas redes la que se ha escogido para el desarrollo de este Proyecto Fin de Carrera es el perceptrón multicapa, consistente en una capa de entrada, una capa oculta y una capa de salida. Sin embargo, como ya se ha comentado, no hemos utilizado un algoritmo de aprendizaje, sino que se ha hecho uso de algoritmos genéticos como sintonizadores de la red. En la Sección 2.2 se explicará en qué consisten estos algoritmos.

2.2. Algoritmos Genéticos

Los algoritmos genéticos tienen muchos campos de aplicación entre los que destacan: búsqueda, optimización y aprendizaje de máquinas. A pesar de que los algoritmos genéticos se llevan utilizando mucho tiempo, no son tan explotados como otros algoritmos de búsqueda y optimización. En nuestro PFC, el algoritmo genético se ha aplicado para sintonizar las redes neuronales, ha sido el encargado de modificar los parámetros libres de la red neuronal conforme a unas pautas dadas según una determinada función.

2.2.1. Introducción

Los algoritmos genéticos son algoritmos de búsqueda probabilística u optimización basados en mecanismos de selección natural y genética inspirados en principios darwinianos. Se encargan de transformar iterativamente un conjunto de objetos matemáticos llamado población, cada uno con un valor de coste (*fitness*) asociado, en una nueva población de descendientes usando operaciones genéticas naturales tales como crossover (reproducción sexual) y mutación (Golberg, 2006). La función de coste es la función que queremos optimizar. Por medio de la función de coste asignamos un valor a cada individuo, de forma que cuanto mayor sea mejor será ese individuo con relación al problema. Las operaciones más comunes en algoritmos genéticos se desarrollan con detalle en la Sección 2.2.2.

El estudio de estos algoritmos se ha llevado a cabo debido a la búsqueda de una mayor robustez en sistemas artificiales. Dicha robustez hay que entenderla como el compromiso entre la eficacia y la eficiencia. La respuesta se encontró de nuevo en el estudio de sistemas biológicos y se llegó a la conclusión de que: cuando la eficiencia y eficacia es deseada, la naturaleza lo hace mejor. Se demostró posteriormente teóricamente y científicamente que los algoritmos genéticos producían búsquedas eficientes en espacios complejos. Estos algoritmos son computacionalmente simples a la par que potentes y además, no están limitados por condiciones restrictivas del espacio de búsqueda (continuidad, derivabilidad, etc.) (Golberg, 2006).

La visión clásica de optimización consiste en mejorar hasta llegar al mejor resultado, pero en los sistemas biológicos no se puede establecer un criterio claro sobre cuál es el mejor criterio o lo que es óptimo. Para los algoritmos genéticos entenderemos la optimización como un proceso de mejora que va a consistir siempre en mejorar la situación anterior. Los algoritmos genéticos se caracterizan por:

- Trabajar con una codificación del conjunto de parámetros en una cadena de caracteres de longitud finita sobre un alfabeto finito (cromosoma).
- Utilizar una población de individuos, de esta forma se tiene una visión del conjunto y no de un sólo punto. Por lo tanto, se realiza con búsqueda en paralelo.
- Usar una función objetiva o información del coste asociado a cada individuo y prescindir de cualquier otro conocimiento; concentrándose en la búsqueda de mejores individuos.
- Usar reglas probabilísticas para las transiciones entre generaciones. Las reglas de transición son operadores estocásticos (Golberg, 2006).

La analogía con los sistemas biológicos puede observarse en que las cadenas de caracteres que describen a cada individuo se denominan cromosomas que son de longitud finita. Al conjunto total del paquete genético (conjunto de cromosomas) se llama genotipo y si juntamos el paquete genético junto con su entorno se denomina fenotipo. Un cromosoma está formado por genes que tienen un determinado valor (*alleles*) y ocupan una determinada posición (*locus*) (Klug et al., 2006). Algunos conceptos importantes son: el de esquema (*schema*), son las similitudes dentro de las cadenas de caracteres o cromosomas, y el de bloques constituyentes (*building blocks*), que son aquellos esquemas mejores y de longitud corta que se propagan de generación en generación (Golberg, 2006; Klug et al., 2006).

Como ya hemos dicho al principio de la Sección el algoritmo genético se encarga de transformar una población de individuos en otra población de descendientes mediante el uso de unos operadores genéticos, pero para entender mejor como se realiza esa

transformación es necesario explicar en qué consisten esos operadores. Los enunciados anteriormente son sólo los más sencillos pero existen otros operadores mucho más complejos que se explicarán a continuación.

2.2.2. Operadores Genéticos

Los operadores genéticos más utilizados son los de reproducción, crossover (reproducción sexual) y mutación. A continuación se explicará en que consiste cada uno de ellos.

- **Reproducción.** Es un proceso en el que el cromosoma de un individuo es copiado de acuerdo a su valor de la función de coste (*fitness*) f (se puede definir la función de coste como la función que queremos maximizar). El que se copien los cromosomas de acuerdo a su valor de coste quiere decir que aquellos individuos dentro de la población con un mayor valor tienen una probabilidad mayor de contribuir una o más veces a la generación siguiente. La forma más sencilla de implementar este operador consiste en crear una ruleta, donde cada individuo de la población de la generación actual ocupará un espacio proporcional al valor asignado por la función de coste. Entonces de manera aleatoria se dará vueltas a la ruleta hasta seleccionar a un candidato para la generación siguiente. De este modo, en probabilidad, aquellos individuos que ocupen mayor parte de la ruleta tendrán un mayor número de descendientes (Golberg, 2006). Además, el operador utilizado en este Proyecto Fin de Carrera obtiene los cinco mejores individuos de la generación anterior sin ser modificados, y a los que se les denomina élite.

- **Crossover (reproducción sexual)** después de la reproducción, se lleva a cabo un crossover simple que se realiza en dos pasos. Primero se coge a los miembros que se acaban de copiar de la generación anterior y se asocian aleatoriamente por parejas (descartamos la élite). En segundo lugar, cada pareja de cromosomas se cruza de la siguiente manera: se selecciona una posición k aleatoriamente dentro del cromosoma entre 1 y la longitud de la cadena menos uno ($[1, l-1]$)

y se intercambian las partes de ambos individuos (Golberg, 2006). Por ejemplo tenemos,

$$x_1 = 10|011$$

$$x_2 = 01|110$$

donde se ha elegido la posición dos a partir de la cual realizar el crossover. Entonces los nuevos individuos quedarían:

$$x_1' = 10110$$

$$x_2' = 01011$$

- **Mutación** consiste en cambiar el valor de un determinado gen aleatoriamente. En un algoritmo genético simple el ratio de mutación tiene una probabilidad muy baja de ocurrir (Golberg, 2006).

Con estos tres simples operadores podemos realizar un algoritmo genético. Los mecanismos de reproducción y crossover son sencillos ya que involucran un número aleatorio de generaciones, copia de cromosomas e intercambios parciales de los cromosomas. En estas operaciones es donde radica el poder de los algoritmos genéticos (Golberg, 2006). Para poder explicar la necesidad de la mutación primero tenemos que explicar lo que hacen los otros dos operadores. En una población de x individuos sobre un determinado alfabeto, cada uno está codificado de tal forma que representa una idea completa o una parte para llevar a cabo una tarea. Cada subcadena dentro de cada individuo representa una noción de lo que es relevante para la tarea; por lo que no sólo tenemos x ideas sino que tenemos multitud de nociones de la tarea a llevar a cabo. Entonces el algoritmo genético se encarga de por un lado copiar aquellas ideas de mejor calidad y por otro lado cruzar estas buenas ideas con otras también que son buenas por lo que en un principio conseguimos mejorar y tener mejores ideas. Pero no siempre cruzando dos ideas puede surgir una idea mejor. Entonces la mutación es necesaria ya que aunque la reproducción y el cruce buscan y combinan las mejores nociones, no muy a menudo el material genético pierde algo de utilidad, no siendo ya

tan bueno y es el operador mutación el que protege a las nociones de esa pérdida de utilidad (Golberg, 2006). Además, la mutación permite que aparezcan nuevas ideas no presentes en la población anterior.

Con estos tres operadores podemos construir el más simple de los algoritmos genéticos. De hecho son estos tres operadores los que se han utilizado en el algoritmo genético del presente PFC. Existen otros operadores avanzados que se explicarán brevemente (Golberg, 2006):

- **Diploicidad y dominación.** En la diploicidad se tienen una o más parejas de cromosomas cada uno de los cuales contienen información de la misma función. La dominación consiste en un operador que sirve para eliminar las redundancias entre dos cromosomas iguales, de tal forma que se concede preferencia a un valor determinado del gen frente al resto de valores que puede tomar ese gen.

- **Inversión.** Se eligen dos puntos dentro del cromosoma, se corta el cromosoma en dichos puntos y se intercambian las posiciones de los extremos de la sección cortada. Existen otras formas de inversión pero esta es la más sencilla, además también existen otros operadores de reordenación del cromosoma que no se explican en este PFC.

- **Partially matched crossover (PMX)** o reproducción sexual parcialmente emparejada. Consiste en establecer dos puntos aleatorios dentro del cromosoma para establecer la zona de cruce entre ambos cromosomas, después con las zonas que no intervienen el cruce se establece una permutación aleatoria de los genes. Por ejemplo, supongamos,

$$x_1 = 9\ 8\ 4 \mid 5\ 6\ 7 \mid 1\ 3\ 2\ 10$$

$$x_2 = 8\ 7\ 1 \mid 2\ 3\ 10 \mid 9\ 5\ 4\ 6.$$

  Entonces,

$$x_1' = 9\ 8\ 4 \mid 2\ 3\ 10 \mid 1\ 6\ 5\ 7$$

$$x_2' = 8\ 10\ 1 \mid 5\ 6\ 7 \mid 9\ 2\ 4\ 3.$$

- **Order crossover (OX)** o reproducción sexual ordenada. Consiste en establecer la zona de intercambio y después realizar el cambio. Ahora en lugar de permutar las otras zonas lo que quedan son huecos de los elementos a intercambiar y lo que se hace es reordenar el cromosoma. Esto se ve mejor con un ejemplo, supongamos,

$$x_1 = 9\ 8\ 4 \mid 5\ 6\ 7 \mid 1\ 3\ 2\ 10$$

$$x_2 = 8\ 7\ 1 \mid 2\ 3\ 10 \mid 9\ 5\ 4\ 6$$

cuando x_2 realiza el intercambio los valores que se van a cambiar fuera de la zona de intercambio dejan huecos:

$$x_2 = 8\ H\ 1 \mid 2\ 3\ 10 \mid 9\ H\ 4\ H$$

Ahora se reordena el cromosoma se ponen todos los huecos en la zona de intercambio y quedaría:

$$x_2 = 2\ 3\ 10 \mid H\ H\ H \mid 9\ 4\ 8\ 1$$

Finalmente obtenemos,

$$x_1' = 5\ 6\ 7 \mid 2\ 3\ 10 \mid 1\ 9\ 8\ 4$$

$$x_2' = 2\ 3\ 10 \mid 5\ 6\ 7 \mid 9\ 4\ 8\ 1.$$

- **Cycle crossover (CX)** o reproducción sexual cíclica. Es totalmente distinto a los anteriores ya que no se establecen zonas sino que se comienza por el principio del cromosoma y se fija el valor del gen que no cambiará. Posteriormente vemos cual es el valor del primer gen del segundo cromosoma, si el valor no coincide con ninguno del primero habríamos acabado y cambiaríamos todos los genes de uno y otro cromosoma, si no fijaríamos el gen del primer cromosoma cuyo valor es igual al primer gen del segundo y volveríamos a repetir la operación. Por ejemplo, supongamos,

$$x_1 = 9\ 8\ 1\ 7\ 4\ 5$$

$$x_2 = 1\ 2\ 4\ 5\ 9\ 7$$

primero tomamos el primer gen de x_1

$$x'_1 = 9\ -\ -\ -\ -\ -$$

el valor del primer gen de x_2 es 1 que se encuentra también en x'_1 por tanto también lo dejamos fijo,

$$x'_1 = 9\ -\ 1\ -\ -\ -$$

el valor en x_2 que corresponde a la posición que hemos fijado en x'_1 es cuatro por lo que también lo fijamos,

$$x'_1 = 9\ -\ 1\ -\ 4\ -$$

por último el valor del gen en x_2 es 9 que ya está fijado por lo tanto se ha completado el ciclo, ahora se intercambian los genes,

$$x'_1 = 9\ 2\ 1\ 5\ 4\ 7$$

$$x'_2 = 1\ 8\ 4\ 7\ 9\ 5$$

- **Segregación.** A la hora de la reproducción sexual se elige únicamente uno de los cromosomas progenitores que será el que continúe en la generación siguiente.
- **Translocación.** Consiste en una reproducción sexual dentro del mismo cromosoma, se intercambian genes de sitio.

Como se acaba de ver los algoritmos genéticos se pueden complicar todo lo que queramos, pero para lo que atañe a este Proyecto Fin de Carrera será suficiente con el uso de los tres primeros operadores. Con esto ya tenemos una visión amplia sobre los algoritmos que se han utilizado en la elaboración de este PFC. En el Capítulo 3 se describe la herramienta que se ha utilizado para su desarrollo.

Capítulo 3

Simulador GedelosSim

En el presente Capítulo describiremos la herramienta software que se ha desarrollado para llevar a cabo el presente Proyecto Fin de Carrera. Primero comenzaremos describiendo la estructura general del simulador (ver Sección 3.1), a continuación se describirá cada uno de las elementos de los que se compone (ver Sección 3.2) y finalmente se expondrán los parámetros necesarios para la ejecución del simulador (ver Sección 3.3).

3.1. Estructura

El simulador se ha realizado en el lenguaje de programación C++, que es un lenguaje orientado a objetos (Aguilar, 2002). El programa se ha hecho totalmente modular utilizando clases distintas para cada una de las diferentes partes que lo integran, permitiendo de este modo una escalabilidad total y generar diferentes modos de operación sin más que añadir nuevas clases. Para su compilación se ha utilizado las herramientas proporcionadas por "autotools"[1]. De esta forma y gracias a las herramientas facilitadas por "autotools" se crean librerías de los diferentes módulos siendo accesibles por el programa principal.

[1]Autotools es un conjunto de herramientas producido por el proyecto GNU. Estas herramientas están diseñadas para ayudar a crear paquetes de código fuente portable a varios sistemas Unix.

El simulador tiene dos modos de operación:

- **Modo virtual** en el que el simulador tiene codificado cada uno de los elementos de la casa objeto de estudio (Magicbox).
- **Modo real** en el que se accede de forma directa a cada uno de los componentes reales de Magicbox sin necesidad de tener ningún modelo de los elementos que intervengan durante la simulación.

Sólo se describirá el modo virtual debido a que durante el desarrollo de este Proyecto Fin de Carrera sólo se ha empleado dicho modo para realizar las investigaciones oportunas. Antes de definir la estructura virtual del simulador es necesario conocer la estructura de los elementos de la casa (ver Figura 3.1):

- Seis generadores fotovoltaicos en diferentes planos de captación.
- Conjunto de electrodomésticos de bajo consumo a los que se puede acceder de forma remota.
- Batería, elemento acumulador de energía para suplir la falta de ésta cuando se demanda en el interior de la casa.
- Inversor de batería, es el encargado de distribuir los flujos de energía provenientes de la red y del sistema fotovoltaico hacia las cargas o hacia la batería cuando ésta no está cargada. También dirige la energía almacenada en la batería, cuando sea necesario, a las cargas.
- Conjunto de contadores de nueva generación capaces de acceder a ellos remotamente para saber la energía que se está consumiendo al instante.

Después de hacer un repaso a la estructura de la casa, nuestro simulador debe ser capaz de integrar todos estos elementos de tal forma que sean lo más reales posibles. La estructura del simulador consta de varias clases que contienen todo lo necesario para poder llevar a cabo la simulación (entorno de trabajo, sistema de control, parámetros de la simulación, etc.) y consta de (ver Figura 3.2):

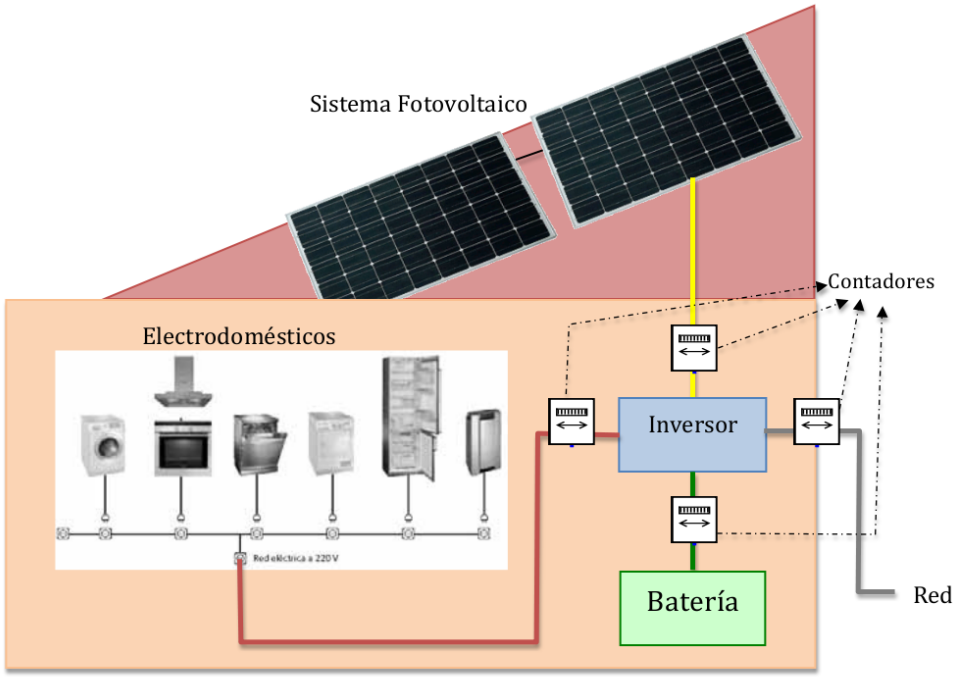


Figura 3.1: Estructura Magicbox.

- El experimento (*experiment.cpp*) donde se define el entorno de la simulación mediante un fichero que contiene los parámetros de simulación como: duración del experimento, tareas a realizar, número de casas, capacidad de batería, etc.

- El simulador (*simulator.cpp*) que crea los elementos necesarios para la simulación y es el encargado de ejecutar en cada tiempo de simulación lo que consiste el experimento.

- La casa (*house.cpp*) formada por todos aquellos elementos de la casa real: electrodomésticos, usuario, sensores, etc.

- Los electrodomésticos (*appliances.cpp*) son modelos de los existentes realmente en Magicbox. Cada uno contiene las diferentes opciones de sus programas así como las duraciones y consumos de cada uno de ellos. Cada electrodoméstico forma parte de una base de datos donde se encuentran todos esos parámetros.

- Los objetos de la casa (*houseobject.cpp*) son aquellos elementos de la casa que sirven para completar el modelo y son: la batería, que posee la misma capacidad que la batería real de Magicbox, el inversor de batería, que realiza el balance de potencia y se encarga de distribuir la energía, y las líneas eléctricas, donde se queda registrada la potencia que se está consumiendo o generando dependiendo de que línea sea. Existen cuatro líneas eléctricas: la de generación fotovoltaica, la de cargas, la de batería y la de red.

- Los perfiles fotovoltaicos que son ficheros de texto de perfiles de generación predichos para Magicbox (Porro, 2010), de esta forma se tienen caracterizados los generadores fotovoltaicos.

- El usuario (*user.cpp*) que es el encargado de establecer las tareas (*task.cpp*) de los electrodomésticos así como de fijar el programa que se quiere ejecutar.

- Los sensores (*sensor.cpp*) que es el conjunto de contadores que toma medidas en cada una de las entradas al inversor.

- Los controladores (*controller.cpp*), es el bloque fundamental de este PFC, y son los encargados de establecer cuándo hay que poner los electrodomésticos en función de unas determinadas condiciones y dependiendo del perfil de usuario.

- Los actuadores (*actuator.cpp*) son los encargados de poner en marcha lo que ha establecido el controlador.

Como podemos observar en la Figura 3.2 el sistema de control está formado por: i) los sensores, que son las entradas a dicho sistema, ii) el controlador, encargado de procesar esas entradas y generar unas salidas acordes a lo que se quiera realizar, y iii) el actuador, que pondrá en marcha lo dicho por controlador.

Ésta es en grandes rasgos la estructura genérica del simulador aunque en este Capítulo no están descritas las herramientas usadas para la implementación de las redes neuronales y algoritmos genéticos, que describiremos en la Sección 4.2. Este

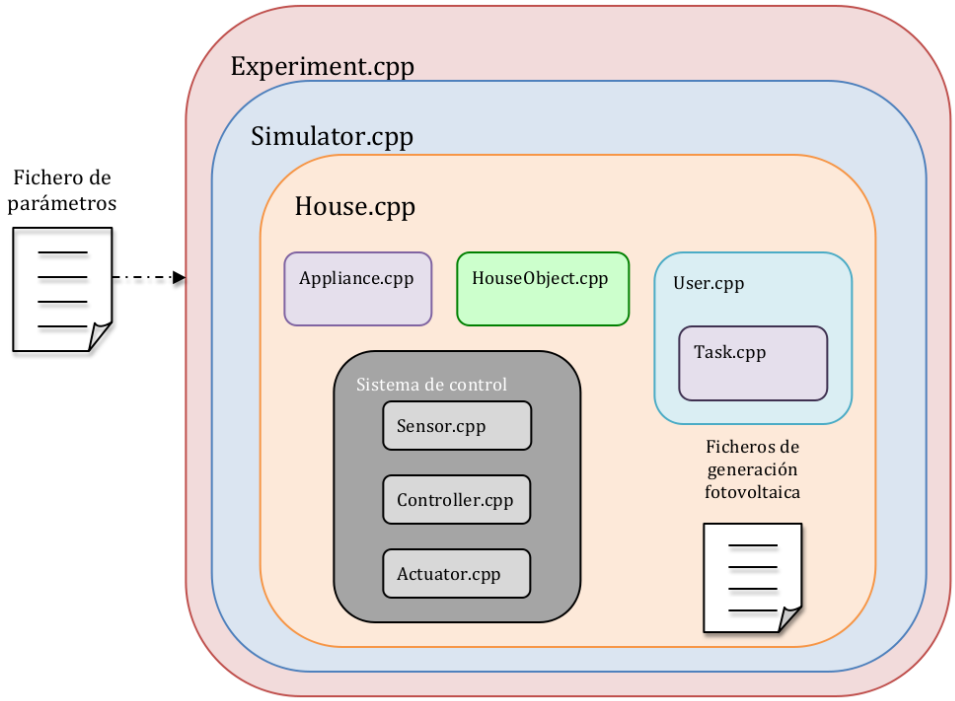


Figura 3.2: Estructura simulador.

Capítulo trata de explicar a grandes rasgos cómo está diseñada la herramienta que se ha utilizado.

3.2. Elementos

En esta parte del Capítulo se describirá cada uno de los elementos que integran el simulador para comprender mejor la naturaleza de este software. Describiremos las distintas librerías del software así como los elementos de la estructura descritos en la Sección 3.1.

3.2.1. Ficheros de entrada

Dentro del simulador existen dos tipos de ficheros para ejecutar la simulación[2] y son: fichero de parámetros (paramFiles) y perfiles fotovoltaicos (simProfiles).

- **Fichero de parámetros.** Es en el que se encuentran definidas todas las opciones necesarias para llevar a cabo la simulación. El fichero contiene las siguientes opciones:
  - *Semilla aleatoria.* Contiene información relativa al algoritmo genético. Puede tomar cualquier valor numérico.
  - *Duración del experimento.* Establece la duración total de ejecución de la simulación. La duración es en minutos.
  - *Duración del tiempo de muestreo.* Podemos ejecutar la simulación con diferentes unidades de tiempo (minutos), cada minuto, cada cinco minutos, etc.
  - *Número de casas.* Establece cuantas casas intervendrán en el experimento. En este PFC se ha trabajado con una vivienda.
  - *Tipo de experimento.* Establece si la ejecución es en real o en simulado. Valor 0 para simulado y 1 para real.
  - *Fichero de generación.* Se establece la ruta donde se encuentra el fichero de generación fotovoltaica de un perfil de Magicbox. Este fichero guarda los valores de potencia generada por el generador fotovoltaico de la casa.
  - *Carga y estado de batería.* Establece cual es la carga de la batería en kWh (la máxima capacidad es de 65 kWh) y los estados de la batería son únicamente dos, cargada (valor 0) y descargada (valor 1).
  - *Máxima potencia instantánea.* El usuario decide cuánta es la máxima potencia que se puede consumir al instante, el valor está en W, para que no haya restricción el valor debe se cero.

[2]Existen también otros dos tipos de ficheros de entrada pero son para establecer los parámetros de las redes neuronales y del algoritmo genético. Estos ficheros se explicarán más adelante.

- *Máxima energía al día.* El usuario establece otra limitación pero esta vez sobre la cantidad de energía consumida en un día, se mide en Wh y para que no exista restricción el valor debe ser cero.
- *Cantidad y tipo de electrodomésticos.* El usuario indicará el número de electrodomésticos que va a activar y después los electrodomésticos que son uno por uno. Para indicar el electrodoméstico se utiliza un código en el que a la lavadora se le ha asignado el número 1, a la secadora el 2, al lavavajillas el 3, al horno el 4, al frigorífico el 5 y a la campana el 6.
- *Tarea.* El usuario una vez que ha elegido los electrodomésticos a usar establecerá las tareas. Primero pondrá el número de tareas que se van a realizar, después comenzando por el primer electrodoméstico indicará cual es (lavadora, secadora, etc), los parámetros del programa del electrodoméstico (por ejemplo en el caso de la lavadora serán la temperatura y las revoluciones), después indicará el día en que empezará a realizarse la tarea, el minuto del día en que empezará, el día en que finalizará y el minuto del día que quiere que se termine. Por último se indicará si la tarea es diferible, es decir, la tarea puede moverse en el tiempo para conseguir maximizar el autoconsumo. Si es diferible valdrá 1 y sino valdrá 0.

  A continuación tenemos un ejemplo del fichero de entrada necesario para la simulación con un único electrodoméstico y una tarea para ese electrodoméstico.

RANDOM SEED	=	*123456*
EXPERIMENT DURATION (Minutes)	=	*141*
TIME STEP DURATION (Minutes)	=	*1*
NUMBER OF HOUSES	=	*1*
EXPERIMENT TYPE (0 SIMULATED, 1 REAL)	=	*0*
PROFILE FILE	=	*pvGen01*
INITIAL BATTERY'S CAPACITY (kWh)	=	*64.5*
INITIAL BATTERY'S STATE (0 CHARGE, 1 DISCHARGE)	=	*0*
MAX INSTANT POWER (Watts, 0 is infinity)	=	*2500*
MAX ENERGY CONSUMPTION PER DAY (Wh, 0 is inf)	=	*4000*
NUMBER OF APPLIANCES	=	*1*
APPLIANCE 1 (WM 1, DR 2, DW 3, OV 4, FR 5, HO 6)	=	*1*
NUMBER OF TASKS	=	*1*
APPLIANCE (WM 1, DR 2, DW 3, OV 4, FR 5, HO 6)	=	*1*
PARAM PROGRAM 1	=	*90*
PARAM PROGRAM 2	=	*1200*
INITIAL DAY TIME	=	*1*
INITIAL MINUTE TIME	=	*500*
END DAY TIME	=	*1*
END MINUTE TIME	=	*1000*
DEFERRABLE (1 YES, 0 NO)	=	*1*

Tabla 3.1: Ejemplo de fichero de parámetros.

- **Perfiles fotovoltaicos** estos ficheros contienen la potencia predicha para el día siguiente del generador fotovoltaico de Magicbox. Estos ficheros son generados por otra aplicación desarrollada para el proyecto GeDELOS-FV (Castillo, 2009; Porro, 2010) y se generan a través de las predicciones meteorológicas de la Agencia Estatal de Meteorología. El fichero consta de 24 valores, uno por cada hora del día, y la potencia se encuentra expresada en vatios. Ejemplo de fichero:

  Hora 1: 0

  Hora 2: 0

  Hora 3: 0

Hora 4: 0

Hora 5: 0

Hora 6: 0

Hora 7: 0

Hora 8: 365.57

Hora 9: 899.43

Hora 10: 1517.81

Hora 11: 2003.52

Hora 12: 2331.48

Hora 13: 2513.86

Hora 14: 2543.23

Hora 15: 2419.73

Hora 16: 2157.79

Hora 17: 1742.57

Hora 18: 1186.32

Hora 19: 550.93

Hora 20: 154.62

Hora 21: 0

Hora 22: 0

Hora 23: 0

Hora 24: 0

El fichero corresponde a la curva de generación prevista para el día siguiente que se observa en la Figura 3.3.

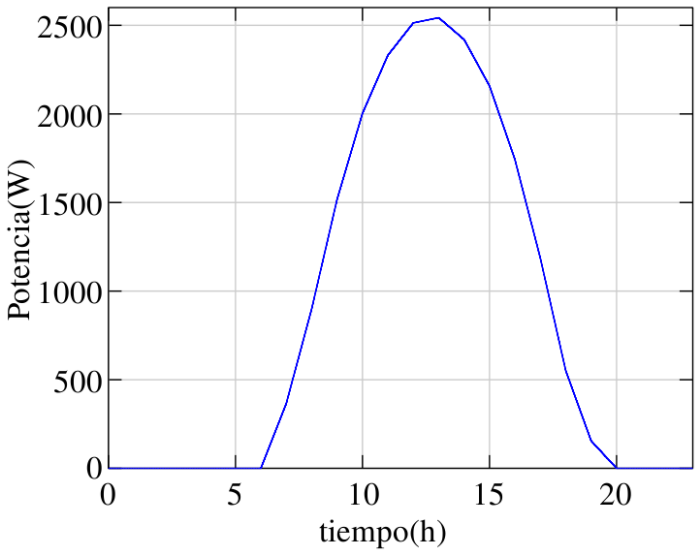


Figura 3.3: Perfil Fotovoltaico de potencia prevista.

3.2.2. Electrodomésticos

La librería de los electrodomésticos está compuesta por: lavadora, lavavajillas, secadora, horno, campana y frigorífico (ver Figura 3.4). Cada uno de ellos tiene su propia base de datos con la información necesaria sobre el consumo, duración y fases de programa. Cada electrodoméstico cuenta con distintos programas de funcionamiento pero todos utilizan las mismas funciones para obtener los datos y para programarlos. Además el simulador cuando opera en modo real es capaz de comunicarse, a través de una pasarela, con los electrodomésticos de Magicbox ya que cuentan con el sistema domótico "Serve@Home"de Siemens (Castillo, 2009; Castillo et al., 2009; Caamaño-Martín et al., 2009a).

Para obtener información de los electrodomésticos existen tres bloques de funciones que se pueden utilizar:

- **Funciones de lectura de la base de datos**, este bloque está compuesto por cinco funciones. Todas las funciones de este bloque necesitan que se pase como parámetro el programa con el que va a funcionar el electrodoméstico mediante

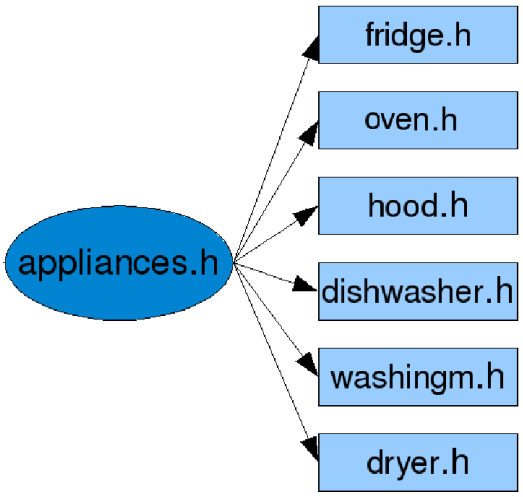


Figura 3.4: Estructura de la librería de los electrodomésticos.

una estructura de datos en la que se almacenan como máximo los dos parámetros necesarios para seleccionar un programa. Las funciones son:

- *ReadTimeInfo (ProgramStruct sProgram, TimeInfo *sOutput)*, para leer el tiempo de duración de ese programa en concreto. Devuelve la información en una estructura de datos que indica la duración, el número de fases y la duración de cada una de las fases.
- *ReadEnergyInfo(ProgramStruct sProgram, EnergyInfo* sOutput)*, para leer la energía que consume el electrodoméstico con el programa seleccionado. Devuelve la información en una estructura de datos que indica la energía total y la energía de cada una de las fases.
- *ReadPower (ProgramStruct sProgram, int nMin)*, para leer la potencia consumida en un determinado instante de tiempo. Devuelve un número de tipo *float*.
- *ReadEnergy (ProgramStruct sProgram, int nMin)*, para leer la energía consumida en un determinado instante de tiempo. Devuelve un número de tipo *float*.

  - *ReadPhaseofWT (ProgramStruct sProgram, int nInputWT)*, para leer en qué fase del programa seleccionado se encuentra el electrodoméstico en un tiempo determinado.

- **Funciones de lectura de estado de tarea**, el siguiente bloque de funciones se compone de:

  - *GetStatus (void)*, para saber en qué estado se encuentra la tarea. Devuelve un entero.
  - *GetProgram (ProgramStruct *sProgram)*, para saber qué programa está ejecutando el electrodoméstico. Devuelve la información en la estructura que le pasamos como parámetro.
  - *GetStartTime (void)*, para saber el tiempo de inicio de la tarea que se está ejecutando. Devuelve un entero.
  - *GetWorkedTime (void)*, para saber cuánto tiempo lleva ejecutado la tarea.

- **Funciones de ejecución**, las funciones que forman este bloque son:

  - *JobStart (ProgramStruct sProgram)*, para iniciar el electrodoméstico con el programa que pasamos por parámetro.
  - *JobPause (void)*, para pausar la tarea en progreso.
  - *JobStop (void)*, para parar la tarea en progreso.

Con todas estas funciones somos capaces de simular el comportamiento de los electrodomésticos además de poder extraer la información necesaria para realizar los experimentos con ellos.

3.2.3. Objetos de la casa

Dentro de los objetos de la casa se encuentran todos aquellos elementos que pertenecen a ésta. Esta librería está formada por:

- **Línea eléctrica (electricline.cpp).** Este elemento sirve para depositar la potencia, tanto consumida por las cargas como generada, de tal forma que los contadores leerán de ella el dato de potencia. Así pues existen cuatro líneas eléctricas diferenciadas: la de las cargas, la del generador fotovoltaico, la de la batería y la de la red. La necesidad de crear este objeto es debido a que el modelo del simulador trata de reproducir el sistema existente en Magicbox, de este modo las líneas eléctricas tratan de imitar el cableado eléctrico de la vivienda.

- **Batería (battery.cpp).** Es otro elemento cuya función es almacenar energía eléctrica para suplir la demanda cuando no haya electricidad disponible proveniente del generador ni de la red. Con ella podemos incorporar un grado más de libertad en la ecuación del balance de potencia y en la maximización del autoconsumo. Este componente está formado por una capacidad inicial en kWh y un determinado estado de carga o descarga. Además para hacerlo más real se le han puesto dos limitaciones: una de carga máxima que consiste en no superar un determinado umbral y otro umbral de descarga.

- **Inversor de batería (batinverter.cpp).** Este aparato realiza el balance de potencia de las diferentes líneas eléctricas su funcionamiento consiste en una serie de reglas para saber por dónde distribuir la energía en el interior de la casa. En el siguiente pseudocódigo se explica su funcionamiento:

```
/*Primero se decide si las cargas necesitan mas potencia*/
if((PotConsumida - PotGenerada) > 0)

 /*Si lo necesitan primero tomara potencia de la bateria*/
 if(Estado_Bateria) //Si esta en descarga

   /* Mira si la potencia que las cargas necesitan es mayor que la
    * que la bateria puede entregar
    */
   //Necesitan mas que lo que da la bateria
   if ((PotConsumida - PotGenerada) > MaxDescarga)
```

```
      //Toma la energia que falta de la red.
      //La potencia de cargas esta cubierta por: la generada,
      //la bateria y la red.
      //Se actualiza la carga de bateria.

    else //Con la bateria es suficiente para cubrir el consumo
      //La bateria entrega la maxima energia de descarga.
      //El excedente de potencia se manda a la red.
                  //Se actualiza la capacidad de la bateria.

  /*Bateria en carga*/
  else

    /*La bateria no esta totalmente cargada*/
    if(Capacidad < MaxCapacidad)
      //Se utiliza la red para cubrir la demanda de las cargas
      //y cargar la bateria.

    /*Bateria cargada*/
    else
      //Se cambia el estado de bateria a cargada.

/*La potencia generada cubre la demanda de las cargas*/
else

  /*Comprueba si la bateria esta en descarga*/
  if(Estado_Bateria)
    //El excedente de potencia generada se manda a la red.

  /*Bateria en carga*/
  else

    /*Bateria no esta totalmente cargada*/
    if(Capacidad < MaxCapacidad)
      //El excedente de potencia generada carga la bateria.

    /*Bateria cargada cambia a descarga*/
    else
      //El excedente de potencia generada pasa de nuevo a la red.
```

3.2.4. Sensores

Los sensores del simulador son los contadores que se encargan de leer la energía que circula por cada una de las líneas eléctricas. Los sensores sirven como entradas al sistema de control ya que son los encargados de recoger la información necesaria

para saber que el sistema de control funciona correctamente. La librería de sensores sólo cuenta con los contadores pero podrían incorporarse sensores de temperatura o humedad para controlar, por ejemplo, el aire acondicionado.

El contador tiene los dos modos, real y virtual, implementados. En el *modo virtual* se encarga de leer las líneas eléctricas e imprimir por pantalla la potencia leída. En el *modo real* se ha utilizado el contador monofásico Domotax de la empresa Orbis. Se ha creado un software específico para comunicarse con él. La comunicación consiste en el envío de tramas, de la forma:

- Para transmisión: *comienzoTX — longitudTrama — peticiónDato — númeroDeAbonado — comprobación — finalTx.*
- Para recepción: *comienzoTX — longitudTrama — peticiónDato — númeroDeAbonado — datos — comprobación — finalTx.*

El protocolo de comunicación es el siguiente: primero mandamos una trama para abrir la comunicación con el contador, si no hay ningún problema el contador enviará otra trama de aceptación de la comunicación, y una vez abierta la comunicación ya podemos preguntar por los datos necesarios al contador.

Toda la información obtenida, tanto en real como en virtual, es almacenada en un histórico. Hay dos históricos uno para potencia y otro para energía. En el modo virtual, se crea un histórico por cada línea eléctrica, es decir, que tenemos un histórico para las cargas, otro para el generador fotovoltaico, otro para la batería y por último, otro para la red. En el modo real se almacenan los mismos datos obtenidos directamente de los contadores y no de la línea. También se almacena en un fichero la última potencia instantánea leída.

3.2.5. Controladores

Dentro de esta librería se encuentran diferentes controladores desarrollados en el proyecto GeDelos-FV (Castillo, 2009; Castillo et al., 2009; Caamaño-Martín et al., 2009a). En lo que respecta a este PFC nos centramos únicamente en el controlador

neuronal, que desarrollaremos en detalle en el Capítulo 4. También dentro de esta librería se encuentran las redes neuronales que explicaremos también en el Capítulo 4 al igual que el algoritmo genético empleado.

3.3. Ejecución

Para poder llevar a cabo una simulación es necesario:

- Establecer un experimento dentro del cual se crea la simulación que se quiere realizar. Se le pasa un fichero de parámetros para establecer las condiciones del experimento, cuando se ejecute el programa (ver Sección 3.2.1).
- El simulador se encargará de establecer los elementos necesarios obtenidos mediante la información que le han pasado por el fichero.
- Se creará la casa, junto con todos los elementos que se han descrito anteriormente.
- Comienza la ejecución de la simulación en la que cada intervalo de tiempo simulado se realizará lo que el controlador, con el que se quiere experimentar (ver Capítulo 4), establezca.
- Finalmente se termina la ejecución y se obtienen los resultados.

Para poder invocar el simulador deberemos llamar al simulador y después seleccionar un experimento y pasarle el fichero de parámetros para llevar a cabo la simulación. El comando para poder ejecutar el programa es el siguiente:

>./gedelosSim -E 2 -p paramFiles/meterParam3.txt .

El identificador de experimento es -E y para indicar el fichero de parámetros es -p; existen también otros identificadores que desarrollaremos posteriormente. Un ejemplo de ejecución es el siguiente:

> ./gedelosSim -E 2 -p paramFiles/meterParam3.txt

TEST EXPERIMENT: 2 CHOOSEN.

filename: paramFiles/meterParam3.txt CHOOSEN.

TEST METER EXP

RANDOM SEED: 123456

EXPERIMENT DURATION: 999

TIME STEP DURATION: 1.000000

HOUSE NUMBER: 1

TYPE OF SIMULATION: 0

PROFILE FILE: simProfiles/pvGeneration1

INITIAL BATTERY CAPACITY: 65.000000

INITIAL BATTERY STATE: 1

MAX INSTANT POWER: 2500.000000

MAX DAILY ENERGY: 4000.000000

NUMBER OF APPLIANCES: 3

WASHING MACHINE TO BE INSTALLED

WASHING MACHINE TO BE INSTALLED

DISHWASHER TO BE INSTALLED

NUMBER OF TASKS: 3

Creating Washing Machine

Creating Dishwasher

Creating Battery

Creating Loads Electric Line

Creating PV Electric Line

Creating Battery Electric Line

Creating Net Electric Line

Creating Battery inverter

LOADS ELECTRIC LINE: 850.0 W 0.0 W

PV ELECTRIC LINE: 400.0 W 0.0 W

BATTERY ELECTRIC LINE: 2000.0 W 0.0 W

NET ELECTRIC LINE: -1550.0 W 0.0 W

Este ejemplo consiste en leer la potencia de cada una de las líneas eléctricas. Como vemos, el simulador se ha encargado de imprimir por pantalla todos los elementos que componen la simulación. Una vez impresa comienza a ejecutarse y vemos las medidas que se han obtenido.

Esta es la herramienta que se ha utilizado durante el desarrollo de este Proyecto Fin de Carrera. El simulador trata de ser una copia virtualizada de la realidad, por lo que a partir de ahora sólo nos centraremos en el controlador neuronal y su implementación.

Capítulo 4

Controlador Neuronal: Implementación y Resultados

En los Capítulos 2 y 3 se han presentado los algoritmos y herramientas que se han utilizado en este PFC. En este Capítulo desarrollamos la implementación del controlador dentro del simulador así como los resultados obtenidos. Como se ha repetido varias veces, el objetivo del controlador no es otro que buscar la máxima eficiencia energética posible.

En la Sección 4.1 se introducirá el concepto de controlador neuronal. En la Sección 4.2 se explicará el software específico que se ha desarrollado para implementar los algoritmos inteligentes descritos en el Capítulo 2. En las Secciones 4.3 y 4.4 explicaremos los experimentos desarrollados para el planificador y el coordinador respectivamente y los resultados obtenidos de las simulaciones. Y por último explicaremos en la Sección 4.5 el funcionamiento del sistema completo.

4.1. Introducción

La introducción de un controlador energético en un entorno residencial tiene un objetivo claro: realizar una gestión de la demanda eléctrica. En nuestro caso al contar con diferentes fuentes energéticas (la red eléctrica, la batería y energía solar fotovoltaica) el controlador primará el uso de unas sobre otras. En concreto trataremos de maximizar con respecto al autoconsumo, es decir, al uso de la energía generada,

frente aquella proveniente de la red eléctrica.

El usuario ofrece al sistema las diferentes tareas que desea ejecutar en ese día (los electrodomésticos que quiere activar). El sistema de control planifica el momento de trabajo de cada uno de los electrodomésticos teniendo en cuenta las restricciones temporales del usuario, los perfiles de generación fotovoltaica y la duración de cada uno de los electrodomésticos. La arquitectura del sistema de control (ver Figura 4.1) está formada por:

- **Capa de planificación.** Se encarga de establecer el tiempo en el que se debe ejecutar la tarea que quiere llevar a cabo el usuario, teniendo en cuenta las restricciones de tiempo del usuario (t_{oi}, t_{fi}) y el perfil de generación fotovoltaica (P_{fv0-23}).

- **Capa de coordinación.** Reparte en el dominio temporal todas las tareas que se quieren llevar a cabo, de tal forma que no se solapen en el tiempo. Para ello necesita las salidas de planificación (t_i) y la duración de cada una de las tareas (Δt_{ai}).

- **Capa de ejecución.** Lleva a cabo la comunicación entre el coordinador y las cargas. Pone en marcha cada tarea cuando el coordinador lo establece. Para ello necesita la tupla de tiempos coordinados (t'_i).

En este PFC, se han diseñado el planificador y el coordinador, ya que la capa de ejecución consiste únicamente en iniciar la tarea cuando lo indique el coordinador (Castillo, 2009). Para ello, dentro de la simulación, es el propio experimento quien se encarga de iniciar la tarea cuando el tiempo simulado corresponde con el tiempo indicado por el coordinador. La forma de implementar el planificador y el coordinador, como ya se ha indicado anteriormente, ha sido mediante el uso de redes neuronales sintonizadas con algoritmos genéticos. Tanto la capa de planificación como la capa de coordinación son distribuidas, debido a la propia naturaleza distribuida de las redes neuronales (Haykin, 2009). Este controlador trata de poner en entredicho la visión

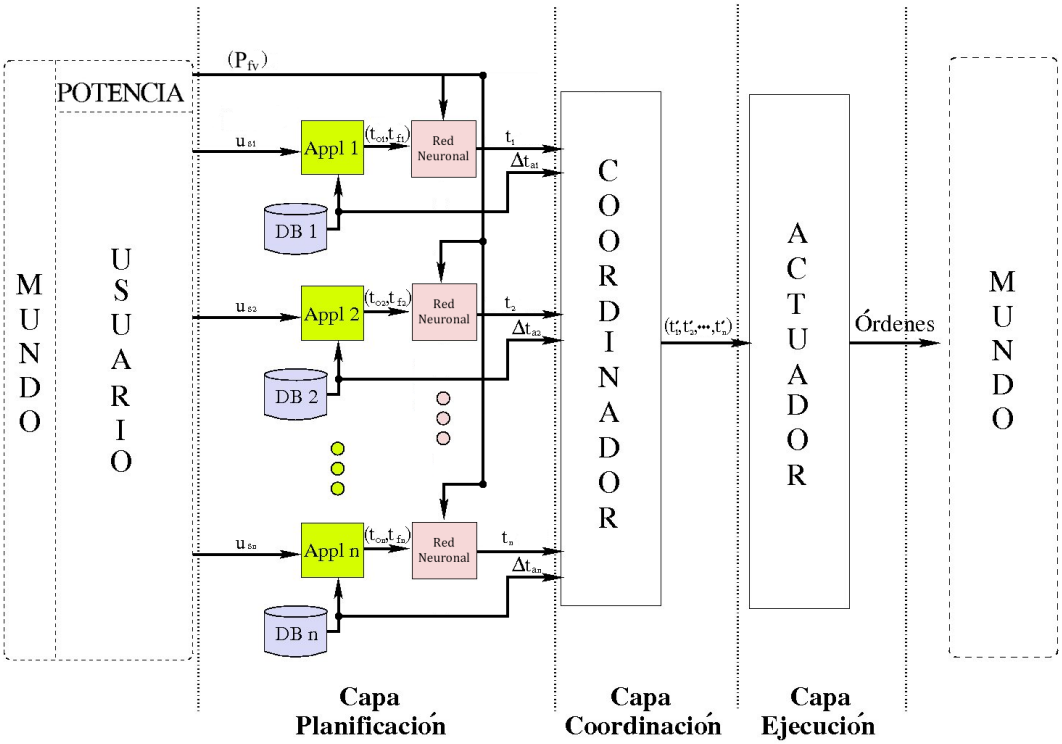


Figura 4.1: Arquitectura del controlador.

cognitivista del control clásico[1] frente a la visión conexionista que trataremos de aplicar en este controlador. A continuación se describirá cómo se han implementado cada una de las capas en el simulador.

4.2. Software

Para implementar el controlador ha sido necesario crear un software específico que nos permitiera implementar redes neuronales, en concreto un perceptrón multicapa, y evolucionar estas redes mediante un algoritmo genético. Para ello el software desarrollado se describe a continuación:

[1]Se necesita conocer todo perfectamente y tener un modelo de lo que se va a controlar para efectuar un control correcto de la planta.

- **Redes neuronales (perceptronMl.cpp)** Se encuentran dentro de la librería de controladores y tratar de realizar las operaciones del modelo de neurona estándar descrito en el Capítulo 2. La clase *perceptronMl.cpp* crea un perceptrón multicapa con: una capa de entrada, una única capa oculta y una capa de salida. Al crear este objeto se establecen las neuronas de cada capa así como los límites superiores e inferiores de los valores de: ganancias de entrada, sesgos (*bias*) y pesos para la capa oculta y de salida.

  El procesamiento de información de la red se divide en tres fases una por capa:

  - **Capa de entrada:** multiplica cada una de las entradas ($x_{in,i}$) por su ganancia de entrada ($g_{in,i}$): $y_{in,i} = x_{in,i} \cdot g_{in,i}$. Luego normaliza el resultado entre los límites superiores e inferiores de las ganancias de entrada impuestos por el fichero de parámetros de entrada. El código es el siguiente:

```
/* Compute Input Readings */
for ( int i = 0 ; i < m_unNumberOfInputs ; i++ )
{
  pf_inputs[i] = pf_inputs[i] * pf_input_gains[i];
  //Normaliza los valores de entrada.
  pf_inputs[i] = Normalize(pf_inputs[i],m_fInputGainUpperBounds,
                                        m_fInputGainLowerBounds );
}
```

  - **Capa oculta:** Cada neurona de la capa oculta realiza primero la suma de los pesos (normalizados entre los límites superior e inferior impuestos) por cada entrada, más el sesgo (θ_i, cuyo valor también ha sido normalizado previamente), $x_{hi,i} = \sum_j y_{in,j} \cdot w_{ij} + \theta_i$. Posteriormente, la salida consiste en aplicar una función de activación sigmoidea, $y_{hi,i} = \frac{1}{1+e^{-x_{hi,i}}}$. El código que realiza esta operación es:

```
/* Compute hidden neuron */
for ( int i = 0 ; i < m_unNumberOfHidden ; i++ )
{
  //Normaliza el parametro de ajuste
  m_pfHidden[i] = Normalize(pf_hidden_bias[i], m_fBiasUpperBounds,
                                               m_fBiasLowerBounds);

  for ( int j = 0 ; j < m_unNumberOfInputs ; j++)
```

```
  {
    //Realiza la suma de pesos normalizados por las entradas
    //de la neurona
    m_pfHidden[i]+= pf_inputs[j] *
          Normalize(pf_hidden_weights[(i * m_unNumberOfInputs)+j],
          m_fWeightsUpperBounds, m_fWeightsLowerBounds);
  }
  //Aplica una funcion de activacion sigmoidea
  m_pfHidden[i] = 1.0 / ( 1.0 + exp ( - m_pfHidden[i] ) );
}
```

- **Capa de salida:** la operación que se realiza en la capa de salida es igual a la realizada en la capa oculta: primero realiza la suma de los pesos normalizados por cada entrada más el sesgo normalizado (θ_i), $x_{out,i} = \sum_j y_{hi,j} \cdot w_{ij} + \theta_i$, y la salida consiste en aplicar una función de activación sigmoidea, $y_{out,i} = \frac{1}{1+e^{-x_{out,i}}}$. El código de la operación es:

```
/* Compute Output neuron */
for ( int i = 0 ; i < m_unNumberOfOutputs ; i++ )
{
  //Normaliza el parametro de ajuste
  m_pfOutputs[i] = Normalize(pf_output_bias[i],
                    m_fBiasUpperBounds, m_fBiasLowerBounds);

  for ( int j = 0 ; j < m_unNumberOfHidden ; j++ )
  {
    //Realiza la suma de pesos normalizados por las entradas
    //de la neurona
    m_pfOutputs[i]+= m_pfHidden[j] *
        Normalize(pf_output_weights[(i*m_unNumberOfHidden)+j],
        m_fWeightsUpperBounds, m_fWeightsLowerBounds);
  }
  //Aplica una funcion de activacion sigmoidea
  m_pfOutputs[i] = 1.0 / ( 1.0 + exp ( - m_pfOutputs[i] ) );
}
```

El número de neuronas, así como los límites superiores e inferiores de normalización de cada una de las capas son establecidos a través de un fichero. Para ello a la hora de ejecutar un experimento se debe introducir por línea de comando con la opción “-n” la ruta donde se encuentra el fichero.

- **Algoritmo genético,** el software que crea el algoritmo genético, está compuesto de los operadores de reproducción (copia de los mejores individuos), crossover y mutación (ver Capítulo 2). Está formado por varias clases y librerías del simulador. En concreto está formado por:

  - **Individuos (individual.cpp),** en esta clase se crea un individuo que tiene un cromosoma determinado. Las funciones principales que conforman los individuos son:

    - *CIndividual(double* pf_chromosome, unsigned int un_chromosome _length),* se crea un individuo con un cromosoma de la longitud especificada.
    - *CIndividual(const CIndividual& original),* se copia un individuo entero que se pasa por parámetro.
    - *double* GetChromosome(void),* para acceder al cromosoma del individuo.
    - *unsigned int GetChromosomeLength(void),* devuelve la longitud del cromosoma.
    - *double GetFitness(void),* para acceder al coste otorgado al individuo de valorar su cromosoma por la función de coste o ajuste.

  - **Población (population.cpp),** se encarga de crear poblaciones de individuos así como la aplicación de los tres operadores genéticos para la creación de nuevas generaciones. Esta clase está formada por las siguientes funciones:

    - *CPopulation(void),* crea un vector de individuos.
    - *CreateRandomPopulation(unsigned int un_number_of_chromosomes),* crea una población de individuos aleatoria. Se ha credo una variante de este método que consiste en crear una serie de individuos a partir de los ya dados y mutarlos de forma aleatoria.

- *unsigned int GetNumberOfCurrentGeneration(void),* devuelve la generación actual en la que se encuentra. Empieza en la generación cero y continua hacia adelante.
- *unsigned int GetNumberOfChromosomes(void),* devuelve el número de cromosomas.
- *unsigned int GetNumberOfChromosomesEvaluated(void),* devuelve el número de cromosomas evaluados hasta la generación actual.
- *bool GetNextChromosomeToEvaluate(void** pc_individual, double** pf_chromosome, unsigned int* un_chromosome_length),* esta función indica si queda algún individuo de la generación por evaluar. Hasta que esta función no devuelva falso no se puede crear la siguiente generación.
- *SetEvaluationResult(void* pc_individual, double f_fitness),* establece el coste para el individuo de la generación que se acaba de evaluar.
- *CreateNextGeneration(void),* esta función se encarga de crear una nueva población, para ello utiliza los tres operadores genéticos principales, reproducción (copia de los mejores individuos), crossover y mutación, desarrollados en el Capítulo 2.
- *GetGenerationStats(double* pf_worst_fitness, double* pf_average_fitness, double* pf_best_fitness),* copia en los argumentos las estadísticas de la generación.
- *GetBestChromosome(double** ppf_chromosome, unsigned int* pun_chromosome_length),* obtiene el mejor cromosoma de la generación.
- *Parámetros,* existen otras funciones para establecer y obtener los parámetros del algoritmo como la semilla aleatoria, el ratio de mutación, el ratio de crossover, el punto donde realizar el crossover, el número de élites a copiar por generación, etc. El simulador nos permite crear distintos tipos de población en los que podríamos usar diferentes operadores genéticos.

- **Función de coste o ajuste (fitnessfunction.cpp),** la librería está compuesta por diferentes funciones, cada una de éstas es creada específicamente para el problema que se desea optimizar, así pues se han creado dos funciones de coste, una para el planificador y otra para el coordinador. Para obtener el valor de coste se llama a la función *double GetFitness(void).*
- **Programa principal (main.cpp),** dentro del programa principal se ejecuta la evolución cuando en la ejecución ponemos la opción "-e". Las funciones que encontramos dentro del programa principal son:
  - *RunEvolutionaryAlgorithm (void),* para iniciar la evolución.
  - *CPopulation* GetPopulation (void),* crea un tipo de población con los parámetros genéticos necesarios para ejecutar los operadores genéticos.
  - *unsigned int GetNumberOfLastGeneration (void),* devuelve el número de la última generación evaluada.
  - *LoadGeneration(CPopulation* pc_population, unsigned int un_generation),* cuando la evolución no comienza desde el principio sino que comienza desde una determinada generación se carga dicha generación y se vuelva a evolucionar.
  - *double EvaluateIndividual(double* pf_chromosome, unsigned int un_chromosome_length),* devuelve el valor de coste asignado al individuo por la función de coste.
  - *CFitnessFunction* GetFitnessFunction(int n_number, CSimulator* pc_simulator),* establece la función de coste para evaluar a los distintos individuos.
  - *void GetGeneticParameters(void),* lee del fichero de parámetros genéticos, pasado por línea de comando con la opción "-g", los valores necesarios para la ejecución del algoritmo.

Ya hemos explicado en qué consiste cada una de las funciones para llevar a cabo el algoritmo genético. A continuación, mostramos como se ejecuta el algoritmo

genético mediante el siguiente pseudocódigo:

```
//Comienzo de ejecucion del algoritmo genetico
void RunEvolutionaryAlgorithm()
{
  //Comienzo de tiempo de evolucion
  //Se establecen las variables de:
    //mejor generacion
    //Variables para las estadisticas

  //Se crea la poblacion de individuos con los parametros geneticos
  CPopulation* pcPopulation = GetPopulation();

  //En caso de volver a ejecutar otra evolucion
  //se obtine la ultima generacion
  if (g_bRestartEvolution)
    g_nRestartFromGeneration = GetNumberOfLastGeneration();

  //Generacion de comienzo y actual
  int          nStartGeneration;
  unsigned int unCurrentGeneration;
  //Variables de individuo
  // cromosoma y longitud de este

  //Inicio de la evolucion
  for (unCurrentGeneration = nStartGeneration;
       unCurrentGeneration < g_unGenerations;
       unCurrentGeneration++)
  {
    //En caso de comenzar la evolucion creamos al primera generacion
    //aleatoriamente el resto las creamos con los operdaores geneticos
    if (unCurrentGeneration == 0 || (g_bRandomizeFirstGeneration
                  && unCurrentGeneration == nStartGeneration))
      pcPopulation->CreateRandomPopulation(g_unChromosomes);
    else
      pcPopulation->CreateNextGeneration();

    //Evaluamos individuo por individuo la generacion
    int unCurrentChromosome = 0;
    while (pcPopulation->GetNextChromosomeToEvaluate(&pvIndividual,
          &pfChromosome,
          &unChromosomeLength))
    {
      for (int nSample = 0; nSample < g_unNumberOfSamplesPerChromosome;
                                                          nSample++)
      {
        //Obtenemos su valor de coste
        double fThisSampleFitness = EvaluateIndividual(pfChromosome,
```

```
                                                    unChromosomeLength);
            //Acumulamos el valor de coste
            fAccumulatedFitness += fThisSampleFitness;

            //Actualizamos los peores y mejores valores de coste
            if (fThisSampleFitness < fWorstSampleFitness)
              fWorstSampleFitness = fThisSampleFitness;
            if (fThisSampleFitness > fBestSampleFitness)
              fBestSampleFitness = fThisSampleFitness;
          }

          //Actualizamos para pasar al siguiente individuo
          unCurrentChromosome++;
          unEvaluations++;
        }

        //Se imprime por pantalla las estadisticas de la generacion actual
        //el mejor, el peor y el valor medio de coste

        //Se comprueba si esta generacion es la mejor del momento
        //si lo es se guarda su valor.
        if (fGlobalFitness > fBestFitnessRecorded)
        {
          unGenerationWithBestFitness       = unCurrentGeneration;
          fBestFitnessRecorded          = fGlobalFitness;
        }

        //Se escribe en un fichero el mejor cromosoma de la generacion, en
        //otro fichero se escribe el mejor cromosoma de todas las generaciones
        //y escribimos en otro un registro de los valores de coste asignados
        //en cada generacion.
      }
    }
```

Ésta es la implementación software de ambos algoritmos. Durante la fase de entrenamiento la red es evolucionada con el algoritmo genético de modo que el cromosoma contiene el valor de los parámetros libres de la red (pesos, ganancias y sesgos) y se evolucionará en base a una función de coste que tendrá la información necesaria para obtener el objetivo que se persigue con la red neuronal. Así se irán modificando los cromosomas, y con ello los parámetros de la red neuronal, hasta llegar a un óptimo o un máximo. Finalizada la evolución, cuando se ejecute la red neuronal, ésta tomará el mejor cromosoma de la evolución y llevará a cabo el objetivo

para el que fue diseñada.

A continuación se describirá la implementación del planificador y del coordinador, en el que mostraremos sus características así como los resultados.

4.3. El Planificador

4.3.1. Introducción

El planificador tiene como objetivo posicionar en el eje temporal las tareas impuestas por el usuario, minimizando el consumo de la red eléctrica en favor del uso de la energía fotovoltaica, dando prioridad al usuario.

El usuario intervendrá en la planificación estableciendo las tareas (programas) a realizar y el intervalo de tiempo en que se desea se ejecute la tarea correspondiente. Por ejemplo, activar la lavadora entre las 10 a.m. y 5 p.m., con un programa a 90°C y 1200 rpm (ver Figura 4.2).

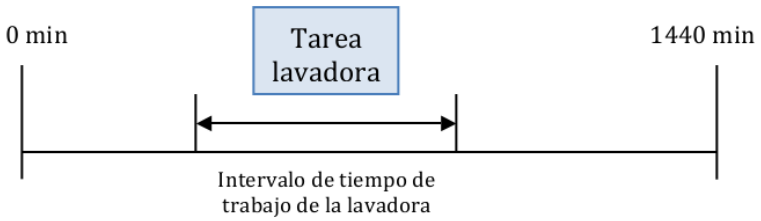


Figura 4.2: Ejemplo de restricción temporal.

El planificador, una vez establecidas las prioridades del usuario, comenzará con la planificación de cada una de las tareas. Cada tarea es planificada de forma independiente al resto. Para obtener el momento en el que debe ejecutarse la tarea se necesitan las restricciones temporales del usuario así como el perfil de generación fotovoltaico predicho del día siguiente (ver Figura 4.3). El perfil fotovoltaico consiste en 24 valores (uno por cada hora del día) de potencia que se estima que generará el sistema fotovoltaico (Porro, 2010).

Ambas informaciones son las entradas a nuestro planificador. El planificador

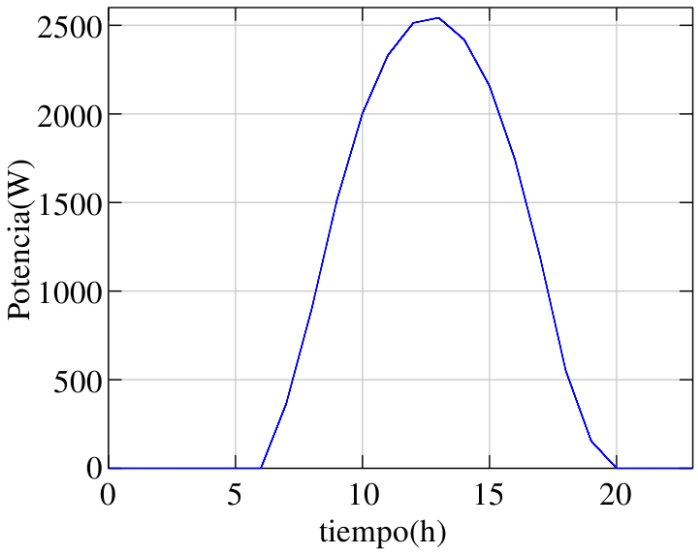


Figura 4.3: Ejemplo de perfil fotovoltaico predicho.

consiste en una red compuesta de tantas redes neuronales como tareas haya establecido el usuario. Cada una de estas redes neuronales es un perceptrón multicapa (ver Figura 4.4). La estructura de la red del planificador siempre será la misma, pero se añadirán tantas redes como número de tareas. Las entradas a cada una de estas redes son:

- Los tiempos de inicio (t_{oi}) y fin (t_{fi}) de la tarea, fijados por el usuario.
- El perfil fotovoltaico de generación del día siguiente (P_{fv0-23}).

Una vez procesadas las entradas y realizados los cálculos obtendremos una salida que representa el momento de ejecución de la tarea (t_i). Este tiempo no es realmente el tiempo de inicio de la tarea sino el centro de la duración de la tarea de tal forma que pueda aprovecharse el máximo de generación fotovoltaica el mayor tiempo posible (ver Figura 4.5). La planificación consiste en encontrar el tiempo que garantice el mayor autoconsumo de energía generada. Para ello la red neuronal escoge el tiempo, dentro del intervalo fijado por el usuario, en que la potencia generada predicha es máxima, de tal forma que consumamos la mayor cantidad de energía fotovoltaica generada en detrimento de la energía proveniente de la red eléctrica. Para conseguir encontrar ese

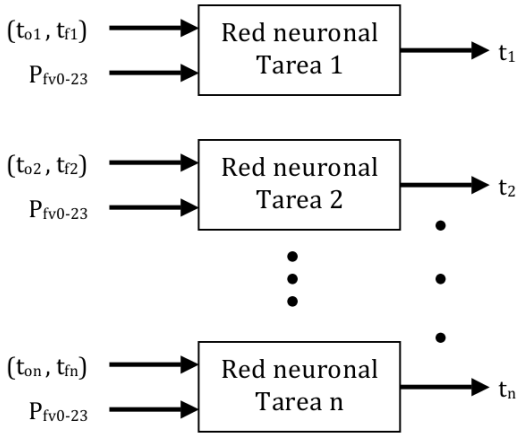


Figura 4.4: Estructura del planificador.

tiempo se ha evolucionado la red neuronal con un algoritmo genético cuya función de coste o ajuste (*fitness*) busca el máximo de potencia dentro del intervalo de tiempo impuesto por el usuario (ver Sección 4.3.3).

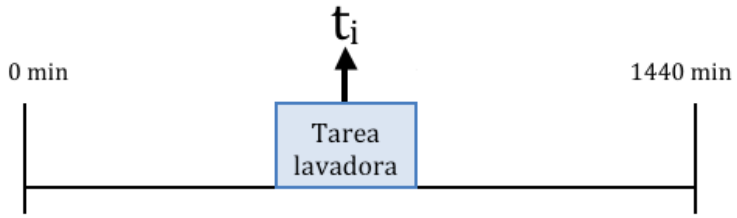


Figura 4.5: Ejemplo de salida de planificación.

Puesto que el planificador se compone de tantas redes neuronales idénticas como tareas haya impuesto el usuario, las redes cuyo intervalo de activación incorpore el máximo del perfil fotovoltaico obtendrán la misma salida. Esta característica de la capa de planificación impone ciertas restricciones al sistema. Puesto que uno de los objetivos del sistema de control es no solapar tareas, el coordinador será el encargado de modificar las planificaciones anteriores (ver Sección 4.4).

Esto puede llevar a la conclusión errónea de que una planificación distribuida, como es ésta, es muy poco eficiente. Sin embargo, esta planificación distribuida es

mucho más eficiente en tiempo de cálculo, ya que una vez evolucionada la red, la respuesta es inmediata, y además al ser distribuida la información no se concentra en un sólo punto; de tal modo que aunque las conexiones de una red se rompan siempre tendremos respuesta y el que se estropee una red no involucra a otras tareas. Esta última característica hace que los sistemas distribuidos sean más robustos que los sistemas centralizados ya que si una tarea no es planificada correctamente la planificación entera es errónea. Además, este planificador es un sistema doblemente distribuido ya que por un lado las planificaciones de tareas distintas se llevan a cabo de forma independiente y en segundo lugar por el carácter distribuido de las redes neuronales ya que la información se encuentra repartida por todas las neuronas que la forman.

Para acabar de entender la planificación neuronal la explicaremos mediante un ejemplo. Tenemos un usuario que requiere tres tareas (ver Figura 4.6):

- Una lavadora entre las 10 a.m. y 6 p.m., con un programa a 90°C y 1200 rpm.
- Una secadora entre las 4 p.m. y las 10 p.m., con un programa a 1600 rpm.
- Un lavavajillas entre las 9 a.m. y las 5 p.m., con el programa de ejecución 5.

Usaremos el perfil de generación fotovoltaica predicho para el día siguiente (ver Figura 4.7). Las salidas de planificación, observando el perfil fotovoltaico junto con las restricciones temporales, serán: para la lavadora (t_{a1}) igual a 2 p.m., que es el máximo para el intervalo de trabajo, para la secadora (t_{a2}) igual a 5 p.m. y para el lavavajillas (t_{a3}) igual a 2 p.m. Obsérvese que las salidas planificadas de la lavadora y el lavavajillas coinciden, pero no es problema ya que ahora pasarán a la siguiente fase del controlador que, junto con la duración de cada tarea, se encargará de solucionar este tipo de situaciones (ver Sección 4.4).

En las siguientes secciones profundizaremos en la implementación de la arquitectura de la red neuronal y en la evolución del planificador.

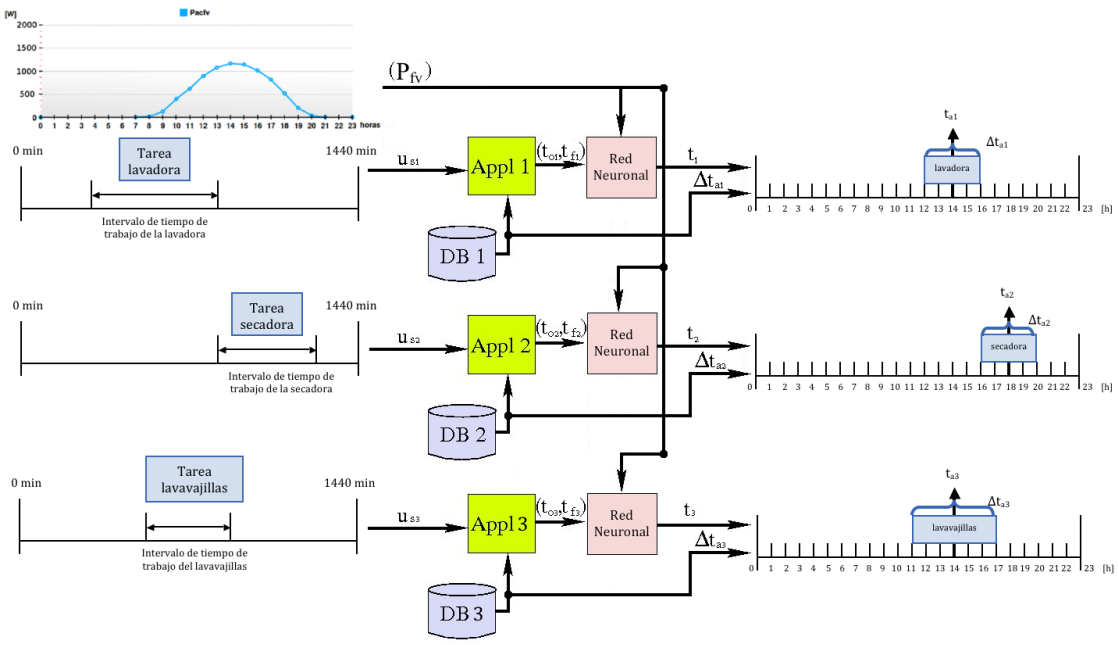


Figura 4.6: Ejemplo de planificación.

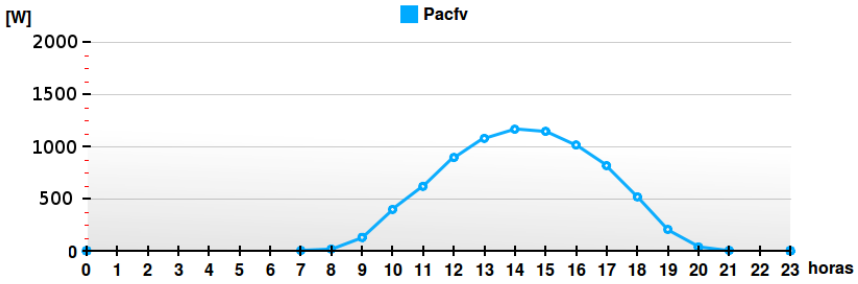


Figura 4.7: Ejemplo de perfil fotovoltaico predicho.

4.3.2. Arquitectura de la red

La arquitectura de las redes neuronales, que forman el planificador, es multicapa, es decir, que cuenta con más capas de neuronas además de la de entrada y salida, como ya se explicó en el Capítulo 2. En concreto se trata de un perceptrón multicapa que consta de tres capas de neuronas: una de entrada, una capa oculta y una capa

de salida (ver Figura 4.8). Como ya se ha resaltado, el objetivo de la planificación es encontrar el tiempo, dentro de un intervalo de trabajo, en el que el autoconsumo sea lo máximo posible. Para ello las entradas a nuestra red deben ser: el intervalo definido por el usuario (t_o, t_f) y el perfil fotovoltaico de potencia predicha para el día siguiente (P_{fv0-23}). De este modo cada red estará formada por:

- 26 neuronas en la capa de entrada: dos para el intervalo temporal fijado por el usuario (t_o, t_f) y las otras 24 para el perfil fotovoltaico predicho, un valor de potencia generada por cada hora del día ($P_{fv0},\ P_{fv1},\ \cdots,\ P_{fv23}$). La operación que realizan las neuronas de esta capa consiste únicamente en multiplicar la entrada por una ganancia ($y_{in,i} = x_{in,i} \cdot g_{in,i}$). Después pasan el resultado a la capa siguiente.

- 13 neuronas en la capa intermedia. Se ha escogido la mitad de neuronas que las de la capa de entrada para que el procesamiento de la información tenga suficientes neuronas para realizar los cálculos adecuadamente. Las entradas a esta capa son las salidas de la capa anterior y a cada neurona de esta capa le llega la salida de cada una de las neuronas de la capa anterior. La operación que realizan las neuronas es $y_{hi,i} = \frac{1}{1+e^{-(\sum w_{ij} \cdot y_{in,j} + \theta_i)}}$. Después sus resultados pasan a la capa siguiente.

- 1 neurona en la capa de salida. Sólo es necesario un valor que será el tiempo planificado que cumple cual es la hora en la que la generación es máxima para el intervalo temporal. Las salidas de la capa oculta convergen todas en esta neurona, la cual realiza la operación siguiente: $t_i = \frac{1}{1+e^{-(\sum w_{ij} \cdot y_{hi,j} + \theta_i)}}$.

La estructura de esta red es única e independiente del número de tareas que se quiera realizar. Por cada tarea añadida por el usuario se creará una red neuronal adicional. De tal modo que es el número de tareas a planificar el que nos indica el número de redes neuronales que tendrá el planificador, mientras que el valor de los pesos de la red será el mismo para cada una de las redes.

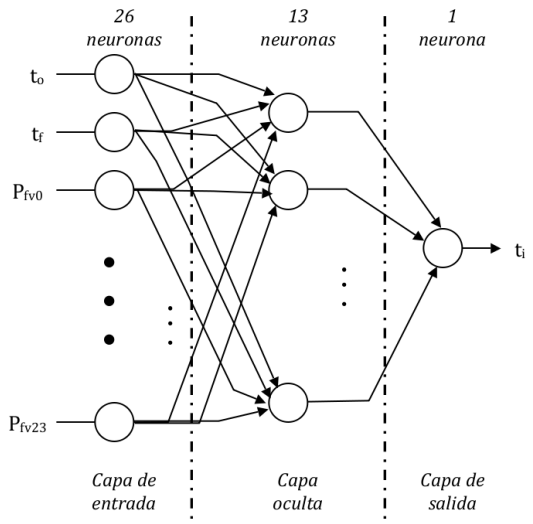


Figura 4.8: Arquitectura del planificador.

Los parámetros de la red neuronal se establecen a través de un fichero y cuando se ejecute el experimento se pasa por línea de comando con la opción "-n". El fichero de parámetros para las redes neuronales del planificador se compone de:

- **Número de neuronas de entrada,** se indica cuantas neuronas debe haber en la capa de entrada. *Ej. NUMBER OF INPUT NEURONS = 26.*
- **Número de neuronas ocultas,** cuántas neuronas forman la capa oculta de la red. *Ej. NUMBER OF HIDDEN NEURONS = 13.*
- **Número de neuronas de salida,** indica las neuronas de la capa de salida. *Ej. NUMBER OF OUTPUT NEURONS = 1.*
- **Límites de ganancia de entrada,** se establece un límite superior e inferior de ganancia que no se pueden superar. *Ej. INPUT GAIN LOWER BOUND = 0.0; INPUT GAIN UPPER BOUND = 1.0.*
- **Límites del parámetro de ajuste (bias),** se establece un límite superior e inferior al valor del parámetro de ajuste que no se puede sobrepasar. *Ej. BIAS LOWER BOUND = -5.0; BIAS UPPER BOUND = 5.0.*

- **Límites de los pesos,** se establece un límite superior e inferior al valor de los pesos de todas las neuronas que no se pueden superar. *Ej. WEIGHTS LOWER BOUND = -5.0; WEIGHTS UPPER BOUND = 5.0.*

Los parámetros anteriores son los que se han utilizado en el experimento del planificador. El software del planificador se encarga de establecer cada uno de los parámetros de la red así como crear el número de redes neuronales en función de las tareas que quiera realizar el usuario. Una vez establecidos todos estos condicionantes comienza la ejecución del planificador y obtendríamos los diferentes tiempos planificados para cada una de las tareas.

4.3.3. El algoritmo genético

El algoritmo genético es usado para sintonizar correctamente la red neuronal consiguiendo de esta manera que la red se pueda utilizar para el objetivo definido. Para llevar a cabo la evolución es necesario fijar los distintos parámetros de los operadores genéticos así como la función de coste a utilizar. Los parámetros del algoritmo genético son fijados en la ejecución pasando un fichero como entrada, con las características que describiremos a continuación, mediante la opción "-g". Los parámetros son:

- **Longitud del cromosoma,** el cromosoma estará formado por todos aquellos parámetros libres de la red neuronal, ganancias de entrada ($g_{in,i}$), sesgos (θ_i) y pesos neuronales (w_{ij}). La longitud del cromosoma viene expresada por la siguiente ecuación:
  $chromosomeLength = N^o InputNeurons + N^o HiddenNeurons + N^o OutputNeurons + (N^o InputNeurons \cdot N^o HiddenNeurons) + (N^o HiddenNeurons \cdot N^o Output Neurons) = N\, g_{in,i} + N\, \theta_{hi,i} + N\, \theta_{out,i} + N\, w_{hi,ij} + N\, w_{out,ij}$.
  En nuestro caso la longitud del cromosoma sería igual a $26 + 13 + 1 + (26 * 13) + (13 * 1) = 391$ genes. Cada gen toma un valor real entre cero y uno. Cuando se guarda un cromosoma este comienza con el valor de la longitud del cromosoma y después vienen los diferentes genes que forman el cromosoma propiamente dicho.

- **Tamaño de la población,** con este parámetro se fija el número de individuos que forman una población.
- **Número de generaciones,** se indica cuantas generaciones como máximo va a evolucionar el algoritmo genético.
- **Tiempo de evaluación,** tiempo que se tardará en evaluar un cromosoma de un individuo.
- **Activar crossover,** permite hacer crossover o no.
- **Número de crossovers,** cuántos crossovers se pueden realizar por generación.
- **Distancia del crossover,** número de puntos donde realizar el crossover.
- **Ratio de mutación,** la probabilidad de que un gen al crear una nueva generación se mute.
- **Número de élites,** indica el número de cromosomas que se copiarán y pasarán a la generación siguiente.
- **Función de coste (fitness),** seleccionamos la función que evaluará los distintos individuos de la población.

Los parámetros genéticos que se han utilizado son:

CHROMOSOME LENGTH = *391*
POPULATION SIZE = *100*
NUMBER OF GENERATIONS = *10000*
EVALUATION TIME = *0*
DO CROSSOVER (0 No, 1 Yes) = *1*
NUMBER OF CROSSOVERS (Always 1) = *1*
CROSSOVER DISTANCE (Always 1) = *1*
MUTATION RATE = *0.05*
NUMBER OF ELITES = *5*
FITNESS FUNCTION = *1*

La función de coste desarrollada evalúa cada individuo para cada vector de entradas de la siguiente manera:

- Busca el máximo de potencia ($P_{fv,max}$) y el tiempo (t_{max}) en el que se produce, en el perfil predicho de generación fotovoltaica ($P_{fv}(t)$), pero restringiéndose al intervalo fijado por el usuario (t_0, t_f).

- Desnormaliza la salida de la red neuronal, obtiene el tiempo planificado (t_{out}) y halla la potencia fotovoltaica predicha ($P_{fv}(t_{out})$) para dicho tiempo.

- Evalúa ahora el resultado generado por ese individuo para un vector de entradas dado y le da un valor de coste según la siguiente función.

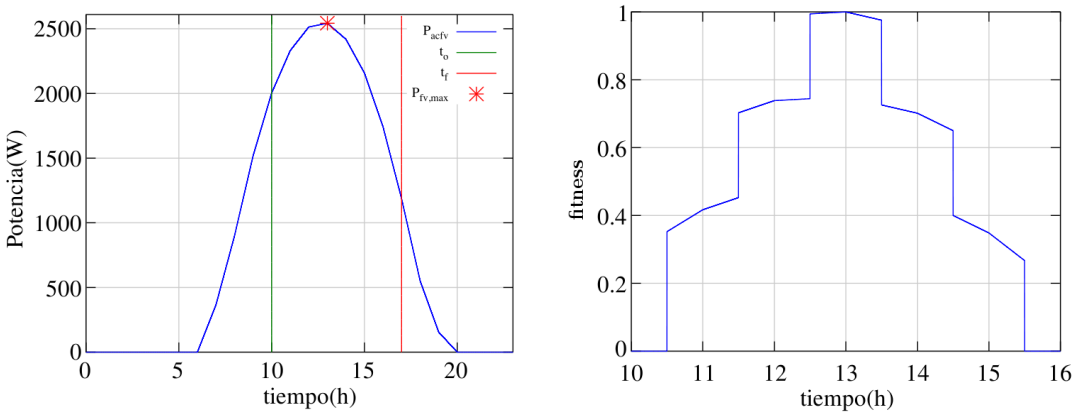
$$fitness(t) = \begin{cases} 0 & \text{si } t < t_{max} - 2, \\ \frac{P_{fv}(t)}{P_{fv,max}} - 0{,}5 & \text{si } t_{max} - 2 \geq t < t_{max} - 1, \\ \frac{P_{fv}(t)}{P_{fv,max}} - 0{,}25 & \text{si } t_{max} - 1 \geq t < t_{max}, \\ \frac{P_{fv}(t)}{P_{fv,max}} = 1 & \text{si } t = t_{max}, \\ \frac{P_{fv}(t)}{P_{fv,max}} - 0{,}25 & \text{si } t_{max} < t \leq t_{max} + 1, \\ \frac{P_{fv}(t)}{P_{fv,max}} - 0{,}5 & \text{si } t_{max} + 1 < t \leq t_{max} + 2, \\ 0 & \text{si } t > t_{max} + 2. \end{cases}$$

- Cada individuo es evaluado para un número determinado de situaciones. El valor de coste (*fitness*) del individuo será la media de los valores de coste para cada una de las situaciones, es decir, $fit_{ind} = \frac{\sum_n (fit_n)}{n}$.

- Se evalúan todos los individuos de esa generación.

- Finalmente el valor de coste de la generación será la media aritmética del coste acumulado, es decir, $fit_{gen} = \frac{\sum_i fit_{ind,i}}{n_{ind}}$.

Para observar mejor cómo es la función de coste la describiremos con un ejemplo. Tomamos el perfil y las restricciones temporales (ver Figura 4.9(a)), el máximo se encuentra en la hora 13. Para construir la función de coste tomamos un entorno dos horas antes del máximo (la hora 11) y dos horas después (la hora 15) y hacemos



(a) (b)

Figura 4.9: Ejemplo de fitness (a) datos de entrada y (b) función de fitness.

el resto cero. Dividimos entre el máximo de potencia ($P_{fv,max}$) todo y aplicamos la ecuación descrita anteriormente. El resultado final puede verse en la Figura 4.9(b).

Ya se han definido todos los parámetros utilizados para evolucionar la red. Ahora puede comenzar la evolución.

4.3.4. Evolución

Una vez establecidos los parámetros de la evolución se puede comenzar a evolucionar el sistema. La evolución consiste en un proceso iterativo que evaluará los individuos de una generación y que mediante los operadores genéticos sólo nos quedaremos con los mejores de cada generación. La evolución siempre mejorará o igualará a la generación anterior nunca la empeorará, gracias a que como se ha comentado en el Capítulo 2, mantendremos los 5 mejores individuos (élites) idénticos de una generación a otra. De este modo la evolución no acabará hasta llegar a un valor de fitness uno o hasta que no hayamos evaluado todas las generaciones impuestas por el fichero de parámetros.

Para nuestras evoluciones, hemos contado con 142 perfiles fotovoltaicos predichos desde el mes de julio al mes de diciembre de 2009 y con restricciones temporales por parte del usuario de 4 horas que se desplazan en saltos de

dos horas a lo largo del día. Con lo que cada individuo en cada una de las generaciones es evaluado, $142 * 11 = 1.562$ veces y cada generación contaría con $1.562 * 100\, individuos = 156.200\, iteraciones$. La duración total del experimento sería de $10.000\, generaciones * 156.200 = 1.562.000.000\, iteraciones$, por lo que la evolución del sistema, en tiempo de ejecución, puede tardar varios días.

Pero, en el caso del planificador, no se han realizado todas las iteraciones de una vez sino que se ha realizado una evolución por tramos, ya que al evolucionar conjuntos menores de individuos logramos evolucionar más rápido y conseguimos que esos pequeños grupos obtengan mejores resultados de fitness que si utilizaremos todo el conjunto. La evolución ha consistido en:

- Realizar una evolución únicamente con 10 perfiles fotovoltaicos y 11 usuarios con un intervalo de restricción de 4 horas diferenciándose en 2 horas entre sí. Obteniendo un resultado de mejor coste de 0,9091740680 en 2189 generaciones (ver Figura 4.10(a)).

- Posteriormente, continúa la evolución con 30 perfiles obteniendo un coste de 0,9172684079 en 996 generaciones para el mejor individuo (ver Figura 4.10(b)).

- Continuamos con 50 perfiles y obtenemos 0,9027720441 en 287 generaciones (ver Figura 4.10(c)). Hasta aquí los resultados son todos más o menos parecidos ya que todos los perfiles pertenecen a la estación del verano.

- En la siguiente evolución seleccionamos 90 perfiles, entre estos perfiles existen mayores diferencias, por lo que el máximo coste que conseguimos es de 0,817121918 en 1186 generaciones (ver Figura 4.10(d)). Esta bajada en el mejor coste se debe a que los perfiles ya no son tan parecidos como antes ya que al tener meses pertenecientes a distintas estaciones sus valores de potencia y horas en las que se generan cambian.

- Evolucionamos con los 142 perfiles obteniendo como máximo valor de coste 0,702618381 en 847 generaciones (ver Figura 4.10(e)).

- Finalmente se ha realizado una evolución variando el intervalo de restricción temporal. En esta última evolución hemos evolucionado con los 142 perfiles y un único usuario con un intervalo de restricción de 24 horas. De esta forma conseguimos que la red seleccione el máximo de generación del perfil y pueda establecer un tiempo de planificación por defecto en caso que el usuario no lo proporcione. El resultado ha sido de 0,9060620785 en 700 generaciones para el mejor individuo (ver Figura 4.10(f)).

Con esta evolución se ha conseguido abarcar el mayor número de casos posibles a los que la red neuronal se puede enfrentar. Como resultado de la evolución finalmente se ha obtenido un cromosoma que contiene en su interior toda la información para ejecutar el planificador. En la Sección 4.3.5 analizaremos el resultado de la evolución y en definitiva el funcionamiento del planificador para el cumplimiento de su objetivo.

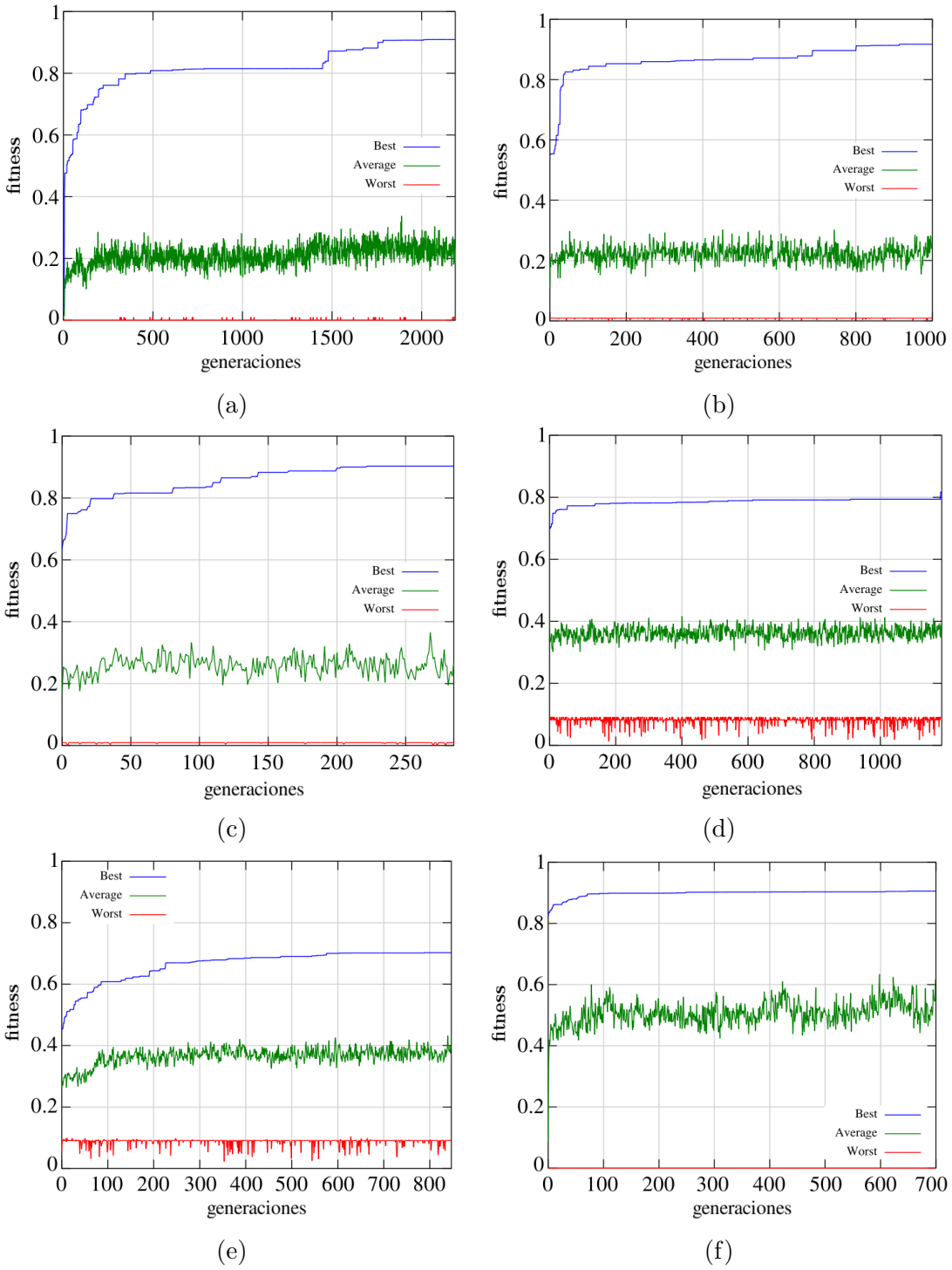


Figura 4.10: Resultados de la evolución: (a) 10 perfiles, (b) 30 perfiles, (c) 50 perfiles, (d) 90 perfiles, (e) 142 perfiles y (f) variación del intervalo temporal.

4.3.5. Análisis de la evolución

El entrenamiento que se ha llevado a cabo ha consistido en evaluar cada uno de los perfiles fotovoltaicos con 11 perfiles de usuario primeramente y después con un único perfil de usuario. En teoría al ejecutar ahora la red neuronal para cada uno de los perfiles con los que se ha entrenado, debería ser capaz de establecer el tiempo del máximo dentro del intervalo en el que nos encontramos. Sin embargo, tal y como se aprecia en los valores de fitness obtenidos, observamos ciertas restricciones:

- La red neuronal después de la evolución localiza el máximo para los 11 perfiles de usuario en un 96 %. En un 3 % de las situaciones la red obtiene un tiempo planificado que dista menos de una hora del máximo del intervalo y sólo en un 1 % de los casos el tiempo planificado se separa del tiempo del máximo en más de una hora.

- Para el caso de un perfil de usuario único, la red ha sido capaz de establecer el tiempo del máximo del perfil en un 90 % de los casos con un intervalo de tiempo suficiente que abarcara al máximo absoluto del perfil. Sin embargo, en un 6 % de las situaciones el tiempo obtenido por la red se ha desviado en menos de una hora del tiempo del máximo y en el 4 % restante la diferencia ha sido mayor de 1 hora.

- Las desviaciones de los tiempos en las salidas de la red se deben a dos factores:

    - El primero es debido a que en el entorno del máximo los valores de potencia son muy parecidos y la red no es capaz de distinguirlo. Además entre perfiles fotovoltaicos el tiempo en el que se produce el máximo de generación fotovoltaica oscila en una hora (la hora 12 o la hora 13) lo que complica la respuesta de la red.

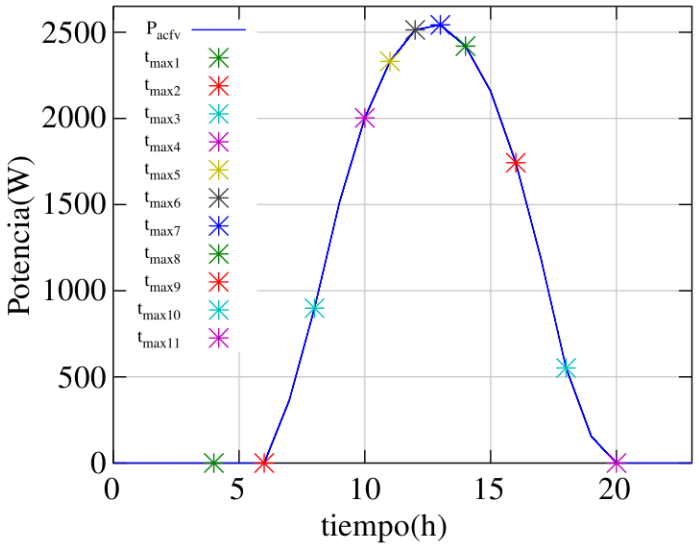
    - Y el segundo factor se debe a que dentro de la evolución han existido diferentes perfiles, que denominaremos con cambios bruscos (un 8 % sobre el total de los perfiles) (ver Figura 4.12). Para dichos perfiles la red no ha

conseguido encontrar el máximo absoluto cuando el intervalo de restricción temporal es único. Sin embargo, cuando el intervalo temporal es de 4 horas sólo en un 2 % la red no ha encontrado el máximo. Estos cambios bruscos en los perfiles son descartados por la evolución ya que son una minoría.

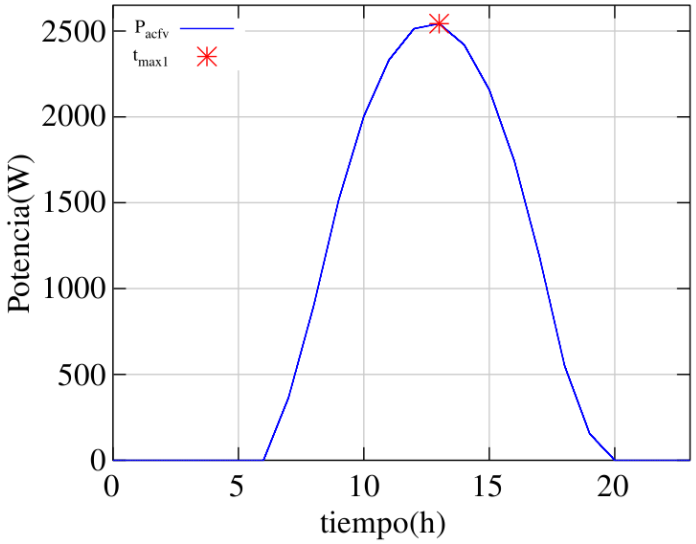
- La explicación de por qué la red responde mejor a los 11 perfiles de usuario en lugar del intervalo de usuario único, se debe al entorno de puntos del perfil fotovoltaico que se toman en cada caso. Para el caso de los 11 perfiles de usuario el máximo del intervalo se encuentra más acotado que en el caso del intervalo de usuario único, ya que contiene menos puntos de potencia del perfil fotovoltaico, siendo más sencilla su localización.

- Otra observación hecha en la evolución es que al realizarla con el perfil único, partiendo de lo ya evolucionado anteriormente, se desplazan ligeramente los tiempos de salida de la red. Aunque esto no tiene mayor relevancia ya que se siguen obteniendo correctamente los tiempos planificados.

En la Figura 4.11 se encuentran sobre el perfil de potencia predicha generada aquellos puntos que la red obtiene como salidas de planificación para el caso de intervalos de 4 horas o para un intervalo único de tiempo.



(a)



(b)

Figura 4.11: Resultado de ejecución: (a) intervalos de 4 horas y (b) intervalo temporal único.

En la Sección 4.5.1, analizaremos el controlador neuronal al completo (planificador + coordinador) donde estudiaremos la post-evaluación del sistema así como el comportamiento del mismo para los vectores de entrada no suministrados a la

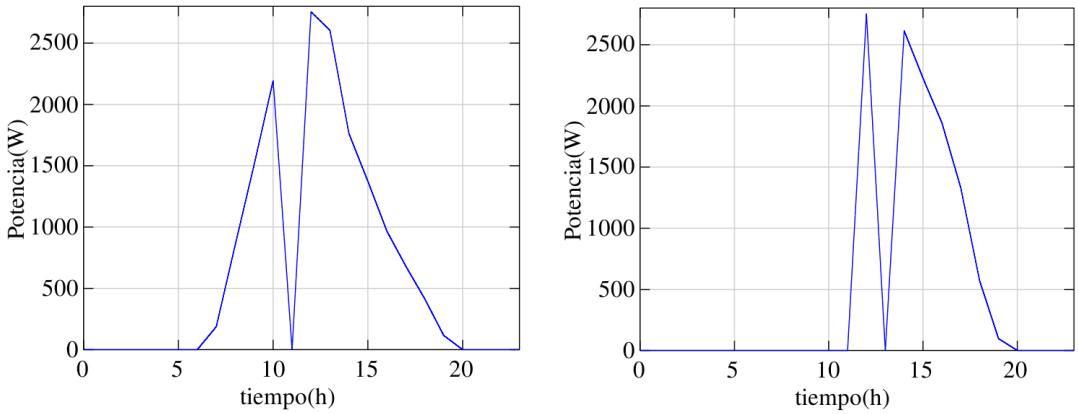


Figura 4.12: Perfiles con cambios bruscos.

evolución.

4.4. El Coordinador

4.4.1. Introducción

La necesidad de coordinar las tareas es evidente, ya que como hemos visto anteriormente, el instante temporal en que se deberían ejecutar cada una de las diferentes tareas puede coincidir y como requisito hemos impuesto que se esté ejecutando una tarea cada vez y no exista solapamiento. Por tanto el objetivo de nuestro coordinador neuronal será que las tareas no se solapen durante su ejecución. Ahora ya no tendremos en cuenta las restricciones temporales del usuario sino que para poder llevar a cabo todas las tareas sin que se solapen tratará de encajar cada tarea en el tiempo correspondiente más cercano al planificado (ver Figura 4.13).

Figura 4.13: Tareas coordinadas.

Para realizar el coordinador se ha utilizado también un perceptrón multicapa. Las entradas de este perceptrón son:

- El tiempo planificado por cada una de las tareas (t_{ai}).
- La duración de cada una de las tareas según el programa seleccionado para cada una de éstas (Δt_{ai}).

Estas entradas se procesarán en el interior de la red neuronal y se obtendrán las salidas deseadas sin que solape ninguna tarea o con el menor solapamiento posible. La salida del coordinador se corresponde con una tupla de tiempos planificados finales para cada una de las tareas (t'_{a1}, t'_{a2}, ..., t'_{an}). Esta vez la red neuronal se ha evolucionado con un algoritmo genético cuya función de coste (*fitness*) evalúa si las tareas se encuentran solapadas o no. En caso afirmativo su función será separar los tiempos planificados para no solapar y en caso negativo dejará el tiempo planificado como estaba. De este modo el algoritmo genético variará los pesos hasta encontrar el mejor cromosoma que cumpla las condiciones que se han establecido. La implementación real de la red neuronal y el algoritmo genético del coordinador se encuentra en las Secciones 4.4.2 y 4.4.3.

Para comprender mejor en que consiste el coordinador neuronal ilustraremos con un ejemplo en lo que consiste (ver Figura 4.14). Retomando el ejemplo de planificación anterior (ver Sección 4.3), hemos llegado a planificar tres tareas (una lavadora entre las 10 a.m. y 6 p.m., con un programa a 90ºC y 1200 rpm, una secadora entre las 4 p.m. y las 10 p.m., con un programa a 1600 rpm y un lavavajillas entre las 9 a.m. y las 5 p.m., con el programa de ejecución 5) cuyos tiempos de planificación han resultado: lavadora, $t_{a1} = 2\,p.m.$, secadora, $t_{a2} = 5\,p.m.$, y lavavajillas, $t_{a3} = 2\,p.m.$ Estos resultados junto con las duraciones de cada una de las tareas ($\Delta t_{a1},\ \Delta t_{a2},\ \Delta t_{a3}$) se pasan como entradas al coordinador.

El coordinador procesará las entradas y a su salida obtendremos los siguientes tiempos que pasarán a la ejecución: lavadora, $t'_{a1} = 9\,a.m.$, secadora, $t'_{a2} = 7\,p.m.$, y lavavajillas, $t'_{a3} = 2\,p.m.$. Como podemos observar hemos respetado las restricciones

temporales de la secadora y del lavavajillas en este caso, pero no de la lavadora. En el coordinador ya no interviene el usuario y como en este caso no era posible cumplir sus preferencias sin que se solaparan las tareas, el coordinador las ha colocado lo más juntas posibles sin que se solapen[2]. Con este coordinador hemos resuelto dos inconvenientes: tiempos de planificación iguales y solapamiento de tareas.

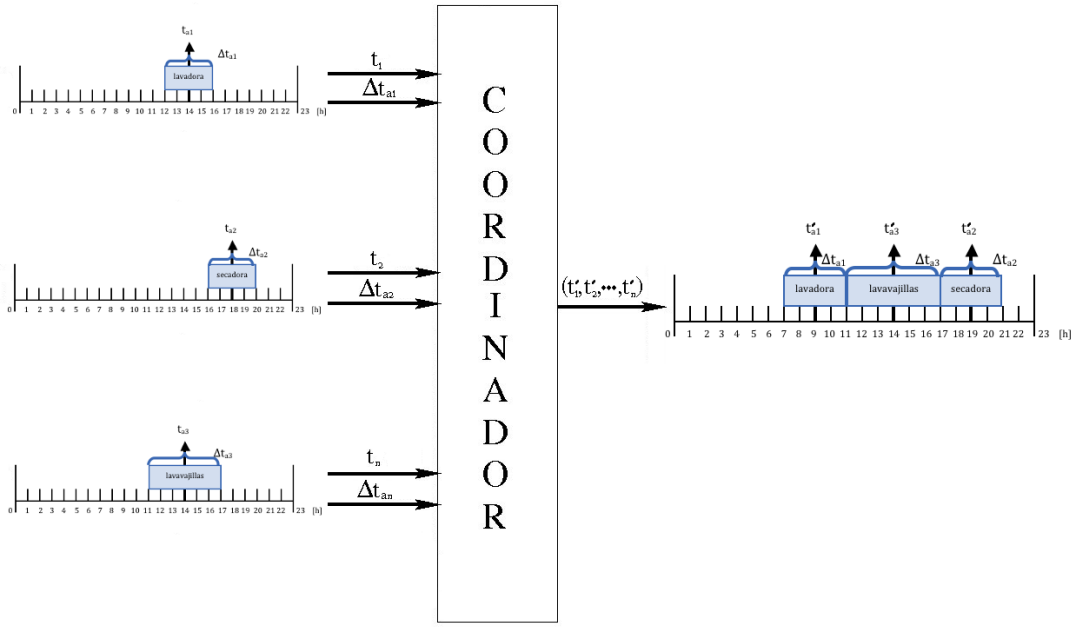


Figura 4.14: Ejemplo de coordinación.

En resumen, el coordinador tiene como objetivo organizar los tiempos planificados de la capa anterior de tal modo que las tareas no se solapen. Para ello se sirve de los tiempos de planificación y de la duración de las distintas tareas que el usuario ha prefijado que se ejecutarán en las próximas 24 horas.

[2]Esta información es la que se suministrará al usuario para ofrecer la mejor planificación de acuerdo a los criterios establecidos. En caso de que el usuario no esté de acuerdo, será necesario que la lavadora se ejecute a las 10 a.m. en vez de las 9 a.m. aunque exista un mayor solapamiento.

4.4.2. Arquitectura de la red

El coordinador está formado por una única red neuronal, cuya arquitectura también es multicapa (ver Capítulo 2), en concreto un perceptrón de 3 capas una de entrada, una oculta y una de salida (ver Figura 4.15), cuyo número de neuronas varía según el número de tareas que se hayan planificado (ver Sección 4.4.1). El objetivo de la coordinación, como ya se ha dicho anteriormente, consiste en organizar sobre el eje temporal los tiempos planificados de la capa anterior de tal forma que sólo se ejecute una tarea cada vez. Para ello las entradas al coordinador deben ser los tiempos planificados así como la duración de cada una de las tareas que quiere ejecutar el usuario. La red estará formada por:

- $2 \cdot n$ neuronas de entradas siendo n el número de tareas a realizar. Las entradas estarán organizadas por pares, cada par corresponde al tiempo planificado (t_i) y la duración del intervalo (Δt_i) de cada una de las tareas planificadas. Al igual que las neuronas de la capa de entrada del planificador, las neuronas de la capa de entrada del coordinador multiplican cada una de las entradas por una ganancia ($y_{in,i} = x_{in,i} \cdot g_{in,i}$). A continuación pasa el resultado a la capa siguiente.

- $n+1$ neuronas en la capa oculta. Se ha escogido este número debido a que en la capa de salida tendremos por lo general más de una neurona de salida por lo que es necesario que la capa intermedia cuente con bastantes neuronas intermedias para realizar los cálculos. La operación que llevan a cabo las neuronas de esta capa consiste en: $y_{hi,i} = \frac{1}{1+e^{-(\sum_j w_{ij} \cdot y_{in,j} + \theta_i)}}$. Sus resultados pasan a la capa siguiente.

- n neuronas de salida. Cada neurona de salida se corresponde con cada uno de los tiempos finales en que se ejecutarán cada una de las tareas. La operación que realiza cada una de estas neuronas es: $t'_{ai} = \frac{1}{1+e^{-(\sum_j w_{ij} \cdot y_{hi,j} + \theta_i)}}$.

En esta capa del controlador sólo existirá una única red neuronal pero su estructura no será única ya que dependiendo del número de tareas que el usuario

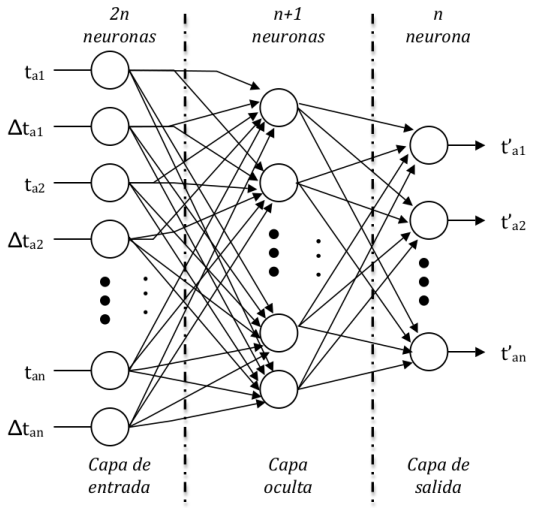


Figura 4.15: Arquitectura del coordinador.

quiere ejecutar la red tendrá más o menos salidas. Esta propiedad hace que por un lado sólo necesitemos una única red, pero por otro lado complica su evaluación ya que el número de situaciones posibles ante las que podemos encontrarnos es mayor. Para este PFC hemos considerado que el coordinador sea una red neuronal para un máximo 7 tareas, una tarea por cada uno de los electrodomésticos. En caso de que no se tengan las 7 tareas activas, las entradas sobrantes serán igual a cero. Pero además para poder recuperar la modularidad descrita, se han establecido unos interruptores, entre la capa oculta y la capa de salida, que inhiben sus conexiones. De esta manera las neuronas de la capa oculta, asignadas a aquellas tareas que no se van a coordinar, no interfieran en el cálculo de los tiempos de las tareas a coordinar.

Como en el planificador, los parámetros de la red neuronal se establecen a través de un fichero que se pasa como parámetro en la ejecución del experimento. Sin embargo este fichero difiere del anterior en que ya no se fija el número de neuronas de cada capa en el fichero sino que es el propio experimento el que las fija. El fichero de parámetros neuronales que usa el coordinador está formado por:

- *Límites de ganancia de entrada,* se establece un límite superior e inferior de ganancia que no se pueden superar. *Ej. INPUT GAIN LOWER BOUND =*

0.0; INPUT GAIN UPPER BOUND = 1.0.

- *Límites del parámetro de ajuste (bias),* se establece un límite superior e inferior al valor del parámetro de ajuste que no se puede sobrepasar. *Ej. BIAS LOWER BOUND = -5.0; BIAS UPPER BOUND = 5.0.*

- *Límites de los pesos,* se establece un límite superior e inferior al valor de los pesos de todas las neuronas que no se pueden superar. *Ej. WEIGHTS LOWER BOUND = -5.0; WEIGHTS UPPER BOUND = 5.0.*

4.4.3. El algoritmo genético

Los parámetros del algoritmo genético son:

- **Longitud del cromosoma,** el cromosoma estará formado por todos aquellos parámetros libres de la red neuronal, ganancias de entrada, sesgos (θ_i) y pesos neuronales. La longitud del cromosoma viene expresada por la ecuación: $chromosomeLength = N\,g_{in,i} + N\,\theta_{hi,i} + N\,\theta_{out,i} + N\,w_{hi,ij} + N\,w_{out,ij}$. Para este caso podemos expresar la longitud del cromosoma en función del número de tareas que se vayan a planificar: $chromosomeLength = 2 \cdot n + (n+1) + (2 \cdot n \cdot (n+1)) + ((n+1) \cdot n) = 3 \cdot n^2 + 7 \cdot n + 1$, siendo n el número de tareas. Para el experimento se ha realizado el estudio con 7 tareas distintas, entonces la longitud del cromosoma sería igual $3 \cdot 7^2 + 7 \cdot 7 + 1 = 197$.

- **Tamaño de la población,** con este parámetro se fija el número de individuos que forman una población.

- **Número de generaciones,** se indica cuantas generaciones como máximo va a evolucionar el algoritmo genético.

- **Tiempo de evaluación,** tiempo que se tardará en evaluar un cromosoma de un individuo.

- **Activar crossover,** permite hacer crossover o no.

- **Número de crossovers,** cuántos crossovers se pueden realizar por generación.
- **Distancia del crossover,** número de puntos donde realizar el crossover.
- **Ratio de mutación,** la probabilidad de que un gen al crear una nueva generación se mute.
- **Número de élites,** indica el número de cromosomas que se copiarán y pasarán a la generación siguiente.
- **Función de coste (fitness),** seleccionamos la función que evaluará los distintos individuos de la población.

El fichero de parámetros genéticos que se ha utilizado para el experimento es:

CHROMOSOME LENGTH = *49*
POPULATION SIZE = *100*
NUMBER OF GENERATIONS = *10000*
EVALUATION TIME = *0*
DO CROSSOVER (0 No, 1 Yes) = *1*
NUMBER OF CROSSOVERS (Always 1) = *1*
CROSSOVER DISTANCE (Always 1) = *1*
MUTATION RATE = *0.05*
NUMBER OF ELITES = *5*
FITNESS FUNCTION = *2*

La función de coste que se ha desarrollado está compuesta por dos funciones que contribuyen a evaluar a un individuo: una para medir la similitud entre el tiempo de la salida coordinada y el tiempo de la entrada planificada y la otra función evalúa el solapamiento entre las salidas coordinadas. La función de coste evalúa un individuo de la siguiente manera:

- Primero busca cual de las tareas tiene el mayor tiempo de ejecución (Δt_{max}).
- Acto seguido compara las salidas iterativamente y evalúa la separación existente entre las diferentes tareas coordinadas según la siguiente función (ver Figura

4.16(a)):

$$fitness_{1,i} = \begin{cases} \frac{1}{\Delta t_{max}} \cdot |t| & \text{si } 0 \leq |t| < \frac{\Delta t_{max}}{4}, \\ \frac{3}{\Delta t_{max}} \cdot |t| - \frac{1}{2} & \text{si } \frac{\Delta t_{max}}{4} \leq |t| < \frac{\Delta t_{max}}{2}, \\ 1 & \text{si } \frac{\Delta t_{max}}{2} \leq |t|. \end{cases}$$

Siendo t la diferencia entre los tiempos de salida de la red ($t_{out,i} - t_{out,j}$). El valor de coste se acumula para contener toda la información de las comparaciones.

- Después se compara la relación entre las entradas y las salidas de este modo estamos garantizando que si no solapa con ninguna otra tarea se mantenga lo que se planificó. La evaluación consiste en (ver Figura 4.16(b)):

$$fitness_{2,i} = \begin{cases} -\frac{3}{2 \cdot \Delta t_{max}} \cdot |t| + 1 & \text{si } 0 \leq |t| < \frac{\Delta t_{max}}{2}, \\ -\frac{1}{2 \cdot \Delta t_{max}} \cdot |t| + \frac{1}{2} & \text{si } \frac{\Delta t_{max}}{2} \leq |t| < \Delta t_{max}, \\ 0 & \text{si } \Delta t_{max} \leq |t|. \end{cases}$$

Siendo t la diferencia entre los tiempos de salida y entrada de la red ($t_{in,i} - t_{out,i}$). El valor de coste se acumula para contener toda la información de las comparaciones.

- El valor de coste del individuo para la situación que se está evaluando sería la multiplicación de los dos costes obtenidos ($fitness_{1,i}$ y $fitness_{2,i}$) pero ponderados sobre el número de salidas de la red, es decir, el valor de coste total del individuo finalmente sería:
$fitness_{tot} = \left(\frac{1}{\sum_{i=1}^{n-1} i} \cdot \sum_{i=1}^{n-1} fitness_{1,i}\right) \cdot \left(\frac{1}{n} \cdot \sum_{i=1}^{n-1} fitness_{2,i}\right)$.

- Cada individuo es evaluado para un número determinado de situaciones. El valor de coste (*fitness*) del individuo será la media de los valores de coste para cada una de las situaciones, es decir, $fit_{ind} = \frac{\sum_n (fit_n)}{n}$.

- Se evalúan todos los individuos de esa generación.

- Finalmente el valor de coste de la generación será la media aritmética del coste acumulado, es decir, $fit_{gen} = \frac{\sum_i fit_{ind,i}}{n_{ind}}$.

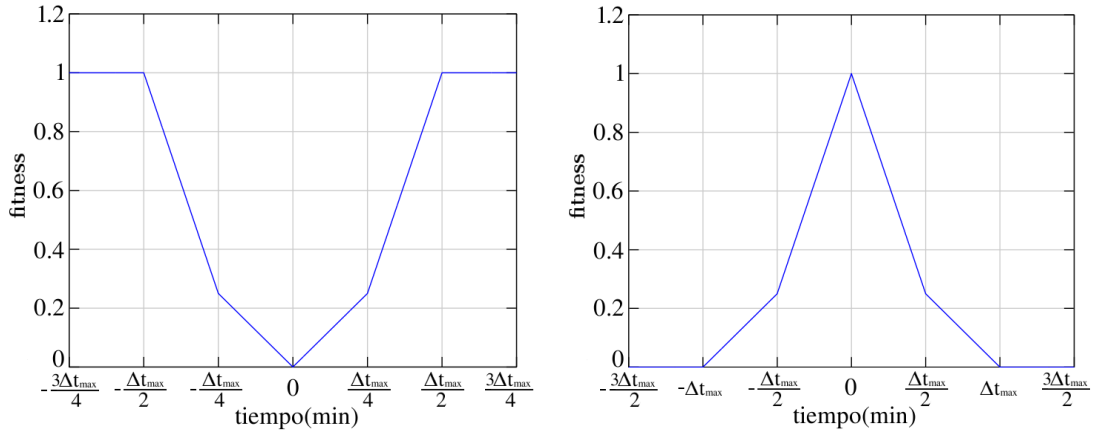


Figura 4.16: Función de fitness (a) comparación entre salidas y (b) comparación entre entrada y salida.

Aunque ambas gráficas pueden parecer opuestas, se debe a que sus objetivos también son contrapuestos ya que para dos tareas que tengan el mismo tiempo planificado es imposible que ambas mantengan sus tiempos coordinados iguales a los planificados y no se solapen, sólo una de ellas podría mantenerlo la otra debe cambiarlo por eso el coste total es la multiplicación de ambas.

4.4.4. Evolución

Para la evolución de la red se han utilizado como entradas las siete duraciones de las tareas (Δt_1, Δt_2, ..., Δt_7), mientras que los tiempos de planificación (t_1, t_2, ..., t_7) varían desde las 10 a.m hasta 8 p.m., los tiempos de planificación varían de una hora en una hora (ver Figura 4.17, para un ejemplo de tres tareas). La evolución de un individuo cuenta con todos los tiempos de planificación, es decir, que su cromosoma se evalúa 10^7 *veces* y cada generación entonces contaría con $10^7 * 100$ *individuos* $= 10^9$ *iteraciones*. La duración total del experimento sería de $10^9 * 10^4$ *generaciones* $= 10^{13}$ *iteraciones*, con lo que el tiempo de simulación aunque menor que la evolución del planificador también tiene una larga duración.

En esta evolución hemos ejecutado todo de una vez, es decir, hemos dejado correr

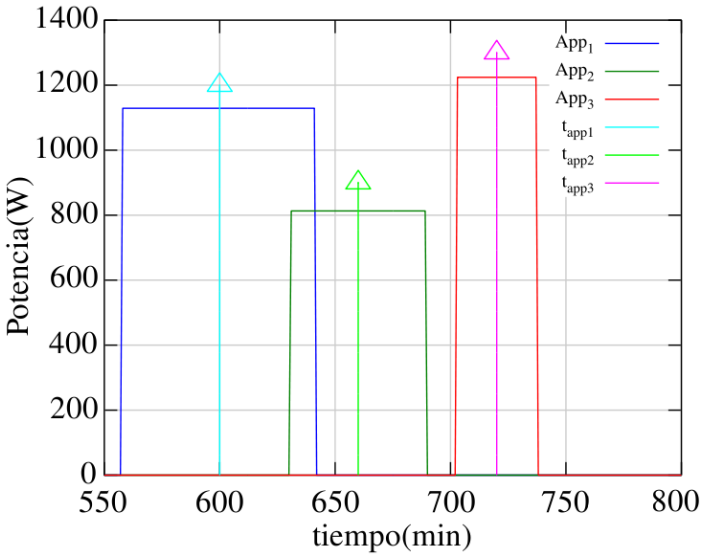
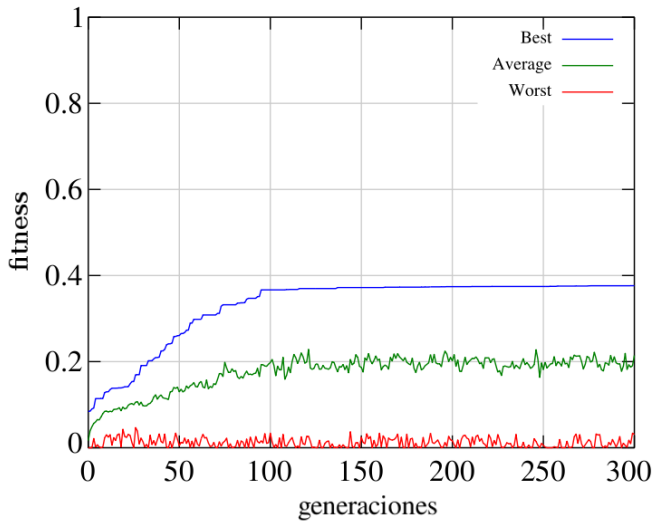
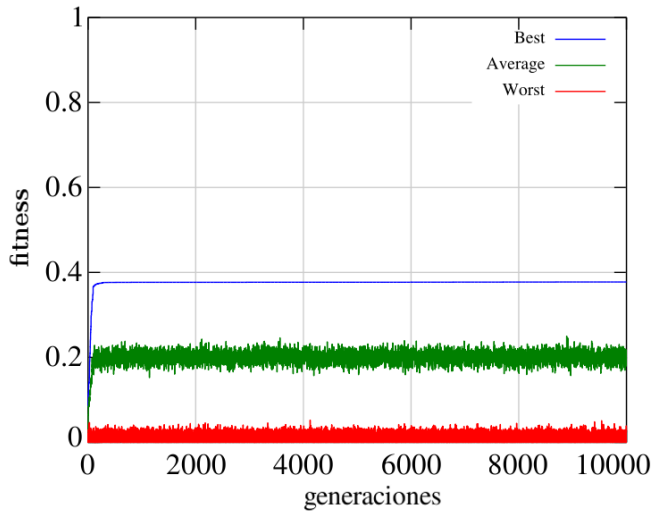


Figura 4.17: Ejemplo de entradas para 3 tareas distintas.

el experimento con todas las iteraciones desde el principio hasta llegar a las 10.000 generaciones. La evolución ha consistido en variar los tiempos planificados de hora en hora entre los límites establecidos, mientras que la duración de las tareas se han mantenido constantes, así pues se ha ido evaluando individuo por individuo si las salidas cumplían el objetivo de no estar solapadas y parecerse lo máximo posible a las entradas. De esta forma el máximo valor de coste que se ha conseguido a lo largo de las 10.000 generaciones ha sido de 0,3774463950 (ver Figura 4.18).



(a)



(b)

Figura 4.18: Resultados de la evolución: (a) 200 generaciones y (b) 10000 generaciones.

Aunque el valor de evaluación no sea tan alto como en la evolución del planificador, hay que recordar que en esta evolución estamos trabajando con dos funciones de coste y el coste total es la media geométrica de las anteriores. Además en caso de que haya solapamiento no se va a poder mantener que se parezca la salida a la entrada y se sigan solapando las tareas, en estos casos el coste será muy pequeño. También hay

que resaltar que en muy pocas generaciones llegamos a un valor cercano al máximo de la evolución.

4.4.5. Análisis de la evolución

Acabada la evolución queda analizar si realmente el coordinador cumple el objetivo para el que se diseñó. Después del entrenamiento el coordinador debería ser capaz de separar todas las tareas que solapasen y las que no dejarlas en la posición de entrada, pero se han observados comportamientos no del todo correctos:

- En el 92 % de los casos se resuelve correctamente las situaciones solapadas. En un 3.5 % de los casos se desvía menos de 10 minutos del tiempo de restricción del usuario y las tareas no se encuentran solapadas, mientras que en un 1.5 % de los casos se desvía más de 30 minutos del intervalo de preferencia del usuario y sin solapamiento. Sólo en el 3 % de las situaciones se produce un solapamiento bajo, de menos de 10 minutos.

- En los casos en las que las tareas no se encuentran solapadas se mantiene el tiempo planificado sin variarlo en un 95 %. El 5 % restante varía el tiempo en menos de 15 minutos.

- Las tareas mantienen espacios de tiempo entre ellas, se debe a que en la evolución una vez que las tareas solapadas estén separadas más de $\frac{\Delta t_{max}}{2}$ se da un valor alto de coste al individuo. Esto podría arreglarse reintroduciendo el perfil de potencia fotovoltaica de tal modo que se penalizara si quedara demasiado separado.

En general, el coordinador neuronal solventa el problema que surgió, como consecuencia de la planificación neuronal, de manera satisfactoria. Se ha conseguido separar todas las tareas solapadas y que se respeten las preferencias del usuario en la mayoría (95 %) de los casos.

En la Figura 4.19 tenemos dos ejemplos del resultado del coordinador, pero sólo para tres tareas de este modo será más sencillo observar el funcionamiento del coordinador. En la Figura 4.19(a) primero se encuentran tres tareas solapadas y se

introducen en el coordinador, éste responde con las salidas mostradas en la Figura 4.19(b). Como podemos comprobar todas las tareas han quedado separadas y se ha mantenido una sobre el tiempo planificado, el resto han sido desplazadas en eje temporal cumpliendo que no hay solapamiento. El caso de la Figura 4.19(c) es un caso favorable de entradas planificadas al coordinador. Como vemos en la Figura 4.19(d) el coordinador desplaza algunos minutos el tiempo planificado pero no sufre ninguna variación con la entrada. Con este ejemplo vemos que si no están solapadas las entradas se mantienen.

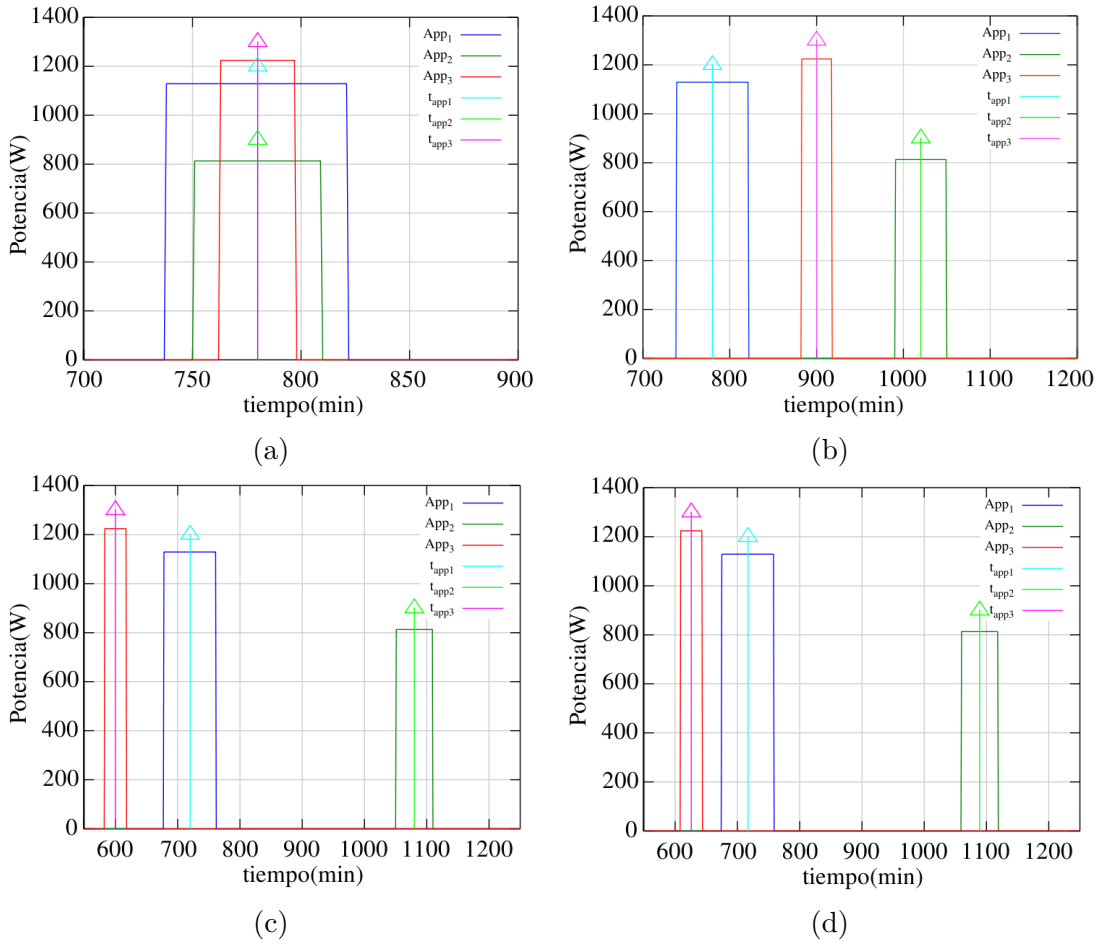


Figura 4.19: Resultados de la ejecución: (a) y (c) entradas al coordinador y (b) y (d) salidas del coordinador.

Estos han sido los resultados obtenidos con las situaciones con que fue entrenado el coordinador, en la Sección 4.5.1 realizaremos un análisis del coordinador con situaciones no entrenadas.

4.5. Sistema Completo

Como se ha visto al principio de este Capítulo el controlador está compuesto por una capa de planificación, una capa de coordinación y una capa de actuación. Para ejecutar el sistema completo es necesario indicar a las distintas redes neuronales de donde obtener los cromosomas, indicar la zona donde ofrecer y obtener los datos para las diferentes capas. De este modo, se ejecutaría el sistema completo y obtendríamos el tiempo ejecución de las tareas. Para ello se ha realizado un programa que cubre estas funcionalidades que mostramos mediante el siguiente ejemplo.

Primero el usuario establece las tareas a realizar con los intervalos de tiempo pedidos (ver Figura 4.20(a)), a través del fichero de parámetros:

- Tarea 1: una lavadora entre las 9 a.m. y 4 p.m., con un programa a 90^{o}C y 1200 rpm.
- Tarea 2: una secadora entre las 12 a.m. y las 7 p.m., con un programa a 1200 rpm.
- Tarea 3: un lavavajillas entre las 8 a.m. y las 11 a.m., con el programa de ejecución 5.

Esta información pasa al planificador y se obtienen las planificaciones (ver Figura 4.20(b)). Para el ejemplo propuesto obtenemos $t_{app1} = t_{app2} = 13\,horas$ y $t_{app3} = 10\,horas$. Estas salidas pasan a la capa de coordinación que trata de eliminar los solapamientos (ver Figura 4.20(c)). Las salidas que obtiene son: $t_{coord1} \simeq 13\,horas$, $t_{coord2} \simeq 15{,}5\,horas$ y $t_{coord3} \simeq 10\,horas$. Finalmente estos resultados se ejecutarían en la simulación utilizando las funciones de los electrodomésticos, descritas en el Capítulo 3, *JobStart* y *JobStop* para el inicio y fin de la tarea.

Para concluir, este es un ejemplo realizado con la aplicación diseñada para este Proyecto Fin de Carrera. Podemos observar que se han respetado las restricciones

temporales impuestas por el usuario. Podemos comprobar que el objetivo de maximizar el autoconsumo que se perseguía se ha visto cumplido ya que las tareas se encuentran en su mayoría por debajo de las potencias de generación predichas. También se puede observar como el coordinador separa las tareas. Aunque la separación entre ellas no es demasiado grande, el coordinador debería establecer tiempos aun más próximos.

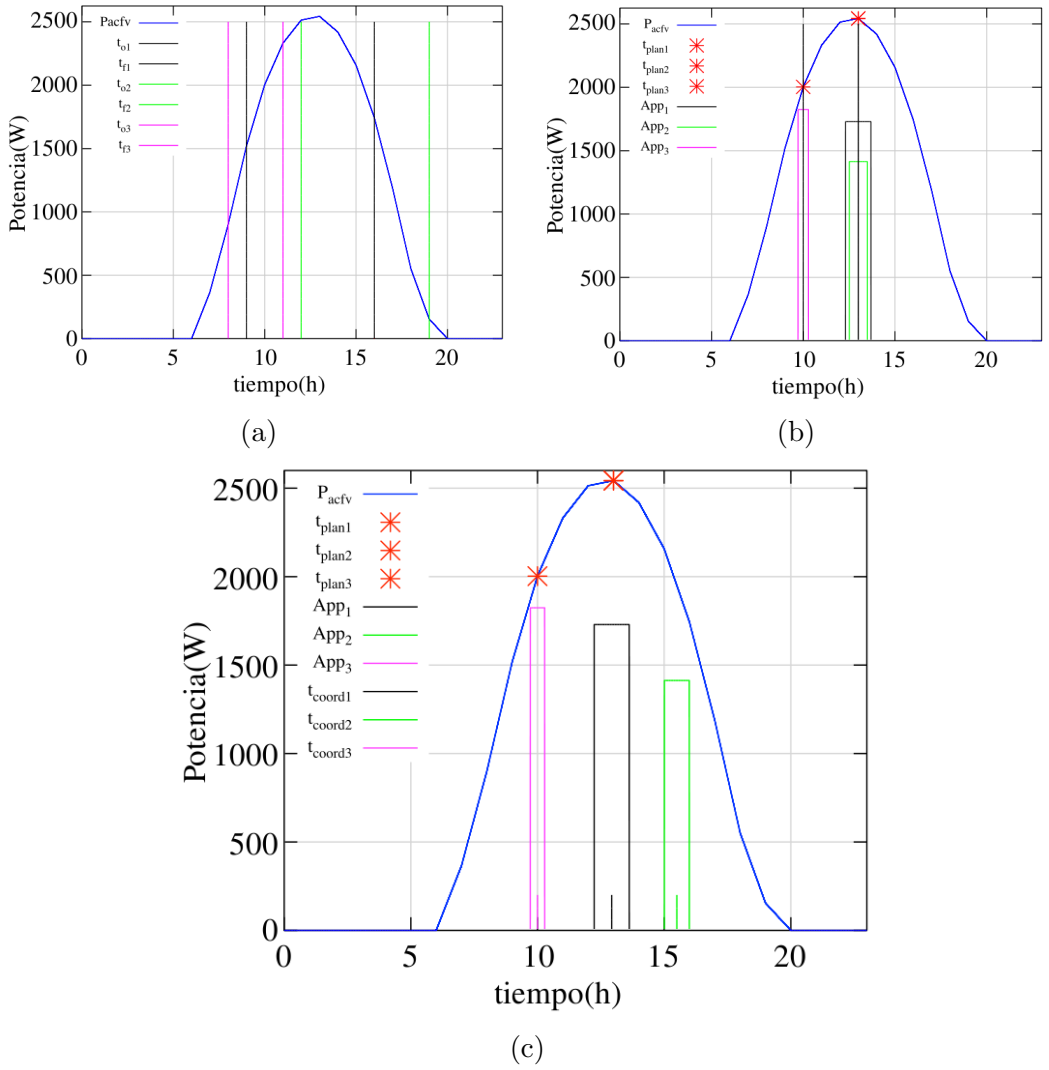


Figura 4.20: Ejemplo sistema completo: (a) entrada, (b) planificación y (c) coordinación.

4.5.1. Postevaluación

Para evaluar el sistema completo hemos tomado el resto de perfiles para completar un año de simulación, es decir, se han utilizado 223 perfiles fotovoltaicos que abarcan el resto de estaciones del año así como distintos fenómenos meteorológicos. Igualmente en el vector de entradas se han definido 3 tareas con 11 intervalos temporales de 4

horas cada una. Los resultados obtenidos son:

- En la capa de planificación se ha acertado al encontrar el máximo en un 87 % de los casos, un 10 % se desvía sólo una hora de encontrar el máximo y el 3 % restante se desvía en más de una hora.

- En el caso del coordinador se han separado las tareas y respetado el intervalo de trabajo impuesto por el usuario en un 89 % de los casos. Sólo en un 7 % se ha desviado del intervalo de usuario en menos de 10 minutos y en el 4 % restante se ha desviado del intervalo de usuario en más de 30 minutos.

- Por tanto, el sistema completo se ha ejecutado correctamente en el 85 % de las situaciones. El 15 % contempla fallos en el planificador, en el coordinador o en ambos.

A continuación ilustraremos con dos ejemplos los casos peor (ver Figura 4.21) y mejor (ver Figura 4.22) de ejecución del sistema para las tres tareas tipo. En ambos casos se trata de establecer el tiempo de ejecución de las siguientes tareas:

- Tarea 1: una lavadora entre las 12 a.m. y 4 p.m., con un programa a 90^{o}C y 1200 rpm.

- Tarea 2: una secadora entre las 10 a.m. y las 7 p.m., con un programa a 1200 rpm.

- Tarea 3: un lavavajillas entre las 8 a.m. y las 8 p.m., con el programa de ejecución 5.

En la Figura 4.21 todo es desfavorable. El perfil de predicción de potencia generada no es parecido al típico, lo que entraña problemas para localizar su máximo, por lo que se ha fallado en la planificación. Pero además tampoco se ha acertado con la coordinación y nos hemos salido de la curva de genración. Mientras que en la Figura 4.22 estamos ante uno de los mejores casos en el que se ha establecido correctamente tanto la planificación como la coordinación. Todas las tareas se encuentran por debajo

de la potencia predicha y se ejecuta una tarea cada vez por lo que estamos cumpliendo el objetivo que nos hemos marcado de maximizar el autoconsumo.

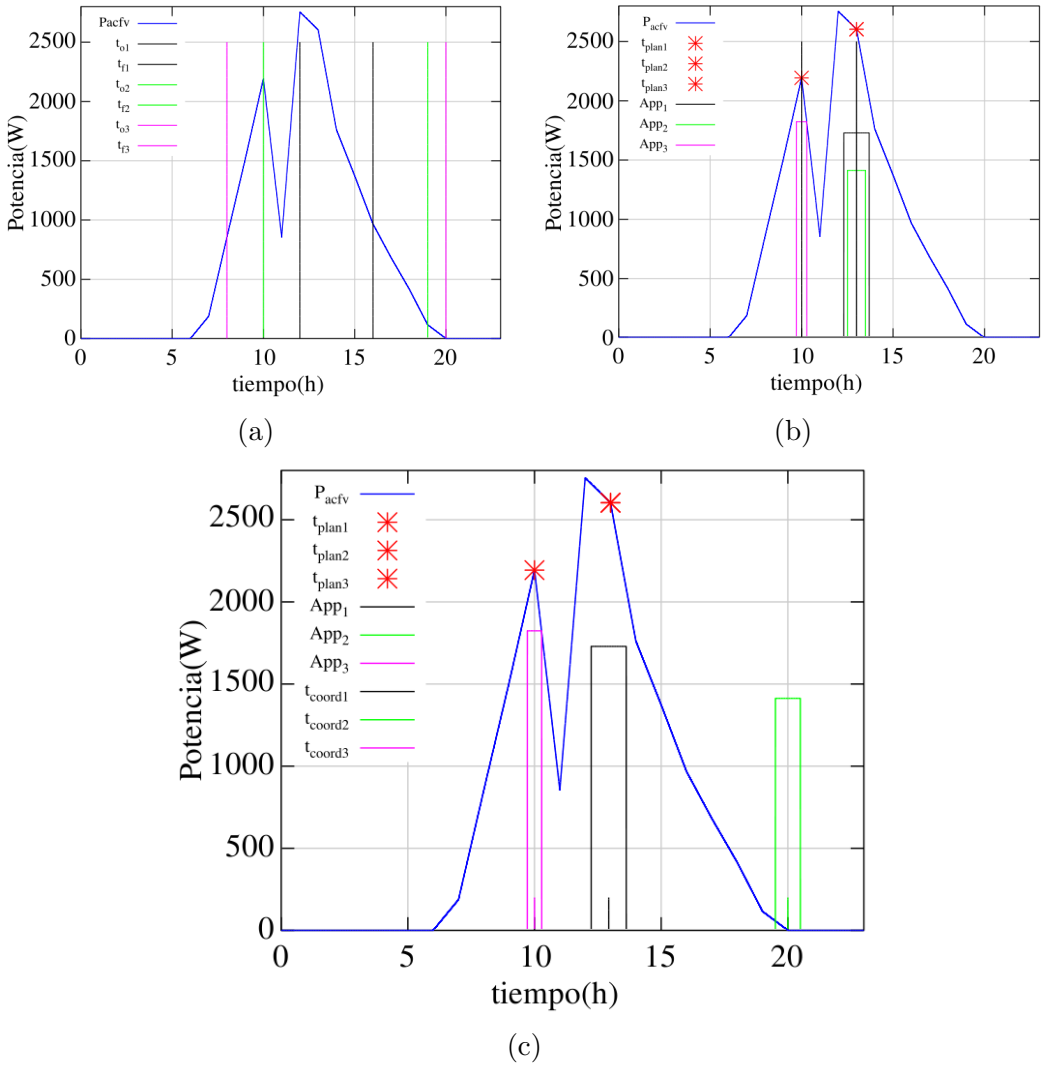


Figura 4.21: Ejemplo sistema completo caso peor: (a) entrada, (b) planificación y (c) coordinación.

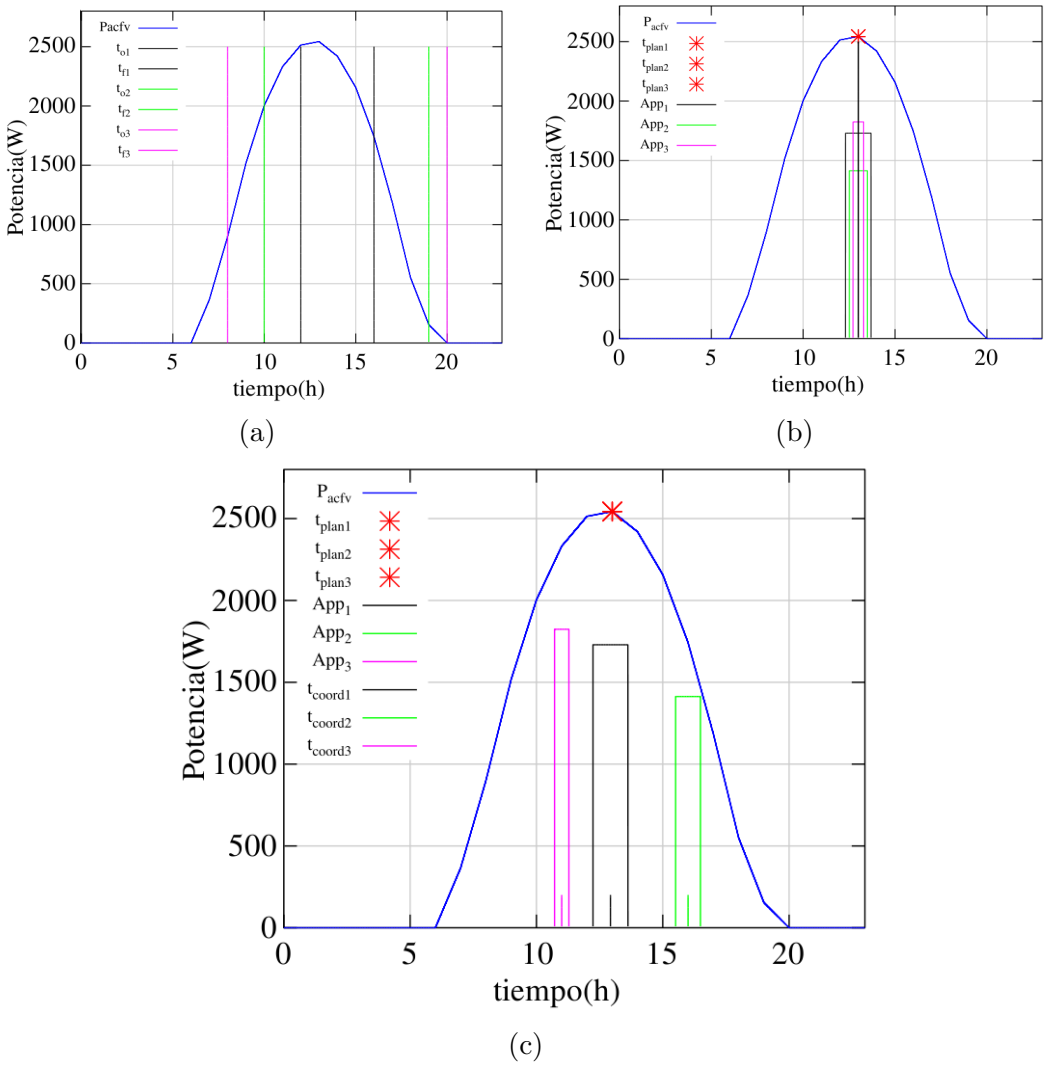


Figura 4.22: Ejemplo sistema completo caso mejor: (a) entrada, (b) planificación y (c) coordinación.

A la vista de los resultados podemos concluir que el controlador neuronal funciona correctamente en la mayoría de los casos. Aunque los fallos que se han encontrado podrían solventarse utilizándose técnicas más complejas (ver Capítulo 5).

Capítulo 5

Conclusiones y líneas futuras

5.1. Conclusiones

En el presente Proyecto Fin de Carrera se ha desarrollado un sistema de control mediante redes neuronales y algoritmos genéticos para la Gestión de la Demanda Eléctrica para viviendas del sector residencial. Se ha buscado la eficiencia energética del usuario mediante planificaciones que tratan de encontrar la máxima eficiencia dentro de la curva de generación fotovoltaica y coordinaciones que reparten finalmente sobre el eje temporal el momento de puesta en marcha de las cargas, para evitar un solapamiento de las tareas.

A la vista de lo desarrollado a lo largo de esta memoria se puede concluir que:

- Se ha logrado usar algoritmos provenientes de la inteligencia artificial para la optimización de la curva de la demanda. Aunque se han utilizado algoritmos básicos, éstos han sido introducidos en un nuevo territorio con muchas posibilidades de exploración y se irán complicado a medida que avance la investigación en este campo.

- El desarrollo de un sistema distribuido, a pesar de las complicaciones que ha entrañado, significa que sistemas de este tipo pueden satisfacer las necesidades del sector residencial sin tener que recurrir a sistemas centralizados.

- La visión conexionista, que se ha querido desarrollar, quizás aún no sea rival para

la visión cognitivista, pero precisa mucha menos información y no requiere tantas entradas del exterior. Con poca información el sistema es capaz de establecer un buen plan de acción.

- La planificación neuronal ha respondido satisfactoriamente ya que ha sido capaz establecer un tiempo óptimo determinado para el cual el usuario está sacando el máximo partido a la energía generada en detrimento de la energía eléctrica proveniente de la red. A pesar de su estructura compleja, el sistema de control cumple con los objetivos marcados en la mayoría de los casos.

- La coordinación neuronal se ha encargado de subsanar algunas deficiencias de la capa anterior pero también han surgido problemas en el reparto de las tareas a lo largo del eje temporal, ya que las tareas pueden llegar a separarse demasiado en el eje temporal. A pesar de este inconveniente el resultado ha sido muy positivo ya que se ha logrado establecer tiempos de ejecución de las tareas que han conseguido maximizar el autoconsumo.

- Las simulaciones han revelado que el controlador desarrollado es capaz de realizar una Gestión de la Demanda Eléctrica que satisface las demandas del usuario y planificar las tareas del día siguiente de forma eficiente energéticamente.

- A la vista de los resultados obtenidos queda más claro que el usuario tiene un gran papel dentro de la industria energética y puede ser un gran activo para el avance de ésta gestionando su consumo.

En resumen, el desarrollo de este Proyecto Fin de Carrera nos ha permitido demostrar que la Gestión de la Demanda Eléctrica provoca una mejora considerable en la eficiencia energética de una vivienda cuyas cargas de trabajo sean programables temporalmente.

5.2. Líneas futuras

Este PFC, como ya se mencionó en el Capítulo 1, se encuentra dentro del proyecto GeDELOS-FV con terminación en Septiembre del 2010. Aunque termine este proyecto se espera que existan futuras ediciones en las que se puedan desarrollar con más profundidad los algoritmos que se han tratado en este Proyecto Fin de Carrera, además de otros sistemas de control desarrollados para GeDELOS-FV.

Otra línea de desarrollo es la coordinación de diferentes sistemas de control permitiendo que varias viviendas compartan recursos. El objetivo debe ser que el balance energético del conjunto sea el más eficiente posible buscando un mejor aprovechamiento de la energía solar entre todas las viviendas.

La variación de los parámetros de los algoritmos genéticos es otra línea de desarrollo a tener en cuenta, como el uso de operadores genéticos más complejos o funciones de coste que expliciten mejor las preferencias del autoconsumo.

El estudio en profundidad de funciones de optimización más avanzadas se perfila como otra línea de desarrollo futura. Se tratará de complicar los algoritmos aquí utilizados para poder realizar mejores planificaciones y llevar a cabo nuevas investigaciones con dichos algoritmos.

Otra línea de investigación tiene que ver con la aplicación de los algoritmos aquí descritos. Dentro del ámbito eléctrico en un futuro podrían utilizarse dentro de la red eléctrica para buscar la eficiencia o en la generación fotovoltaica para conectar o desconectar un generador fotovoltaico.

5.3. Contribuciones

Este Proyecto Fin de Carrera describe investigaciones originales llevadas a cabo por el autor. Todos los resultados expuestos en el presente proyecto estarán recogidos próximamente en un artículo original llevado a cabo por el autor y el grupo de investigadores que están involucrados dentro del proyecto nacional GeDELOS-FV.

Bibliografía

Aguilar, L. J. (2002). *Programación en C++: Algoritmos, estructuras de datos y objetos*. Mc GrawHill, Madrid, España.

Bear, M. F., Paradiso, M. A., and Connors, B. W. (2007). *Neuroscience: Exploring the Brain, Third Edition*. Lippincott Williams and Wilkins, USA.

Bennett, S. (1979). *A History of Control Engineering 1800-1930*. London: Peregrinus on behalf of the Institution of Electrical Engineers, London, UK.

Caamaño-Martín, E., Castillo, M., Masa, D., Gutiérrez, A., Monasterio-Huelin, F., and Jiménez-Leube, J. (2010). Self-consumption of pv electricity with active demand side management: the gedelos-pv system. In *Proceedings of the 25th European Photovoltaic Solar Energy Conference*, Valencia, Spain. WIP-Renewable Energies. In press.

Caamaño-Martín, E., Masa, D., Gutiérrez, A., Monasterio-Huelin, F., Castillo, M., Jiménez, J., and Porro, J. (2009a). Optimizing pv use through active demand side management. In *Proceedings of the 24th European Photovoltaic Solar Energy Conference*, pages 3149–3155, Munich, Germany. WIP-Renewable Energies.

Caamaño-Martín, E., Masa, D., Gutiérrez, A., Monasterio-Huelin, F., Jiménez-Leube, J., Porro, J., and Castillo, M. (2009b). Optimización del uso de un sistema fotovoltaico mediante gestión activa de la demanda. In *I Congreso de Generación Distribuida*, pages 121–126, Madrid, Spain. Fundación de la Energía de la Comunidad de Madrid.

Castillo, M. (2009). *Diseño e Implementación de un Sistema de Gestión de la Demanda Eléctrica en Viviendas Inteligentes, Proyecto Fin de Carrera.* UMA-UPM, Málaga, España.

Castillo, M., Gutiérrez, A., Monasterio-Huelin, F., Masa, D., Caamaño-Martín, E., Jiménez-Leube, J., and Porro, J. (2009). Sistema de control distribuido para la gestión de la demanda en el sector residencial. In *I Congreso de Generación Distribuida*, pages 275–280, Madrid, Spain. Fundación de la Energía de la Comunidad de Madrid.

CCEE (2002). Towards a eu strategy for security of energy supply. Technical Report (2002)321, CEC, Brussels, Belgium.

Degner, T., Schmid, J., and Strauss, P. (2006). Dispower - distributed generation with high penetration of renewable energy sources. Technical Report ENK-CT-2001-00522, ISET, Kassel, Germany.

D.R.E.D. (2007). El consumo eléctrico en el mercado peninsular. informes de los años 1998 a 2006. Technical Report CNE PA004/08, Comisión Nacional de la Energía, España.

Golberg, D. E. (2006). *Genetic Algorithms in Search, Optimization and Machine Learning.* Addison Wesley, Indiana, USA.

Graham, S. and Kumar, P. R. (2003). The convergence of control, communication and computation? *Lecture Notes in Computer Science*, 2775:458–475.

Groppi, F. (2002). Grid connected photovoltaic power systems: power value and capacity value of pv systems. Technical Report IEA PVPS T5-11: 2002, CESI, Italy.

Haykin, S. (2009). *Neuronal Networks and Learning Machines, Third Edition.* Pearson International Edition, New Jersey, USA.

Hopfield, J. J. (1982). Neural networks & physical systems with emergent colective computational abilities. *Proceedings of the National Academy of Sciences*, 79:2554–2558.

Imaz, L. (2007). *La contribución de la gestión de la demanda ala eficiencia del sistema eléctrico*. Jornadas Internacionales de Equipos Eléctricos, Bilbao, España.

Klug, W. S., Cummings, M. R., and Spencer, C. A. (2006). *Conceptos de Genética, Octava Edición*. Pearson Education, Madrid, España.

Liang, J. and Du, R. (2008). Design of intelligent comfort control system with human learning and minimun power control strategies. *Energy Conversion and Management*, 49(ISSN 0196-8904):517–528.

Linder, S. (2007). *Solar Urban Planning - PV in Urban Environment: Germany*. Jornadas Aplicación de la Energía Solar Fotovoltaica desde el Urbanismo, E.T.S. Arquitectura U.P.M., Madrid, España.

Martín, B. and Sanz, A. (2001). *Redes Neuronales y Sistemas Borrosos, Segunda Edición*. Ra-Ma, Madrid, España.

Moon, S., Nahan, R., Warner, C., and Wassmer, M. (2005). Solar decathlon 2005: The event in review. Technical Report DOE/GO-102006-2328, National Renewable Energy Laboratory, U.S.A.

Porro, J. (2010). *Desarrollo de un sistema de monitorización y gestión de la demanda eléctrica en sistemas fotovoltaicos conectados a la red, Proyecto Fin de Carrera*. UPM, Madrid, España.

Purves, D. (2006). *Neurociencia, Tercera Edición*. Editorial Médica Panamericana, Madrid, España.

Pérez, J. I., Sánchez, L. J., and Pardo, M. (2005). *La gestión de la demanda de electricidad*. Ed. Fundación Alternativas, Madrid, España.

R.E.E. (1998). *Atlas de la demanda eléctrica en España. Proyecto INDEL.* Ed. Red eléctrica de España, España.

R.E.E. (2006). *El sistema electrico español 2006.* Ed. Red eléctrica de España, España.

Romero, C. (2004). *Domótica e Inmótica. Vivienda y Edificios Inteligentes.* Ra - Ma, Madrid, España.

S.G.P.E. (2007). Planificación de los sectores de electricidad y gas 2008-2016. Technical report, Ministerio de Industria, Turismo y Comercio, España.

Tecnalia, L. (2007). *Guia básica de la gestión de la demanda eléctrica.* Fundación de la energía de la comunidad de Madrid, Madrid, España.

Valavanis, K. P. and Saridis, G. N. (1992). *Intelligent Robotic Systems: Theory, Design and Applications.* Kluwer Academic Publishers, USA.

Valentí, J. I. F. (2007). *Eficiencia energética en redes eléctricas desde la perpectiva de las empresas distribuidoras.* Jornadas de la Cátedra Endesa Red de Innovación Energética, Barcelona, España.

Vasilakos, A. and Pedriyzc, W. (2006). *Ambient Intelligence, wireless networking and ubiquitous computing.* Artech House Publishers, USA.