VNIVERSITAT DE VALÈNCIA

Aplicación de Algoritmos Genéticos en Extreme Learning Machine

Application of Genetic Algorithms to Extreme Learning Machine

Proyecto Fin de Carrera

Iván Vallés Pérez

Directores

Emilio Soria-Olivas, PhD
Joan Vila-Francés, PhD

Índice general

Índice de figuras

Capítulo 1

Objetivo del proyecto.

ELM (*Extreme Learning Machine*) es un nuevo algoritmo para obtener los parámetros de una red neuronal monocapa. Tiene una serie de ventajas que han hecho que se utilice en un gran número de aplicaciones

La principal desventaja del algoritmo (*ELM*) es que sólo puede utilizar una función de coste cuadrática. El objetivo de este proyecto consiste en analizar el uso de los algoritmos genéticos para la obtención de los parámetros de salida de una *Extreme Learning Machine* (*ELM*) cuando se usan otras funciones de coste diferentes a la cuadrática. Se comprueba que, al utilizar los algoritmos genéticos, se obtiene una solución mejor que con la *ELM* clásica. Además, al utilizar la *ELM* con algoritmos genéticos, aumentan mucho sus posibilidades, ya que pueden utilizarse funciones de coste distintas a la cuadrática.

Capítulo 2

Introducción.

2.1. Minería de datos.

La minería de datos (*Data Mining*, en inglés) es también llamada análisis inteligente de datos o "descubrimiento de conocimiento en bases de datos" (por las siglas *KDD*, *Knowledge Discovery in Databases*). Se ha definido como "el proceso no trivial de identificación en los datos de patrones válidos, nuevos, potencialmente útiles, y finalmente comprensibles"[Fayyad et al., 1996].

Data Mining (DM) es un término relativamente nuevo, de finales del siglo pasado, que ha surgido gracias a cuatro factores: mejora en la calidad y cantidad de los sistemas de almacenamiento de datos, una mayor velocidad de procesamiento informático de los mismos, mejoras en la velocidad y confiabilidad de transmisión de datos y aparición de sistemas de administración de bases de datos más potentes [Félix, 2002]. A ello hay que añadir el avance en nuevos desarrollos de algoritmos matemáticos que hacen posible el análisis avanzado de datos (redes neuronales artificiales, algoritmos genéticos, etc.).

Pueden encontrarse antecedentes de *DM* en la década de los sesenta, los estadísticos hablaban en términos similares de *"data archeology"* o *"data fishing"* con la idea de encontrar correlaciones entre variables de una base de datos sin necesidad de formular ninguna hipótesis previa.

La cantidad de información a la que se puede acceder es inmensa, si se introduce en el buscador Google la palabra *información* aparecen aproximadamente 273.000.000 de sitios que se pueden visitar. Si se piensa dedicar tan solo un minuto a cada uno de ellos serían necesarios más de 500 años para lograrlo. El ejemplo expuesto justifica la necesidad del desarrollo de nuevas teorías que ayuden en la búsqueda y en el análisis de datos. Con la minería de datos se pueden abordar tres tipos de problemas [Fayyad et al., 1996]:

- Problemas de clasificación: se etiquetan los registros en determinadas clases y se crea un modelo que los clasifique bajo algún criterio.
- Problemas de predicción: se asignan unos valores ausentes en un campo, en función de los demás campos presentes en el registro o de los registros existentes.
- Problemas de segmentación: se fracciona el conjunto de los registros (población) en subpoblaciones de comportamiento similar.

En minería de datos se estudian conjuntos de datos con la intención de encontrar una, o varias, hipótesis que después serán validadas con más datos. El proceso general comienza definiendo el conjunto de datos a utilizar, seguidamente se analizan las propiedades estadísticas de éstos; esta parte se conoce como preprocesamiento. Posteriormente se escoge una técnica adecuada al problema en estudio y se aplica a los datos. Se obtiene así un modelado del sistema. El siguiente, y último, paso, consiste en interpretar los resultados para evaluar el modelo y así, aprobarlo o descartarlo.

Las técnicas usadas pertenecen a las áreas de la estadística y la inteligencia artificial. Entre ellas destacan las redes neuronales artificiales (modelos de proceso numérico en paralelo) y los algoritmos genéticos (métodos numéricos de optimización), ambas técnicas se tratan con más detalle en los próximos capítulos. Otros algoritmos de *DM* son la regresión lineal, árboles de decisión, técnicas de clustering o agrupamiento, etc. [Fayyad et al., 1996].

DM es una disciplina apasionante y cambiante, ya que puede conducir al descubrimiento de patrones previamente desconocidos en los datos para obtener una ventaja potencial no sólo en los negocios sino también en los problemas de la ciencia, la ingeniería y la salud. La minería de datos se aplica en múltiples campos, como [López and Herrero, 2006], [Félix, 2002]:

- *En negocios*: detección de clientes con probabilidad de compra, hábitos de compra en supermercados para saber qué y cuándo hacer ofertas, patrones de fuga que permiten conocer qué clientes pueden escapar y hacerles mejores ofertas, etc.
- *Seguridad de países*: vigilancia de datos comerciales de hábitos de compra de consumidores para la detección de posibles terroristas.
- *Física*: proyecto *SKYCAT* de catalogación de objetos estelares.
- *Universidad*: estudio de las actividades profesionales de los recién titulados.
- *Medicina*: la medicina traslacional, se trata de transferir los resultados científicos de los laboratorios a la práctica clínica para el diagnóstico y tratamiento del paciente. En genética se estudian cómo pueden afectar los cambios en la secuencia de ADN de un individuo frente al riesgo de sufrir determinadas enfermedades.

Y el número de aplicaciones sigue aumentando.

2.2. Redes Neuronales Artificiales.

Las Redes Neuronales Artificiales (RNA) son de gran utilidad en la minería de datos puesto que pueden encontrar relaciones no lineales entre conjuntos de datos por medio de algoritmos de aprendizaje basados en los datos de que se dispone, obteniendo modelos en problemas grandes y complejos [Haykin, 1998].

Una RNA es un modelo matemático inspirado en el comportamiento biológico de las neuronas y en la estructura del cerebro [Lin et al., 1996]. Se trata de un sistema inteligente que trabaja de modo distinto a los ordenadores actuales. Problemas aparentemente sencillos como el reconocimiento de un rostro conocido entre una multitud de ellos son de una enorme complejidad y necesitan de una gran cantidad de tiempo de computación para nuestros actuales ordenadores. En cambio, el cerebro humano puede realizar este trabajo en décimas de segundo y esto es debido a la elevadísima interconexión entre sus elementos más pequeños, las neuronas, y a la capacidad de operar en paralelo.

La neurona, unidad básica del cerebro, está compuesta por [Olabe, 1998]:

- El cuerpo central, llamado *soma*, que contiene el núcleo celular.
- Una prolongación del *soma*, el *axón*.
- Una ramificación terminal, las *dendritas*.
- Una zona de conexión con otras neuronas, conocida como *sinapsis*.

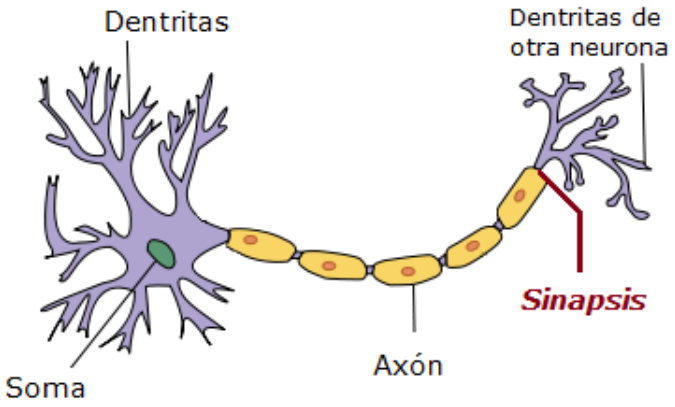

Figura 2.1: Esquema de una neurona biológica.

La función principal de las neuronas es la transmisión de los impulsos nerviosos que viajan por toda la neurona comenzando por las dendritas hasta llegar a las terminaciones del axón, donde pasan a otra neurona por medio de la conexión sináptica.

Se estima que el cerebro humano contiene de 50 a 100 mil millones (10^{11}) de neuronas que transmiten las señales a través de hasta 1000 billones (10^{15}) de conexiones sinápticas.

La manera en que respondemos ante los estímulos del mundo exterior y nuestro aprendizaje del mismo está directamente relacionada con las conexiones neuronales del cerebro siendo las RNA un intento de emular este hecho.

Las neuronas artificiales, o nodos, se comunican entre sí mediante unas conexiones llamadas pesos sinápticos [Serrano et al., 2009]. Los pesos variarán en función del aprendizaje siendo el aprendizaje el proceso por el cual se modifican los pesos sinápticos para obtener el resultado deseado.

Cada neurona artificial tiene una serie de entradas (x_i) conectadas a través de interconexiones (pesos sinapticos). En la figura 2.2 se ilustra el funcionamiento de una neurona:

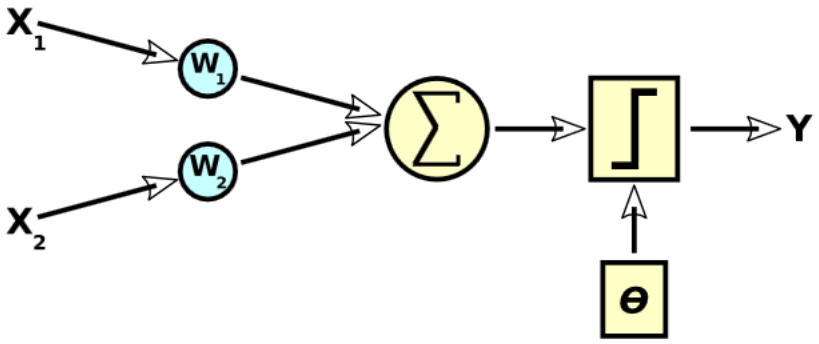

Figura 2.2: Esquema de una neurona artificial.

Las entradas (x_i) son multiplicados por los pesos sinápticos (w_i) que son los parámetros del sistema, de modo que el aprendizaje se reduce a obtener los valores de los pesos que mejor se ajusten a nuestro problema. Los pesos sinápticos w_i pueden ser excitativos ($w_i > 0$) o inhibidores ($w_i < 0$). Todas las entradas, multiplicadas por los pesos correspondientes, son sumadas. Se refleja la situación similar que se produce en los axones de las neuronas biológicas [Lahoz-Beltrà, 2004].

A la salida del sumador se aplica una función no lineal que se conoce como función de activación, pudiendo no existir. Existe un umbral θ que la neurona debe sobrepasar para activarse como ocurre en el cuerpo de la neurona biológica, siendo en este caso la salida la misma función de propagación. Entre las funciones de activación más usadas están [Serrano et al., 2009]:

- *Función Signo*: es la más sencilla y plantea una separación dura de los datos de entrada en 2 clases (1 para valores positivos y -1 para negativos).

- *Función Sigmoide*: se usa en los algoritmos que exigen tratar con funciones de activación diferenciables, se define como $\frac{a}{1+e^{-bx}}$, donde a es la amplitud del sigmoide y b la pendiente en el origen. Sus valores de salida están en el intervalo $[0, a]$.

- *Función Tangente Hiperbólica*: es una variante de la anterior y se usa cuando se necesitan resultados en un intervalo [-a,a]. Se define como a $\frac{1-e^{-bx}}{1+e^{-bx}}$.

La figura 2.3 muestra las tres funciones de activación comentadas.

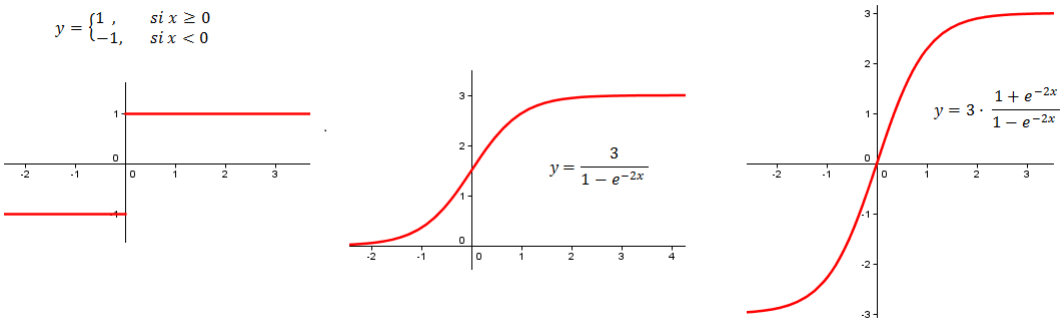

Figura 2.3: Funciones de activación más comunes.

Al conectar varias neuronas de un determinado modo, se consigue una red neuronal. Las RNA, además de ser útiles para la resolución de problemas que implican un elevado tiempo de computación si se abordan por métodos clásicos, presentan las siguientes ventajas [Haykin, 1998]:

- *Aprendizaje adaptativo*. Son capaces de aprender a desempeñar determinadas tareas después de un entrenamiento inicial.
- *Sistemas no lineales*. Al ser la neurona un elemento no lineal, puede abordar problemas no lineales que no se pueden tratar con los métodos clásicos lineales.
- *Autoorganización*. Pueden crear su propia organización de la información que recibe mediante una etapa de aprendizaje.
- *Tolerancia a fallos*. Al almacenar la información de forma redundante, la RNA puede seguir trabajando aceptablemente aun si resulta dañada.
- *Flexibilidad*. Admiten cambios no importantes en la información de entrada (por ejemplo, si la información de entrada es una imagen, la respuesta correspondiente no sufre cambios si la imagen cambia un poco su brillo o hay un ligero cambio de posición).

- *Fácil inserción dentro de la tecnología existente.* Admiten implementación *VLSI* simulando elementos biológicos con chips de silicio dando respuestas en tiempo real.

Existen muchos tipos de RNA siendo el Perceptrón Multicapa o *MLP* (*Multi-Layer Perceptron*) uno de las más conocidas y con mayor número de aplicaciones.

El *MLP* surge de la dificultad que presentan las RNA anteriores a él en la clasificación de conjuntos de datos, solo eran capaces de actuar con regiones linealmente separables [Serrano et al., 2009]. La solución está en la inclusión de capas de neuronas entre las capas de entrada y salida (capas ocultas). Aparece con ello el problema asignar un "error" al proceso de aprendizaje de estas neuronas ocultas.

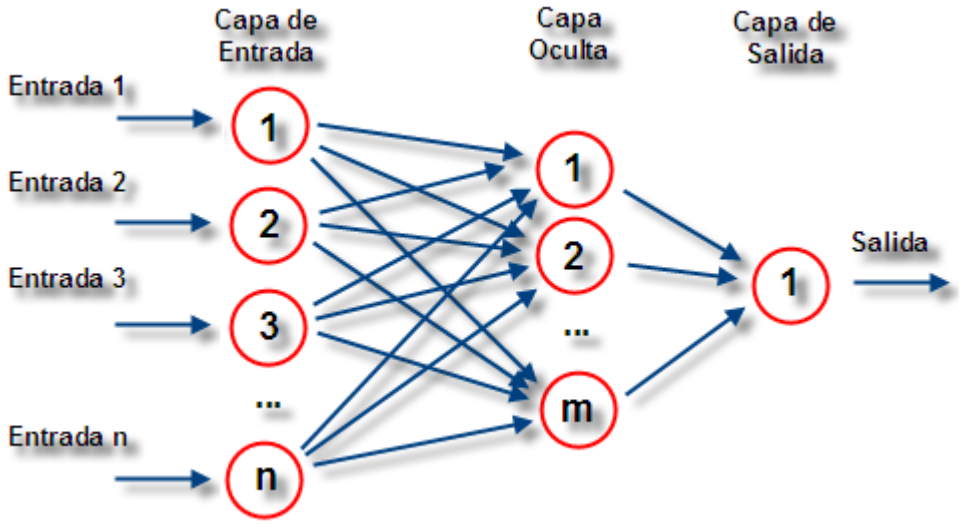

Figura 2.4: Perceptrón multicapa.

Las características principales del MLP están en que se trata de una estructura altamente no lineal, tolerante a fallos, capaz de establecer relaciones entre dos conjuntos de datos y que admite una implementación *hardware*.

Así como el número de neuronas de las capas de entrada y salida vienen determinados por el problema que se esté tratando, el número de capas ocultas y el de neuronas en cada una de ellas no está determinado *a priori*, siendo el diseñador de la red quien ha de decidirlo. Sólo está demostrado que, para un conjunto de datos conexo, es suficiente

con una sola capa oculta, aunque el número de neuronas de ésta no esté determinado, y para conjuntos inconexos son necesarias, al menos, dos capas.

Aunque parecería lógico pensar que la mejor solución estaría en tomar cuantas más capas ocultas y muchas neuronas en las capas ocultas, hay grandes inconvenientes para ello: el aumento de la carga computacional (se necesita más tiempo para los cálculos y aumenta el tiempo de aprendizaje), pérdida en generalización (el modelo no funcionará bien con patrones no usados en su construcción) [Serrano et al., 2009].

2.3. *Extreme Learning Machine.*

Extreme Learning Machine es un novedoso, rápido y eficiente método para el entrenamiento de redes neuronales artificiales de tipo *"feed-forward"* (aprendizaje supervisado) con la estructura de un perceptrón multicapa con una capa oculta.

Las Redes Neuronales Artificiales (RNA) han sido empleadas exitosamente en una diversidad de campos del conocimiento. Aún así, tienen problemas en el proceso de optimización de los parámetros de la red (pesos y número de neuronas) con un elevado tiempo de cálculo y, a veces, con una convergencia a mínimos locales. Se han desarrollado muchos trabajos para obtener más rápidos y precisos algoritmos de entrenamiento supervisado para redes *"feed-forward"* y uno de los más recientes es el método conocido como *Extreme Learning Machine* (*ELM*) [Lara et al., 2011].

El algoritmo *Extreme Learning Machine* (*ELM*) está fundamentado en que un *MLP* compuesto por H neuronas, cuyos pesos de entrada están inicializados aleatoriamente, pueden "aprender" N distintos casos de entrenamiento produciendo un error cero, siendo $N \geq H$, y aproximando cualquier tipo de función continua. Tras inicializar de manera aleatoria los pesos de entrada, un *MLP* puede ser considerado como un sistema lineal y los pesos de salida pueden obtenerse de manera analítica mediante un simple cálculo de la pseudo-inversa de *Moore-Penrose* de la matriz de las salidas de las H neuronas ocultas para un determinado conjunto de entrenamiento [Laencina et al., 2009]. Ya se han aplicado en diferentes problemas:

- Pronóstico de ventas con aplicaciones en el comercio minorista de la moda [Sun et al., 2008].

- Predicción de estructuras secundarias de proteinas [Wang et al., 2008].
- Enfoques inteligentes para la protección de líneas de transmisión [Malathi et al., 2010].
- Reconocimiento de la acción humana a partir del vocabulario visual [Minhas et al., 2010].

2.4. Algoritmos Genéticos.

El algoritmo genético (AG), desarrollado por *John Holland* de la Universidad de Michigan a finales de los 70, es una técnica de búsqueda basada en la teoría de la evolución de *Darwin* y el mecanismo de selección natural: los individuos más aptos de una población son los que sobreviven, adaptándose más fácilmente a los cambios que se producen en su entorno [Serrano et al., 2009]. Los cambios se efectúan en los genes del individuo y los atributos más deseables del mismo se transmiten a sus descendientes en la reproducción.

El AG consiste en una búsqueda heurística que imita el proceso de evolución natural con técnicas de herencia, mutación, selección y cruce. Forma parte de la informática de la inteligencia artificial [Mitchell, 1998].

La aplicación más común de los algoritmos genéticos ha sido la solución de problemas de optimización, donde han mostrado ser muy eficientes. Para problemas no lineales, o lineales de muchas variables, los métodos tradicionales de optimización de funciones no son útiles. Una característica que debe tener el método es la capacidad de *castigar* las malas soluciones, y de *premiar* las buenas, de forma que sean estas últimas las que se propaguen con mayor rapidez.

La codificación más común de las respuestas es a través de cadenas binarias, aunque se han utilizado también números reales y letras [Sivanandam and Deepa, 2008]. El funcionamiento de un algoritmo genético simple es el siguiente: al principio se debe generar aleatoriamente la población inicial, que estará constituida por un conjunto de cromosomas (o cadenas de caracteres) que representan las soluciones posibles del problema. A cada uno de los cromosomas de esta población se le aplicará la función a optimizar a fin de saber qué tan buena es la solución que está codificando. Sabiendo

el valor que da cada cromosoma, se procede a la selección de los que se cruzarán en la siguiente generación (se deberá escoger a los "mejores"). Dos son los métodos de selección más comunes [Sivanandam and Deepa, 2008]:

- La ruleta: consiste en crear una ruleta en la que cada cromosoma tiene asignada una fracción proporcional a su aptitud, los mejores individuos recibirán una porción de la ruleta mayor que los peores.

- El torneo: se baraja la población y después se hace competir a los cromosomas que la integran en grupos de tamaño predefinido (normalmente compiten en parejas) en un torneo del que resultarán ganadores los mejores individuos.

Una vez realizada la selección, se procede a la reproducción sexual o cruce de los individuos seleccionados. En esta etapa, los supervivientes intercambiarán material cromosómico y sus descendientes formarán la población de la siguiente generación.

El cruce se escoge de forma aleatoria sobre la longitud de la cadena que representa el cromosoma y, a partir de él, se realiza el intercambio de material de los dos individuos. Normalmente el cruce se produce dentro del algoritmo genético como un porcentaje que indicará la frecuencia con la que se ha de efectuar. Esto significa que no todas las parejas de cromosomas se cruzarán, algunas pasarán intactas a la siguiente generación.

Además de la selección y el cruce, existe otro operador llamado mutación, que realiza un cambio en los genes de un cromosoma elegido aleatoriamente. Cuando se usa una representación binaria, el gen seleccionado se sustituye por su complemento (un cero cambia a uno y viceversa). Este operador permite la introducción de nuevo material cromosómico en la población tal y como sucede con sus equivalentes biológicos. La mutación, normalmente, se produce con un porcentaje que indica con qué frecuencia tiene lugar, aunque se distingue del cruce por ocurrir mucho más esporádicamente (el porcentaje de cruce normalmente es de más del 60 % mientras que el de mutación no suele superar el 5 %).

Si se supiera la solución del problema de antemano, detener el algoritmo genético sería algo trivial. Sin embargo, eso casi nunca es posible, por lo que, normalmente, se usan dos criterios principales de detención [Sivanandam and Deepa, 2008]: ejecutar el algoritmo genético durante un número máximo de generaciones o detenerlo cuando la

población se haya estabilizado, es decir, cuando todos, o la mayoría, de los individuos tengan el mismo error.

Los AG, cuando se usan para problemas de optimización -maximizar o minimizar una función objetivo-. resultan menos afectados por los extremos locales (soluciones subóptimas) que las técnicas tradicionales.

Alguna aplicaciones de los algoritmos genéticos son:

- Diseño automatizado de componentes automovilísticos: mejoras de comportamiento ante choques, ahorro de peso, mejoras en la aerodinámica, etc.
- Diseño automatizado de equipamiento industrial.
- Optimización de carga de contenedores.
- Diseño de sistemas de distribución de aguas.
- Diseño de topologías de circuitos impresos.
- Aprendizaje de comportamiento de robots.
- Análisis lingüístico, incluyendo inducción gramática, y otros aspectos de procesamiento de lenguajes naturales, tales como eliminación de ambigüedad de sentido.
- Optimización de sistemas de compresión de datos.
- Construcción de horarios en grandes universidades, evitando conflictos de clases.

Capítulo 3

Algoritmos genéticos.

Los algoritmos genéticos [Sivanandam and Deepa, 2008] son métodos heurísticos adaptativos que usan una analogía directa con el comportamiento natural para optimizar funciones . Se trabaja con una población de individuos en la que cada uno de ellos representa una solución al problema. A cada individuo se le asigna una puntuación o *fitness* en función de su aptitud; por ejemplo, en un problema de búsqueda del mínimo de una función, el individuo cuyo valor sea más bajo será el que mayor *fitness* tenga y viceversa. A continuación, se cruzan los indivíduos produciendo o no descendencia y dando lugar a una nueva generación que puede haber sufrido mutaciones.

Existen distintas ventajas a destacar de los algoritmos genéticos que los hacen preferibles a otros métodos de búsqueda, según qué aplicaciones. Las características más importantes son [Sivanandam and Deepa, 2008]:

- Operan simultaneamente con varias soluciones, a diferencia de las técnicas tradicionales, que operan de forma secuencial. Esto hace al algoritmo más robusto para problemas de muchas dimensiones.
- De todos los algoritmos de optimización estocásticos, los algoritmos genéticos son considerados unos de los más exploratorios, debido a que realizan un barrido muy grande del espacio de posibles soluciones.
- Son poco sensibles a la inicialización, por lo que una inicialización aleatoria suele bastar para obtener buenos resultados, siempre que la población sea lo suficientemente grande.

- Es posible realizar, mediante subpoblaciones, algoritmos genéticos que funcionen paralelamente y que compartan información genética a fin de encontrar mejores soluciones, o la mejor solución, en un tiempo menor.

3.1. Óptimos locales y globales.

Otros métodos de optimización, como el método del gradiente que tiene en cuenta el ritmo de máximo crecimiento de una función para encontrar su óptimo, tienen un grave inconveniente: pueden dar soluciones subóptimas cayendo en óptimos locales [Serrano et al., 2009].

En la figura 3.1 se puede ver la caída a un mínimo local al aplicar el método del gradiente para la función: $y(x) = x^4 + x^3 - 2x^2$. El método del gradiente convergerá en el mínimo local, como indican las flechas rojas de la ilustración.

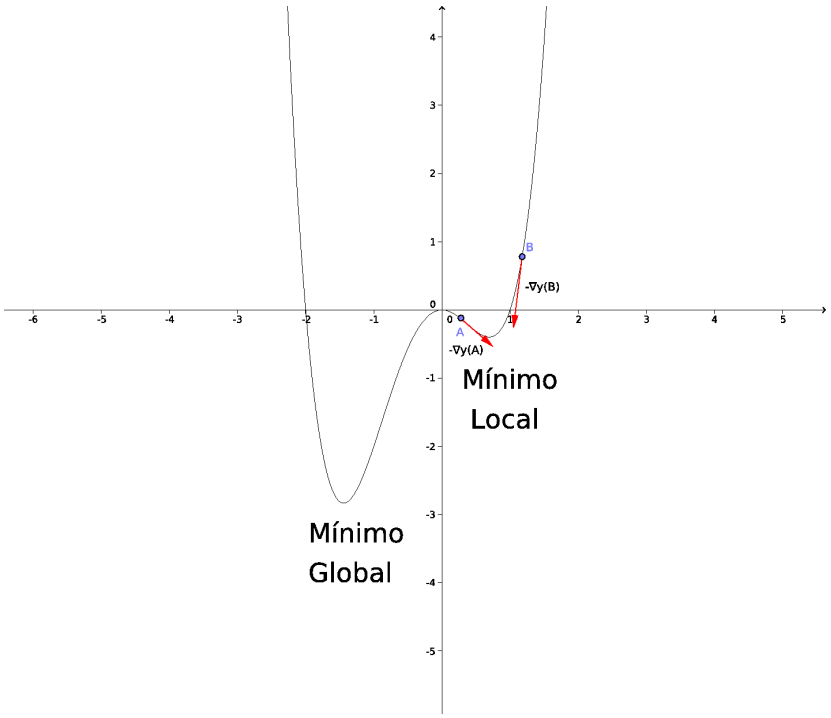

Figura 3.1: Ejemplo de caída en mínimo local del método del gradiente.

Se puede utilizar una analogía para entender, a grandes rasgos, el método del gradiente para encontrar mínimos. Si se dejara caer una bola desde uno de los lados del

mínimo local, quedaría estancada en éste, sin percatarse de que hay un valor de la función más pequeño.

Los algoritmos genéticos no presentan este problema ya que, configurados con los parámetros adecuados, cubren sin problemas todo el espacio de búsqueda. El proceso de mutación ayuda a que, si el algoritmo se estanca en un mínimo local, pueda encontrar, en siguientes generaciones, el mínimo global. [Sivanandam and Deepa, 2008]

3.2. Funciones.

3.2.1. Codificación.

Los individuos se codifican mediante una combinación de parámetros llamados genes. Éstos se forman, normalmente, a partir de un número determinado de *bits* o alelos. El número de *bits* determina la precisión de cada parámetro y no tiene por qué ser igual para todos. De esta manera, cada individuo tiene tantos genes como variables tiene el problema [Sivanandam and Deepa, 2008].

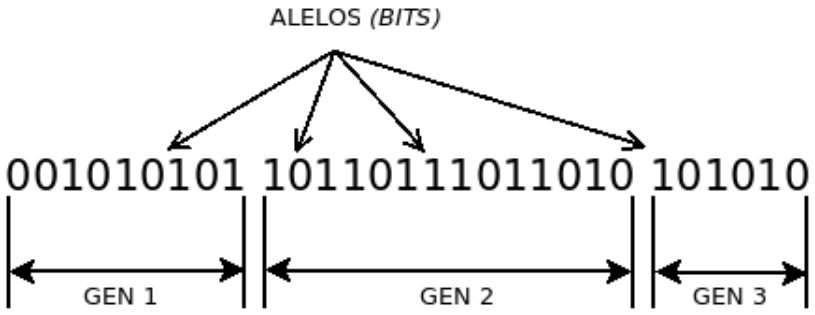

Figura 3.2: Individuo genético binario.

Generalmente, la codificación de individuos es binaria aunque pueden utilizarse otros tipos como la codificación octal, hexadecimal, etc. A continuación se explican los tipos de codificación [Sivanandam and Deepa, 2008].

Codificación binaria.

Esta codificación es la más usada. Se codifican los individuos en una cadena de *bits*. Cada una de ellas es una solución al problema, pero no necesariamente la mejor. La longitud de cada gen y, por tanto, la longitud del individuo depende del problema.

La codificación binaria da muchos posibles cromosomas con un número pequeño de alelos. Por otra parte esta codificación puede no funcionar bien para determinados problemas. La longitud de la cadena de *bits* depende de la precisión que se necesite en el problema.

A continuación se representan dos ejemplos de individuos de la misma población con codificación binaria:

- Individuo 1: 00101101011011101010
- Individuo 2: 10111011000000101110

Codificación octal.

Esta codificación utiliza cadenas de números octales (0-7) para representar a los individuos. Por ejemplo:

- Individuo 1: 726253607453215
- Individuo 2: 162570023401624

Codificación hexadecimal.

Esta codificación utiliza cadenas de números hexadecimales (0-9,A-F); por ejemplo:

- Individuo 1: A13F813EA2
- Individuo 2: 98C93DB32A

Codificación de permutación.

Se utilizan numeros naturales, con los que se construye un *string*. Sólamente se utiliza para problemas de ordenación. A modo de ejemplo:

- Individuo 1: 1 2 9 7 4 5 2 3
- Individuo 2: 4 3 8 2 1 7 4 8

3.2.2. Poblaciones.

Una población es un conjunto de individuos que son probados en la función objetivo. Los aspectos más importantes son la generación de la poblacion inicial y el tamaño. El tamaño de la poblacion depende totalmente de la complejidad del problema. La mayoría de las veces se inicializa aleatoriamente, sin embargo, existen procedimientos heurísticos para determinar una mejor inicialización.

Fitness.

Es el valor de una función objetivo para un determinado fenotipo. Para calcularlo es necesario decodificar primero el individuo y después evaluar la función objetivo. El valor de *fitness* no sólo indica la calidad de la solución, sinó que también nos da información sobre la cercanía al óptimo [Sivanandam and Deepa, 2008].

3.2.3. Selección.

En este paso se escogerán parejas de individuos para la reproducción. Esta selección otorga más posibilidades de reproducción a los individuos más aptos, como en la naturaleza [Sivanandam and Deepa, 2008]. Existen diferentes tipos:

Selección Aleatoria.

Este es el método de selección más simple. Se seleccionan N progenitores de forma totalmente aleatoria.

Selección mediante el método del torneo.

Este método se basa principalmente en realizar una selección usando comparaciones directas entre individuos. Existen dos versiones de este método:

- *Determinista*: Se seleccionan T individuos aleatoriamente (donde T es el número de individuos que competirán entre si, normalmente T=2). De estos individuos, se selecciona el que tiene un valor más alto de *fitness* para pasarlo a la siguiente generación.

- *Probabilística*: Esta versión difiere de la anterior únicamente en la fase de la selección del individuo que pasará a la siguiente generación. En vez de escoger al individuo de mayor *fitness*, se genera un número aleatorio $n \in [0, 1]$ y se compara con un parámetro constante p (generalmente $p \in [0.5, 1]$). Si $n > p$ se escoge al más apto y, si $n < p$, se escoge al menos apto.

Del método del torneo, se dice que éste es un método elitista.

Selección mediante el método de la ruleta.

Los individuos se representan sobre un círculo, de modo que, a cada uno, le corresponde un sector circular de ángulo proporcional a su *fitness* de forma que la suma de todos los sectores ocupe todo el círculo [Blickle and Thiele, 1995].

El ángulo que le corresponde a cada individuo (α_i) puede calcularse mediante la ecuación 3.1. Se escoge al individuo correspondiente al obtener un ángulo entre 0 y 2π mediante una distribución uniforme.

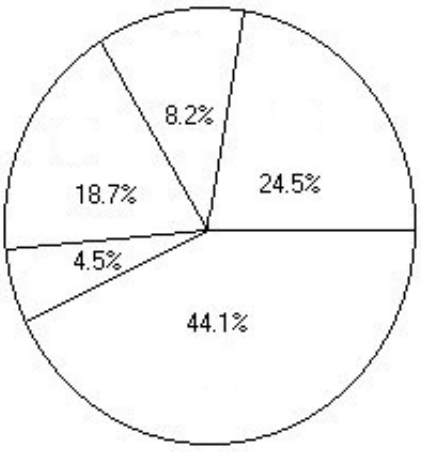

Figura 3.3: Método de la ruleta.

$$\alpha_i = \frac{360^o}{\sum_i fitness_k} \cdot fitness_i \tag{3.1}$$

Selección mediante el método de la clasificación.

Este método se utiliza cuando la diferencia de *fitness* entre individuos es muy elevada, lo que es un problema en el caso del método de la ruleta, ya que si, por ejemplo, el mejor individuo tiene un *fitness* del 90 % ocupará el 90 % de la circunferencia, mientras que los demás estarán repartidos en el 10 % restante. Ello conduce a que se tengan que hacer muchas iteraciones para seleccionar individuos diferentes del mejor.

El método de la clasificación ordena los individuos de menor a mayor *fitness*, a continuación se le asigna como nuevo *fitness* su numero de orden en la lista.

Se plantea un ejemplo para 15 individuos donde uno de ellos tiene un *fitness* mucho más alto que los demás. En la figura 3.4 se representan los individuos junto a sus *fitness* originales y su nueva asignación de *fitness*. En la figura 3.5 se representa, mediante el metodo de la ruleta, el problema de tener un individuo con un *fitness* mucho mayor que el resto. Se puede observar como el método de la clasificación corrige perfectamente este problema en la figura 3.6

	Fitness Original	Nuevo *Fitness*
Individuo 1	5	1
Individuo 2	11	2
Individuo 3	15	3
Individuo 4	17	4
Individuo 5	23	5
Individuo 6	24	6
Individuo 7	29	7
Individuo 8	31	8
Individuo 9	33	9
Individuo 10	42	10
Individuo 11	43	11
Individuo 12	61	12
Individuo 13	64	13
Individuo 14	90	14
Individuo 15	10983	15

Figura 3.4: Individuos ordenados de menor a mayor *fitness* y con nuevo *fitness* asignado.

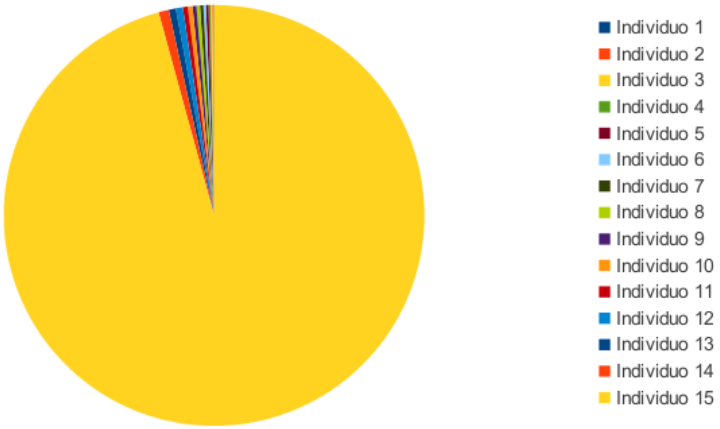

Figura 3.5: Ruleta generada con los *fitness* originales de los individuos.

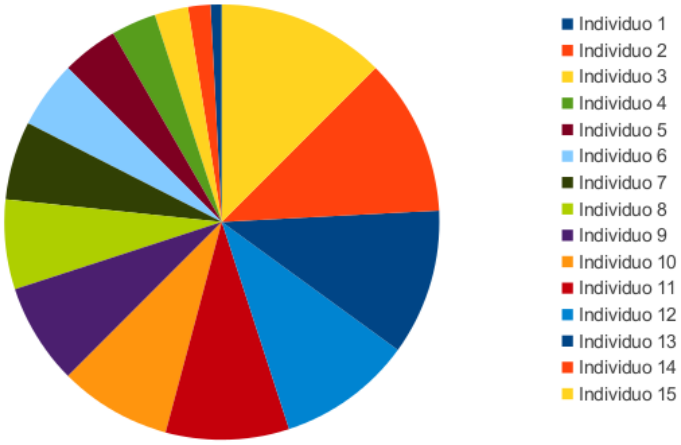

Figura 3.6: Ruleta generada con los *fitness* nuevos de los individuos.

Selección estocástica uniforme.

En éste método se representan los individuos consecutivamente sobre una recta de modo que cada uno de ellos ocupe un intervalo de amplitud proporcional a su *fitness*. Sobre el segmento de recta así obtenido se disponen tantos punteros, igualmente espaciados, como individuos se desea seleccionar [Sivanandam and Deepa, 2008]. Al espaciado entre punteros se le llama paso. El primer puntero se coloca a una distancia aleatoria del principio menor que el paso.

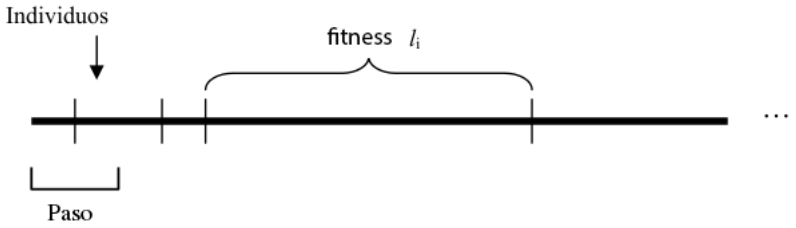

Figura 3.7: Selección estocástica uniforme.

A continuación se muestra un ejemplo de este método aplicado a un problema en el que hay 7 individuos y se desea seleccionar 5. Si llamamos L a la amplitud del segmento que ocupan, el paso será $P = \frac{L}{5}$. El primer puntero se inicializa aleatoriamente en el segmento $[0, \frac{L}{5}]$.

El puntero entonces recorrerá, como se puede ver en la figura 3.8, la recta a saltos de paso $P = \frac{L}{5}$, empezando desde el número aleatorio calculado hasta llegar al final. Los punteros ocuparán las posiciones P1, P2, P3, P4 y P5. Se seleccionan los individuos cuya amplitud sea apuntada por el puntero: I1, I2, I3, I4 e I6. Se desprecian los que no son apuntados: I5 e I7.

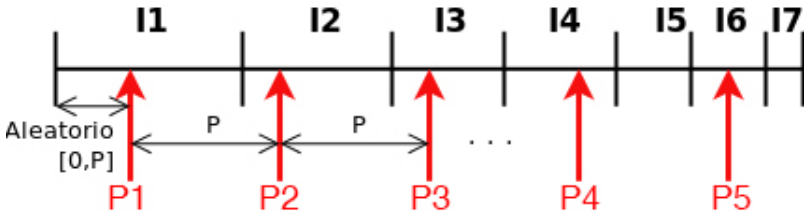

Figura 3.8: Ejemplo de selección estocástica uniforme en detalle.

3.2.4. Cruce (Recombinación).

Esta operación es una de las más importantes de los algoritmos genéticos. Mediante este proceso, se obtiene una nueva generación de individuos [Sivanandam and Deepa, 2008].

Los métodos de cruce más utilizados se detallan a continuación.

Cruce por el método de un punto.

Es el método más simple; se fija un punto de corte al azar entre los *bits* de los dos progenitores para diferenciar entre la parte derecha e izquierda de ambos. Para generar los hijos, se intercambian las partes derechas entre sí tal y como se muestra en la figura 3.9.

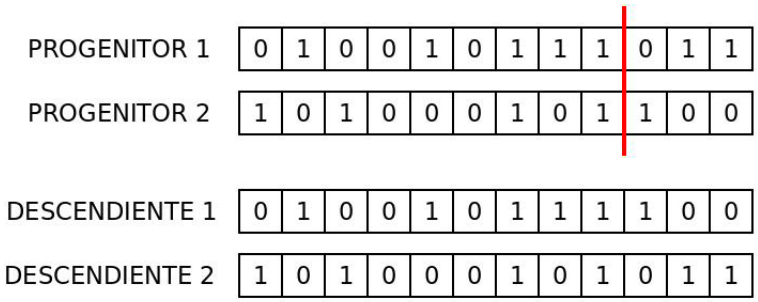

Figura 3.9: Ejemplo del método de cruce un punto.

Cruce por el método de dos puntos.

Este método es similar al anterior, la diferencia radica en que se fijan dos puntos en vez de uno. Una vez fijados, para producir la descendencia los progenitores intercambian los *bits* que están entre los dos puntos, produciéndose así dos descendientes. Se puede ver un ejemplo en la figura 3.10.

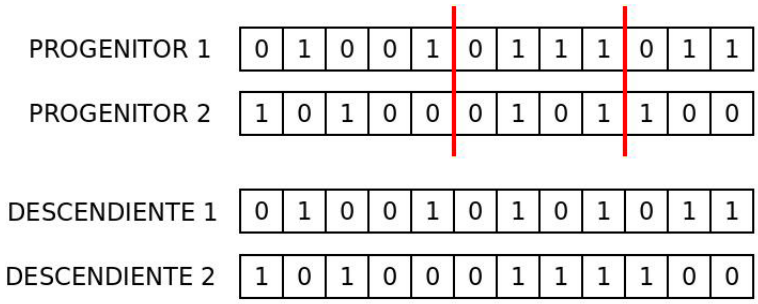

Figura 3.10: Ejemplo del método de cruce dos puntos.

Cruce heurístico.

Esta función se basa en el valor de *fitness* de los progenitores. Se obtienen dos descendientes que cumplen las siguientes ecuaciones:

$$Hijo_1 = Progenitor_1 + r * (Progenitor_1 – Progenitor_2) \tag{3.2}$$

$$Hijo_2 = Progenitor_1 \tag{3.3}$$

Siendo r un valor aleatorio entre $[0, 1]$ y $Fitness_{Progenitor_1} > Fitness_{Progenitor_2}$

De forma que el $Hijo_1$ se obtiene por la ecuación 3.2 y el $Hijo_2$ es igual que el progenitor que más *fitness* tiene. Con este método se premia al progenitor más apto haciendo que pase a la siguiente generación, castigando al progenitor menos apto.

Cruce uniforme.

En este método intervienen dos progenitores. Se genera una cadena de *bits* llamada máscara de cruce de longitud igual a la de los progenitores. Cada alelo de los hijos se crea por copia de los progenitores. Se lee *bit a bit* la máscara de cruce y, para el hijo 1, donde en la máscara hay un 1, se pone el *bit* del primer progenitor y donde hay un 0 se pone el del segundo progenitor. A continuación se ilustra en la figura 3.11 un ejemplo.

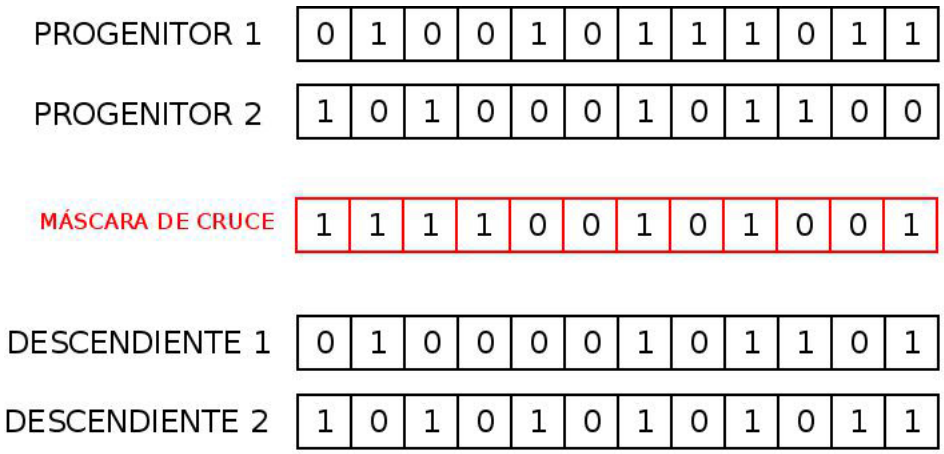

Figura 3.11: Ejemplo del método de cruce uniforme.

Cruce de tres progenitores.

En este método intervienen tres progenitores escogidos al azar. Estos producen un solo descendiente comparando sus alelos. Cada *bit* del primer progenitor es comparado con el del segundo progenitor. Si ambos son iguales, el descendiente heredará ese *bit*. Si son diferentes, el descendiente heredará el *bit* que tiene el tercer progenitor en esa posición. Obsérvese el ejemplo de la figura 3.12.

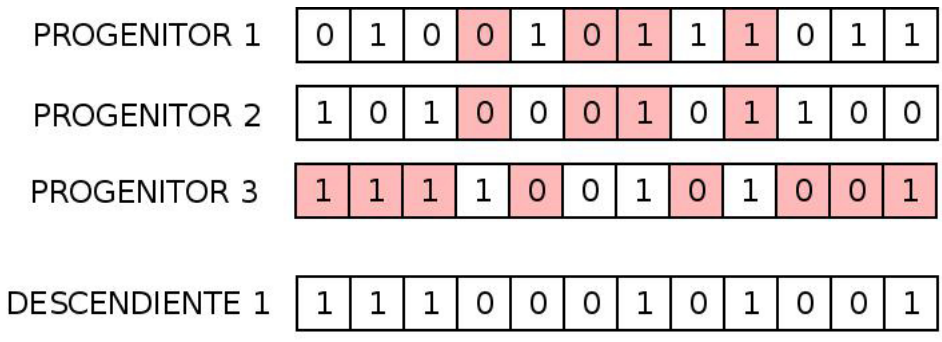

Figura 3.12: Ejemplo del método de cruce de tres progenitores.

3.2.5. Mutación.

El objetivo principal de esta operación es producir diversidad entre la población. Esta función es la principal encargada de que el algoritmo genético no se estanque en mínimos locales [Sivanandam and Deepa, 2008]. Existen diferentes procedimientos:

Método *flipping*.

Se genera aleatoriamente una cadena de *bits*, llamada cromosoma de mutación, de la misma longitud que los individuos. Esta contiene un mayor número de ceros que de unos. Se compara *bit a bit* con el individuo que mutará: cuando aparece un 1 en el cromosoma de mutación, se cambia el alelo del individuo (donde había un 1, se cambia a 0 y viceversa). Véase el ejemplo de la figura 3.13.

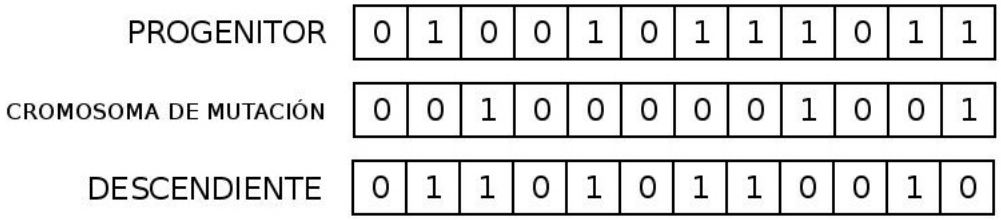

Figura 3.13: Ejemplo del método *flipping*.

Función de mutación gaussiana.

Suma un número aleatorio a cada término (o gen) del individuo. Este número pertenece a una distribución gaussiana con media 0 y desviación típica en la primera generación definida por el parámetro *Scale*. La desviación típica disminuye, de generación en generación, en función del parámetro *Shrink*, el cual pertenece al intervalo [0,1].

Si se escoge un *Shrink* muy próximo a uno, a medida que pasen las generaciones se verá muy pronto como los individuos empiezan a ser parecidos hasta que convergen en el mismo valor.

Por el contrario, si se escoge un *Shrink* muy próximo a cero, tendrán que pasar muchas generaciones para que los individuos empiecen a tomar valores parecidos.

Probabilidad de mutación.

Este método se basa en una probabilidad (P_m) fijada *a priori* por el usuario, mediante este parámetro es posible fijar la frecuencia de mutación. Si en un progenitor no se produce mutación es porque la probabilidad de mutación se ha seleccionado muy baja y pasará a la siguiente generación sin problemas. Si la mutación se produce, una o más partes del cromosoma sufrirán un cambio.

3.2.6. Reemplazo.

El reemplazo de individuos es la última fase del algoritmo. En el caso general, dos individuos de la población se han convertido en progenitores, se han cruzado y han producido dos hijos. Como la población de nuestro algoritmo genético es fija, no pueden volver a adaptarse los cuatro en la población y, por tanto, los descendientes podrán o no reemplazar a sus progenitores. Existen diferentes formas de reemplazarlos, se comentan a continuación las más utilizadas [Sivanandam and Deepa, 2008].

Reemplazo aleatorio.

Es el método más sencillo. Los descendientes reemplazan a dos individuos de la población elegidos al azar.

Reemplazo de débiles progenitores.

De entre los progenitores y los descendientes, los dos individuos con mayor *fitness* reemplazan a los dos de menor *fitness*.

Reemplazo de ambos progenitores.

Los progenitores son reemplazados por los descendientes siempre.

3.2.7. Criterio de parada.

Una de las dudas que surgen al tratar con algoritmos genéticos es: ¿cuándo hay que detenerlo?. Existen diferentes versiones según el criterio escogido, de las cuales se comentarán las más utilizadas a continuación [Sivanandam and Deepa, 2008]:

- *Máximo número de generaciones*: Se fija un número máximo de generaciones y cuando el algoritmo genético lo alcanza, para.
- *Tiempo empleado*: El algoritmo genético para cuando se ejecuta durante un tiempo determinado.
- *No hay cambios en el fitness*: El algoritmo genético para cuando no se producen cambios en el *fitness* en un número determinado de generaciones. En este método se ajustan dos parámetros, la sensibilidad del *fitness* y el número de generaciones sin cambio de *fitness*.

Capítulo 4

Extreme Learning Machine.

4.1. Arquitectura y funcionamiento.

El algoritmo *Extreme Learning Machine* (*ELM*, Máquina de Aprendizaje Extremo) fue propuesto por Huang et al. [Huang et al., 2006]. Se usa en una estructura multicapa con una sola capa de neuronas oculta (*Single Layer Feedforward Network*, *SLFN*). Se empieza iniciando aleatoriamente los pesos que unen la capa de entrada y la capa oculta. De esta manera, sólo será necesario optimizar los pesos que están entre la capa oculta y la capa de salida. Para optimizar estos últimos pesos se utiliza la matriz pseudoinversa de *Moore-Penrose* [Rao and Mitra, 1972].

ELM disminuye notablemente el tiempo computacional empleado para ajustar los pesos debido a que no utiliza ningún método de búsqueda para los coeficientes ocultos. En cambio, como se detallará a continuación, el método de la matriz pseudoinversa de *Moore-Penrose* es un simple y rápido cálculo algebraico.

Dado un conjunto de N patrones, $D = (\mathbf{x_i}, \mathbf{o_i})$, $i = 1..N$, donde las entradas $\mathbf{x_i} \in \Re^{d1}$ y la salida deseada $\mathbf{o_i} \in \Re^{d2}$. El objetivo es encontrar la relación entre $\mathbf{x_i}$ y $\mathbf{o_i}$. Siendo M el número de neuronas en la capa oculta e y_j la salida j-ésima del *SLFN*, se tiene:

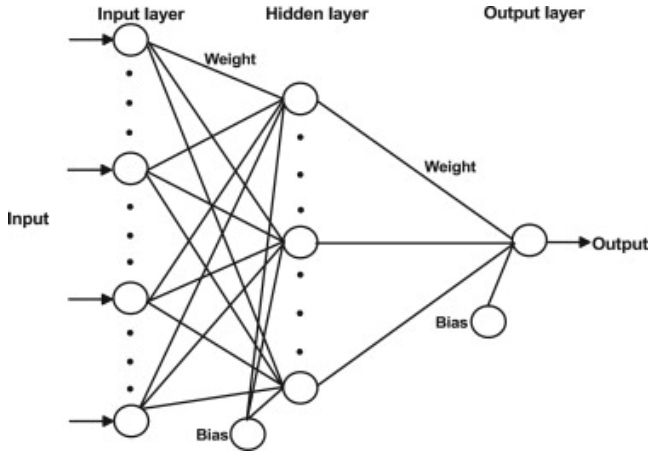

Figura 4.1: Arquitectura utilizada en la *ELM*.

$$y_j = \sum_{k=1}^{M} h_k \cdot f(\mathbf{w_k}, \mathbf{x_j}) \tag{4.1}$$

Donde $1 \leqq j \leqq N$, $\mathbf{w_k}$ representa los parámetros del k-ésimo elemento de la capa de entrada, h_k es el k-ésimo peso que conecta la capa de entrada con la capa de salida y f es una función de activación aplicada al producto escalar del vector de entrada y los pesos de la capa de salida.

La ecuación 4.1 también puede expresarse como $\mathbf{y} = \mathbf{G} \cdot \mathbf{h}$ donde $\boldsymbol{h}$ es el vector de pesos de la capa de salida, $\boldsymbol{y}$ es el vector de salida y $\boldsymbol{G}$ se calcula mediante:

$$\mathbf{G} = \begin{pmatrix} f(\mathbf{w_1}, \mathbf{x_1}) & \dots & f(\mathbf{w_M}, \mathbf{x_1}) \\ \vdots & \ddots & \vdots \\ f(\mathbf{w_1}, \mathbf{x_N}) & \dots & f(\mathbf{w_M}, \mathbf{x_N}) \end{pmatrix} \tag{4.2}$$

$$\mathbf{h} = \begin{pmatrix} h_1 \\ \vdots \\ h_M \end{pmatrix} \tag{4.3}$$

$$\mathbf{y} = \begin{pmatrix} y_1 \\ \vdots \\ y_N \end{pmatrix} \tag{4.4}$$

Como se comentó anteriormente, se inicializan aleatoriamente los pesos de la capa de entrada. Los pesos de la capa de salida se obtienen mediante la matriz pseudoinversa de *Moore-Penrose* ($\mathbf{G}^+$) con las siguientes expresiones [Rao and Mitra, 1972]:

$$\mathbf{h} = \mathbf{G}^+ \cdot \mathbf{o} \tag{4.5}$$

Donde la matriz pseudoinversa $\mathbf{G}^+$ se calcula mediante la expresión:

$$\mathbf{G}^+ = (\mathbf{G}^\top \cdot \mathbf{G})^{-\mathbf{1}} \cdot \mathbf{G}^\top \tag{4.6}$$

El superíndice T significa trasposición.

Y $\mathbf{o}$ es el vector de salida deseada definido como:

$$\mathbf{o} = \begin{pmatrix} o_1 \\ \vdots \\ o_N \end{pmatrix} \tag{4.7}$$

Como puede observarse en la ecuación 4.5, una vez calculada $\mathbf{G}^+$, basta con multiplicarla con la salida deseada ($\mathbf{o}$) para obtener, automáticamente, el vector de pesos de la capa de salida ($\mathbf{h}$).

4.2. Ventajas e inconvenientes de la *ELM*.

Como se ha comentado, este método es mucho más rápido que el método de descenso por gradiente que se usa de forma clásica [Haykin, 1998], debido a que obtiene los coeficientes mediante un cálculo algebráico.

Por otra parte, a pesar de su rapidez, al iniciar los pesos de la capa de entrada aleatoriamente, se necesitarán muchas neuronas en la capa oculta para modelizar bien el sistema. Por tanto, se tendrán muchos parámetros (pesos sinápticos) a optimizar y la red sobreentrenará produciendo un gran error de generalización.

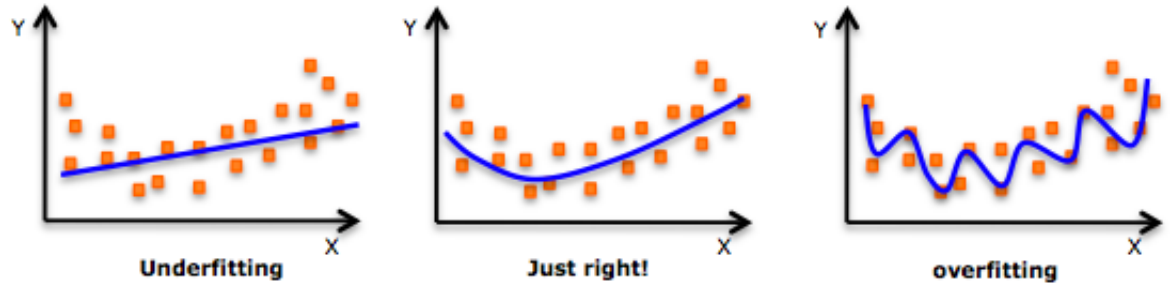

Figura 4.2: Ejemplo de sistema poco entrenado, correctamente entrenado y sobreentrenado.

La ELM solamente puede entrenarse con función de coste cuadrática, lo cual es un inconveniente.

4.3. Aproximación planteada.

Como se ha comentado, el algoritmo *ELM* solamente puede utilizar función de coste cuadrática. Esto es un problema ya que, aparte de las desventajas que conlleva usar este tipo de coste [Haykin, 1998], en determinadas ocasiones, dependiendo de la aplicación, puede interesar utilizar cualquier otra funcion de coste. La función de coste cuadrática, al contener el término elevado al cuadrado, ante un conjunto de datos con algunos parámetros dispares puede devolver resultados erróneos. La penalización, para el criterio de error absoluto, aumenta linealmente con la magnitud del error. El coste cuadrático penaliza cualquier error de forma proporcional al cuadrado del mismo. Esto puede llegar a ser un grave problema en el caso de que en el conjunto de datos aparezcan datos dispares, pues el sistema se modelizará mal debido a la alta influencia de estos datos

Se pretende utilizar, en vez de la matriz pseudoinversa de *Moore-Penrose*, algoritmos genéticos para obtener los pesos de la capa oculta de la *ELM*. De esta manera será posible utilizar otras funciones de coste diferentes a la cuadrática. También podrán

utilizarse funciones de coste de regularización que permiten disminuir el sobreentrenamiento. A este nuevo algoritmo se le llamará en adelante *GELM* (*Genetic Extreme Learning Machine*, del inglés, Máquina Genética de Aprendizaje Extremo).

Se obtendrán los resultados para los algoritmos *ELM* y *GELM*, mediante una serie de conjuntos, y se compararán los resultados. Todo este proceso y los resultados obtenidos se detallarán en el siguiente capítulo.

Capítulo 5

Resultados.

En este capítulo se dan los resultados obtenidos aplicando la aproximación planteada, *GELM*, a una serie de bases de datos estándar.

5.1. Procedimiento experimental.

En esta sección se explicarán y justificarán las técnicas que se han utilizado para obtener los resultados. El *software* utilizado en los experimentos ha sido *Matlab R2011a*.

Algoritmos genéticos.

A continuación se describe el procedimiento experimental seguido para definir los parámetros de los algoritmos genéticos.

- *Codificación*: Se utiliza codificación decimal con vectores de tipo *double*. Se ha elegido ésta porque estaba configurada por defecto y no se ha necesitado cambiarla.
- *Población*: Se ha decidido inicializar la población del algoritmo genético de forma completamente aleatoria usando una distribución uniforme. Los valores iniciales de la población se han fijado al rango [-0.5, 0.5] debido a que, al inicializar la

distribución de inividuos más cercana a 0, se ha observado, experimentalmente, una convergencia mas rápida.

- *Número de individuos*: Se utilizará un número de individuos para el algoritmo genético proporcional al número de neuronas en la capa oculta de la ELM. Así, la proporción del número de individuos respecto del tamaño del problema será constante. Dependiendo del tamaño del conjunto de datos, el factor multiplicador será mayor o menor. Al aumentar el número de individuos tambien aumenta el tiempo de cálculo, éste es un aspecto importante que se ha tenido en cuenta.

$$PopulationSize = n_{nodos} * FactorMultiplicador \tag{5.1}$$

  Se realizó una prueba para determinar el *FactorMultiplicador* óptimo en función del problema. Se hizo un barrido con este parámetro (fijando el número de nodos a 120), ejecutando el algoritmo una vez por cada ajuste y analizando cuál producía un menor error en la salida de la *ELM*. La conclusión obtenida al realizar este experimento fue que, debido a la inicialización aleatoria, el número óptimo de individuos en la población varía. En el anexo (archivo *optimiza_individuos.m*) se añade el algoritmo que realiza el barrido para el conjunto de datos *housing* (ver sección 5.1.1).

  Ante estos resultados, se ha escogido un $FactorMultiplicador \in [10, 25]$, en base al tamaño del conjunto de datos: para el conjunto más grande tomará el valor 25 y para el más pequeño el valor 10.

- *Selección*: Se ha utilizado la función de selección estocástica uniforme ya que era la que tenía el *software* asignada por defecto y funcionaba perfectamente

- *Cruce*: Se ha empleado la función de cruce heurístico porque, con la función por defecto del *software*, el algoritmo no llegaba a converger. El parámetro r de esta función se ha configurado a $r = 1.2$ que es el valor por defecto y funciona adecuadamente.

- *Mutación*: La función de mutación empleada es la predeterminada en el *software*. Se trata de la función de mutación gaussiana, donde los parámetros utilizados han sido el *Shrink*=*Scale*=1 (valor por defecto).

- *Criterio de parada*: Se ha escogido la opción de parar el algoritmo cuando se llegue a un número determinado de generaciones, 800 en concreto. Si el valor de la función de *fitness* del mejor individuo no varía en 100 generaciones, aunque

no se haya completado el ciclo de generaciones, también se dará por termindo el algoritmo.

Extreme Learning Machine.

A continuación se describen las pruebas que se han hecho para comparar la *ELM* y la *GELM*.

En primer lugar, los pesos de la capa de entrada se inicializan aleatoriamente. Estos valores se conservarán para entrenar los pesos de la capa de salida en la *ELM* y en la *GELM* para que, al ser iguales, ambos algoritmos estén en igualdad de condiciones y se puedan, *a posteriori*, comparar los resultados.

Se ha realizado un barrido en el número de neuronas de la capa oculta. Éste es siempre proporcional al número de variables que tiene el conjunto de datos. Siendo L el número de variables, el experimento se ha repetido para $Num.deNodos = 2L, 3L, 4L...XL$ donde X se ajusta experimentalmente dependiendo del tamaño del conjunto de datos escogido teniendo en cuenta que a más número de neuronas en la capa oculta, hay más parámetros a optimizar y el tiempo de cálculo aumenta. Normalmente se escoge un número alto, del orden de $[25, 35]$. Para las funciones de coste de regularización solamente se han estudiado los extremos, es decir, $Num.deNodos = 2L, XL$

Además, todo lo anterior se repite 100 veces y se obtiene la media de los resultados. Esto se realiza para obtener unos resultados estadísticamente fiables, ya que como se escogen los pesos de la capa de entrada y los subconjuntos de entrenamiento y generalización de forma aleatoria, podría darse la casualidad de elegirlos muy buenos, o muy malos, obteniendo conclusiones erróneas.

Las funciones de coste utilizadas en la modelización son dos: la cuadrática (ecuación 5.2) y la absoluta (ecuación 5.3). Las utilizadas en la regularización son dos también: la función de coste regularizada cuadrática (ecuación 5.4) [Hoerl and Kennard, 1970], y la funcion de coste regularizada absoluta (ecuación 5.5). Se ha repetido el procedimiento anterior para cada función de coste.

$$J_{cuadrática} = \frac{1}{N} \cdot \sum_{i=1}^{N} (o_i - y_i)^2 \tag{5.2}$$

$$J_{absoluta} = \frac{1}{N} \cdot \sum_{i=1}^{N} |o_i - y_i| \tag{5.3}$$

$$J_{Reg_{cuadrática}} = \frac{1}{N} \cdot \sum_{i=1}^{N} |o_i - y_i| + \lambda \cdot \sum_{j=1}^{M} h_j^2 \tag{5.4}$$

$$J_{Reg_{absoluta}} = \frac{1}{N} \cdot \sum_{i=1}^{N} |o_i - y_i| + \lambda \cdot \sum_{j=1}^{M} |h_j| \tag{5.5}$$

Aquí J es la función de coste correspondiente y λ es un parámetro a configurar. El código utilizado (en *Matlab R2011a*) puede ser consultado en el anexo (archivos *ejecutar.m*, *genera.m*, *minimiza_error.m*).

5.1.1. Datos usados.

A continuación se da una introducción a los conjuntos de datos con los que se han testeado los algoritmos. Estos datos han sido descargados del repositorio *online UCI Machine Learning Repository* (http://archive.ics.uci.edu/ml/).

- *Parkinson.*
  Este conjunto de datos fue creado por *Max Little* de la Universidad de Oxford, en colaboración con el Centro Nacional de Voz y Habla, Denver, Colorado. Está compuesto de una serie de medidas vocales biomédicas de 31 personas de las cuales 23 tienen la enfermedad de Parkinson. Está organizado en forma de tabla de tamaño 5875x22.

  Cada columna se corresponde con una medida particular de la voz, y cada fila es una de 195 voces grabadas a estos individuos. Una de las columnas es la salida deseada, de forma que si en una fila hay un 0, la persona cuya columna pertenece a la grabación de su voz está sana, si hay un 1, significa que esa persona tiene la enfermedad de Parkinson.

Con este conjunto de datos se pretende aprender a discernir entre gente sana y gente enferma de Parkinson mediante grabaciones biomédicas de su voz.

En la aplicación del nuevo algoritmo GELM, este conjunto fue descartado debido a su gran tamaño, después de un mes corriendo el algoritmo, no acabó ni siquiera una de las 100 pruebas realizadas.

- *Abalone.*
  Con este conjunto de datos se pretende predecir la edad de una serie de abulones a partir de determinadas medidas físicas [Nash et al., 1994]. El procedimiento tradicional utilizado para el cálculo de la edad de estos se basa en la realización de un corte en su cáscara y la examinación mediante microscopio.

  Los datos se organizan en una tabla de tamaño 4177x9 en el que cada columna es una medida (la primera es el sexo) y cada fila es un abulón diferente. Una de las columnas es el sexo y otra es el número de anillos, que se corresponde proporcionalmente con su edad y éste, por tanto, es el valor a predecir.

  Este conjunto se descartó ya que al ejecutar el algoritmo *GELM*, logró acabar al cabo de un mes de ejecución. Se obtuvieron los resultados para las primeras pruebas (coste cuadrático) pero no se repitió para las siguientes.

- *Deltaelevators.*
  Este conjunto de datos se ha obtenido de un control de elevadores del avión F16. La variable objetivo es la variación del lugar del avión en valor absoluto. Se organiza en forma de tabla de tamaño 9517x7 donde cada columna representa una variable.

  Este conjunto se descartó debido al alto coste computacional y el gran tiempo que suponía testearlo.

- *Housing.*
  Este conjunto de datos fue donado por la librería *StatLib*, mantenida por la universidad Carnegie Mellon [David Harrison and Rubinfeld, 1978]. Estudia el valor de las viviendas en los suburbios de Boston. Se organiza en forma de tabla de tamaño 506x14 donde cada fila corresponde a una vivienda y cada columna a una variable. La última columna es la salida deseada, que es el valor a predecir: el valor medio de las casas habitadas en miles de dólares. Las variables a estudiar son datos sociales tales como la tasa de criminalidad *per cápita* por localidad o la *ratio* profesor-alumno por pueblo.

- *AutoMPG.*
  Este conjunto de datos fue donado por la librería *StatLib*, como en el caso anterior [David Harrison and Rubinfeld, 1978]. Fue usado en 1983 en la Exposición de la Asociación Americana de Estadística.

  Se basa en el cálculo del consumo de combustible de un automóvil en MPG (*Milles Per Gallón*) a partir de atributos como el número de cilindros, la aceleración del coche, su peso, etc.

  Los datos se organizan en una matriz de datos de tamaño 392x8 donde las columnas representan los datos del coche y cada fila es un coche diferente.

- *Autoprice.*
  Este conjunto de datos trata de valorar automóviles de segunda mano asignándoles un precio según sus características (por ejemplo un índice de seguridad). Los datos se organizan en una matriz de tamaño 159x16 en la que las columnas representan los datos de los coches y las filas los coches. La última columna representa el precio asignado por un técnico.

- *Forestfires.*
  Se predice la superficie de bosque quemada a partir de datos como la temperatura ambiente, la velocidad del viento, el mes del año, la humedad relativa, etc [Cortez and Morais, 2007]. Este conjunto de datos tiene un tamaño de 517x13.

- *Servo.*
  Estos datos fueron cortesía de *Karl Ulrich* del MIT (*Massachussets Institute of Technology*, Instituto de Tecnología de Massachussets) en 1986. Los datos son de una simulación de un sistema servomotor con su amplificador de un robot. Mediante este conjunto, se pretenden estudiar ciertos parámetros para obtener la mejor respuesta del motor; este conjunto tiene un tamaño de 167x5.

5.1.2. Esquema del procedimiento experimental seguido.

La figura 5.1 resume, *grosso modo*, el procedimiento experimental utilizado en este proyecto por lo que puede ser útil para tener una visión general del mismo.

Cabe destacar que la inicializacion aleatoria de pesos es la misma para el algoritmo *ELM* que para el *GELM*. La doble flecha que aparece entre los bloques de resultados indica una comparación.

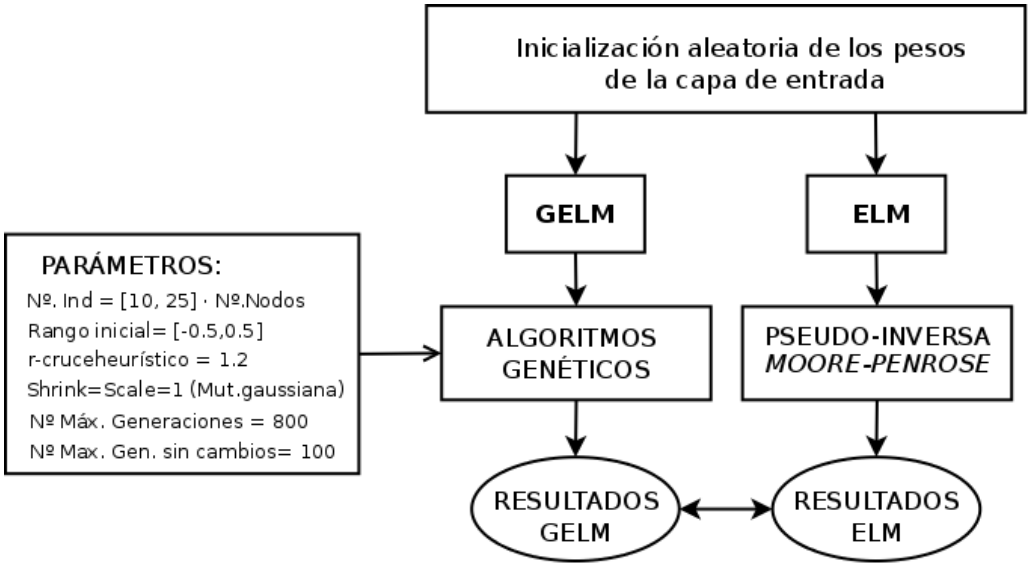

Figura 5.1: Esquema del procedimiento general.

Los resultados se comparan mediante el error que comente el sistema modelizado. Se calculará el error *ME* (*Mean Error*, del inglés error medio), el error *RMSE* (*Root Mean Squared Error*, del inglés raíz cuadrada del error cuadrático medio) y el error *MAE* (*Mean Absolute Error*, del inglés error medio absoluto).

- *RMSE*: Se calcula mediante la expresión:

$$RMSE = \sum_{i=1}^{N} \frac{(o_i - y_i)^2}{N} \tag{5.6}$$

  Este índice nos da la medida del promedio de las diferencias entre los valores de salida del sistema deseados y los obtenidos. Siempre será positivo.

- *MAE*: Se calcula mediante la expresión:

$$MAE = \sum_{i=1}^{N} \frac{|o_i - y_i|}{N} \tag{5.7}$$

  Este índice nos da una información similar al anterior. Siempre será positivo.

- *ME*: Se calcula mediante la expresión:

$$ME = \sum_{i=1}^{N} \frac{(o_i - y_i)}{N} \tag{5.8}$$

  Este índice nos aporta información sobre la tendencia del modelo a sobreestimar o subestimar una variable, es decir, nos determina el sesgo del modelo.

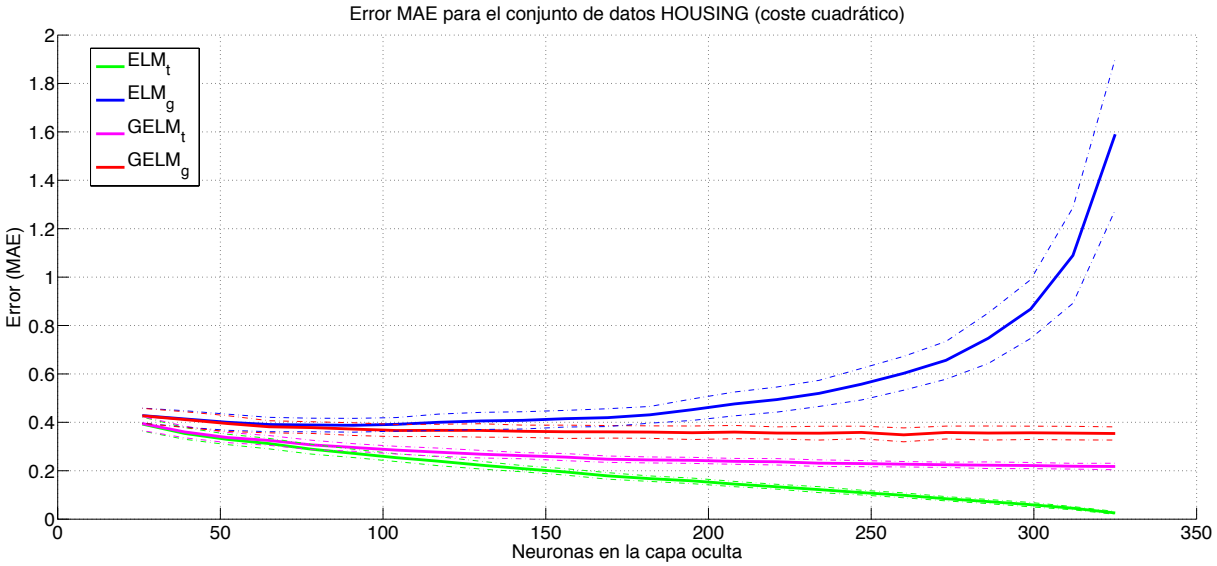

Figura 5.2: MAE para el conjunto de datos *housing* con función de coste cuadrática.

5.2. Resultados para funciones de coste sin regularizar.

En las gráficas representadas en este apartado, las líneas discontínuas representan la desviación estándar de las 100 pruebas realizadas, y las contínuas los valores medios.

5.2.1. Función de coste cuadrática.

Datos *Housing*.

En las figuras 5.2 y 5.3 se representan los índices de error para entrenamiento y validación obtenidos al ejecutar el algoritmo. Puede observarse como el error de la *GELM* se estabiliza a los pocos nodos (100 aprox.) y se mantiene constante con su aumento, mientras que el error de la *ELM* se estabiliza con menos nodos (50 aprox.) y sobreentrena cuando estos aumentan.

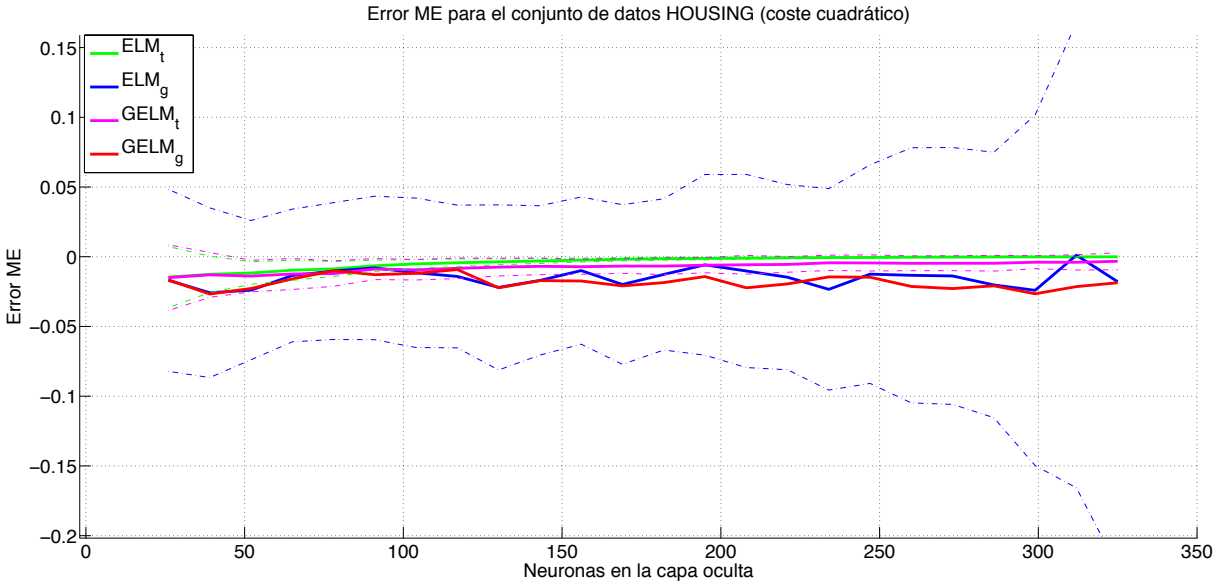

Figura 5.3: ME para el conjunto de datos *housing* con función de coste cuadrática.

También se puede ver que, en este caso, el error de generalización obtenido por la *ELM* es menor que el de la GELM, por tanto, podría concluirse que la *GELM* obtiene una mejor solución y es estable cuando se configura con muchos nodos.

Datos *Abalone*.

En las figuras 5.4 y 5.5 se representan los índices de error para entrenamiento y validación que se han obtenido al ejecutar el algoritmo.

Se han añadido estas graficas para mostrar como se comportan ambos algoritmos ante un conjunto de datos grande. Como puede observarse aparece el mismo efecto que en el conjunto de datos anterior, al subir mucho el número de nodos, la *ELM* empieza a sobreentrenar mientras que el error de la *GELM* permanece constante. En cuanto a los resultados, para unos 120 nodos, la *ELM* obtiene un resultado mínimamente mejor que la GELM.

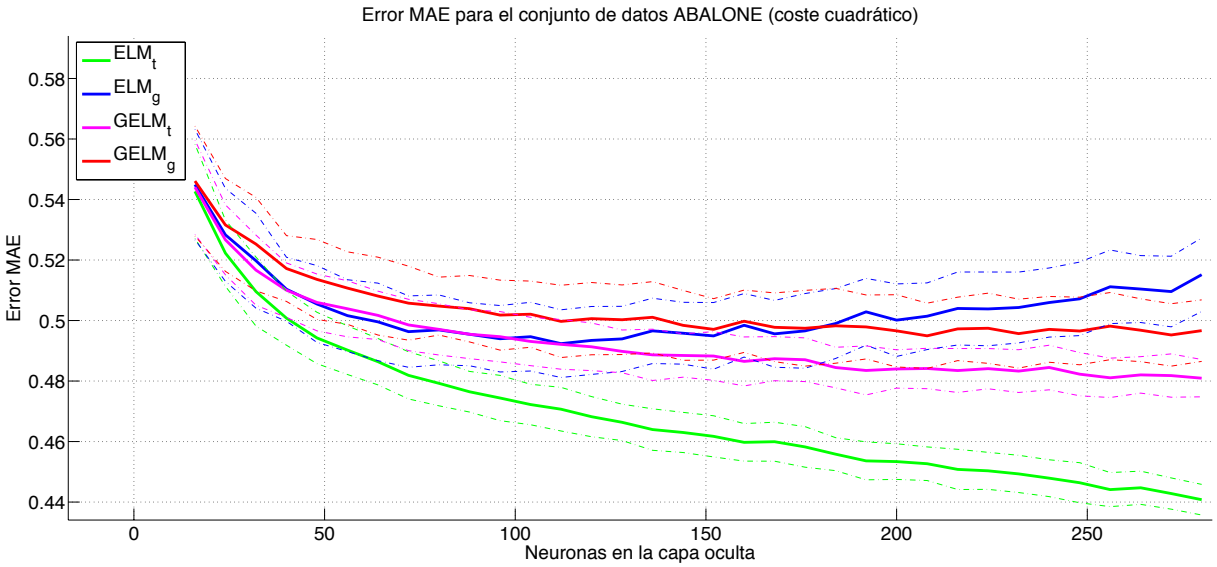

Figura 5.4: MAE para el conjunto de datos *abalone* con función de coste cuadrática.

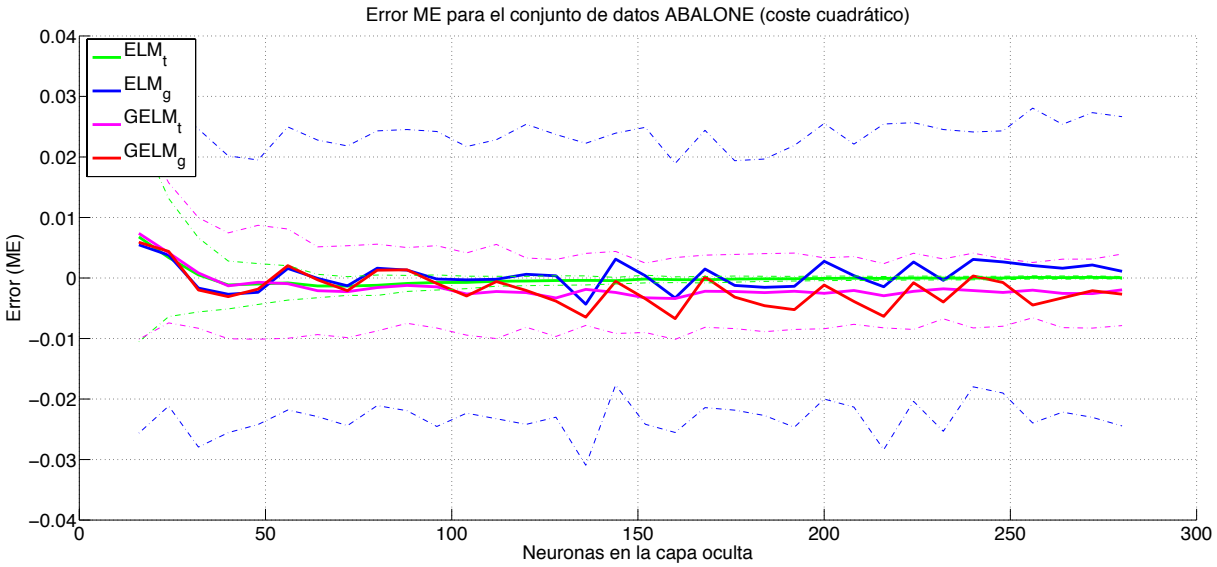

Figura 5.5: ME para el conjunto de datos *abalone* con función de coste cuadrática.

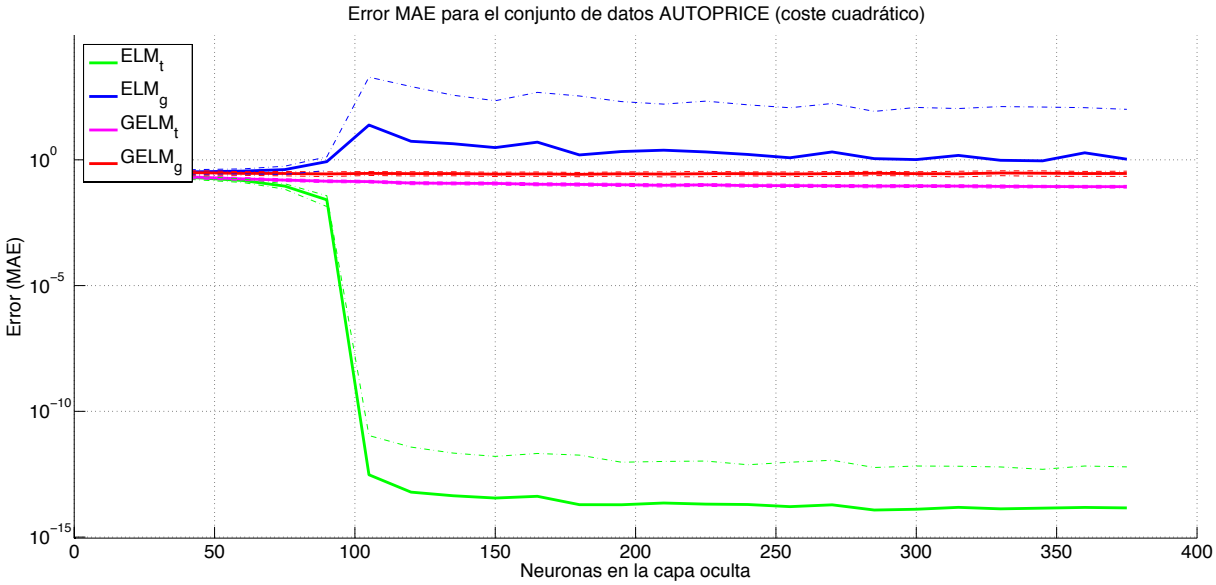

Figura 5.6: MAE para el conjunto de datos *autoprice* con función de coste cuadrática.

Datos *Autoprice.*

En las figuras 5.6 y 5.7 se representan los índices de error para entrenamiento y validación que se han obtenido al ejecutar el algoritmo. En este conjunto de datos ha surgido un problema grave; la *ELM* tiene un sobreentrenamiento precoz, es decir, con muy pocos nodos, el error de generalización se hace muy grande y el de entrenamiento muy pequeño. Se ha usado, para calcular el error medio, la mediana en vez de la media y, para representarlo, una escala semilogarítmica. Por otra parte, la *GELM* permanece con un error estable sin sobreentrenar.

Datos *Servo.*

En las figuras 5.8 y 5.9 se representan los índices de error para entrenamiento y validación que se han obtenido al ejecutar el algoritmo. Se han añadido estas gráficas para mostrar cómo funcionan los algoritmos con un conjunto de datos muy pequeño. Se observa el mismo comportamiento comentado en los casos anteriores. La *GELM* obtiene un error menor que la *ELM* y ésta, cuando el número de nodos es elevado, sobreentrena.

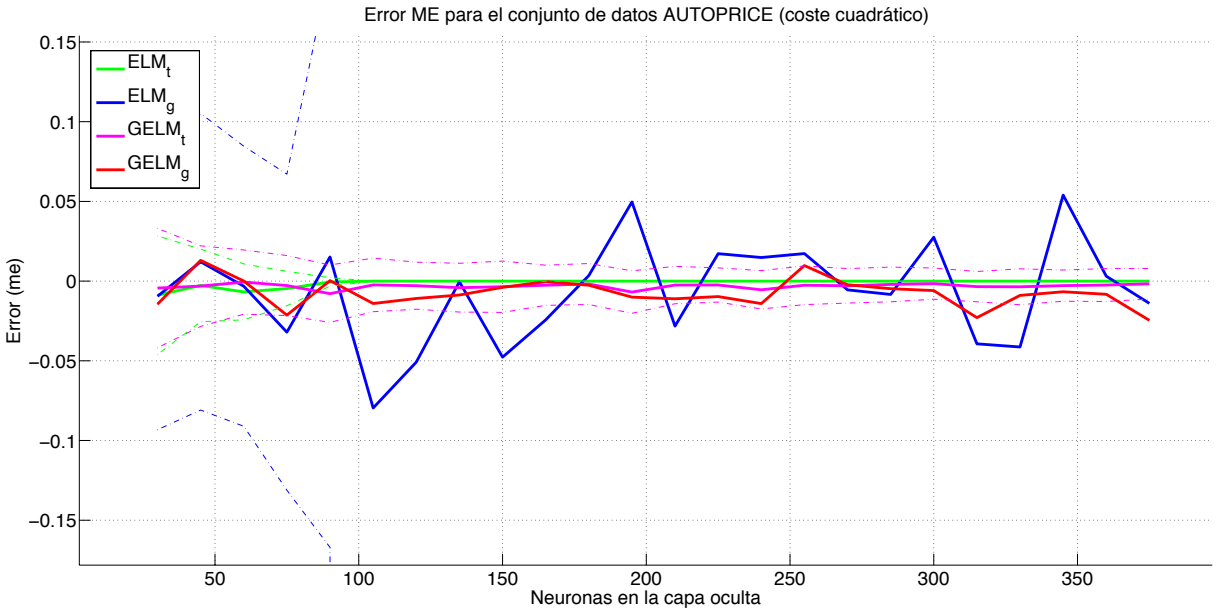

Figura 5.7: ME para el conjunto de datos *autoprice* con función de coste cuadrática.

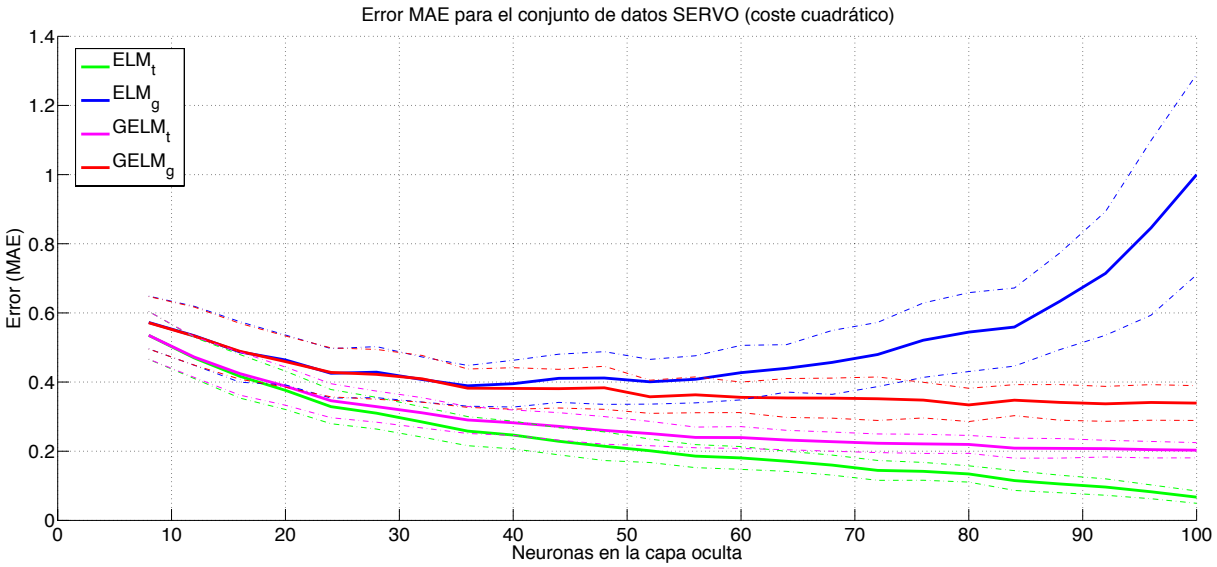

Figura 5.8: MAE para el conjunto de datos *servo* con función de coste cuadrática.

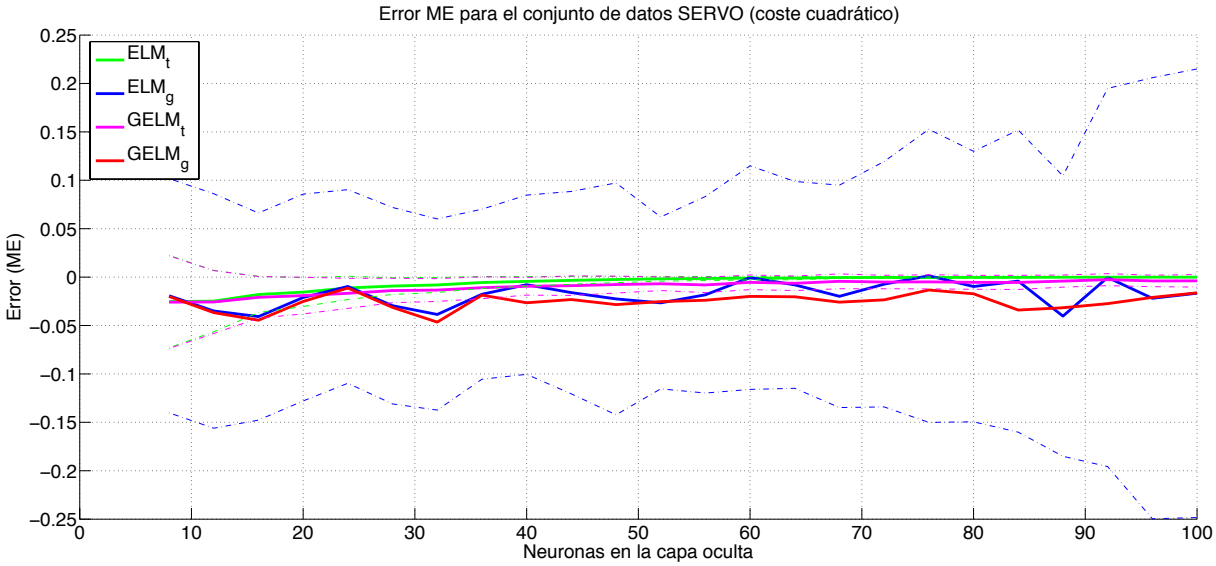

Figura 5.9: ME para el conjunto de datos *servo* con función de coste cuadrática.

5.2.2. Función de coste valor absoluto.

Datos *Housing*.

En las figuras 5.10 y 5.11 se representan los índices de error para entrenamiento y validación obtenidos al ejecutar el algoritmo. Los resultados obtenidos son similares a los comentados anteriormente (función de coste cuadrática).

Datos *Autoprice*.

En las figuras 5.12 y 5.13 se representan los errores de entrenamiento y validación que se han obtenido al ejecutar el algoritmo. Los resultados obtenidos son similares a los comentados anteriormente (función de coste cuadrática). Las gráficas se han representado con escala semilogarítmica.

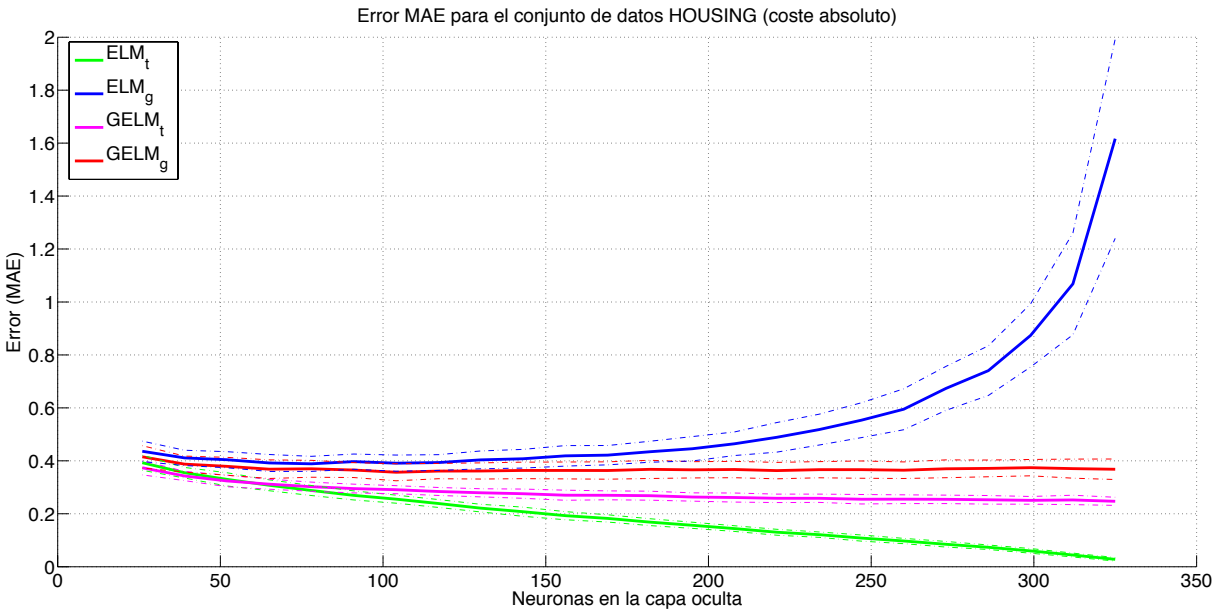

Figura 5.10: MAE para el conjunto de datos *housing* con función de coste valor absoluto.

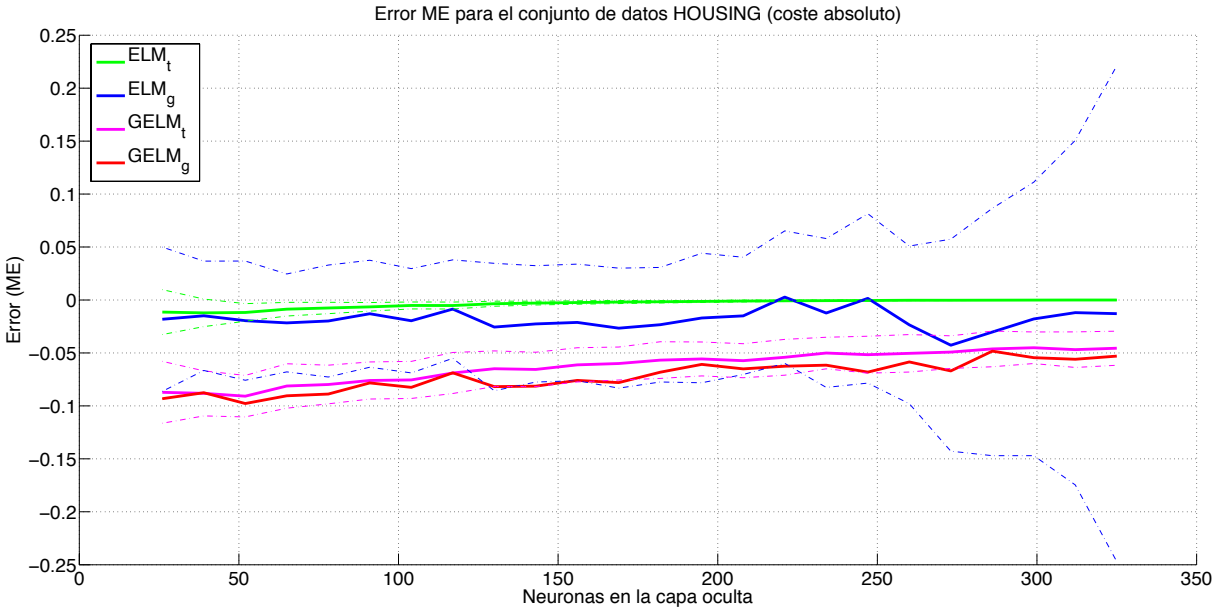

Figura 5.11: ME para el conjunto de datos *housing* con función de coste valor absoluto.

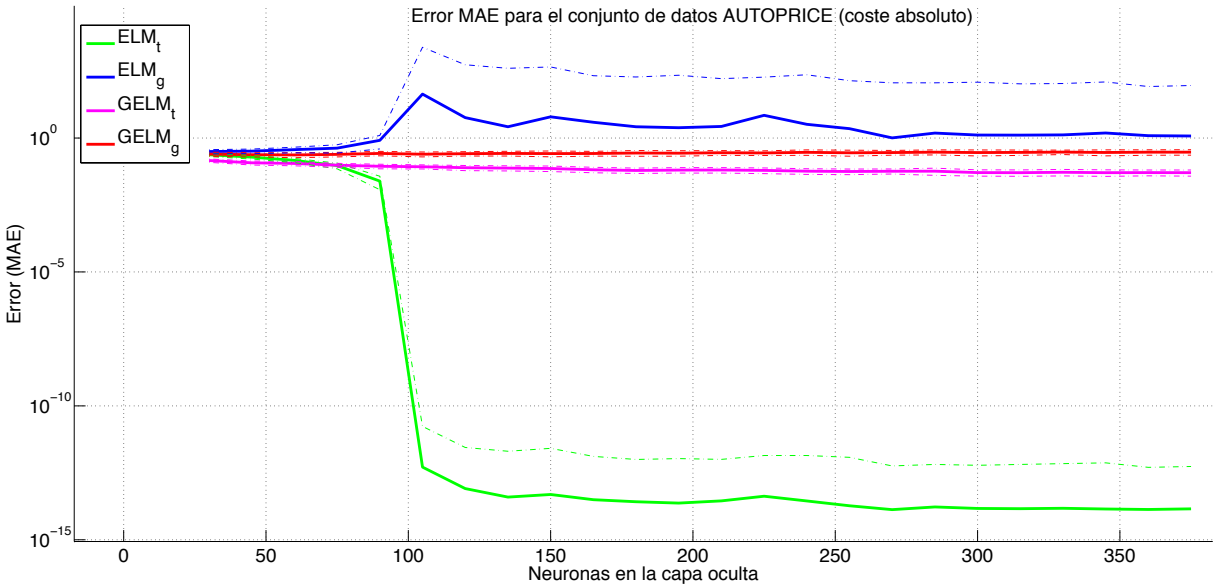

Figura 5.12: MAE para el conjunto de datos *autoprice* con función de coste valor absoluto.

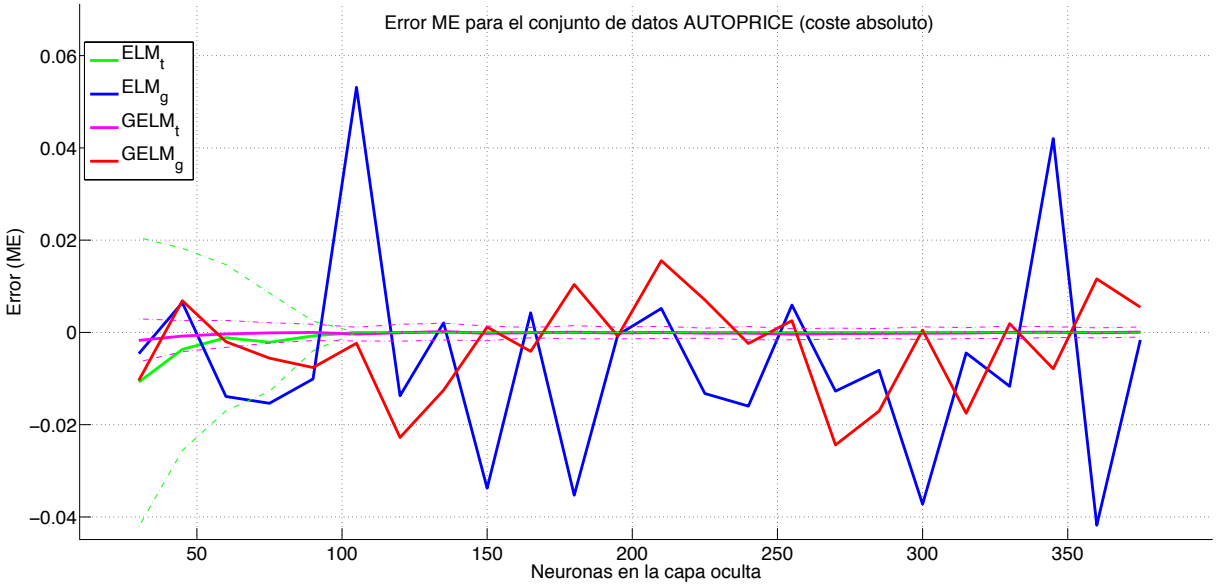

Figura 5.13: ME para el conjunto de datos *autoprice* con función de coste valor absoluto.

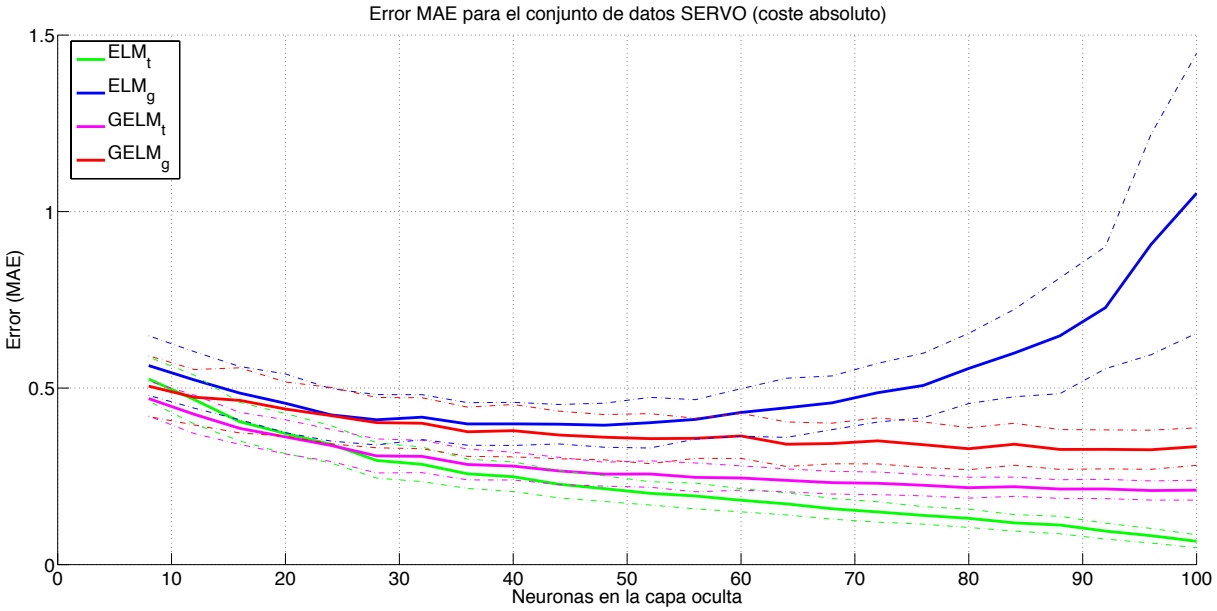

Figura 5.14: MAE para el conjunto de datos *servo* con función de coste valor absoluto.

Datos *Servo*.

En las figuras 5.14 y 5.15 se representan los errores de entrenamiento y validación que se han obtenido al ejecutar el algoritmo. Los resultados obtenidos con coste absoluto son similares a los obtenidos con coste cuadrático.

5.3. Resultados para funciones de coste regularizadas.

Estas funciones de coste hacen tender el valor de los pesos a 0, por lo que de esta manera, se pueden simplificar las estructuras eliminando los pesos más cercanos a 0.

En esta sección se han realizado 50 iteraciones en vez de 100 debido al alto coste computacional y al elevado tiempo de cálculo. Se han utilizado, en las funciones de coste, dos valores distintos de λ, $\lambda_1 = 0.001$ y $\lambda_2 = 0.1$.

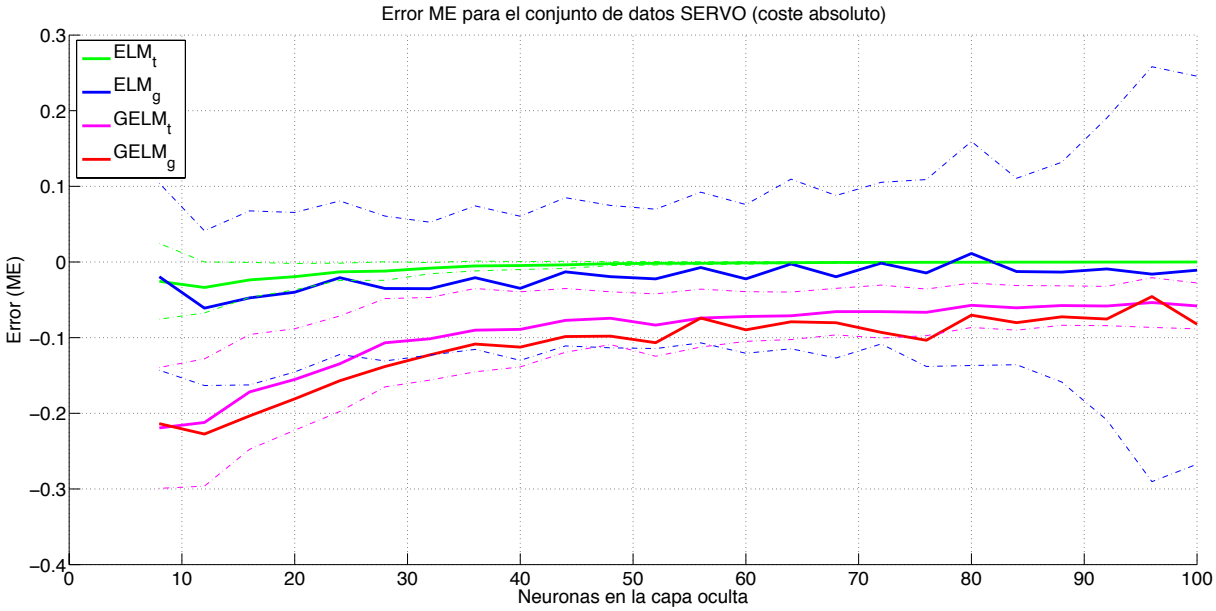

Figura 5.15: ME para el conjunto de datos *servo* con función de coste valor absoluto.

5.3.1. Función de coste regularizada cuadrática.

Datos *Housing*.

Se ha escogido este conjunto de datos para representar los resultados debido a que todas las gráficas obtenidas son similares. Las figuras 5.16 y 5.17 muestran el valor medio de los pesos en las 50 iteraciones realizadas para los casos de 325 y 26 neuronas en la capa oculta, respectivamente.

Como se puede observar en estas gráficas, la media del valor de los pesos, disminuye mucho.

Las figuras 5.18 y 5.19 muestran los índices de error obtenidos con las funciones de coste de regularización para el algoritmo ELM, el algoritmo *GELM* con λ_1 y el algoritmo *GELM* con λ_2, para 26 y 325 neuronas en la capa oculta.

Se puede ver que con pocos nodos, la *GELM* obtiene una solución ligéramente mejor que la ELM, y que con muchos nodos la *ELM* sobreentrena mientras que la *GELM* mantine el error constante.

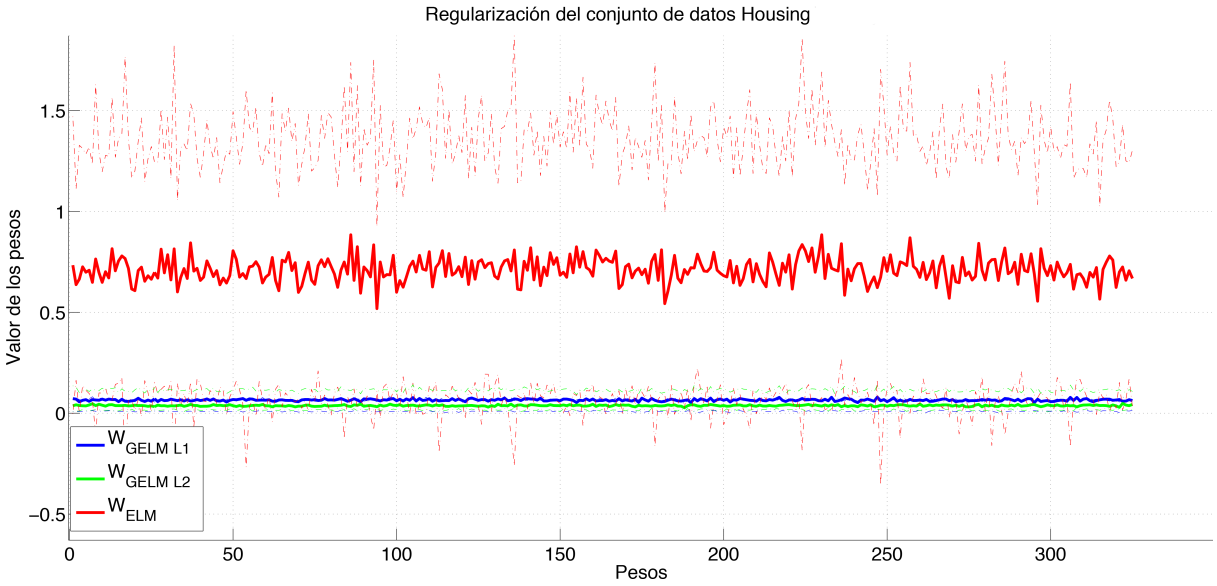

Figura 5.16: Valor medio de los pesos para 325 neuronas en la capa oculta con la función de coste regularizada cuadrática.

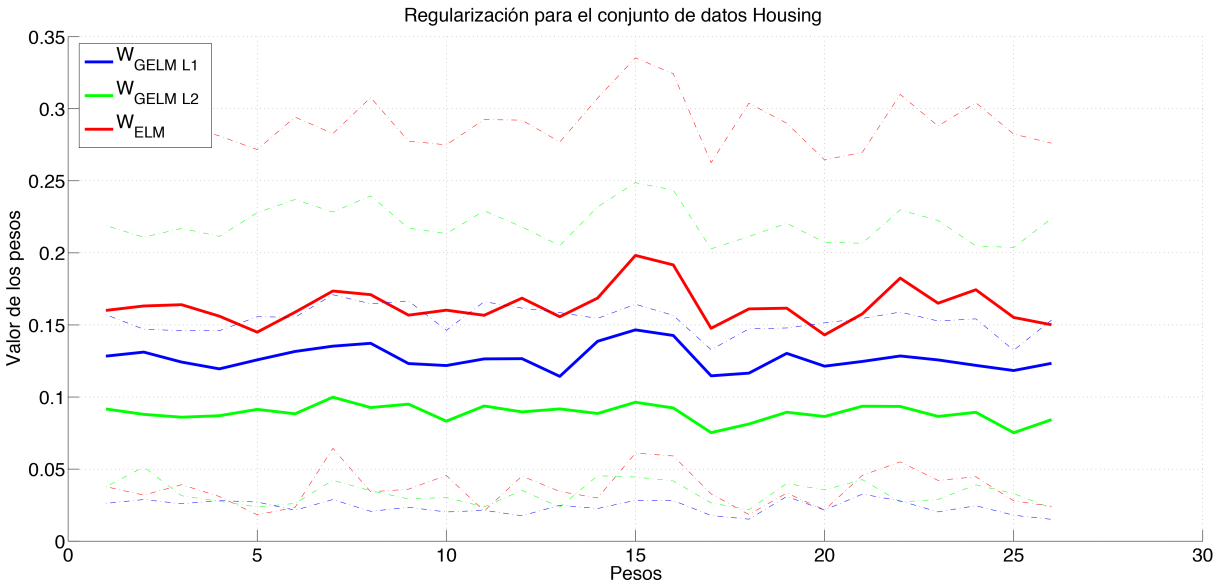

Figura 5.17: Valor medio de los pesos para 26 neuronas en la capa oculta con la función de coste regularizada cuadrática.

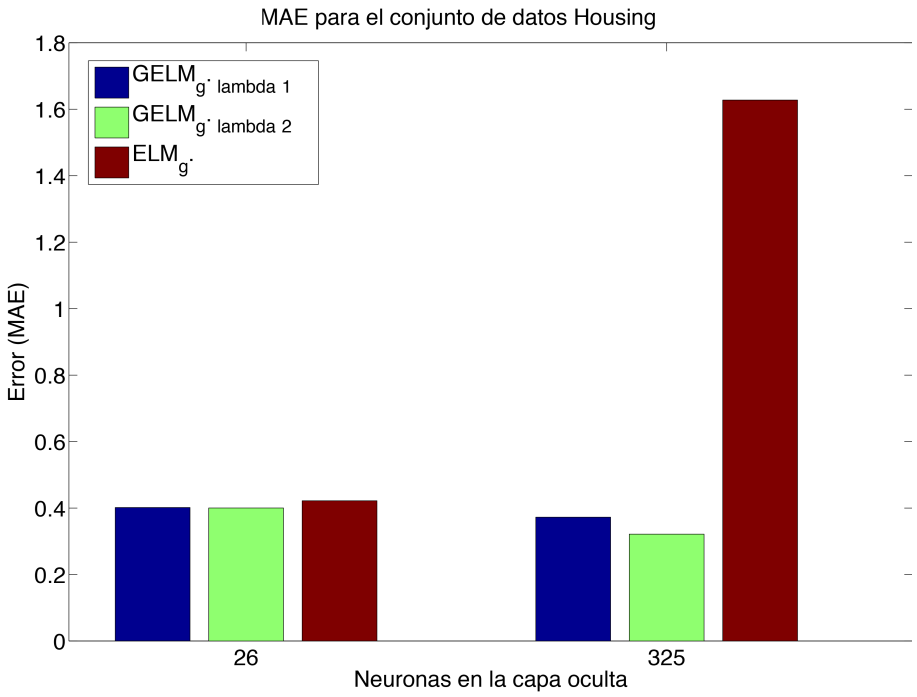

Figura 5.18: MAE de salida de los patrones utilizados en la generalización del sistema usando función de coste cuadrática para la regularización.

Los demás conjuntos tienen un comportamiento similar. Pueden consultarse las gráficas en el CD adjunto al proyecto.

5.3.2. Función de coste regularizada valor absoluto.

Datos *Housing*.

Se ha escogido el mismo conjunto de datos que en el caso anterior para, así, poder comparar ambas funciones de coste. Las figuras 5.20 y 5.21 muestran el valor medio de los pesos en las 50 iteraciones realizadas para los casos de 325 y 26 neuronas en la capa oculta respectivamente.

Los resultados son similares a los anteriores. Al aumentar el valor de λ, es decir, en el caso de λ_2, se observa una mayor disminución del valor medio de los pesos. Las figuras 5.22 y 5.23 muestran los índices de error obtenidos con las funciones de coste

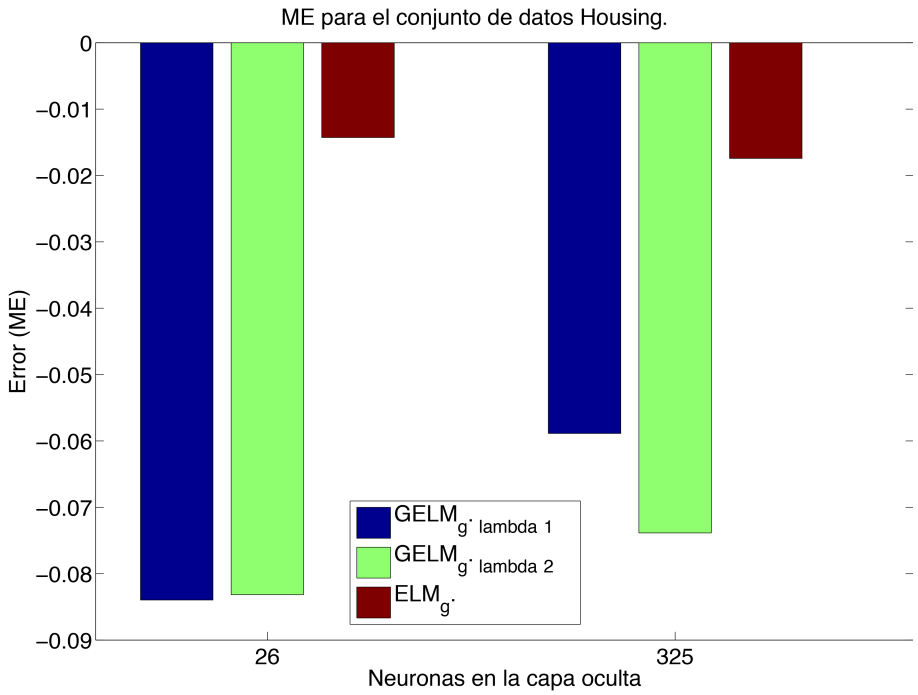

Figura 5.19: ME de salida de los patrones utilizados en la generalización del sistema usando función de coste cuadrática para la regularización.

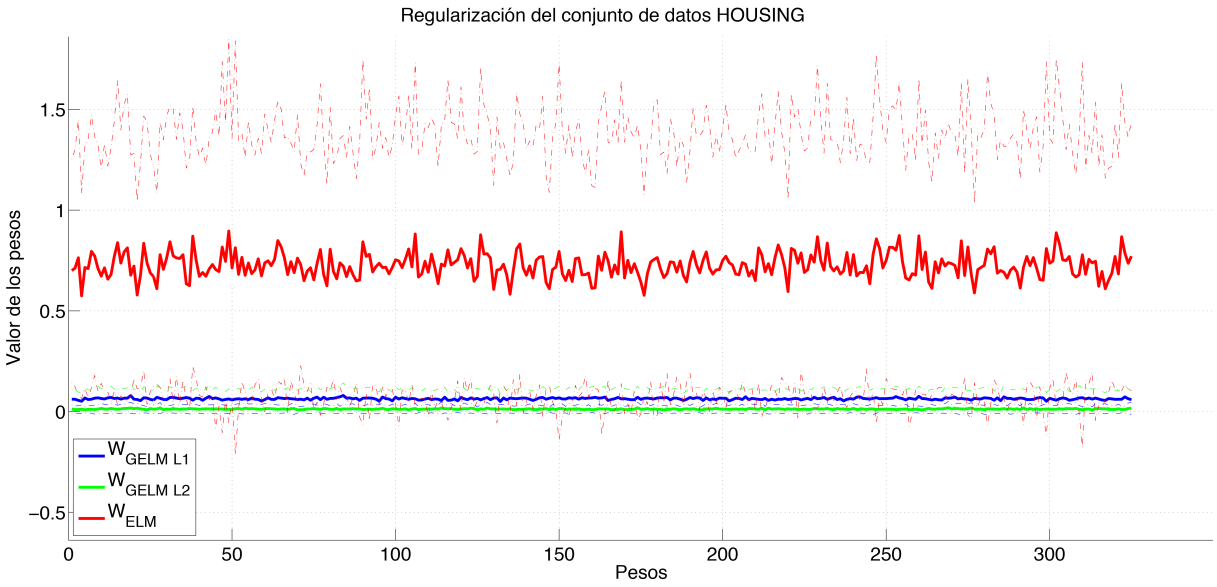

Figura 5.20: Valor medio de los pesos para 325 neuronas en la capa oculta con la función de coste regularizada valor absoluto.

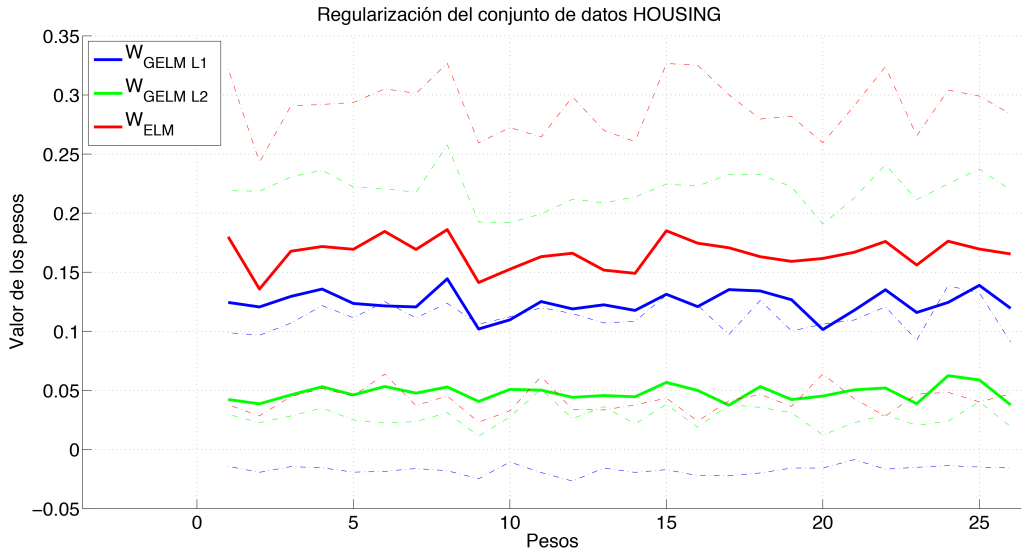

Figura 5.21: Valor medio de los pesos para 26 neuronas en la capa oculta con la función de coste regularizada valor absoluto.

de regularización para el algoritmo ELM, el algoritmo *GELM* con λ_1 y el algoritmo *GELM* con λ_2, para 26 nodos y 325 neuronas en la capa oculta.

Puede observarse que los errores son muy similares a los de la función de coste anterior.

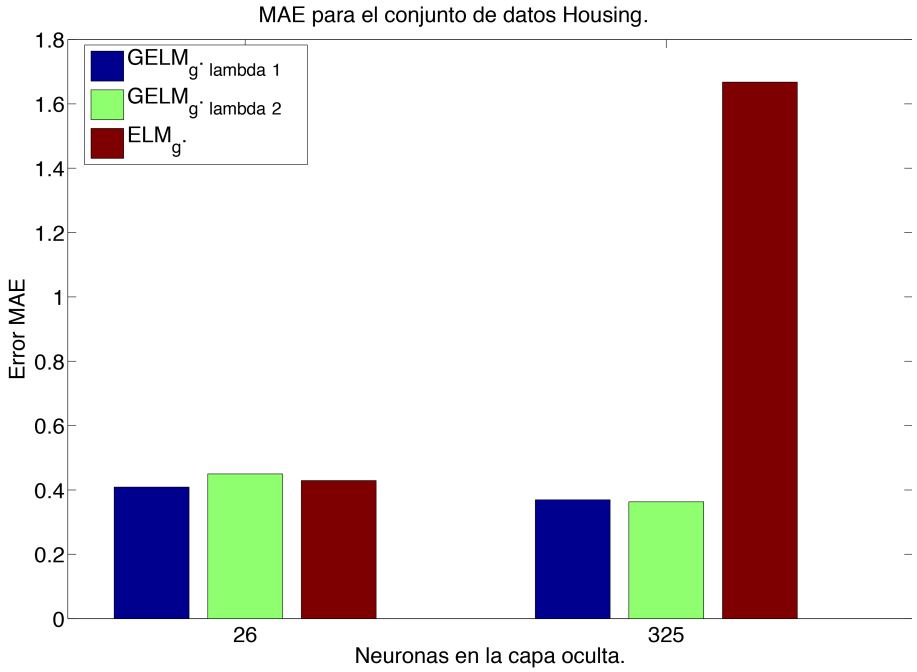

Figura 5.22: MAE de salida de los patrones utilizados en la generalización del sistema usando función de coste valor absoluto para la regularización.

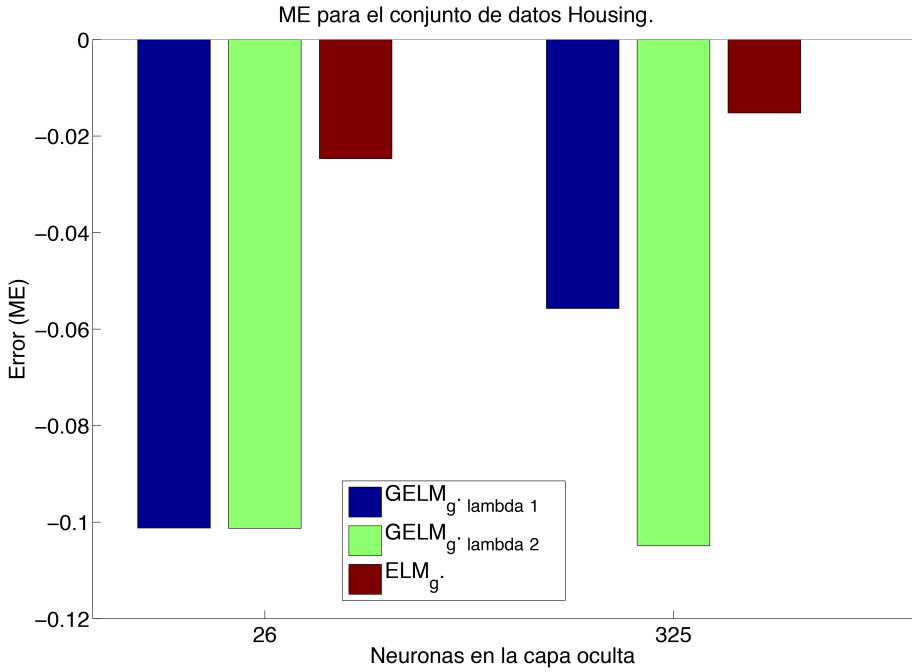

Figura 5.23: ME de salida de los patrones utilizados en la generalización del sistema usando función de coste valor absoluto para la regularización.

Capítulo 6

Conclusiones.

El algoritmo *GELM* logra obtener una solución mejor que el *ELM* en validación. Si se modeliza un sistema con el algoritmo *GELM* se obtendrá menos error de generalización. En cambio, el tiempo de computación aumenta enormemente al introducir algoritmos genéticos en una *ELM*, por lo que, si el conjunto de datos es grande, hay que plantearse reducir el subconjunto de entrenamiento.

Por otra parte, ante conjuntos de datos con los que nunca se ha trabajado, a la hora de ajustar el número de nodos de la capa oculta, si se utiliza el algoritmo *ELM* se tendrá un grave problema: si excede del número óptimo o se queda corto, el modelo sobreentrenará o entrenará poco y tendrá un error considerablemente alto. Por el contrario, al aplicar una *GELM*, invirtiendo un tiempo mucho mayor que la *ELM*, si se ajusta el número de nodos por exceso, se obtendrá una solución buena debido a que la *GELM* no sobreentrena, como se ha podido comprobar experimentalmente.

Bibliografía

[Blickle and Thiele, 1995] Blickle, T. and Thiele, L. (1995). *A comparison of selection schemes used in genetic algorithms.* Computer Engineering and Communication Network Lab (TIK), Gloriastrasse 35, 8092 Zurich, Switzerland.

[Cortez and Morais, 2007] Cortez, P. and Morais, A. (2007). A data mining approach to predict forest fires using meteorological data. *Department of Information Systems/RD Algoritmi Centre, University of Minho, 4800-058 Guimaraes, Portugal. http://www.dsi.uminho.pt/ pcortez.*

[David Harrison and Rubinfeld, 1978] David Harrison, J. and Rubinfeld, D. L. (1978). *Hedonic prices and the demand for clean air*, volume 5 of *81-102.* Journal of Environmental Economics and Management.

[Fayyad et al., 1996] Fayyad, U., Piatetsky-Shapiro, G., Smyth, P., and Uthurusamy, R. (1996). *Advanced Tecniques in Knowledge Discovery and Data Mining.* MIT Press, MA.

[Félix, 2002] Félix, L. C. M. (2002). *Data Mining: torturando a los datos hasta que confiesen.* UOC, http://www.uoc.edu/web/esp/art/uoc/molina1102/molina1102.html.

[Haykin, 1998] Haykin, S. (1998). *Neural Networks: A Comprehensive Foundation.* Prentice-Hall.

[Hoerl and Kennard, 1970] Hoerl, A. E. and Kennard, R. W. (1970). Ridge regression: Applications to nonorthogonal problems. *Technometrics 12*, 12(1):69–82.

[Huang et al., 2006] Huang, G. B., Zhu, Q. Y., and Siew, C. K. (2006). *Extreme learning machine: Theory and applications, Neurocomputing*, volume 70, 489-501.

[Laencina et al., 2009] Laencina, P. G., Monedero, R. V., Ruiz, J. L., Sánchez, J. M., and Gómez, J. L. S. (2009). Nuevas tendencias en redes neuronales artificiales:

Extreme learning machine. *Repositorio Digital de la Universidad Politécnica de Cartagena http://hdl.handle.net/10317/871.*

[Lahoz-Beltrà, 2004] Lahoz-Beltrà, R. (2004). *Bioinformática. Simulación, Vida Artificial e Inteligencia Artificial.* Díaz de Santos SA.

[Lara et al., 2011] Lara, R. M. M., Jumilla, M. C. B., Sánchez, J. M., and Gómez, J. L. S. (2011). Segmentación de imágenes mediante redes neuronales para el análisis de señales acústicas emitidas por cetáceos. *Repositorio Digital de la Universidad Politécnica de Cartagena, http://hdl.handle.net/10317/1751.*

[Lin et al., 1996] Lin, Ch.T., and Lee, G. (1996). *Neural Fuzzy Systems. A Neuro-Fuzzy Synergism to Intelligent Systems.* Prentice-Hall.

[López and Herrero, 2006] López, J. M. M. and Herrero, J. G. (2006). *Téecnicas de Análisis de Datos. Aplicaciones prácticas utilizando Microsoft Excel y Weka.* Universidad Carlos III, Madrid.

[Malathi et al., 2010] Malathi, V., Marimuthu, N., and Baskar, S. (2010). Intelligent approaches using support vector machine and extreme learning machine for transmission line protection. 73(10-12):2160–2167.

[Minhas et al., 2010] Minhas, R., Baradarani, A., Seifzadeh, S., and Wu, Q. J. (2010). Human action recognition using extreme learning machine based on visual vocabularies. *Neurocomputing*, 73(10-12):1906–1917.

[Mitchell, 1998] Mitchell, M. (1998). *An Introduction to Genetic Algorithms.* First MIT Press paperback edition.

[Nash et al., 1994] Nash, W. J., Sellers, T. L., Talbot, S. R., Cawthorn, A. J., and Ford, W. B. (1994). *The Population Biology of Abalone (Haliotis species) in Tasmania. I. Blacklip Abalone (H. rubra) from the North Coast and Islands of Bass Strait.* Sea Fisheries Division, Technical Report No. 48 (ISSN 1034-3288).

[Olabe, 1998] Olabe, X. B. (1998). *Redes Neuronales Artificiales y sus Aplicaciones.* Publicaciones de la Escuela Superior de Ingenieros de Bilbao.

[Rao and Mitra, 1972] Rao, C. R. and Mitra, S. K. (1972). *Generalized Inverse of Matrices and It's Applications.* Wiley.

[Serrano et al., 2009] Serrano, A., Soria, E., and Martín, J. (2009). *Redes Neuronales Artificiales*. OpenCourseWare Valencia, http://ocw.uv.es/ciencias/1-2/Course_listing.

[Sivanandam and Deepa, 2008] Sivanandam, S. and Deepa, S. (2008). *Introduction to Genetic Algorithms*. Springer.

[Sun et al., 2008] Sun, Z. L., Choi, T.-M., Au, K.-F., and Y.Yu (2008). Sales forecasting using extreme learning machine with applications in fashion retailing. *Decision Support Systems*, 46(1):411–419.

[Wang et al., 2008] Wang, G., Y-Zhao, and Wang, D. (2008). A protein secondary structure prediction framework based on the extreme learning machine. *Neurocomputing*, 72(1-3):7.

Anexo.

Aquí se añadirá todo el código utilizado en el proyecto. En la cabecera de cada una de las siguientes páginas aparece el nombre del archivo del que se trata.

```
clear
clc
close all


rep=10; % Repeticiones del algoritmo
IND_MAX=20; % Número máximo de indivíduos a testear
n_nodos=120; % Número de neuronas en la capa oculta FIJAMOS 120 PARA OPTIMIZAR LOS INDIVIDUOS
% Cargamos el conjunto de datos
load datos_housing
data=housing;
datan=zscore(data(1:end-1,:)); % Estandarizamos toda la matriz de datos
                               % Exceptuamos un patrón para que sean pares
% Definición de ENTRADAS y DESEADA
entradas=([1:13]);
deseada=14;

% Ponemos las ENTRADAS en X
X=datan(:,entradas);

% Ponemos la señal DESEADA en des
des=datan(:,deseada);

% N=Número de patrones  ##  L=Número de entradas
[N,L]=size(X);

IND_MAX=20; % Número máximo de individuos a testear
n_nodos=L*10; % Número de neuronas en la capa oculta


l=length(n_nodos); % Número de neuronas ocultas


% Inicializamos a 0 los errores del validación para la GELM
error_gelm_g=zeros(IND_MAX,l);


% Opciones del GA a modificar
options = gaoptimset;
% Modify options setting
options = gaoptimset(options,'Display', 'final');
options = gaoptimset(options,'MigrationFraction', 0.4);
options = gaoptimset(options,'Generations', 500);
options = gaoptimset(options,'StallGenLimit', 100);
options = gaoptimset(options,'CrossoverFcn', {  @crossoverheuristic [] });
options = gaoptimset(options,'PopInitRange', [-.5;.5]);
%% Bucle Repeticiones
for r=1:rep
%options = gaoptimset(options, 'MutationFcn',{@mutationuniform, rate});
%options = gaoptimset(options,'StallGenLimit', 150);
%options = gaoptimset(options,'Generations', 300);
for i=1:IND_MAX % Iteraciones IndivÃ-duos
tic
fprintf('####################\n')

    % Usamos un número de individuos igual al número de nodos
    % options = gaoptimset(options,'EliteCount', 2*n_nodos(m));
    options = gaoptimset(options,'PopulationSize', i*n_nodos);
    fprintf('Repetición %i/%i: Evaluando el error para %i x Num.Nodos Individuos...\n  ',r,rep, i);

%================================================================================
[Xent, Xgen, desent, desgen]=genera(X,des);

[N,L]=size(Xent);% Tamaño de Entradas para Entrenamiento
rho=zeros(N,n_nodos); % Inicialización de la matriz G (Matriz 3000xN.Neuronas)
pesos_oculta=1*randn(n_nodos,L+1); % Matriz N.Neuronasx20
```

```
 67
 68 %Obtención de matriz rho (G) para el entrenamiento
 69 for tt=1:N
 70     x=[1 Xent(tt,:)]';% Patrón de entrada precedido de un 1 (umbral) (20x1)
 71     sal=tanh(pesos_oculta*x); %FNL (20x20)*(20x1)
 72     rho(tt,:)=sal'; % Matriz (3000x20)
 73 end
 74
 75 % Planteamos Genéticos
 76 deseada=desent;
 77 FitnessFunction =@(w) coste_elm(w,rho,deseada); %@ designa variable a optimizar
 78 numberOfVariables =n_nodos;
 79
 80 [w,fval] = ga(FitnessFunction,numberOfVariables,options);
 81
 82
 83 [N,L]=size(Xgen); % (2000x1)
 84 rhog=zeros(N,n_nodos); %inicialización de la matriz G (Matriz 3000xN.Neuronas)
 85
 86 % Obtención de matriz rhog (G) Para la generalización
 87 for tt=1:N
 88     x=[1 Xgen(tt,:)]';% Patrón de entrada precedido de un 1 (umbral) (20x1)
 89     sal=tanh(pesos_oculta*x); %FNL (20x20)*(20x1)
 90     rhog(tt,:)=sal';  % Matriz (2000x20)
 91 end
 92
 93  % CÁLCULO DE LAS SALIDAS GENERALIZACIÓˆN GELM
 94  salidas=rhog*w'; % (2000x1) Una salida por patrón
 95  error_gelm_g(i)=mean(abs(desgen-salidas));
 96
 97 %================================================================================
 98
 99     fprintf('\n');
100
101 toc
102
103 end
104 [error_minimo(r),individuo_mejor(r)]=min(error_gelm_g);
105 end
106 %% Representamos soluciones
107 %mean_error_gelm_g=mean(error_gelm_g, 1); %media GELM generalización
108 %std_error_gelm_g=std(error_gelm_g, 1); %std ELM generalización
109 clc
110 % Representamos el error MAE
111 figure
112 plot(error_minimo, 'r')
113
114 fprintf('####################\n')
115 fprintf('El mejor número de Individuos obtenido para %i nodos es de %i x Num.Nodos
\n', n_nodos,individuo_mejor);
116 fprintf('####################\n')
117
```

```
clear
clc
close all

%% Preprocesado de datos y características del conjunto de datos
% Cargamos el conjunto de datos
load datos_housing
data=housing;
datan=zscore(data(1:end-1,:)); % Estandarizamos toda la matriz de datos exceptuando un patrón
% Definición de columnas ENTRADAS y DESEADA
entradas=[1:13];
deseada=14;

% Ponemos las ENTRADAS en X
X=datan(:,entradas);
% Ponemos la señal DESEADA en des
des=datan(:,deseada);

%% Opciones e inicializaciones
% N=Número de patrones  ##  L=Número de entradas
[N,L]=size(X);

NR=100; % Número de repeticiones del experimento
n_nodos=2*L:L:25*L; % Número de neuronas en la capa oculta
l=length(n_nodos); % Número de pruebas de neuronas ocultas

% Inicializamos a 0 los errores para la ELM y la GELM
error_gelm_t=zeros(NR,l);
error_elm_t=error_gelm_t;
error_gelm_g=zeros(NR,l);
error_elm_g=error_gelm_g;

%% Opciones del GA a modificar
options = gaoptimset; % Inicialización
options = gaoptimset(options,'Display', 'off');
options = gaoptimset(options,'Generations', 800);
options = gaoptimset(options,'StallGenLimit', 100);
options = gaoptimset(options,'CrossoverFcn', {  @crossoverheuristic [] });
options = gaoptimset(options,'PopInitRange', [-.5;.5]);

%% Bucle principal
for k=1:NR % Repeticiones del entrenamiento
  fprintf('####################\n')
  for m=1:l % Iteramos para distinto número de nodos
    tic % Inicia cuenta de tiempo
    options = gaoptimset(options,'PopulationSize', 15*n_nodos(m));
    fprintf('IteraciÃ³n %i para %i nodos...\n  ', k, n_nodos(m));
    % FunciÃón principal;
    [salida_elm_t(m,:,k), salida_elm_g(m,:,k), salida_gelm_t(m,:,k), salida_gelm_g(m,:,k), des_t(m,:,k), des_g(m,:,k)]=minimiza_error(X, des, n_nodos(m), options);
    fprintf('  ');
    toc % Termina cuenta de tiempo y saca por la command window el tiempo transcurrido
    fprintf('\n');
  end
end

%% Salvado de datos y salida
save datos_obtenidos salida_elm_t salida_elm_g salida_gelm_t salida_gelm_g des_t des_g n_nodos
quit % Cerrar matlab. Libera memoria en servidor
```

```
 1 function [salida_elm_t, salida_elm_g, salida_gelm_t, salida_gelm_g, des_t, des_g]↵
=minimiza_error(X, des, n_nodos, options)
 2
 3 % Dividimos el conjunto de datos en Entrenamiento y Generalización
 4 [Xent, Xgen, desent, desgen]=genera(X,des);
 5
 6 [N,L]=size(Xent);% Tamaño de Entradas para Entrenamiento
 7 rho=zeros(N,n_nodos); % Inicialización de la matriz G (Matriz 3000xN.Neuronas)
 8 pesos_oculta=1*randn(n_nodos,L+1); % Matriz N.Neuronasx20
 9
10 %Obtención de matriz rho (G) para el entrenamiento
11 for tt=1:N
12     x=[1 Xent(tt,:)]';% Patrón de entrada precedido de un 1 (umbral) (20x1)
13     sal=tanh(pesos_oculta*x); %FNL (20x20)*(20x1)
14     rho(tt,:)=sal'; % Matriz (3000x20)
15 end
16
17 % Planteamos ELM
18 welm=pinv(rho)*desent; % Cálculo de la Pseudoinversa de G (20x3000)(3000x1)
19
20 % CÁLCULO DE LAS SALIDAS ENTRENAMIENTO ELM
21 salida_elm_t=rho*welm; % (3000x1) Una salida por patrón
22
23 % Planteamos Genéticos
24 deseada=desent;
25 FitnessFunction =@(w) coste_elm(w,rho,deseada); %@ designa variable a optimizar
26 numberOfVariables=n_nodos;
27
28 %options = gaoptimset(options,'InitialPopulation',[welm+1*(rand(n_nodos,1)-0.5)]'); %↵
Intento aproximar pesos...
29 [w,fval] = ga(FitnessFunction,numberOfVariables,options);
30
31 % CÁ□LCULO DE LAS SALIDAS ENTRENAMIENTO GELM
32 salida_gelm_t=rho*w'; % (3000x1) Una salida por patrón
33
34 [N,L]=size(Xgen); % (2000x1)
35 rhog=zeros(N,n_nodos); %inicialización de la matriz G (Matriz 3000xN.Neuronas)
36
37 % Obtención de matriz rhog (G) Para la generalización
38 for tt=1:N
39     x=[1 Xgen(tt,:)]';% Patrón de entrada precedido de un 1 (umbral) (20x1)
40     sal=tanh(pesos_oculta*x); %FNL (20x20)*(20x1)
41     rhog(tt,:)=sal';  % Matriz (2000x20)
42 end
43  % CÁLCULO DE LAS SALIDAS DE GENERALIZACIÓN DE LA ELM
44  salida_elm_g=rhog*welm; % (2000x1) Una salida por patrón
45
46  % CÁLCULO DE LAS SALIDAS DE GENERALIZACIÓN DE LA GELM
47  salida_gelm_g=rhog*w'; % (2000x1) Una salida por patrón
48
49  des_t=desent;
50  des_g=desgen;
51 end
```

```
function [Xent, Xgen, desent, desgen]=genera(X,des)


 % GENERAMOS CONJUNTOS DIFERENTES DE ENTRENAMIENTO Y VALIDACIÓN

 [dd,ee]=size(X);
 r=randperm(dd);
 l=fix(2*dd/3);

 Xent=zeros(l,ee);
 desent=zeros(l,1);
 Xgen=zeros(dd-l,ee);
 desgen=zeros(dd-l,1);



 % CONJUNTOS ENTRENAMIENTO

 for kk=1:l

     Xent(kk,:)=X(r(kk),:);
     desent(kk)=des(r(kk));


 end

i=1;


 % CONJUNTOS VALIDACIÓN
 for kk=l+1:dd


     Xgen(i,:)=X(r(kk),:);
     desgen(i)=des(r(kk));
     i=i+1;


 end
```

```
1  function y = coste_elm(w,rho,deseada)
2
3  % En esta parte generamos el coste a minimizar;
4  y=mean((deseada-rho*w').^2);
5  %y=mean(abs(deseada-rho*w'));
6
7  end
```