UNIVERSIDAD DE GRANADA

E.T.S. de Ingeniería Informática

Universidad
de Granada

Departamento de

Ciencias de la Computación e Inteligencia Artificial

Modelos de Regresión Basados en Redes Neuronales de Unidades Producto Diseñadas y Entrenadas Mediante Algoritmos de Optimización Híbrida. Aplicaciones

Tesis Doctoral

Alfonso Carlos Martínez Estudillo

Marzo de 2005

Universidad
de Granada

Modelos de Regresión Basados en Redes Neuronales de Unidades Producto Diseñadas y Entrenadas Mediante Algoritmos de Optimización Híbrida. Aplicaciones

MEMORIA QUE PRESENTA

Alfonso Carlos Martínez Estudillo

PARA OPTAR AL GRADO DE DOCTOR EN INFORMÁTICA

MARZO 2005

DIRECTORES

César Hervás Martínez

Francisco José Martínez Estudillo

DEPARTAMENTO DE

CIENCIAS DE LA COMPUTACIÓN E INTELIGENCIA ARTIFICIAL

E.T.S DE INGENIERIA INFORMÁTICA UNIVERSIDAD DE GRANADA

La memoria titulada **Modelos de regresión basados en redes neuronales de unidades producto diseñadas y entrenadas mediante algoritmos de optimización híbrida. Aplicaciones,** que presenta D. Alfonso Carlos Martínez Estudillo para optar al grado de doctor, ha sido realizada en el Departamento de Ciencias de la Computación e Inteligencia Artificial de la Universidad de Granada bajo la dirección de los doctores D. César Hervás Martínez y D. Francisco José Martínez Estudillo.

Granada, Marzo de 2005

El doctorando:

Alfonso Carlos Martínez Estudillo

Los directores:

César Hervás Martínez Francisco José Martínez Estudillo

Tesis Doctoral parcialmente subvencionada por la

Comisión Interministerial de Ciencia y Tecnología

Con el proyecto TIC2001-1143-C03-02

TIC2002-04036-C05-02 e INSA-ETEA

CYCIT

TIC2001-1143-C03-02

TIC2002-04036-C05-02

A M^a José

A Pedro

Agradecimientos

Dicen que *"Es de bien nacidos ser agradecidos"*. En primer lugar, quiero agradecer a mi director de tesis, César Hervás, quien un día, en el Puerto de Sta. María, despertó mi inquietud investigadora y me animó para realizar esta tesis doctoral. Su confianza, paciencia e ilusión son difíciles de superar. También quiero agradecer a mi otro director de tesis, Francisco José Martínez, hermano y compañero; por su apoyo, en especial en los malos momentos, su disciplina, sistemática matemática y afán por explicar el por qué de las cosas me han ayudado especialmente para la realización de este trabajo de tesis.

Al grupo de investigación AYRNA, que me ha dotado del ambiente e infraestructura idóneo para la realización de esta tesis. Gracias por vuestra acogida, cercanía y apoyo.

A INSA-ETEA en general, y a mis compañeros de trabajo en particular, por su ayuda, apoyo y especial motivación.

A los grandes sufridores de mis ausencias, mis madrugones y retrasos. Por su comprensión, ánimo y cariño. Ellos son mi principal fuente de motivación. Mami, chico, ¡papá ya la ha terminado!.

A mi padre quien desde pequeño, me inculcó el amor por las matemáticas. A mi madre, en quien siempre encontré la palabra de ánimo. A mis hermanos, Carmen, Juan Antonio y Adela por ser como son. A Joaquina, Pepe y Juanra.

A mi familia política, Rosy, Lourdes, Silvia. Gracias por esos cafés a las siete y media. Alfonso gracias por estar siempre ahí.

A los aires que respiré en el Colegio Mayor Loyola, en especial, a mi amigo Juan Vázquez, compañero de aventuras. Al hermano Martínez, él me ayudó a estar donde estoy.

A todos los amigos que descuidé durante la realización de este trabajo de tesis, os prometo que recuperaremos el tiempo perdido.

Gracias a todos.

"...sin juramento me podrás creer que quisiera que este libro, como hijo del entendimiento, fuera el más hermoso, el más gallardo y más discreto que pudiera imaginarse. Pero no he podido yo contravenir al orden de naturaleza, que en ella cada cosa engendra a su semejante...."

Miguel de Cervantes Saavedra del Prólogo del Quijote

Índice

Índice de Figuras

Índice de Tablas

1 Introducción

El modelado de sistemas es, en la actualidad, uno de los problemas de mayor interés en numerosas ramas de la ciencia. Los modelos son abstracciones de la realidad que pueden ser aplicados para mejorar nuestra comprensión de fenómenos en el mundo. En diversas áreas de aplicación, como la economía, ingeniería, biología, sociología y medicina, surge el problema de establecer una relación funcional entre las diferentes variables que intervienen en el fenómeno que se está estudiando.

De forma general, existen dos maneras de abordar el problema. La primera es de carácter deductivo. Básicamente, el método consiste en la división del sistema en subsistemas que son modelados a partir de leyes físicas y relaciones previamente aceptadas como hipótesis, usando como herramienta fundamental las ecuaciones diferenciales. La segunda aproximación es de naturaleza inductiva, ya que realiza la estimación de los modelos a partir del análisis y medida de los datos. Este proceso de estimación se denomina comúnmente aprendizaje. En este

contexto, aprendizaje significa obtener una representación de los datos que pueda ser usada para diferentes objetivos como la predicción o la clasificación.

Las redes neuronales artificiales (Bishop, 1995; Haykin, 1994), que constituyen una de las áreas de la ingeniería del conocimiento que más se han desarrollado en los últimos años, pueden situarse dentro de esta forma de abordar el problema de modelado. Sus propiedades y características han hecho de ellas una herramienta usada con frecuencia en la resolución con éxito de problemas reales de gran complejidad, pertenecientes a diferentes áreas, como por ejemplo, el reconocimiento de patrones, diagnóstico médico, modelado de datos financieros, predicción de datos de crecimiento microbiano, etc.

Entre las diferentes arquitecturas de redes neuronales es preciso destacar, por su importancia, las redes basadas en funciones de base sigmoide o perceptrón multicapa (MLP). Estas redes poseen una importante propiedad: la familia de funciones reales que representan es un subconjunto denso en el espacio de las funciones continuas definidas sobre un compacto con la convergencia uniforme. Esta propiedad de densidad significa que pueden aproximar cualquier función continua con suficiente precisión, con tal de que el número de nodos ocultos no esté acotado, y proporciona una sólida base de carácter teórico que ha sido esencial para el desarrollo del estudio y de las aplicaciones de estas redes (Cybenko, 1989; Hornik et al., 1989; Leshno et al., 1993).

La capacidad de aproximación de las redes neuronales artificiales de base sigmoide, ha hecho de ellas una herramienta alternativa para la resolución de problemas de modelado. Sin embargo, esta propiedad de carácter teórico presenta numerosas limitaciones en la práctica. En primer lugar, el número de nodos ocultos de la red necesarios para obtener una buena aproximación suele ser muy alto, lo que provoca la escasa interpretabilidad de los modelos obtenidos. Esto supone un grave inconveniente en muchas áreas de investigación, en las que la comprensión del modelo de regresión es tan importante como la precisión

alcanzada en la aproximación. Por otra parte, los diferentes resultados teóricos sobre la capacidad de aproximación de las redes neuronales, son teoremas de existencia que no construyen la función que aproxima. La construcción de esta función no es tarea sencilla, ya que implica determinar la estructura de la red y los pesos de las conexiones necesarios para conseguir un determinado nivel de aproximación. A este hecho hay que unir que, con frecuencia, los algoritmos de entrenamiento que se utilizan para obtener los pesos de la red, o equivalentemente los coeficientes del modelo, tienen el riesgo de quedar atrapados en óptimos locales de una superficie de error que con frecuencia está plagada de ellos y que además es posible que tenga regiones planas.

Como alternativa a las redes neuronales basadas en unidades sigmoides, se han desarrollado redes con otros tipos de funciones de base como las funciones gaussianas (Kosko, 1991), funciones de base radial (Lee and Hou, 2002), regresión mediante la proyección[1] (Friedman and Stuetzle, 1981), redes de regresión generales[2] (Specht, 1991; Tomandl and Schober, 2001), enlaces funcionales[3] (Pao, 1989), etc.

Entre las diferentes tipologías de funciones de base, destacaremos en este trabajo de tesis el modelo de red neuronal basado en unidades producto (PU) introducido por Durbin y Rumelhart en 1989 (Durbin and Rumelhart, 1989). Se trata de redes neuronales en las que la función de computación de los nodos de la capa oculta en lugar de ser de tipo aditivo, es de tipo multiplicativo, y viene dada por:

[1] Proyection pursuit regression

[2] General Regression Neural Networks, GRNN

[3] Functional links

$$x_1^{w_1} x_2^{w_2} \cdots x_k^{w_k},$$

donde x_1, x_2,$\cdots$,x_k son variables de entrada de la red y los exponentes (números reales) w_1, w_2,$\cdots$,w_k de las variables son coeficientes del modelo. Si se considera que las funciones de salida de los nodos de las capas ocultas son iguales a la función identidad, el modelo funcional obtenido por una red de este tipo es:

$$f(x_1, x_2, ..., x_k) = \sum_{j=1}^{m} \beta_j \left(\prod_{i=1}^{k} x_i^{w_{ji}} \right)$$

Las redes PU presentan las siguientes ventajas:

- La posibilidad de usar exponentes negativos permite expresar cocientes y ratios entre las variables.
- Los exponentes del modelo son números reales. Esta característica tiene especial importancia, sobre todo si se tiene en cuenta que son frecuentes las situaciones en el modelado de datos reales en las que la relación entre las variables responde a una estructura de tipo potencial donde las potencias no están restringidas a los números naturales o enteros (Saito and Nakano, 1997a; Saito and Nakano, 1997b).
- Permiten implementar funciones polinómicas de orden superior. De esta forma, cuando existen interacciones entre las variables que intervienen en un determinado fenómeno, las redes neuronales basadas en unidades producto permiten obtener modelos más simplificados que las redes neuronales de tipo sigmoide.
- A diferencia de lo que ocurre con las redes de tipo sigmoide o de base radial, las funciones derivadas parciales del modelo obtenido a partir de una red neuronal PU son funciones del mismo tipo. Este hecho ayuda con frecuencia al estudio de las tasas de cambio de la variable dependiente del modelo con respecto a cada una de las variables, facilitando la interpretabilidad de los modelos.

Sin embargo, las redes neuronales basadas en unidades producto presentan un inconveniente importante, la superficie de error asociada es especialmente compleja (Engelbrecht and Ismail, 1999), con numerosos óptimos locales y regiones planas. La estructura potencial del modelo provoca que pequeños cambios en los exponentes tengan como consecuencia un cambio significativo en los valores de la función y en la función de error. Así, los algoritmos de entrenamiento de redes basados en el gradiente quedan con frecuencia, y de forma especial en este tipo de redes neuronales, atrapados en óptimos locales. Por ejemplo, está estudiado el hecho de que el clásico algoritmo de retropropagación no funciona bien en este tipo de redes (Janson and Frenzel, 1993).

A nuestro juicio, esta dificultad relacionada con el entrenamiento, es una de las razones por las que, a pesar de las ventajas anteriormente señaladas, la teoría de redes neuronales basadas en unidades producto ha tenido poco desarrollo y han sido menos utilizadas como herramientas para problemas de clasificación y de regresión que otros tipos de redes neuronales.

Una alternativa interesante al entrenamiento de redes neuronales con métodos de tipo gradiente, es el uso de algoritmos evolutivos que reduzcan la probabilidad de que el proceso de búsqueda de la solución quede atrapado en óptimos locales lejanos del óptimo global. Los algoritmos evolutivos son un tipo de algoritmos de búsqueda estocástica que permiten encontrar la solución en espacios complejos. Numerosos investigadores (Houck *et al.*, 1996; Houck *et al.*, 1997; Joines *et al.*, 2000; Michalewicz and Schoenauer, 1996) han demostrado que los algoritmos evolutivos son una herramienta adecuada para la búsqueda global, ya que son capaces de hallar rápidamente regiones del espacio de búsqueda en las que se encuentran los óptimos locales del problema o bien la solución global. Dichos algoritmos se han utilizado con éxito en el campo de las redes neuronales para encontrar la estructura adecuada de la red y determinar el valor de los pesos de las conexiones. Existen diversos trabajos que versan sobre el aprendizaje y

diseño de redes neuronales de tipo sigmoide (Angeline et al., 1994; García-Pedrajas et al., 2002; Yao and Liu, 1997) utilizando la computación evolutiva. La principal ventaja de estos algoritmos es que a la vez que van diseñando la estructura de la red, van entrenando y ajustando los pesos de las conexiones con el objeto de minimizar la función de error. Por la propia naturaleza de la computación evolutiva basada en una población de individuos, el espacio de búsqueda se amplía, reduciendo drásticamente el riesgo de quedar atrapado en un óptimo local alejado del global.

En lo referente a redes neuronales basadas en unidades producto, no tenemos constancia de ningún trabajo en el que se apliquen las técnicas de computación evolutiva al diseño de la estructura y al entrenamiento de una red neuronal de este tipo. Sólo hemos encontrado en la revisión bibliográfica realizada un trabajo (Janson and Frenzel, 1993), en el que se aplica un algoritmo genético estándar para el entrenamiento de una red neuronal de tipo producto, fijada previamente la estructura de la red neuronal.

Uno de los objetivos de partida y de las aportaciones de esta memoria será, como detallaremos a continuación, el desarrollo de algoritmos evolutivos como herramienta para el diseño y entrenamiento de redes neuronales basadas en unidades producto.

Sin embargo, los algoritmos evolutivos presentan también diversos inconvenientes, entre los que merece destacar el siguiente. Los algoritmos evolutivos (AEs) son eficientes explorando el espacio de búsqueda en un problema de optimización, si bien está probada su ineficiencia para encontrar el óptimo local dentro de la región del espacio en la que el algoritmo converge, ya que la convergencia hacia el óptimo correspondiente es, con frecuencia, muy lenta.

Por otra parte, se sabe que determinados procedimientos, conocidos como procedimientos de mejora local (LIPs), son capaces de encontrar con precisión el óptimo local cuando la búsqueda se realiza en una pequeña región del espacio.

Las nuevas metodologías, basadas en la combinación de los algoritmos evolutivos como buscadores globales y los procedimientos de búsqueda local, se denominan comúnmente algoritmos híbridos. Numerosas investigaciones utilizan los algoritmos híbridos para realizar esta combinación entre la búsqueda global y local, permitiendo así mejorar la precisión de los algoritmos evolutivos clásicos. La combinación de las estrategias de búsqueda global con los procedimientos de búsqueda o mejora local es la clave para encontrar de manera eficiente soluciones de calidad. Es necesario, por tanto, encontrar un equilibrio entre los procesos de exploración del espacio de soluciones y la explotación de las regiones más prometedoras que establezca un compromiso entre las búsquedas globales y locales.

En los últimos años ha habido un amplio desarrollo de algoritmos híbridos para la resolución de problemas de optimización. Una revisión bibliográfica muestra la variedad de métodos y de técnicas implementadas, siempre basadas en la adecuada combinación de la búsqueda global realizada por un algoritmo evolutivo y la búsqueda local realizada con la metodología correspondiente.

En esta memoria, presentamos una propuesta diferente de método híbrido para el diseño y entrenamiento de una red neuronal basada en unidades producto para la resolución de un problema de regresión.

1.1 Objetivos y aportaciones

El objetivo general trazado para el desarrollo de esta memoria ha sido profundizar en el estudio de un tipo de redes neuronales artificiales: las redes neuronales basadas en unidades producto, propuestas en 1989 por Durbin y Rumelharth (Durbin and Rumelhart, 1989), y que sin embargo no han tenido posteriormente en la literatura científica el mismo desarrollo que otro tipo de redes neuronales como el perceptrón multicapa o las redes de base radial.

De forma más precisa, el objetivo general fijado en los primeros pasos de la investigación fue el uso de las redes neuronales de tipo producto como herramienta para la resolución de problemas de regresión.

Para abordar este problema, se optó por utilizar algoritmos evolutivos para el diseño de la estructura y el entrenamiento de este tipo de redes. Tras revisar la bibliografía existente, comprobamos que los métodos de entrenamiento de estas redes consistían sobre todo en métodos de tipo gradiente y alguna aplicación de un algoritmo genético estándar o del método de optimización basado en enjambre de partículas, siempre aplicados a problemas sencillos de baja dimensionalidad. Como se ha comentado con anterioridad, los algoritmos de tipo gradiente, aún en sus versiones más refinadas y complejas, quedaban con frecuencia atrapados en óptimos locales.

Por este motivo, en una primera fase se construyó un algoritmo evolutivo adaptado especialmente a las redes PU (Martínez-Estudillo et al., 2002). Posteriormente, se construyó un algoritmo híbrido con el objetivo de afinar y mejorar los resultados obtenidos por el algoritmo evolutivo.

En cualquiera de los casos, los algoritmos diseñados se han utilizado para la resolución de problemas de regresión, tanto con datos artificiales como con datos reales.

A continuación, se expresan de forma resumida las aportaciones fundamentales incluidas en esta memoria.

- **Primera aportación:**

  Hemos realizado una revisión bibliográfica exhaustiva de las redes neuronales basadas en unidades producto. Es importante señalar que son escasos y bastante dispersos los trabajos relacionados con este tipo de redes neuronales desde su aparición en 1989.

- **Segunda aportación:**

  Se han estudiado las propiedades de las redes neuronales basadas en unidades producto. Los trabajos de carácter teórico sobre este tipo de redes se reducen a los realizados por Schmitt en (Schmitt, 2001) centrados en el estudio de la dimensión de Vapnik-Chervonenkis (VC) y de la pseudo dimensión para redes de orden superior y basadas en unidades producto.

  Concretamente, el primer paso dado en el estudio teórico es la demostración de que las redes neuronales basadas en unidades producto son aproximadores universales, al igual que otros tipos de redes neuronales, como las redes neuronales basadas en unidades sigmoide y las redes de base radial. Usando el Teorema de Stone-Weierstrass demostramos que la familia de funciones obtenida a partir de las redes PU es un subconjunto denso dentro de las funciones continuas. Esta propiedad de aproximación proporciona una sólida base teórica necesaria para el uso de estas redes como herramienta de garantía para la resolución de problemas de regresión.

  Por otra parte, también se han estudiado algunas propiedades de tipo analítico y geométrico que diferencian a las redes PU del resto de modelos. Se ha realizado además el estudio comparado de estas propiedades con las redes neuronales de base sigmoide, siendo especialmente interesantes las diferencias obtenidas al analizar la curvatura de Gauss y la concavidad y convexidad en el caso bidimensional.

- **Tercera aportación:**

  Hemos construido un algoritmo evolutivo para el diseño y entrenamiento de una red neuronal basada en unidades producto

aplicado a la resolución de problemas de regresión (Martínez-Estudillo et al., 2005c). De esta forma, se proporciona una alternativa a los métodos basados en el gradiente usados hasta ahora para el entrenamiento de las redes PU. Estos métodos han resultado en general poco eficientes para problemas complejos debido a la rugosidad de la superficie de error asociada a este tipo de redes.

El algoritmo comienza el proceso de búsqueda con una población inicial de individuos, en este caso de redes de unidades producto, donde generamos aleatoriamente, tanto la estructura como los pesos de las conexiones de cada red. A partir de este momento, la población se actualiza en cada generación mediante los operadores de replicación y dos tipos de operadores de mutación: paramétrica, que afecta a los pesos de las conexiones y estructural, que modifica la estructura (número de nodos y conexiones) de la red. Cabe destacar la adaptación en línea de diversos parámetros que intervienen en el algoritmo y el término de regularización introducido que permite obtener soluciones con una buena capacidad de generalización.

Otra de las ventajas del algoritmo es que no necesita la diferenciabilidad de la función de error, ya que no se aplica ningún método de tipo gradiente.

Muy recientemente se han aplicado técnicas de extracción de reglas para redes neuronales en problemas de regresión no lineal (Saito and Nakano, 2002; Setiono et al., 2002) con el objetivo de mejorar la interpretabilidad de los modelos propuestos por los algoritmos de aprendizaje de la estructura y los pesos de la red. Estas técnicas, tratan de implantar reglas que se correspondan con una subregión del espacio de entrada y de una función lineal que dependa de los atributos más importantes de entrada y que aproxime las salidas de la

red para todos los datos muestrales asociados a esta subregión. En nuestra metodología y con el objetivo de analizar los modelos obtenidos, consideramos subregiones donde una o varias funciones de base potencial tiene una acción sobre las salidas de la red mínimas, esto es, su aportación al valor final de la variable dependiente es inferior a un 5% por lo que son eliminadas dichas funciones en esa subregión, obteniéndose un modelo más simplificado.

- **Cuarta aportación:**

  Presentamos dos versiones de un algoritmo evolutivo híbrido para el diseño y entrenamiento de redes neuronales PU (Martínez-Estudillo et al., 2005a) (Martínez-Estudillo et al., 2005b). En la primera, los procedimientos de búsqueda local se realizan cuando el algoritmo evolutivo finaliza, mientras que en la segunda versión, denominada versión dinámica, los procedimientos de búsqueda local se realizan de forma periódica, cada determinado número de generaciones, durante el proceso evolutivo. El tamaño de la población de individuos hace que sea ineficiente la aplicación del proceso de búsqueda local a cada individuo de la población, como puede ocurrir en un algoritmo memético estándar. Por este motivo, sólo se realizará el proceso de búsqueda local sobre un reducido número de individuos pertenecientes a un porcentaje de los mejores individuos de la población. Con el fin de reducir el tiempo de computación dedicado a la búsqueda local, la selección de los individuos a los que se les aplica dicha búsqueda, se realiza a partir de un proceso de agrupación que previamente se realiza sobre un porcentaje de los mejores individuos. Una vez obtenidas las diferentes clases se elige el individuo de cada clase con mejor aptitud. El algoritmo de búsqueda local aplicado es el algoritmo de Levenberg-Marquardt (L-M), indicado especialmente para resolver problemas de

regresión en los que se considera la suma de residuos al cuadrado como función de error (Bishop, 1995; Levenberg, 1944; Marquardt, 1963).

Aunque el algoritmo está enfocado a la resolución de problemas de regresión mediante el diseño y el aprendizaje de redes neuronales basadas en unidades producto, pensamos que la estructura básica del algoritmo puede ser utilizada sin demasiados cambios en otros contextos, tanto en problemas de clasificación como en problemas de regresión con redes neuronales de otro tipo.

- **Quinta aportación:**

  Los algoritmos evolutivos con y sin hibridación se han aplicado a la resolución de problemas de regresión con datos sintéticos y reales (Martínez-Estudillo et al., 2004).

  Los datos sintéticos corresponden a diferentes funciones de prueba que aparecen en la literatura asociadas a problemas de regresión. La función de Sugeno, la función de Friedman#1 y otras funciones construídas por nosotros han sido estudiadas aplicando los algoritmos sin y con hibridación, obteniéndose resultados comparables a los obtenidos por otras técnicas de regresión

  Por otra parte, es importante señalar que el modelo desarrollado a partir de las redes neuronales basadas en PU y los algoritmos evolutivos propuestos para su diseño y entrenamiento, se han aplicado a la resolución de un problema de modelado de datos reales dentro del área de microbiología predictiva. Los datos manejados están asociados al crecimiento de microorganismos que deterioran los alimentos precocinados y envasados al vacío. Concretamente, el problema real abordado consiste en la estimación de los parámetros de un modelo secundario de una curva de crecimiento microbiano (Martínez-

Estudillo et al., 2003). Debido probablemente a la complejidad de los datos, con fuertes interacciones entre las variables, los resultados obtenidos han mejorado a los existentes usando técnicas clásicas de estimación y redes neuronales estándar. Por último, es interesante destacar como una apartación del presente trabajo de tesis, la mayor interpretabilidad, a juicio de los investigadores del área de microbiología predictiva, que tienen los modelos obtenidos con redes PU frente a los obtenidos hasta el momento con redes estándar de tipo sigmoide. Este hecho, unido a la precisión de los modelos, han supuesto un avance en este área de microbiología predictiva.

Por otra parte, cabe destacar aquí la aplicación que de este tipo de modelos de red neuronal estamos realizando en problemas de cinética química. En concreto hemos aplicado recientemente los modelos basados en unidades producto a la cuantificación de los analitos que proporcionan curvas de señal tiempo, altamente solapadas, obtenidas mediante instrumentos de detección. Con estos modelos de previsión obtenemos información de calidad acerca del problema químico objeto de estudio. En nuestro caso se estudiaron dos N-methylcarbamato pesticidas, Carbofuran y Propoxur, cuyas concentraciones se determinaron mediante una clásica reacción quimioluminescente de peroxyoxalato como un sistema de detección para el análisis del cromatografo (Hervás et al., 2005). Estos modelos, y sus resultados y conclusiones, no los hemos introducido en este trabajo de tesis por no extenderlo más, dado que la mayoría de las conclusiones coinciden con lo mostrado en el problema de microbiología predictiva.

1.2 Estructura de la memoria y contenido de los capítulos

Detallamos ahora la estructura de la memoria y el contenido de cada capítulo.

En el Capítulo 2 se introducen las redes neuronales basadas en unidades producto. Presentamos en primer lugar la estructura de este tipo de redes neuronales así como el modelo funcional asociado. Posteriormente se estudian desde un punto de vista geométrico algunas propiedades que diferencian las redes neuronales basadas en unidades producto de otras redes como las redes neuronales estándar de base sigmoide. El estudio en el caso bidimensional de propiedades geométricas como la curvatura de Gauss y la concavidad y convexidad resultó especialmente significativo, mostrando el carácter especial de este tipo de redes. El capítulo finaliza con la demostración a partir del Teorema de Stone-Weierstrass de la densidad de la familia de funciones asociada a las redes PU dentro del espacio de las funciones continuas. El carácter de aproximadores universales de las redes PU proporciona una sólida base teórica necesaria para el uso de estas redes como herramienta de garantía para la resolución de problemas de regresión.

El Capítulo 3 está dedicado al estudio de técnicas de computación evolutiva aplicadas al diseño y entrenamiento de redes neuronales. Después de una introducción en la que se presentan aspectos esenciales de la computación evolutiva y de la evolución de redes neuronales, se presenta el estado del arte de las diferentes técnicas usadas hasta el momento para el diseño y entrenamiento de redes neuronales PU, indicando las ventajas e inconvenientes encontradas. La parte central de este capítulo lo ocupa la construcción de un algoritmo evolutivo que permite diseñar y entrenar redes neuronales PU y que supone una alternativa

a los métodos, especialmente basados en el gradiente, utilizados hasta el momento para entrenar este tipo de redes.

El Capítulo 4 es un capítulo de carácter experimental en el que el algoritmo evolutivo diseñado se enfrenta a la resolución de diferentes problemas de regresión con datos sintéticos y reales. Los datos sintéticos utilizados corresponden por una parte a funciones de prueba que aparecen en la literatura asociadas a problemas de regresión como la función de Sugeno y la función de Saito y Nakano, y por otra a funciones construidas por nosotros. Los resultados comparativos obtenidos muestran el buen comportamiento de las redes PU y del algoritmo evolutivo utilizado para diseñarlas y entrenarlas con respecto a técnicas usadas por otros autores en la resolución de problemas de regresión.

Los datos reales manejados corresponden a un problema de microbiología predictiva y están asociados al crecimiento de microorganismos que deterioran los alimentos precocinados y envasados al vacío. Los resultados obtenidos y el análisis estadístico realizado muestran en general significativas mejoras con respecto a los resultados obtenidos hasta el momento al aplicar a estos datos técnicas clásicas de estimación y redes neuronales estándar. El capítulo presenta en su parte final un estudio de los mejores modelos obtenidos para cada parámetro de crecimiento microbiano, analizando diferentes regiones del espacio de entradas en las que el modelo se simplifica considerablemente y que ayudan a interpretar los modelos obtenidos.

El Capítulo final está dedicado integramente a la presentación de dos versiones de un algoritmo evolutivo híbrido para el diseño y entrenamiento de redes PU. Los algoritmos combinan de forma equilibrada la búsqueda global (exploración del espacio) con la búsqueda local (explotación). La primera propuesta realiza la búsqueda local cuando el algoritmo evolutivo finaliza y la segunda propuesta, denominada versión dinámica, realiza la búsqueda local de forma periódica durante el proceso evolutivo. En ambos casos, la búsqueda local

está precedida de un proceso de agrupamiento de parte de los individuos de la población. A partir de este agrupamiento, se elige el mejor individuo de cada clase obtenido y sobre él se aplica la búsqueda local. Los algoritmos híbridos propuestos son aplicados posteriormente a las funciones de prueba presentadas en el Capítulo 4 y a los datos reales de crecimiento microbiano. El capítulo contiene un estudio estadístico detallado de los resultados obtenidos por los algoritmos híbridos, realizando diferentes comparaciones entre los algoritmos híbridos y otros algoritmos de regresión, y entre las diferentes versiones del algoritmo híbrido entre sí y el algoritimo evolutivo sin hibridación. El análisis estadístico es especialmente minucioso en el caso de la función de Friedman#1 y en los datos reales de crecimiento microbiano.

2 Redes Neuronales de Unidades Producto

Comenzaremos introduciendo la terminología y los conceptos fundamentales que usaremos a lo largo de este epígrafe.

2.1 Terminología de redes neuronales

Una red neuronal consiste en un conjunto de unidades de procesamiento, conocidas como nodos o unidades, las cuales están conectadas entre sí.

La conectividad de una red neuronal viene dada en términos de una arquitectura o topología, la cual es un grafo con conexiones entre los nodos o unidades[4]. Aquellos nodos que no tienen conexiones de entrada se denominan

[4]A lo largo de este trabajo utilizaremos indistintamente las denominaciones de nodo y de unidad.

nodos de entrada y los nodos de los que no sale ninguna conexión se denominan nodos de salida. El resto de nodos se denominan nodos ocultos. Los nodos de computación de una red son los nodos de salida y los nodos ocultos. Todos los nodos que se encuentran a la misma distancia en el grafo de los nodos de entrada forman una capa.

Las redes neuronales pueden dividirse en diferentes tipos en función de su conectividad, siendo las más importantes: redes con conexiones hacia adelante (feedforward) y redes recurrentes (véase la Figura 2-1). En el primer caso existe una forma de numerar los nodos de la red de tal forma que no exista ninguna conexión desde un nodo con mayor número a un nodo con menor número. Todas las conexiones van desde nodos con menor número hacia conexiones con mayor número. En el caso de las redes neuronales recurrentes este método de numeración no existe.

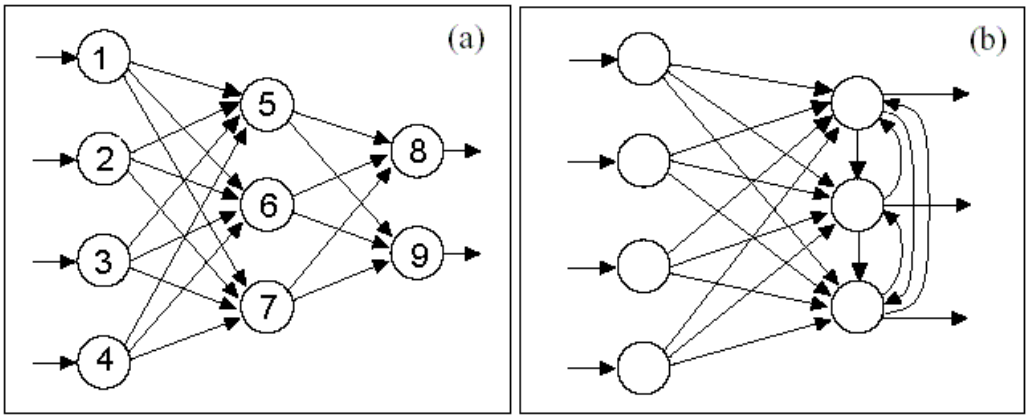


Figura 2-1. (a) Ejemplo de red feedforward. (b) Ejemplo de red recurrente.

En las aplicaciones que aparecerán en el presente trabajo consideraremos redes con conexiones hacia adelante (feedforward) con un solo nodo de salida que permitirán computar funciones escalares, es decir, funciones en las que la imagen

es un número real. Además, las redes consideradas tendrán una única capa intermedia o capa oculta (véase la Figura 2-1 (a)).

Cada nodo i-ésimo de una red neuronal está caracterizado por un valor numérico denominado valor o estado de activación y_i asociado a la unidad. Existe una función de salida o de activación f_i que transforma el estado actual de activación en una señal de salida $f_i(y_i)$. La composición de la activación de la unidad y la función de salida se denomina función de transferencia de la unidad. Las funciones de activación son en general diferentes según se trate de unidades de las capa de salida o de unidades pertenecientes a la capa oculta. Usualmente, las funciones de salida de las unidades de capas de salida son lineales cuando planteamos un problema de regresión y de diferente tipo las correspondientes a la capa oculta. Existen numerosas referencias a modelos de redes neuronales entre las que cabe destacar las de (Bishop, 1995; Haykin, 1994).

2.2 Modelos de redes neuronales

La naturaleza de las unidades o nodos presentes en la estructura de la red y la tipología de las funciones de activación dan lugar a distintos tipos de redes neuronales. Veamos a continuación diferentes tipos de unidades (aditivas y multiplicativas) y de funciones de activación de los nodos en una red neuronal.

2.2.1 Modelo aditivo

El caso usado con más frecuencia dentro de la teoría de redes neuronales es el que corresponde a unidades de tipo aditivo. En este caso, si $w_1, w_2, ..., w_k$ son un conjunto de pesos y t corresponde al sesgo considerado, siendo k el número de variables de entrada, la salida de cada nodo está dada por la expresión:

$$f(w_1x_1 + w_2x_2 + ... + w_kx_k - t) \tag{2.1}$$

donde $x_1, x_2, ..., x_k$ son las variables de entrada del modelo con dominio en $\mathbb{R}$ y f es una función real de variable real, denominada función de salida de la unidad.

La elección de la función de salida de cada nodo de la capa oculta o de la capa de salida, da lugar a diferentes tipos de unidades. Señalemos las funciones de salida más usadas y las unidades a las que dan lugar:

- Se denomina unidad umbral o perceptrón, también conocida como neurona de McCulloch-Pitts (McCulloch and Pitts, 1943), cuando la función de activación es una función escalón como la función signo dada por:

$$\operatorname{sgn}(y) = \begin{cases} 1 & y \geq 0 \\ -1 & y < 0 \end{cases}$$

o la función umbral

$$\theta(y) = \begin{cases} 1 & y \geq 0 \\ 0 & y < 0 \end{cases}.$$

Se trata de funciones que proporcionan una salida binaria dependiente de si el valor de entrada está por encima o por debajo del valor umbral definido.

- Unidad de tipo sigmoide estándar, cuando consideremos como función de activación la función logística sigmoidal dada por:

$$f(y) = \frac{1}{1 + e^{-cy}},\ c > 0$$

Una característica importante de la función logística sigmoidal[5] (Figura 2-2) es que se trata de una función derivable, a diferencia de la función escalón,

[5] El término sigmoidal se utiliza para referirse a la clase de funciones $\sigma(y)$ que cumplen $\lim_{y \to +\infty} \sigma(y) = 1$ y $\lim_{y \to -\infty} \sigma(y) = 0$. Sin embargo, existe cierta falta de consistencia en esta

con derivada siempre positiva que cumple la ecuación $f'(y) = f(y)(1 - f(y))$ que permite calcularla con facilidad, donde el valor máximo de la derivada se encuentra en el punto cero. En este caso la salida del nodo está dada por la expresión:

$$\frac{1}{1 + e^{-c(w_1x_1 + w_2x_2 + ... + w_kx_k - t)}} \tag{2.2}$$

y el rango de salida se encuentra dentro del intervalo $[0,1]$.

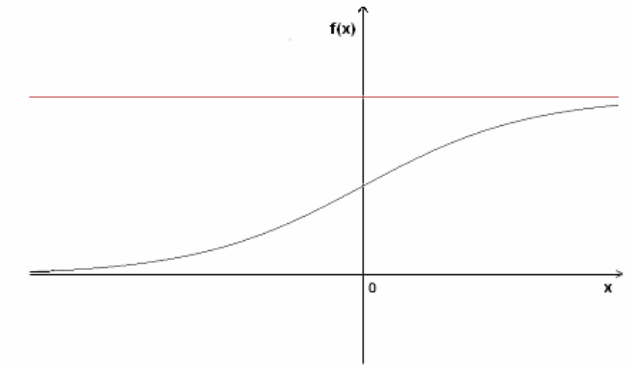


Figura 2-2. Ejemplo gráfico de una función de transferencia sigmoide.

Además de la función logística sigmoidal, existen otros tipos de funciones sigmoides como la función tangente hiperbólica (Figura 2-3):

$$f(y) = \tanh cy\ ,\ c > 0$$

En este caso la salida del nodo está dada por la expresión

$$\tanh\ c(w_1x_1 + w_2x_2 + ... + w_kx_k - t) \tag{2.3}$$

con rango en el intervalo $[-1,1]$.

terminología. Algunos autores exigen además, que σ sea también continua y/o monótona (incluso estrictamente monótona) en todo $\mathbb{R}$, mientras que otros autores no exigen esta condición.

Igualmente se definen otros tipos de funciones sigmoides como la gaussiana,

$$\sigma(y) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{y} e^{-\frac{x^2}{2}} dx$$

o bien, la función arcotangente,

$$\sigma(y) = \frac{1}{\pi} arctg(y) + \frac{1}{2}$$

En estos casos se trata de funciones monótonas acotadas que dan una salida gradual no lineal para las entradas.

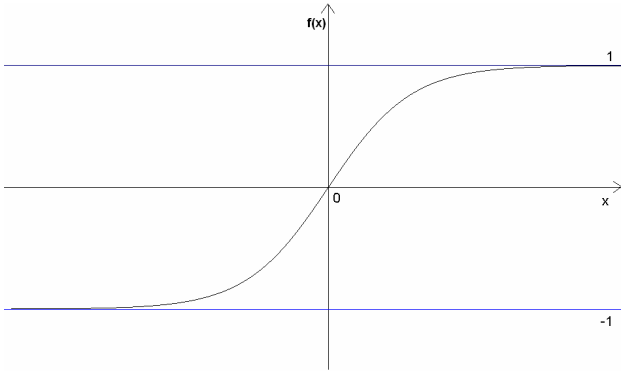


Figura 2-3. Ejemplo gráfico de una función de transferencia tangente hiperbólica.

- Unidad lineal, cuando la función de activación es la función identidad. En este caso la salida del nodo es la función afín:

$$y = w_1 x_1 + w_2 x_2 + ... + w_k x_k - t$$

siendo el rango de salida el conjunto de los números reales $\mathbb{R}$. Observemos que el uso de unidades lineales en una red neuronal puede ser redundante si las unidades de salidas de la red son también de tipo aditivo. Por ejemplo una red neuronal en la que todas las unidades son lineales es equivalente a una red con una única unidad lineal.

Las funciones de transferencia señaladas son sólo las usadas con más frecuencia. Una descripción amplia y detallada de otros tipos de funciones de transferencia puede consultarse en (Duch and Jankowsky, 2001).

2.2.2 Modelo multiplicativo: unidades de orden superior

El éxito de los modelos de redes neuronales basadas en unidades de tipo aditivo está basado en el hecho de que en muchos problemas reales y en diversas aplicaciones es posible modelar y clasificar los datos a través de transformaciones de combinaciones lineales de las variables independientes. Sin embargo, son también numerosos los casos en los que existe una interacción entre las variables y en los que las regiones de decisión no pueden ser separadas por hiperplanos y por tanto no es posible realizar eficientemente la tarea de modelado o clasificación a partir de combinaciones lineales de las variables. Un ejemplo sencillo de este hecho es el siguiente:

Consideremos el espacio de dimensión 1 representado en la Figura 2-4, dividido en dos regiones R_1 y R_2 que forman una partición de $\mathbb{R}$. Una función discriminante lineal no puede generar los bordes de decisión señalados en el espacio, sin embargo, si consideramos una función cuadrática de la forma $f(x) = w_2x^2 + w_1x + w_0$ los bordes de decisión pueden obtenerse fácilmente eligiendo convenientemente los pesos w_0, w_1 y w_2. Un valor x pertenecerá a R_1 si $f(x) > 0$ y a R_2 en caso contrario.

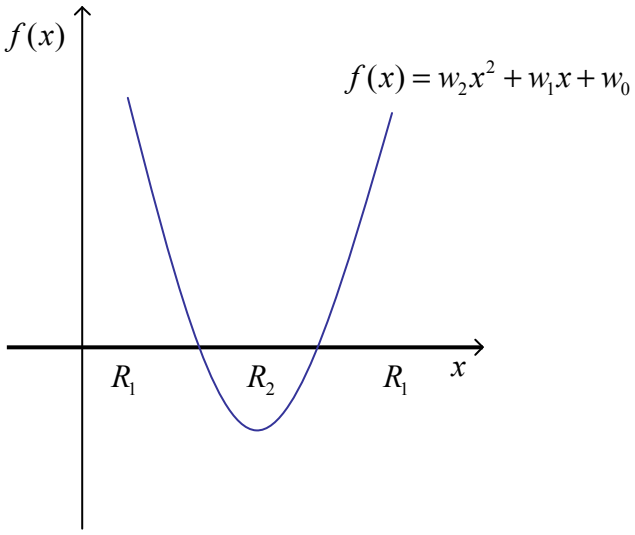


Figura 2-4. Ejemplo de una función cuadrática a partir de la cual se divide la recta real en dos regiones R_1 y R_2.

La existencia de interacción entre las variables independientes en un conjunto de datos que se desea modelar, o bien la presencia de bordes de decisión cóncavos o de regiones no conexas en un problema de clasificación, motiva la introducción de otro tipo de redes neuronales en las que las unidades son de tipo multiplicativo en lugar de aditivas (Rumelhart et al., 1986; Schmitt, 2001).

El caso más sencillo de modelo multiplicativo corresponde a una expresión de la forma

$$M = x_1^{d_1} x_2^{d_2} \cdots x_k^{d_k} , \tag{2.4}$$

denominada monomio, donde los exponentes de las variables son números enteros no negativos. El valor $d_1 + d_2 + ... + d_k$ se llama orden del monomio.

Si $M_1, M_2, ..., M_m$ son monomios, se define una unidad de orden superior (Ghosh and Shin, 1992; Giles and Maxwel, 1987), o unidades sigma-pi (Rumelhart et al., 1986) como el polinomio dado por:

$$w_1 M_1 + w_2 M_2 + ... + w_m M_m - t$$

donde $w_1, w_2, ..., w_m$ son un conjunto de pesos y t corresponde al sesgo considerado. Los pesos y el sesgo son parámetros de la red, donde suponemos fijada la estructura de los monomios $\left\{ M_1, M_2, ..., M_m \right\}$. Observemos que una unidad de orden superior puede interpretarse como una red neuronal con una capa intermedia formada por los diferentes monomios y una unidad lineal en el nodo de salida de la red, donde sólo las conexiones entre la capa intermedia y la capa de salida tienen pesos modificables. Por este motivo, una unidad de orden superior es a menudo llamada red neuronal de orden superior.

A partir de una red neuronal de orden superior es posible construir funciones polinómicas en varias variables de la forma:

$$P(\mathbf{x}) = w_0 + \sum_{i=1}^{k} w_i x_i + \sum_{i=1}^{k} \sum_{j=1}^{k} w_{ij} x_i x_j + \sum_{i=1}^{k} \sum_{j=1}^{k} \sum_{k=1}^{k} w_{ijk} x_i x_j x_k + ...$$

Para unidades de orden superior es frecuente considerar, dependiendo del tipo de aplicación, como funciones de activación de la unidad de salida, funciones de tipo sigmoide o bien la función umbral, en vez de una función lineal. En estos casos la salida de la unidad de orden superior está dada por:

$$f(w_1 M_1 + w_2 M_2 + ... + w_m M_m - t)$$

Concretamente, en el caso de que f sea la función umbral, tenemos las unidades umbral de orden superior y si la función de activación f es de tipo sigmoide, nos referiremos a ellas como unidades sigmoides de orden superior. Una de las principales dificultades que presenta el uso de redes neuronales basadas en unidades de orden superior es que el número de monomios y por tanto de nodos en la capa oculta crece rápidamente cuando se abordan problemas de alta dimensionalidad. Por ejemplo, para un problema de dimensión $n = 10$, en el que las unidades estén totalmente conectadas, las unidades de primer orden tienen 11 parámetros, una de segundo orden 66 y una unidad de tercer orden 572 pesos

independientes. Esta dificultad es a veces compensada con propiedades de invarianza con respecto a transformaciones de los datos. Así, una unidad de tercer orden es invariante simultáneamente a las translaciones y rotaciones de las variables independientes en el caso bidimensional (Bishop, 1995).

2.2.3 Modelo multiplicativo: unidades producto

El tipo más general de modelo multiplicativo es el denominado: unidades producto dado por:

$$x_1^{w_1} x_2^{w_2} \cdots x_k^{w_k} \,, \tag{2.5}$$

donde los exponentes de las variables en este caso son números reales y coeficientes del modelo, a diferencia de lo que ocurría con las unidades de orden superior en donde los exponentes eran números naturales fijos presentes en la estructura de red.

Obsérvese que las unidades producto generalizan a las unidades de orden superior y por tanto su poder computacional es, al menos, el que poseen los monomios. Además, la posibilidad de usar exponentes negativos, permite expresar ratios entre las variables y cocientes. Esta posibilidad que ofrecen este tipo de unidades fue la que motivó su definición por Durbin y Rumelhart (Durbin and Rumelhart, 1989).

Por otra parte, son frecuentes las situaciones en el modelado de datos reales en las que la relación entre las variables responde a una estructura de tipo potencial en la que las potencias no están restringidas a los números naturales o enteros. Como ejemplos sencillos de estas situaciones podemos señalar los siguientes casos estudiados por Saito y Nakano (Saito and Nakano, 1997a; Saito and Nakano, 1997b):

- La tercera ley de Kepler relaciona la distancia r al sol y el periodo de revolución T de cinco planetas según la expresión de tipo potencial:

$$T = kr^{3/2}$$

- La ley de Hagen-Rubens relaciona la frecuencia ν de un rayo de luz, la conductividad eléctrica del metal σ y la reflectancia R según la expresión:

$$R = 1 - 2\left(\frac{\nu}{\sigma}\right)^{\frac{1}{2}}$$

- La ley de Boyle relaciona la presión p y el volumen V de un gas según la expresión:

$$V = 29,05p^{-1.08} - 0,61$$

Una cuestión que puede plantearse en este punto es si la tarea que realizan las unidades producto puede ser realizada por una red con un número razonable de unidades aditivas. La multiplicación y potenciación de enteros en representación binaria puede realizarse a partir de redes con unidades umbral y un pequeño número de capas intermedias (Hofmeister, 1994; Siu et al., 1995). Sin embargo, el tamaño de estas redes crece considerablemente cuando se trabaja con números reales con infinita precisión. Por tanto, las unidades aditivas no pueden sustituir adecuadamente a las unidades tipo producto.

En el siguiente ejemplo se puede observar la capacidad de las unidades producto para generar bordes de decisión más complejos, no conexos y no convexos.

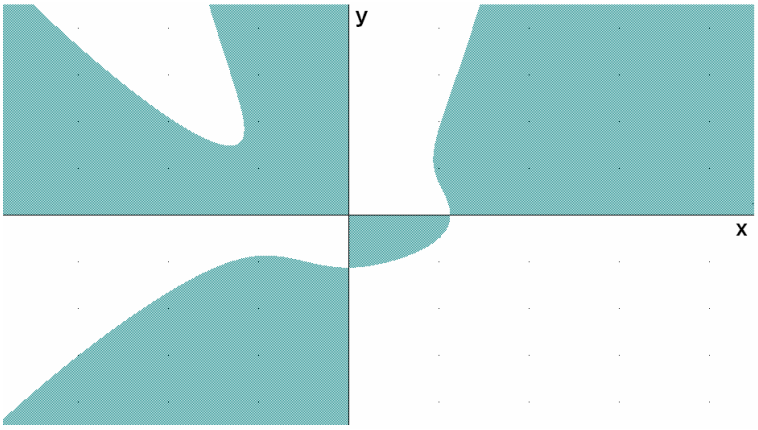


Figura 2-5. Ejemplo de borde de decisión generado por $\frac{x^4}{y} + 2xy - \frac{y^2}{x} - \frac{1}{xy} > 0$

2.3 Modelo de red neuronal basada en unidades producto

En nuestro caso, la arquitectura de las redes neuronales basadas en unidades producto que consideraremos en el resto del trabajo de tesis presenta las siguientes características:

- Una capa de entrada con un nodo para cada una de las variables de entrada.
- Una sola capa oculta con varios nodos formados por unidades producto.
- Una única capa de salida con un nodo de tipo lineal.
- Se añade sesgo al nodo de la capa de salida de la red neuronal propuesta.

Además, los nodos de una misma capa no pueden estar conectados entre sí y no existen conexiones directas entre las capas de entrada y salida.

Hemos considerado este tipo de redes con nodo de salida lineal con el fin de ganar en la interpretabilidad de los modelos que se obtengan al resolver el problema de regresión considerado.

La estructura de la red neuronal con unidades producto que utilizaremos en el resto del trabajo queda representada en la Figura 2-6:

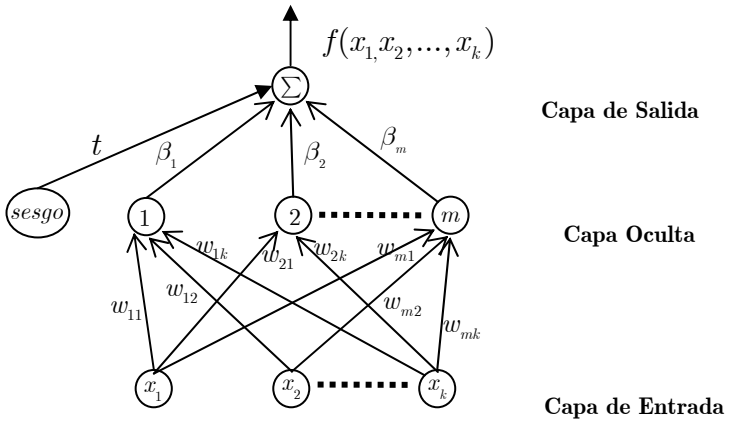


Figura 2-6. Estructura de red neuronal basada en unidades producto.

Suponemos que la red tiene k entradas representadas por las variables independientes, $(x_1, x_2, ..., x_k)$, m nodos en la capa oculta, que representa el número de términos del modelo y un nodo en la capa de salida, que representa la única variable dependiente del modelo. La activación de cada nodo j de la capa oculta está dada por

$$\phi_j = \prod_{i=1}^{k} {x_i}^{w_{ji}} \ , \quad w_{ji} \in \mathbb{R} \tag{2.6}$$

donde w_{ji} son los pesos que conectan el nodo j de la capa oculta con el nodo i de la capa de entrada. Por otra parte, la activación de cada nodo de la capa de salida está dada por

$$\sum_{j=1}^{m} \beta_j \phi_j + t \,, \quad \beta_j \in \mathbb{R} \tag{2.7}$$

siendo β_j el peso que conecta el nodo j de la capa oculta con el nodo de salida y t el sesgo. Además, las funciones de salida de cada nodo de la capa oculta y la función de salida del único nodo de la capa de salida son iguales a la función identidad. Obsérvese que debido al tipo de función de transferencia de las unidades producto, aunque el modelo no llevara expresamente sesgo, el sesgo se podría considerar como un nodo de la capa oculta que no tiene conexión con ningún nodo de la capa de entrada y con conexión a la capa de salida.

Analíticamente, las redes neuronales con unidades producto que serán utilizadas durante el presente trabajo pueden expresarse de la siguiente forma[6]:

$$f : A \subset \mathbb{R}^k \rightarrow \mathbb{R}$$

$$f(x_1, x_2, ..., x_k) = \sum_{j=1}^{m} \beta_j \left(\prod_{i=1}^{k} x_i^{w_{ji}} \right) \tag{2.8}$$

donde

$$\beta_j \in \mathbb{R},\, w_{ji} \in \mathbb{R},$$
$$i = 1, 2, ..., k,\; j = 1, 2, ..., m$$

siendo k el número de variables independientes y m el número de términos o sumandos del modelo. El dominio de definición de f es

$$A = \left\{ (x_1, x_2, ..., x_k) \in \mathbb{R}^k : 0 < x_i \right\}$$

En el caso más sencillo, con dos variables independientes, la función definida anteriormente vendría dada por:

[6] Obsérvese que el sesgo del modelo aparece en la expresión analítica (2.8), considerando cuando para algún j, los exponentes $w_{ij} = 0,\, i = 1, ..., k$.

$$f(x_1, x_2) = \sum_{j=1}^{m} \beta_j x_1^{w_{j1}} x_2^{w_{j2}}$$

La tipología de funciones que acabamos de definir puede considerarse por tanto una generalización de las superficies de respuesta (Myers and Montgomery, 2002). Observemos, por ejemplo, que si elegimos convenientemente los exponentes $w_{ji} \in \{0,1\}$ se obtiene una superficie de respuesta cuadrática del tipo:

$$f(x_1, x_2, ..., x_k) = c_0 + \sum_{i=1}^{k} c_i x_i + \sum_{i,j=1}^{k} c_{ij} x_i x_j$$

Por otra parte, es interesante señalar que la función

$$f(x_1, x_2, ..., x_k) = \sum_{j=1}^{m} \beta_j \left(\prod_{i=1}^{k} x_i^{w_{ji}} \right)$$

puede expresarse de forma equivalente como:

$$f(x_1, x_2, ..., x_k) = \sum_{j=1}^{m} \beta_j \left(\prod_{i=1}^{k} x_i^{w_{ji}} \right) = \sum_{j=1}^{m} \beta_j \exp\left(\sum_{i=1}^{k} w_{ji} \ln x_i \right)$$

Por tanto, una red neuronal basada en unidades producto puede verse desde otra perspectiva como una red neuronal con una capa oculta, donde la función de salida de cada nodo de esta capa es la función exponencial $\exp(z) = e^z$ y en donde los valores de entrada x_i han sufrido una transformación logarítmica.

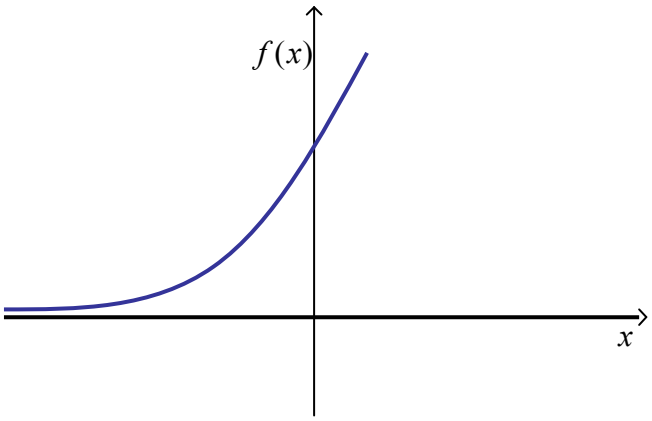


Figura 2-7. Representación gráfica de la función $f(x) = e^x$

Las redes neuronales con unidades tipo producto pueden construir funciones en varias variables con valores en el cuerpo de los números complejos $\mathbb{C}$ de la forma

$$f(x_1, x_2, ..., x_k) = \sum_{j=1}^{m} \beta_j \left(\prod_{i=1}^{k} x_i^{w_{ji}} \right)$$

donde $\beta_j , w_{ji} \in \mathbb{R}$.

Teniendo en cuenta que $x_i^{w_{ji}} = e^{w_{ij} \ln x_i}$, la función anterior puede representarse equivalentemente a partir de la siguiente estructura, en términos de la función exponencial y logarítmica:

$$f(x_1, x_2, ..., x_k) = \sum_{j=1}^{m} \beta_j \left(\prod_{i=1}^{k} e^{w_{ji} \ln x_i} \right) = \sum_{j=1}^{m} \beta_j e^{\sum_{i=1}^{k} w_{ji} \ln x_i} \tag{2.9}$$

Cuando el valor x_i es negativo, entonces x_i puede escribirse en la forma de número complejo como $x_i = i^2 |x_i|$. Teniendo en cuenta esta expresión de x_i y considerando la rama principal del logaritmo complejo, resulta que:

$$\ln x_i = \ln i^2 |x_i| = \ln i^2 + \ln |x_i| = i\pi + \ln |x_i|$$

Sustituyendo esta expresión en (2.9), se tiene que,

$$f(x_1, x_2, ..., x_k) = \sum_{j=1}^{m} \beta_j e^{\sum_{i=1}^{k} w_{ji}(i\pi + \ln|x_i|)} = \sum_{j=1}^{m} \beta_j e^{\sum_{i=1}^{k} w_{ji} \ln|x_i|} e^{i\pi \sum_{i=1}^{k} w_{ji}} \tag{2.10}$$

por otra parte, como

$$e^{i\pi \sum_{i=1}^{k} w_{ji}} = \cos(\pi \sum_{i=1}^{k} w_{ji}) + i \sin(\pi \sum_{i=1}^{k} w_{ji}) \tag{2.11}$$

sustituyendo (2.11) en (2.10) se tiene que

$$f(x_1, x_2, ..., x_k) = \sum_{j=1}^{m} \beta_j e^{\sum_{i=1}^{k} w_{ji} \ln|x_i|} \left[\cos(\pi \sum_{i=1}^{k} w_{ji}) + i \sin(\pi \sum_{i=1}^{k} w_{ji}) \right]$$

El modelo funcional obtenido a partir de una red neuronal con unidades producto puede por tanto expresarse de esta forma equivalente:

$$f(x_1, x_2, ..., x_k) = \sum_{j=1}^{m} \beta_j e^{\sum_{i=1}^{k} w_{ji} \ln|x_i|} \left[\cos(\pi \sum_{i=1}^{k} w_{ji} I_i) + i \sin(\pi \sum_{i=1}^{k} w_{ji} I_i) \right], \; x_i \neq 0 \tag{2.12}$$

donde:

$$I_i = \begin{cases} 0 & si \;\; x_i > 0 \\ 1 & si \;\; x_i < 0 \end{cases}$$

Es importante señalar que para valores negativos de las variables independientes es posible que la salida de la red sea un número complejo, al ser los pesos números reales. Como no es frecuente en aplicaciones reales el uso de redes neuronales con salidas en el campo de los números complejos, (Durbin and Rumelhart, 1989) sugiere considerar sólo la parte real de la salida obviando la parte imaginaria en (2.12), en este caso la salida de la red vendría dada por la expresión:

$$f(x_1, x_2, ..., x_k) = \sum_{j=1}^{m} \beta_j e^{\sum_{i=1}^{k} w_{ji} \ln|x_i|} \cos(\pi \sum_{i=1}^{k} w_{ji}) \tag{2.13}$$

Por otra parte, Schmitt (Schmitt, 2001) propone, sin embargo, considerar que si una entrada x_i es negativa, el correspondiente peso w_{ji} es un número entero. Esta condición evita que la red neuronal tenga salidas complejas y permite ver a las redes neuronales basadas en unidades producto como una generalización de las redes basadas en unidades de orden superior. Además, considera que el valor de salida de la unidad es cero cuando el valor de alguna de las variables es cero y alguno de los pesos es negativo.

En algunos de los modelos usados en este trabajo, se han considerado los exponentes no negativos, con el fin de aumentar la interpretabilidad de los modelos obtenidos. En los casos en los que las entradas sean negativas, realizando una translación de las variables independientes x_i, se puede considerar sin falta de generalidad que el dominio considerado es $A = \left\{(x_1, x_2, ..., x_k) \in \mathbb{R}^k : 0 < x_i\right\}$.

Por último, es interesante señalar que las unidades producto aparecen a menudo en redes con otro tipo de funciones de activación. Una estructura usada con frecuencia en redes neuronales de tipo producto está formada por una capa oculta con unidades producto y una capa de salida con un nodo tipo sigmoide que se utiliza como clasificador.

Las redes neuronales basadas en unidades producto han demostrado buenos resultados en el modelado de datos en los que existen interacciones de diferentes órdenes entre las variables independientes del problema. Se ha comprobado la eficiencia en el modelado de determinado tipo de datos usando un número de nodos en la capa intermedia, inferior al que se necesitarían si se consideraran redes neuronales estándar. Durbin y Rumelhart (Durbin and Rumelhart, 1989) demostraron que la capacidad de información de una única unidad de tipo producto (medida como la capacidad para el aprendizaje de

patrones booleanos aleatorios[7]) es aproximadamente igual a $3N$, comparado con el valor de $2N$ que corresponde a una unidad de tipo aditivo, siendo N el número de entradas de la unidad. Por este motivo, para aproximar funciones usando unidades producto, en determinados casos, se necesitarán en general menos unidades de procesamiento que si se utilizan unidades de tipo aditivo.

La Tabla 2-1, véase (Ismail and Engelbrecht, 1999), muestra el número mínimo de unidades de procesamiento necesarias para modelar funciones sencillas de tipo polinómico. El número mínimo de unidades de tipo aditivo ha sido calculado usando el algoritmo de poda de Engelbrecht, (Engelbrecht et al., 1999), mientras que el número de unidades producto es simplemente el número de sumandos de la función.

Función	Unidades Aditivas	Unidades Producto
$f(x) = x^2$	2	1
$f(x) = x^6$	3	1
$f(x) = x^2 + x^5$	3	2
$f(x_1, x_2) = x_1^3 x_2^7 - 0,5x_1^6$	8	2

Tabla 2-1. Número mínimo de unidades necesarias en la capa oculta para modelar las funciones según el tipo de unidad utilizada.

[7] random boolean patterns

2.4 Propiedades geométricas de las unidades producto: homogeneidad, convexidad y curvatura de Gauss

En este epígrafe se realiza una comparación, desde un punto de vista geométrico y analítico, entre los modelos de redes neuronales basados en unidades producto y los modelos de redes basados en unidades de base sigmoide.

Para comparar los modelos de redes neuronales basados en unidades de base sigmoide y los modelos de redes basados en unidades tipo producto, comenzamos estudiando algunas propiedades de las funciones de base que conforman cada uno de los modelos. Haremos un estudio más detenido en el caso bidimensional en el que es posible la representación gráfica.

Una red neuronal basada en unidades producto puede expresarse como combinación lineal de funciones elementales de la forma $B(\mathbf{x}, \mathbf{w}_j) = \prod_{l=1}^{k} x_l^{w_{jl}}$ que denominaremos funciones de base potencial:

$$f(x_1, x_2, ..., x_k) = \sum_{j=1}^{m} \beta_j B(\mathbf{x}, \mathbf{w}_j) \tag{2.14}$$

Para analizar las características de este tipo de redes y realizar una comparación con las redes neuronales de tipo sigmoide, comenzamos señalando algunas propiedades interesantes de tipo geométrico de las funciones elementales

$$B(\mathbf{x}, \mathbf{w}_j) = \prod_{l=1}^{k} x_l^{w_{jl}} \tag{2.15}$$

que forman una red neuronal basada en unidades producto, donde $\mathbf{x}$ es el vector de variables independientes y $\mathbf{w}_j$ es el vector de pesos. Las propiedades enunciadas a continuación muestran la variedad y la versatilidad de este tipo de unidades.

Propiedades de las unidades de tipo producto

1) La función $B(\mathbf{x}, \mathbf{w}_j) = \prod_{l=1}^{k} x_l^{w_{jl}}$ es homogénea de grado $r = \sum_{l=1}^{k} w_{jl}$

2) La función $B(\mathbf{x}, \mathbf{w}_j) = \prod_{l=1}^{k} x_l^{w_{jl}}$ es cóncava $\forall \mathbf{x} \in A$ si $\sum_{l=1}^{k} w_{jl} \leq 1$ y estrictamente cóncava en A si $\sum_{l=1}^{k} w_{jl} < 1$.

3) La función $B(\mathbf{x}, \mathbf{w}_j) = \prod_{l=1}^{k} x_l^{w_{jl}}$ es cuasicóncava en A para todo $w_{jl} > 0$.

4) En el caso bidimensional, la curvatura de Gauss de la superficie obtenida como grafo de la función $B(x_1, x_2; w_1, w_2) = x_1^{w_1} x_2^{w_2}$ en un punto $P = (x_1, x_2)$ es igual a

$$K(P) = \frac{-w_1\ w_2\ (w_1 + w_2 - 1)\ x_1^{2(w_1+1)}\ x_2^{2(w_2+1)}}{\left[x_1^{2w_1} x_2^{2w_2} (w_2^2 x_1^2 + w_1^2 x_2^2) + x_1^2 x_2^2 \right]^2}$$

Además se cumple que cuando los pesos son positivos:

- Si $0 < w_1 + w_2 < 1$ entonces $K(P) > 0$, para cada $(x_1, x_2) \in A$
- Si $w_1 + w_2 > 1$ entonces $K(P) < 0$, para cada $(x_1, x_2) \in A$

Demostración

1) Se cumple que

$$B(t\mathbf{x}, \mathbf{w}_j) = \prod_{l=1}^{k} (tx_l)^{w_{jl}} = t^{\sum_{l=1}^{k} w_{jl}} \prod_{l=1}^{k} x_l^{w_{jl}} = t^{\sum_{l=1}^{k} w_{jl}} B(\mathbf{x}, \mathbf{w}_j)$$

por tanto la función es homogénea de grado $r = \sum_{l=1}^{k} w_{jl}$.

2 y 3) En el caso bidimensional, la matriz hessiana de la función $B(x_1, x_2; w_1, w_2) = x_1^{w_1} x_2^{w_2}$ está dada por

$$H(x_1, x_2) = \begin{pmatrix} w_1(w_1 - 1)x_1^{w_1-2}x_2^{w_2} & w_1 w_2 x_1^{w_1-1}x_2^{w_2-1} \\ w_1 w_2 x_1^{w_1-1}x_2^{w_2-1} & w_2(w_2 - 1)x_1^{w_1}x_2^{w_2-2} \end{pmatrix}$$

Como el determinante de la matriz hessiana es igual a

$$DetH(x_1, x_2) = w_1 w_2 x^{2w_1-2} y^{2w_2-2}(1 - w_1 - w_2)$$

la matriz es definida positiva cuando $0 < w_1 + w_2 < 1$ y por tanto la función es estrictamente convexa, y semidefinida positiva si $w_1 + w_2 \leq 1$ y la función es convexa. En el caso de dimensión mayor que dos la demostración es similar usando criterios de clasificación de funciones cóncavas y cuasicóncavas a partir de la matriz hessiana (Barbolla et al., 1991).

4) El valor de la curvatura de Gauss se obtiene realizando un cálculo sencillo a partir del valor de la curvatura de Gauss (Do·Carmo, 1976) de una superficie del espacio euclídeo obtenida como grafo de una función $g(x_1, x_2)$, dado por

$$K(x_1, x_2) = \frac{g_{x_1x_1} g_{x_2x_2} - g_{x_1x_2}^2}{\left[1 + g_{x_1}^2 + g_{x_2}^2\right]^2}$$

donde las funciones g_{x_i} y $g_{x_i x_j}$, $j = 1,2$, corresponden a las derivadas parciales de primer y segundo órdenes respectivamente de la función $g(x_1, x_2) = B(x_1, x_2; w_1, w_2) = x_1^{w_1} x_2^{w_2}$.

A continuación señalamos algunas propiedades interesantes de las funciones de base sigmoide

$$B(\mathbf{x}, \mathbf{w_j}) = \frac{1}{1 + e^{-\left(w_0 + \sum_{i=1}^{n} w_i x_i\right)}}$$

Propiedades de la unidades de tipo sigmoide

1) La función $B(\mathbf{x}, \mathbf{w_j}) = \dfrac{1}{1 + e^{-\left(w_0 + \sum_{i=1}^{n} w_i x_i\right)}}$ es cóncava en el dominio:

$$A^+ = \{(x_1, x_2, ..., x_n) : w_0 + w_1 x_1 + w_2 x_2 + ... + w_n x_n > 0\}$$

y convexa en

$$A^+ = \{(x_1, x_2, ..., x_n) : w_0 + w_1 x_1 + w_2 x_2 + ... + w_n x_n < 0\}.$$

2) En el caso dos dimensional, la curvatura de Gauss de la superficie obtenida como grafo de la función $B(x_1, x_2) = \dfrac{1}{1 + e^{-w_0 - w_1 x_1 - w_2 x_2}}$ en cada punto $P = (x_1, x_2)$ del dominio A es igual a cero.

Demostración

1) La propiedad se demuestra con facilidad teniendo en cuenta que $B(\mathbf{x}, \mathbf{w_j})$ es la composición de la función estrictamente creciente $F(t) = \dfrac{1}{1 + e^{-t}}$ y la función lineal $h(\mathbf{x}, \mathbf{w_j}) = w_0 + w_1 x_1 + w_2 x_2 + ... + w_n x_n$. Como $F(t)$ es cóncava para $t > 0$ y convexa para $t < 0$ la función $B(\mathbf{x}, \mathbf{w_j}) = F(h(\mathbf{x}, \mathbf{w_j}))$ es cóncava en A^+ y convexa en A^-.

2) Por otra parte, al igual que hicimos con las funciones de base potencial, el valor de la curvatura de Gauss se obtiene realizando un cálculo sencillo a partir de la expresión de la curvatura de Gauss de una superficie del espacio euclídeo obtenida como grafo, en este caso, de la función.

$$g(x_1, x_2) = B(x_1, x_2) = \frac{1}{1 + e^{-w_0 - w_1 x_1 - w_2 x_2}}$$

A continuación presentamos varios ejemplos gráficos de funciones construidas a partir de redes neuronales de base sigmoide y de redes neuronales

basadas en unidades producto en el caso bidimensional. En los ejemplos se podrá observar gráficamente algunas de las diferencias geométricas existentes entre los dos tipos de funciones, descritas analíticamente con las propiedades anteriores.

Ejemplo 1

En la **Figura 2-8** podemos ver dos ejemplos de funciones de base potencial. En el caso a) representamos la función $g_1(x_1, x_2) = x_1^{0,5} x_2^{0,3}$ con los pesos $w_{11} = 0,5$ y $w_{12} = 0,3$. Mientras que en b) representamos la función $g_2(x_1, x_2) = x_1^{1,5} x_2^{0,9}$ donde utilizamos los mismos pesos multiplicados por 3, esto es, $w_{11} = 1,5$ y $w_{12} = 0,9$, siendo x_1 y $x_2 \in [0,1]$.

a) $g_1(x_1, x_2) = x_1^{0,5} x_2^{0,3}$ b) $g_2(x_1, x_2) = x_1^{1,5} x_2^{0,9}$

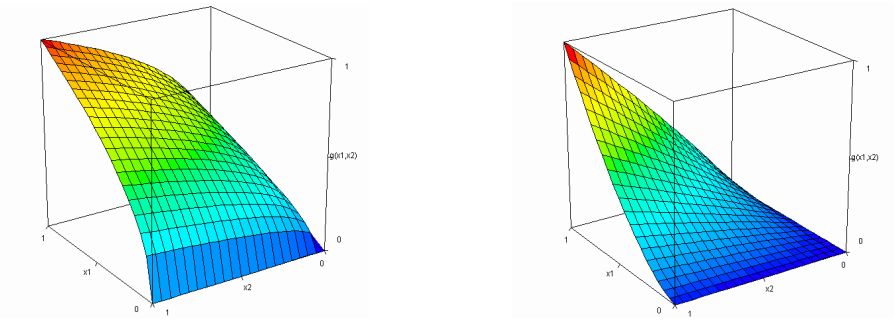


Figura 2-8. Ejemplos de funciones de base potencial

Observamos que la función representada en a) es cóncava en todo el dominio, puesto que $w_{11} + w_{12} < 1$, mientras que la función representada en b) es cóncava en una parte del dominio y convexa en otra. Se puede observar también la diferencia que existe entre la curvatura de ambas superficies en cada punto. Concretamente, de acuerdo con la Propiedad 4, la curvatura de Gauss tiene una fuerte dependencia de los valores de los parámetros w_{ij}.

Ejemplo 2

En la **Figura 2-9** podemos ver dos ejemplos de funciones de base sigmoide utilizando los mismos pesos usados para las funciones de base potencial g_1 y g_2 del Ejemplo 1. En el primer caso, representamos la función $g_3(x_1,x_2)=\dfrac{1}{1+e^{-0,5x_1-0,3x_2}}$ donde $w_{11}=0,5$ y $w_{12}=0,3$, mientras que en el caso b) representamos la función $g_4(x_1,x_2)=\dfrac{1}{1+e^{-1,5x_1-0,9x_2}}$ obtenida multiplicando por 3 los pesos anteriores, esto es, $w_{11}=1,5$ y $w_{12}=0,9$, siendo x_1 y $x_2 \in [-10,10]$.

a) $g_3(x_1,x_2)=\dfrac{1}{1+e^{-0,5x_1-0,3x_2}}$ b) $g_4(x_1,x_2)=\dfrac{1}{1+e^{-1,5x_1-0,9x_2}}$

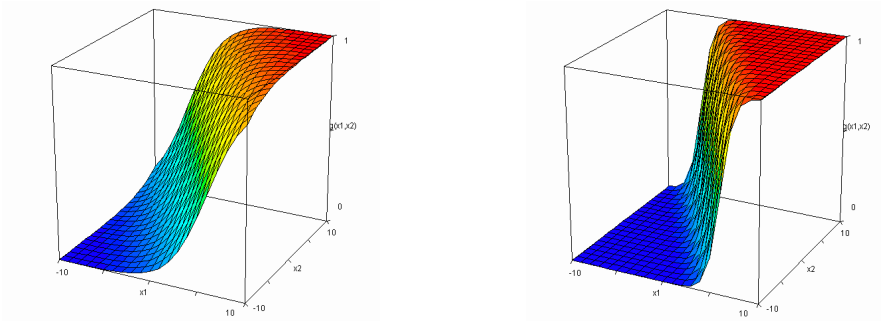


Figura 2-9. Ejemplos de funciones de base sigmoide.

La función $g_3(x_1,x_2)$ es cóncava en $A^+=\{(x_1,x_2): 0,5x_1+0,3x_2>0\}$ y convexa en $A^-=\{(x_1,x_2): 0,5x_1+0,3x_2<0\}$.

La función $g_4(x_1,x_2)$ es cóncava en $A^+=\{(x_1,x_2): 1,5x_1+0,9x_2>0\}$ y convexa en $A^-=\{(x_1,x_2): 1,5x_1+0,9x_2<0\}$.

En ambos casos, la curvatura de Gauss es igual a cero.

Si comparamos las funciones de base potencial con las funciones de base sigmoide con los mismos parámetros, observamos que las segundas son mucho

más alisadas que las primeras, presentando una menor curvatura. Teóricamente, este hecho se justifica observando los valores de las curvaturas de Gauss de las superficies obtenidas como grafos de las funciones potencial y sigmoide respectivamente. En el primer caso, la curvatura de Gauss es, en general, no constante; dependiendo su signo de los parámetros w_1 y w_2, y en el segundo, la curvatura de Gauss es idénticamente nula en cada punto del dominio y para cualquier valor de los parámetros w_1 y w_2.

Es interesante señalar que las propiedades enunciadas (sobre concavidad, homogeneidad y curvatura de Gauss) de las funciones de base potencial $B(\mathbf{x}, \mathbf{w}_j) = \prod_{l=1}^{k} x_l^{w_{jl}}$ no se conservan cuando se considera el modelo funcional completo $f(x_1, x_2, ..., x_k) = \sum_{j=1}^{m} \beta_j \left(\prod_{i=1}^{k} x_i^{w_{ji}} \right)$ obtenido como suma de funciones de tipo potencial.

Los siguientes ejemplos muestran la gráfica de funciones de dos variables obtenidas como suma de funciones de base potencial:

a) Función $f(x_1, x_2) = 2x_1^2 x_2^{0,5} - 2x_1 x_2^2$ en el dominio $[-2,2] \times [0,3]$

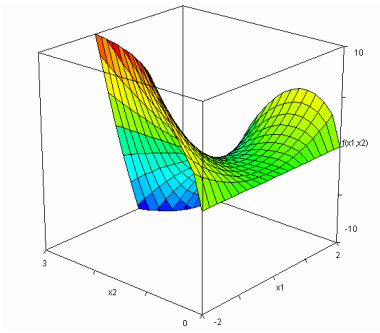


Figura 2-10. Representación gráfica de $f(x_1, x_2) = 2x_1^2 x_2^{0,5} - 2x_1 x_2^2$

b) Función $f(x_1, x_2) = x_1^{-1}x_2^{0,5} - 2x_1^{0,6}x_2^{0,3} + x_1x_2$ en el dominio $[0,3]\times[0,3]$.

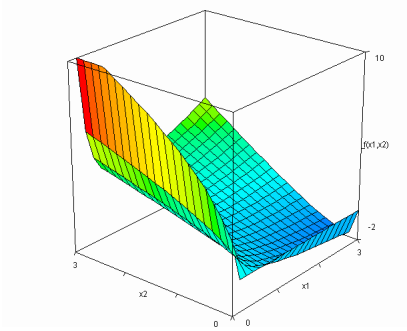


Figura 2-11. Representación gráfica de $f(x_1, x_2) = x_1^{-1}x_2^{0,5} - 2x_1^{0,6}x_2^{0,3} + x_1x_2$

c) Función $f(x_1, x_2) = x_1x_2 - x_1^2x_2^3 + 2x_1^2x_2^2$ en el dominio $[0,3]\times[0,3]$.

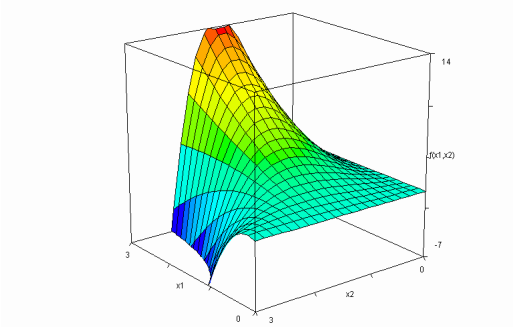


Figura 2-12. Representación gráfica de $f(x_1, x_2) = x_1x_2 - x_1^2x_2^3 + 2x_1^2x_2^2$

Las propiedades enunciadas de las funciones de base sigmoide y potencial, unidas a los ejemplos señalados anteriormente, dan una idea del comportamiento diferente que existe entre ambos tipos de modelos. Los modelos basados en unidades de tipo sigmoide son, en general, más alisados que los modelos multiplicativos, presentando además una menor variabilidad en su curvatura en función de los valores de los pesos de la correspondiente red. La variabilidad de la curvatura y de la concavidad de los modelos obtenidos a partir de las redes

neuronales basadas en unidades producto es, sin embargo, mayor ya que una ligera variación en cualquiera de los pesos, significa una variación en los exponentes de la correspondiente función potencial y como consecuencia una variación en la concavidad o convexidad del modelo y en su curvatura. Esta circunstancia habrá que tenerla en cuenta cuando definamos los operadores de mutación paramétrica de los exponentes y de los coeficientes el modelo evolutivo de entrenamiento y diseño, asignando diferentes valores a las varianzas de las distribuciones normales utilizadas.

Este hecho tendrá como consecuencia un mejor ajuste de este tipo de redes cuando tratemos el modelado de sistemas en los que existe una mayor complejidad debida a las posibles interacciones entre las variables, aunque también una mayor varianza cuando se apliquen algoritmos evolutivos para el diseño y entrenamiento de las redes.

2.5 Las redes neuronales basadas en unidades producto como reconocedores universales

En 1989 aparecieron los 4 artículos más citados, estudiando el problema de densidad para una amplia clase de funciones de salida o de activación (Carroll and Dickinsion, 1989; Cybenko, 1989; Funahashi, 1989; Hornik et al., 1989). Utilizando diferentes argumentos, los autores citados demostraron la capacidad de aproximación de las redes neuronales de base sigmoide bajo ciertas hipótesis sobre las funciones de activación.

Trabajos posteriores (Hornik et al., 1990; Ito, 1991a; Ito, 1991b; Stinchcombe and White, 1989; Stinchcombe and White, 1990) estudiaron con mayor profundidad el problema de aproximación y diversas cuestiones relacionadas, como el uso de funciones no sigmoides, restricciones sobre el dominio de los pesos de la red utilizada, diversas métricas, etc.

Sin embargo, es en 1993 cuando Leshno y colaboradores, resuelven el problema de densidad de una forma sorprendentemente simple, estableciendo una condición necesaria y suficiente para que una función de activación tenga la propiedad de densidad. Concretamente, como queda expresado en el siguiente Teorema, la condición necesaria y suficiente es que la función de activación sea no polinómica (Leshno et al., 1993).

TEOREMA

Sea $\sigma \in C(\mathbb{R})$ una función continua. El conjunto

$$M(\sigma) = \bigcup_{r\in\mathbb{N}} \left\{ \sum_{i=1}^{r} c_i \sigma(w_1 x_1 + w_2 x_2 + ... + w_k x_k - \theta_i) : c_i, \theta_i \in \mathbb{R}, w^i \in \mathbb{R}^n \right\}$$

es denso en $C(\mathbb{R}^n)$ si y solo si la función σ es no polinómica.

Como consecuencia del resultado anterior, si se considera como caso particular $\sigma \in C(\mathbb{R})$ una función sigmoide, se obtiene que las redes neuronales basadas en unidades de tipo sigmoide son aproximadores universales. Además, el resultado proporciona una forma sencilla de generar familias de funciones que forman un subconjunto denso de las funciones continuas.

Un estudio más detallado de la teoría de aproximación de redes neuronales puede verse en (Pinkus, 1999).

Esta característica que poseen las redes neuronales de poder aproximar cualquier función contínua con una precisión determinada, unida al rápido avance experimentado por la velocidad de cómputo, ha tenido como consecuencia que las redes neuronales se utilicen en el campo de la regresión como método alternativo para predecir los valores de la variable y. Aunque distintos modelos de redes neuronales comparten el objetivo de aproximar el modelo dado, las diferentes arquitecturas de red varían en su capacidad de aproximación sobre diferentes problemas, dependiendo su eficacia del problema al que se enfrenten. En la bibliografía podemos encontrar distintos tipos de redes neuronales artificiales

aplicadas al problema del modelado de sistemas: perceptrón multicapa (MLP), funciones de base radial (RBF) y redes neuronales de regresión (GRNN) (Lee et al., 2004; Schioler and Hartmann, 1992; Specht, 1991; Tomandl and Schober, 2001). En MLP la superficie de regresión se construye como combinación lineal de funciones de base sigmoide (Denison et al., 2002), mientras que las redes RBF aproximan funciones mediante la combinación lineal de funciones semiparamétricas no lineales, por ejemplo, funciones de núcleo o Gaussianas. Las GRNN aproximan mediante funciones de núcleo utilizando técnicas Bayesianas. No obstante, el número de nodos ocultos o el número de núcleos y el radio de estos modelos de red deben ser fijados de antemano, para posteriormente, entrenar las redes y aunque todas son aproximadores universales, en la práctica la capacidad de aprendizaje puede variar significativamente.

En el presente epígrafe mostramos como las redes neuronales basadas en unidades producto tienen también esta propiedad pudiendo, por tanto, usarse para la resolución de problemas de regresión. El uso de un tipo u otro de modelo, sigmoide o de unidades producto, dependerá de la estructura de los datos del problema que se desee resolver.

Para demostrar el carácter de aproximadores universales de las redes neuronales basadas en unidades producto aplicaremos herramientas basadas en el álgebra de funciones y en el Teorema de Stone-Weierstrass. Es importante señalar que esta metodología es diferente a la utilizada por Cybenko (Cybenko, 1989) para la demostración de la densidad de las redes neuronales de tipo sigmoide en las funciones continuas a partir del Teorema de Hahn-Banach y la representación de Riesz (Rudin, 1966; Rudin, 1973).

Antes de enunciar el Teorema de aproximación, necesitamos fijar la notación e introducir el Teorema de Stone-Weierstrass. Este teorema es una generalización del clásico Teorema de Weierstrass en el que se establece que cualquier función real continua de una variable, definida sobre un intervalo

compacto, es límite uniforme de funciones polinómicas. El Teorema de Stone-Weierstrass proporciona un resultado similar, determinando condiciones suficientes para que un determinado subconjunto sea denso en el espacio de las funciones reales continuas provisto de la convergencia uniforme sobre un conjunto compacto.

TEOREMA DE STONE-WEIERSTRASS

Sea E un espacio compacto y representemos por $C(E)$ el espacio de las funciones continuas sobre el dominio E. Si H es un subespacio vectorial de $C(E)$ que cumple las siguientes condiciones:

1) Las funciones constantes pertenecen a H

2) Para cada par de puntos x e y de E, existe una función $\varphi \in H$ tal que $\varphi(x) \neq \varphi(y)$

3) Para cada $\varphi \in H, \phi \in H$ se cumple que $\phi\varphi \in H$.

entonces H es un subconjunto denso en el espacio $C(E)$.

La hipótesis 3) y la condición de subespacio vectorial juntas son equivalentes a afirmar que H es una subálgebra del espacio $C(E)$. Mientras que la condición 2) afirma que el subconjunto H separa puntos.

La demostración de este importante resultado de análisis funcional puede verse, por ejemplo, en (Rudin, 1973).

Sea $\mathbb{R}^n$ el espacio euclídeo n dimensional y sea K un subconjunto compacto del espacio $\mathbb{R}^n$ definido por $K = \left\{(x_1, x_2, ..., x_n) \in \mathbb{R}^n : 0 \leq x_i \leq c, i = 1, 2, ..., n\right\}$. Representamos por $C(K)$ el espacio de las funciones continuas sobre el conjunto K y denotamos por $\|f\|$ la norma uniforme de cada función $f \in C(K)$. Por otra parte, consideremos el conjunto $F(K)$ que representa la familia de funciones $f : K \subset \mathbb{R}^n \rightarrow \mathbb{R}$ dadas por:

$$f(x_1, x_2, ..., x_n) = \sum_{j=1}^{m} \beta_j \prod_{i=1}^{n} x_i^{w_{ji}}$$

donde los coeficientes $\beta_j, w_{ji} \in \mathbb{R}$, siendo $w_{ji} \geq 0$ y $m \in \mathbb{N}$. Es fácil comprobar que la función f es continua en el dominio K y por tanto $F(K)$ es un subconjunto del conjunto de las funciones continuas $C(K)$.

El objetivo de esta sección es demostrar que $F(K)$ es un subconjunto denso del conjunto de funciones continuas $C(K)$ con respecto a la norma uniforme. Es decir, cualquier función continua sobre el conjunto compacto K puede ser aproximada por una función de la familia de funciones $F(K)$. La demostración está basada en el Teorema de Stone-Weierstrass.

TEOREMA DE APROXIMACIÓN

La familia de funciones $F(K)$ es un subconjunto denso del conjunto de funciones continuas $C(K)$ definidas sobre el conjunto compacto K.

En otras palabras, dada cualquier función continua $g \in C(K)$ y un número real $\varepsilon > 0$, existe una función $f \in F(K)$ tal que $|f(x) - g(x)| < \varepsilon$, para cada $x \in K$

Demostración

La familia de funciones continuas $F(K)$ es una subálgebra de $C(K)$. En efecto; por una parte, $F(K)$ es un subespacio vectorial del espacio de funciones continuas. Si f_1 y f_2 pertenecen a $F(K)$ es fácil comprobar que la suma $f_1 + f_2$ y el producto por un número real cf_1, con $c \in \mathbb{R}$, pertenecen ambas a $F(K)$.

Además, el producto de dos funciones f_1 y f_2 de la familia $F(K)$, dado por $f_1 f_2$ pertenece a $F(K)$.

Por otra parte, la familia de funciones $F(K)$ cumple las condiciones 1) y 2) del Teorema de Stone-Weierstrass. Es claro que las funciones constantes pertenecen a la familia $F(K)$ tomando los exponentes $w_{ji} = 0$. Además, la familia $F(K)$ separa puntos: si tomamos dos puntos cualesquiera $x, y \in K$, con $x \neq y$, ha de existir, al menos, una coordenada en la que $x_i \neq y_i$, de donde considerando la función $f(x) = x_i \in F(K)$, se cumple que $f(x) \neq f(y)$.

Como el subconjunto $F(K)$ cumple las hipótesis del Teorema de Stone-Weierstrass, se puede afirmar que $F(K)$ es un subconjunto denso del espacio de las funciones continuas $C(K)$ sobre el conjunto compacto K.

Observaciones

1. El Teorema de Stone-Weiertrass establece condiciones suficientes para que un subconjunto sea denso en el espacio de las funciones continuas, sin embargo las condiciones no son necesarias. La familia de funciones de base sigmoide son aproximadores universales, es decir, forman un subconjunto denso de las funciones continuas definidas sobre un compacto y, sin embargo, no verifican todas las hipótesis del Teorema de Stone-Weierstrass. Concretamente, si f y g son dos funciones de tipo sigmoide, el producto de ambas $f\,g$ no es, en general, de tipo sigmoide.

2. El resultado demuestra la capacidad de una red neuronal basada en unidades producto para aproximar cualquier función continua definida sobre un conjunto compacto. Al igual que ocurre con la aproximación mediante redes de tipo sigmoide, el resultado no proporciona ninguna información sobre el número de sumandos del modelo, o lo que es equivalente, sobre el número de unidades necesarias en la capa intermedia de la red para realizar esta aproximación. Varios resultados que establecen cotas inferiores en el grado de aproximación de redes neuronales MLP pueden verse en (Maiorov and Pinkus, 1999).

3. El Teorema de aproximación se ha demostrado suponiendo que los exponentes de las funciones son no negativos. Sin embargo, es importante señalar que en el caso en el que los exponentes puedan tomar valores negativos, basta exigir que el dominio de definición K no contenga ningún punto con alguna de las coordenadas igual a cero, es decir, los valores de las variables independientes no son iguales a cero, para que se cumpla el Teorema de Aproximación. Bastaría para ello hacer translaciones a dominios donde las variables independientes tomaran sólo valores positivos.

2.6 Justificación desde un punto de vista biológico del uso de las unidades de tipo multiplicativo

Existen numerosas razones que han llevado a los neurobiologos a estudiar las unidades multiplicativas como un mecanismo computacional subyacente al funcionamiento de los sistemas neuronales (Schmitt, 2001):

a) Las unidades multiplicativas pueden usarse para modelar las no linealidades implicadas en el procesamiento de las conexiones sinápticas. Se ha demostrado que sucede en la actividad informativa de neuronas aisladas o en poblaciones neuronales.

b) Puede emplearse en el estudio de cómo los modelos simples de redes pueden llevar a un comportamiento complejo con métodos reales, desde el punto de vista biológico.

c) Se han encontrado neuronas de tipo multiplicativo en redes neuronales reales.

En la mayoría de los modelos neuronales la interacción de las conexiones sinápticas se modela como una operación linear. No obstante, numerosos estudios

han puesto de manifiesto que las conexiones sinápticas pueden interaccionar de manera no lineal cuando las sinapsis están localizadas en zonas de la membrana dendrítica con propiedades específicas. En este sentido, se ha argumentado que estas no linealidades dendríticas se podrían modelar mediante unidades multiplicativas.

Las operaciones de tipo multiplicativo en las dendritas parece que tienen lugar en forma de división, una operación que no se puede desarrollar por monomios o unidades de orden superior pero que pueden ser modeladas mediante unidades producto introducidas por Durbin y Rumelhart (Durbin and Rumelhart, 1989) usando exponentes negativos. Los resultados neurobiológicos llevados a cabo con sinapsis de tipo inhibitorio muestran la división como el mecanismo principal.

Se han identificado, en numerosos sistemas nerviosos de diversos animales, neuronas que llevan a cabo cálculos del tipo multiplicativo.

Andersen y colaboradores han analizado datos, derivados de neuronas procedentes del cortex visual de monos, y han demostrado que la selectividad de su campo receptivo cambia con el ángulo de la mirada. Además, la interacción del estimulo visual y la posición del ojo en estas neuronas es multiplicativa. De esta forma, se contribuye a codificar la localización espacial independientemente de la posición del ojo (Andersen et al., 1985).

Otros autores (Suga et al., 1990), han descrito el conjunto de filtros neuronales en el sistema auditivo del murciélago, que proporciona un modo de procesar señales de sonido complejas, operando como multiplicadores.

Investigando el sistema visual de la langosta, Hatsopoulos y colaboradores (Hatsopoulos et al., 1995) han demostrado que una neurona aislada, del tipo sensible al movimiento, conocida como detector gigante del lóbulo lleva a cabo multiplicación de dos señales de salida independientes. De los datos

experimentales se ha derivado un algoritmo que podría usarse en el sistema visual para anticipar el tiempo de colisión con un objeto que se esta aproximando. Los resultados revelan que la multiplicación es un bloque elemental subyacente a la detección del movimiento en insectos. El trabajo de Gabbiani, (Gabbiani et al., 1999) ha continuado con estas investigaciones y ha concluido con una confirmación y generalización del modelo.

Por último, en el estudio experimental del sistema visual del gato, realizado por Anzai y colaboradores (Anzani et al., 1999), se ha demostrado que el mecanismo neurológico subyacente a la interacción binocular también está basado en la multiplicación.

3 Algoritmo Evolutivo para el Entrenamiento y Diseño de Redes Neuronales de Unidades Producto

3.1 Computación Evolutiva

La Computación Evolutiva es una de las múltiples ramas de la Inteligencia Artificial cuya filosofía se basa en desarrollar algoritmos de búsqueda estocástica mediante técnicas inspiradas en la teoría de la evolución[8]. La

[8] Sin embargo, el objetivo no es explicar los fenómenos naturales y diseñar sistemas biológicamente coherentes, sino el de diseñar métodos eficientes que resuelvan problemas aunque no sean factibles en la naturaleza (Yao, X., 2002. Evolutionary computation: A gentle introduction. Chapter 2. In: M.M. R. Sarker, and X. Yao (Editor), Evolutionary Optimization. Kluwer Academic Publishers, pp. 27-53.)

evolución es un proceso de optimización, donde el objetivo es mejorar la habilidad de los individuos para sobrevivir adaptándose al medio. Los organismos tienen determinadas características que influyen en su capacidad de adaptación y reproducción. Estas características, (genes) se encuentran representadas en los cromosomas. Tras la reproducción sexual se produce una mezcla de material genético generando un hijo con la información genética de los padres. Es de esperar que los hijos vayan heredando las mejores características de sus padres. No obstante, el proceso de la selección natural asegura que los individuos más aptos vayan perpetuándose a lo largo de la evolución, siendo los mejores individuos de la población cada vez más aptos.

Ocasionalmente, se producen mutaciones en los cromosomas que causan cambios en las características de los individuos y que, algunas veces, suelen mejorar las capacidades de los individuos.

La evolución natural puede verse como un proceso de búsqueda aleatorio dentro del espacio de búsqueda de los posibles cromosomas con el objetivo de encontrar un cromosoma que mejore cierta característica. Desde este punto de vista, un algoritmo evolutivo se entiende como un proceso de búsqueda aleatoria de la mejor solución a un problema dado.

La idea que subyace bajo estos algoritmos es siempre la misma: dada una población de individuos en un entorno bajo condiciones de presión selectiva (sobreviven los más aptos) ir mejorando la aptitud de los individuos de la población. Dada una función capaz de medir la capacidad o calidad de los individuos es fácil generar aleatoriamente un conjunto de posibles soluciones[9], y aplicar sobre éstas la función denominada de aptitud, como una medida abstracta para ver la capacidad de adaptación al medio de cada individuo. Según esta

[9] Elementos del dominio de la función de aptitud.

medida podemos ordenar los individuos de mejor a peor. Los mejores individuos, como regla general[10], se elegirán como candidatos, también denominados padres, para formar la siguiente generación aplicando técnicas de recombinación o cruce y/o mutación. El cruce o recombinación es un operador que se aplica a más de un padre y cuyo cometido es obtener uno o más individuos –hijos- mediante el intercambio de información. Por otra parte, la mutación siempre se aplica sobre un padre y se obtiene un hijo, modificando de alguna manera al padre. Aplicando dichos operadores a partir del conjunto de individuos padres obtenemos otro conjunto de individuos hijos que compiten entre sí y/o con los padres para formar una nueva población de individuos en la siguiente generación. Este proceso se repite hasta llegar a obtener un individuo con una determinada aptitud suficientemente aceptable, o bien, cuando se ha llegado a un límite computacional fijado de antemano.

3.2 Características de los Algoritmos Evolutivos

En todo algoritmo evolutivo existen dos elementos fundamentales que siempre existen y que además distingue a unos algoritmos de otros, a saber:

- Operadores de Modificación: cuyo principal objetivo es crear diversidad en la población explorando nuevas zonas dentro del espacio de búsqueda.
- Mecanismos de Selección: cuyo principal objetivo es dirigir la búsqueda explotando las zonas más prometedoras.

[10] Es posible que no siempre sean los mejores individuos dependiendo del método de selección utilizado, aunque en la mayoría de los métodos los más aptos tienen más probabilidad de ser seleccionados.

La combinación de ambos elementos de manera reiterada permite a los algoritmos ir mejorando la aptitud conforme vamos avanzando en las generaciones. Se podría ver, por tanto, como un proceso de optimización en el que cada vez los individuos están más cerca de la solución. Desde el punto de vista biológico, se podría decir que los individuos se van adaptando cada vez mejor a las condiciones del medio.

Otro aspecto importante es observar que los algoritmos evolutivos son procesos aleatorios, ya que tanto los operadores de modificación como los operadores de selección tienen un fuerte componente aleatorio.

Un algoritmo evolutivo típico puede responder al esquema mostrado en la Figura 3-1. A la luz de esta figura, se puede observar que estos métodos pueden encuadrarse dentro de la categoría de algoritmos generar y probar[11].

La evaluación de la función de aptitud representa una estimación heurística de la calidad de la solución y el proceso de búsqueda está dirigido por los operadores de modificación y selección.

```
1.  Asignar i=0;
2.  Generar la Población Inicial P(i) aleatoriamente;
3.  Repetir
        a.  Evaluar la aptitud de cada individuo de P(i);
        b.  Seleccionar los individuos padre de P(i) basándose
            en su aptitud;
        c.  Aplicar los operadores de modificación sobre los
            individuos padre para generar la población P(i+1);
4.  Hasta que la población converja o se haya alcanzado el
    tiempo
```

Figura 3-1. Descripción de un algoritmo evolutivo genérico.

[11] Generate-and-test

3.3 Algoritmos Evolutivos vs Optimización Clásica

El teorema NFL[12] propuesto por Wolpert y Macready asegura que *"no puede existir un algoritmo capaz de resolver todos los problemas en media mejor que cualquier otro algoritmo"* (Wolpert and Macready, 1996). Este teorema motiva a que se siga buscando nuevos algoritmos de optimización. Mientras que los algoritmos clásicos de optimización han sido efectivos en entornos continuos, lineales o cuadráticos, unimodales y convexos; los algoritmos evolutivos se muestran más eficientes en entornos discontinuos, no diferenciables, no convexos, multimodales y con problemas de ruido o faltos de información a priori (Yao, 1997).

Los algoritmos de optimización clásica utilizan reglas deterministas para pasar de un punto del espacio de búsqueda al siguiente, además comienzan desde un único punto. Sin embargo, los algoritmos evolutivos realizan una búsqueda paralela desde distintos puntos y las transiciones tienen ciertas componentes aleatorias. Por otra parte, los algoritmos de optimización clásica hacen uso de la primera o segunda derivada de la superficie de error para la búsqueda del óptimo, mientras que los algoritmos evolutivos solo utilizan la información de la aptitud del individuo.

3.4 Componentes de un Algoritmo Evolutivo

En esta sección, se describirán los componentes fundamentales de un algoritmo evolutivo. Dependiendo de la definición y uso de cada uno de estos componentes se obtendrán los distintos tipos de algoritmos evolutivos.

[12] No-Free-Lunch

3.4.1 Representación (definición de los individuos)

Un algoritmo evolutivo hace uso de una población de individuos donde cada uno de ellos representa una posible solución al problema real. El primer paso, en la construcción de un algoritmo evolutivo, es el establecimiento de una aplicación entre el "mundo real" y el "mundo del algoritmo evolutivo". Se trata de buscar una representación de cada solución real para que pueda trabajar el algoritmo evolutivo en el espacio de las representaciones para que se puedan aplicar los operadores de búsqueda o modificación. Desde el punto de vista biológico, podemos hablar de fenotipo (individuo real) y genotipo que es la codificación cromosómica de dicho individuo. Este paso es fundamental dentro del diseño del algoritmo evolutivo pues afecta de manera directa a la eficiencia y complejidad del algoritmo.

El proceso de traducción entre el fenotipo (individuo) al genotipo (cromosoma) se denomina codificación y el inverso se denomina decodificación.

En la Figura 3-2, podemos observar términos equivalentes dependiendo del espacio en donde se encuentren.

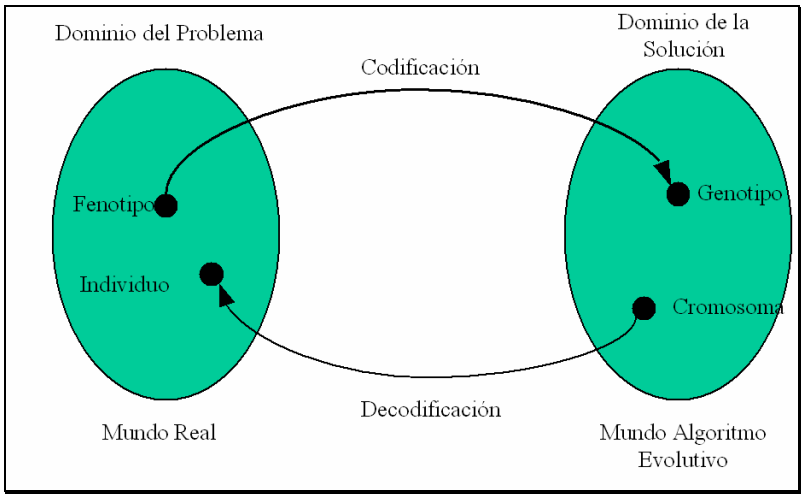


Figura 3-2. Equivalencia de términos utilizados en la Computación Evolutiva.

No obstante, algunos algoritmos evolutivos no trabajan sobre los genotipos, pues no se puede definir una función de codificación y decodificación efectiva. En estos casos, tanto los mecanismos de modificación, como los de selección trabajan sobre el dominio del problema directamente.

3.4.2 Función de aptitud

El papel de esta función dentro del proceso evolutivo es la de medir la capacidad de adaptación del individuo a los requerimientos del medio. Es decir, mide la calidad del individuo desde el punto de vista de la solución. Esta función se aplica en el dominio del fenotipo y le asigna un valor real al genotipo para ser utilizado por el algoritmo evolutivo en su proceso de selección y modificación.

$$F_{EA} : C^I \rightarrow \mathbb{R}$$

Donde, F_{EA} es la función de aptitud, y C^I representa el conjunto de cromosomas de longitud I. El problema suele consistir en maximizar esta función aunque en algunos casos, cuando la función de aptitud equivale a una función de error se trata de minimizar dicha función. En los problemas de optimización también se puede denominar función objetivo.

3.4.3 Población

La misión de la población es la de representar todas las posibles soluciones distintas con las que contamos en un momento dado dentro del proceso de evolución. La población es, por tanto, un conjunto de fenotipos.

El tamaño de la población suele permanecer constante durante el proceso de la evolución. Existen diferentes medidas, que se hacen sobre la población, que nos sirven para analizar cómo va la evolución: aptitud del mejor y peor individuo, media y desviación típica de las aptitudes, etc.

3.4.4 Mecanismo de selección de padres

El objetivo de este mecanismo, dentro del proceso de evolución, es el de elegir los mejores padres para asegurar que el proceso de evolución vaya mejorando la aptitud media de la población.

La elección de los padres, no suele ser un proceso determinista y juega un papel importante la aptitud del individuo. Cuanto mejor aptitud tenga un individuo, más posibilidades tendrá de ser elegido como padre para la formación de la siguiente generación.

No obstante, los individuos menos aptos no tienen porqué descartarse en su totalidad, pues ésta suele ser una forma de evitar quedar atrapados en mínimos locales y dejar de explorar otras zonas del espacio de búsqueda que aunque no parezcan en principio prometedoras, puedan serlo una vez explotada la zona.

Existen diferentes métodos de selección. Entre ellos destacamos los siguientes:

- **Selección aleatoria.** Los individuos se eligen de manera aleatoria sin tener en cuenta su aptitud.

- **Selección proporcional.** La probabilidad de seleccionar a un individuo de la población viene dada por la siguiente expresión:

$$Prob(\vec{c}_n) = \frac{F_{EA}(\vec{c}_n)}{\sum_{n=1}^{N} F_{EA}(\vec{c}_n)}$$

  donde $Prob(\vec{c}_n)$ es la probabilidad de que el individuo $\vec{c}_n$ sea seleccionado y N es el tamaño de la población.

  Un caso particular de este tipo es el **método de la ruleta,** donde la aptitud de los individuos es normalizada dividiéndola por el máximo

de la aptitud. La distribución de probabilidad puede ser considerada como una ruleta en la que el ancho de cada sector es proporcional a la aptitud normalizada de cada individuo.

- **Selección por torneo.** Se extrae de forma aleatoria un grupo de k individuos de la población y se elige el de mayor aptitud.
- **Selección basada en la clasificación[13].** Este método utiliza la posición del individuo dentro de la población atendiendo a su aptitud. La probabilidad de elegir a un individuo depende directamente de la posición que ocupe en esta clasificación. Sin embargo, no depende tan directamente de su aptitud aunque se haya ordenado según ésta. Por ejemplo, una técnica dentro de este método de selección atendería a la siguiente función de probabilidad:

$$Prob(\vec{c}_n) = \frac{1 - e^{-r(\vec{c}_n)}}{\mu}$$

  donde $r(\vec{c}_n)$ es la posición que ocupa el individuo en la población C^I, y μ es una constante de normalización.
- **Elitismo.** Éste método consiste en pasar a la siguiente generación individuos seleccionados de diversas maneras. La más común es la de elegir los k individuos más aptos. De esta forma la aptitud máxima de la población no decrece a lo largo de la evolución.

[13] ranking

3.4.5 Operadores de modificación

Se puede hablar de operadores de cruce y mutación dependiendo de su aridad. Su principal objetivo es la explotación de una determinada zona dentro del espacio de búsqueda. Modifica individuos con la esperanza de obtener mejores.

3.4.5.1 Mutación

La aridad de este operador es unaria. Se aplica, generalmente, sobre el genotipo mediante la modificación aleatoria de uno o varios genes del cromosoma elegidos al azar. Genera un único hijo, y representa un salto dentro del espacio de búsqueda, que puede servir tanto para explotar una determinada zona como para explorar nuevas zonas. Estos cambios se realizan con una probabilidad $p_m \in [0,1]$, denominada *tasa de mutación* y suele ser un valor pequeño con el objeto de proteger al individuo padre de grandes cambios que haga al individuo hijo poco apto. Se trata del único operador de modificación que introduce nuevo material genético a la población de individuos favoreciendo la diversidad. Cuando se utiliza el cruce como operador de modificación se debe combinar con el operador de mutación ya que de esta manera se asegura recorrer otras zonas del espacio de búsqueda no abarcadas en la población inicial de la evolución.

Existen distintos tipos de mutación dependiendo del paradigma. Dentro de los Algoritmos Genéticos podemos destacar los siguientes:

- **Mutación aleatoria.** Se modifica cada posición del cromosoma con una probabilidad p_m.

- **Mutación intercalada.** Se eligen dos posiciones del cromosoma y se modifican las posiciones intermedias con una probabilidad p_m.

Cabe destacar, que la modificación referida depende de la codificación utilizada. Es decir, en el caso de utilizar codificación binaria se trata de pasar de

un valor del gen a su complementario, en caso de utilizar valores reales, se habla de introducir ruido gaussiano a un determinado gen del cromosoma.

El esquema general del operador de modificación mutación puede observarse en la Figura 3-3:

1. Sea i la posición elegida, o gen, del cromosoma $\vec{c}_n$ para ser mutada.
2. Se genera un valor aleatorio $\xi_i \sim U(0,1)$
3. Si $\xi_i \leq p_m$, entonces
4. Si se utiliza valores binarios, entonces $c_n[i] = \overline{c}_n[i]$
   a. Si se utilizan valores reales, entonces
      i. Se calcula el ruido $\eta_i \sim N(0,\sigma^2)$
      ii. $c_n[i] = c_n[i] + \eta_i$

Figura 3-3. Pseudo-código del operador de mutación.

No obstante, existen otros paradigmas donde se utiliza el operador de mutación sobre el propio fenotipo. En estos casos, las mutaciones son más variadas y dependen de las características propias de los individuos, además de la representación utilizada para que surjan nuevos tipos de mutaciones. Por ejemplo, en la Programación Genética se utilizan árboles para representar a los individuos. En este caso, el operador de mutación suele consistir en eliminar una rama, añadir una nueva rama, cambiar el comportamiento de un determinado nodo terminal, etc. El algoritmo propuesto en esta tesis consta de seis tipos distintos de mutaciones (ver 3.9.7 y 3.9.8). En algunos paradigmas de la Computación Evolutiva es el único operador de modificación que se utiliza.

3.4.5.2 Recombinación o cruce

Este operador suele ser binario aunque también puede tener una aridad mayor que dos. Igualmente se trata de un operador que genera dos hijos aunque puede generar uno o más. Se trata igualmente de un operador con una componente aleatoria pues tanto la decisión del punto de cruce en ambos individuos como la parte a cruzar, es aleatoria.

Existen multitud de tipos de cruces y al igual que el operador de mutación se aplica sobre el genotipo. Entre ellos destacamos

- **Cruce uniforme.** Se elige de manera aleatoria una mascara m, $m \in \{0,1\}^I$, donde I es la longitud del cromosoma. Siendo $\vec{c}_{p_1}, \vec{c}_{p_2}$ los padres y $\vec{c}_{h_1}, \vec{c}_{h_2}$ los hijos se tiene que:

$$Si, \begin{array}{l} m[l] = 1, \rightarrow \vec{c}_{h_1}[l] = \vec{c}_{p_2}[l], \vec{c}_{h_2}[l] = \vec{c}_{p_1}[l] \\ m[l] = 0, \rightarrow \vec{c}_{h_1}[l] = \vec{c}_{p_1}[l], \vec{c}_{h_2}[l] = \vec{c}_{p_2}[l] \end{array}, \text{ donde } l = 1...I$$

- **Cruce en un punto.** Se elige de manera aleatoria un valor k, $1 < k < I$, de tal manera que

$$Si, \begin{array}{l} l \leq k \rightarrow \vec{c}_{h_1}[l] = \vec{c}_{p_2}[l], \vec{c}_{h_2}[l] = \vec{c}_{p_1}[l] \\ l > k \rightarrow \vec{c}_{h_1}[l] = \vec{c}_{p_1}[l], \vec{c}_{h_2}[l] = \vec{c}_{p_2}[l] \end{array}, \text{ donde } l = 1...I$$

- **Cruce en dos puntos.** Se elige de manera aleatoria dos valores k_1, k_2, $1 < k_1 < k_2 < I$, de tal manera que

$$Si, \begin{array}{l} l \leq k_1 \rightarrow \vec{c}_{h_1}[l] = \vec{c}_{p_2}[l], \vec{c}_{h_2}[l] = \vec{c}_{p_1}[l] \\ k_1 < l \leq k_2 \rightarrow \vec{c}_{h_1}[l] = \vec{c}_{p_1}[l], \vec{c}_{h_2}[l] = \vec{c}_{p_2}[l] \\ l > k_2 \rightarrow \vec{c}_{h_1}[l] = \vec{c}_{p_2}[l], \vec{c}_{h_2}[l] = \vec{c}_{p_1}[l] \end{array}, \text{ donde } l = 1...I$$

No obstante, el diseño del operador de cruce está íntimamente relacionado con el tipo de codificación utilizado, ya que al menos el operador de cruce debe garantizar que el individuo hijo resultante del cruce sea viable en el dominio del mundo real, es decir, que el proceso de decodificación dé como resultado un individuo real.

3.4.6 Reemplazamiento

Este mecanismo de selección tiene como cometido seleccionar individuos entre los hijos generados, mediante los mecanismos de modificación, y los individuos padres, para formar la población de la siguiente generación.

Al igual que el proceso de selección de padres, se trata de un proceso aleatorio, que no tiene porqué buscar los más aptos, buscando siempre la manera de no quedar atrapados en mínimos locales. Este mecanismo controlará, en nuestro caso, que el tamaño de la población permanezca constante.

Es común también utilizar otros criterios a la hora de seleccionar individuos como puede ser la edad del individuo, esto es, el número de generaciones que sobrevive un mismo individuo. De tal manera que si este es elevado, podremos concluir que se trata de un mínimo local y se puede desechar.

3.4.7 Inicialización

Todo proceso de evolución necesita un mecanismo que genere la población de individuos de la primera generación. Este proceso es aleatorio, y computacionalmente no suele ser relevante ya que solo se realiza una vez. Se desarrolla en el dominio del genotipo, y al igual que el cruce, ésta relacionado con el tipo de codificación utilizada, con el objeto de no generar individuos inviables en el dominio real.

3.4.8 Condición de parada

Generalmente, se pueden distinguir dos tipos de condiciones: Si se conoce cual es la aptitud del individuo solución, cuando se encuentre un individuo con dicha aptitud, o bien cuando se observe que la evolución ha quedado estancada, de forma tal que no mejore en un número de generaciones sucesivas la aptitud media o del mejor individuo de la población.

En ciertos entornos, se suele introducir otra condición que siempre se va a cumplir, y que tiene que ver con valores computacionales tales como número de generaciones, número de evaluaciones, tiempo de CPU, etc.

Por tanto, el esquema general de un algoritmo evolutivo puede ser el mostrado en la Figura 3-4:

1. Asignar $g = 0$
2. Inicializar la población C_g de N individuos como $C_g = \{\vec{c}_{g,n} \,|\, n = 1, ..., N\}$
3. Mientras no se cumpla la condición de parada
    a. Evaluar la aptitud $F_{EA}(\vec{c}_{g,n})$ de cada individuo de la población C_g.
    b. Aplicar operador de modificación de cruce
        i. Seleccionar dos individuos $\vec{c}_{g,n_1}$ y $\vec{c}_{g,n_2}$
        ii. Generar hijos a partir de los individuos $\vec{c}_{g,n_1}$ y $\vec{c}_{g,n_2}$.
    c. Aplicar operador de modificación de mutación
        i. Seleccionar un individuo $\vec{c}_{g,n}$
        ii. Mutar el individuo $\vec{c}_{g,n}$
    d. Reemplazar los individuos para formar la población C_{g+1}
    e. Pasar a la siguiente generación. Asignar $g = g + 1$
4. Fin Mientras

Figura 3-4. Pseudo-código de un algoritmo evolutivo general

3.5 Metodología de Deb

Deb propone una nueva forma de describir y estructurar los algoritmos de optimización basados en poblaciones de individuos (Deb, 2005). Según Deb, la mayoría de los algoritmos de optimización basados en poblaciones de individuos generalmente responden a una misma estructura. La mayoría de los algoritmos evolutivos son algoritmos de optimización basados en poblaciones de individuos, por tanto, se les puede aplicar esta metodología. Igualmente en (Deb, 2005) se le aplica esta metodología a otros métodos de optimización clásicos como el método del simplex propuesto por Nelder and Mead (Nelder and Mead, 1965), y el método de búsqueda aleatoria adaptativa (Brooks, 1958).

La principal ventaja de esta metodología, es que podemos comparar distintos algoritmos de este tipo de una manera sencilla ya que ésta nos propone una descomposición funcional común, en forma de planes o etapas independientes

entre sí. De esta manera, nos puede servir para optimizar los algoritmos, a partir de otros, analizando y comparando cada uno de los planes y extrayendo de cada algoritmo lo mejor de cada plan.

Según Deb, el algoritmo comienza con la generación de una población inicial de individuos, posibles soluciones al problema, y en cada generación la población es modificada usando un algoritmo de actualización. Por tanto, si suponemos que nos encontramos en la generación t el conjunto de individuos $B^{(t)}$ (con $N = \left|B^{(t)}\right|$) al final de la generación obtendremos una nueva población $B^{(t+1)}$ siguiendo el algoritmo de actualización que consta de los siguientes pasos:

Paso 1: Plan de Selección (PS)

Dicho paso consiste en seleccionar los μ individuos del conjunto $B^{(t)}$ para formar el conjunto $P^{(t)}$. Normalmente se suelen elegir los individuos más aptos pero, como se ha visto anteriormente, en ocasiones se incluyen individuos menos aptos para evitar quedar atrapado en mínimos locales y poder explorar otras zonas dentro del espacio de búsqueda. En la Figura 3-5, podemos observar como se seleccionan los individuos marcados de color verde.

Paso 2: Plan de Generación (PG)

En este paso se generan λ individuos a partir del conjunto $P^{(t)}$ formando el conjunto $C^{(t)}$. Este paso se trata del más importante y donde suele haber mayor diferencia de unos a otros algoritmos. Dentro de este paso se utilizan los operadores de modificación generando nuevos individuos. En la Figura 3-5, podemos observar cómo se generan los individuos marcados de color rojo a partir de los marcados en el paso anterior mediante color verde.

Paso 3: Plan de Reemplazamiento (PR)

En este paso se seleccionan τ individuos del conjunto $B^{(t)}$ formando el conjunto $R^{(t)}$ con el objeto de ser sustituídos. Generalmente, se trata de los

individuos menos aptos de la población. En la Figura 3-5, podemos observar cómo se seleccionan los individuos marcados de amarillo.

Paso 4: Plan de Sustitución (PSU)

Por último, en este paso se genera la población $B^{(t+1)}$ formada por los individuos de $B^{(t)}$ una vez sustituídos los individuos de $R^{(t)}$ por los individuos seleccionados de los conjuntos $P^{(t)}$ *y* $C^{(t)}$ utilizando el plan de sustitución. En la Figura 3-5, se sustituyen los individuos marcados en el plan de reemplazamiento de color amarillo, por los seleccionados en este paso de color azul. Hay que observar que el número de elementos seleccionados en el plan de reemplazamiento coincide con los elementos seleccionados en el plan de sustitución. De esta forma mantenemos constante el número de individuos de la población, es decir, $\left|B^{(t)}\right| = \left|B^{(t+1)}\right| = N$. Siendo $|\ |$ el cardinal del conjunto.

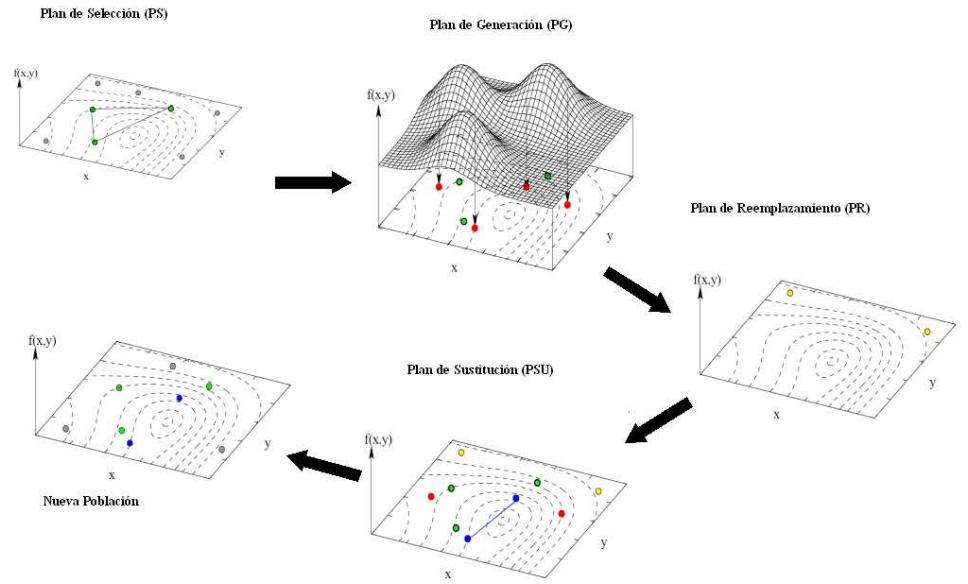


Figura 3-5. Esquema de los planes definidos en la Metodología de Deb[14]

3.6 Paradigmas de la Computación Evolutiva

En el presente epígrafe se presenta de forma breve los diferentes paradigmas de la Computación Evolutiva. Dependiendo de los componentes de los algoritmos evolutivos, definidos en 3.3, que se utilizan y dependiendo del tipo de representación considerada se pueden establecer dentro del campo de la Computación Evolutiva diferentes ramas o paradigmas.

Dentro de la computación evolutiva se distinguen principalmente tres ramas: Estrategias Evolutivas (EE) (Schwefel, 1981) (Schwefel, 1995),

[14] Fuente: Deb, K., 2005. A population-based algorithm-generator for real-parameter optimization. Soft Computing, (in press). Springer.

Programación Evolutiva (PE) (Fogel et al., 1966) (Fogel, 1995) y Algoritmos Genéticos (AG) (Holland, 1975) (Goldberg, 1989; Michalewicz, 1996).

Las Estrategias Evolutivas fueron propuestas por Rechenberg y Schwefel en 1965 como una técnica de optimización numérica. En la estrategia originaria no se usaba población de individuos. Fue Schwefel más tarde, (Schwefel, 1981) el que introdujo la población de individuos dentro de la estrategia evolutiva. La idea que subyace es pensar que el proceso de evolución, como un proceso más dentro de la naturaleza, está sujeto a su vez de la evolución. Por tanto, se trata de evolucionar el proceso de evolución.

Por su parte, los primeros trabajos de Programación Evolutiva fueron los de Fogel y colaboradores(Fogel et al., 1966). Fogel evolucionó autómatas finitos demostrando la eficacia de la técnica evolutiva propuesta. Más tarde, en los años 80, se comenzó a aplicar también a problemas combinatorios y de optimización numérica.

El conjunto de los Algoritmos Genéticos es seguramente la rama de la computación evolutiva más desarrollada y mejor conocida. Fue propuesta por Holland (Holland, 1975) en 1975, aunque algunas de las ideas utilizadas ya habían sido propuestas por Fraser (Fraser, 1957). Los AG se proponen en principio como algoritmos de búsqueda adaptativos, aunque su principal aplicación ha sido la resolución de los problemas combinatorios y de optimización numérica como métodos de optimización global.

Podemos considerar dentro de esta última rama, la Programación Genética (PG), como una aplicación particular de los Algoritmos Genéticos donde se evolucionan árboles de cromosomas. Originariamente, estos árboles eran programas en LISP. El primero que utiliza este término es Koza en (Koza, 1989).

Básicamente la diferencia fundamental entre ellos es la forma cómo se aplican los operadores de búsqueda, los mecanismos de selección de individuos utilizados y la representación de los individuos.

Los AG utilizan el cruce y la mutación como operadores de búsqueda. Además la representación clásica utilizada en los AG es mediante un vector de números binarios o, en menor medida, números enteros reales. Incluso, la dimensión del vector puede no ser fija (Goldberg, 1989). En los AG se hace especial hincapié en el esquema de representación del individuo –genotipo- ya que es crucial para la eficacia del operador de cruce.

La Programación Evolutiva difiere sustancialmente de los AG y PG ya que la PE enfatiza sobre la evolución del comportamiento del modelo en vez de su material genético. Es decir la PE trabaja sobre la evolución del fenotipo. El proceso evolutivo consiste en encontrar un conjunto de comportamientos óptimos dentro del espacio de comportamiento observables. La PE solo utiliza el operador de modificación mutación.

Por su parte, las EE consideran tanto la evolución del fenotipo como la del genotipo, enfatizando sobre el comportamiento del fenotipo de los individuos. Cada individuo es representado mediante bloques de material genético y un conjunto de parámetros de estrategia que modelan el comportamiento del individuo dentro de su entorno. Una diferencia fundamental entre la EE y el resto de paradigmas de la Computación Evolutiva es que las mutaciones únicamente se aceptan en caso de mejorar al individuo. Dentro de las Estrategias Evolutivas se realiza recombinación de los individuos, pero se huye del concepto de cruce ya que se trabaja a nivel evolutivo y no a nivel genético (Yao, 2002).

En los últimos años se están abriendo nuevos paradigmas dentro de la Computación Evolutiva como son: La Evolución Diferencial similar a los algoritmos genéticos difiriendo en los mecanismos de reproducción utilizados. La Evolución Cultural donde se evoluciona la cultura de la evolución y cómo dicha

cultura afecta a la evolución de los fenotipos y genotipos de los individuos. Por último, los Algoritmos Coevolutivos que evolucionan distintas poblaciones de individuos cuyo objetivo es cooperar o competir entre ellos para obtener una solución global al problema.

Dentro de la Computación Evolutiva se abren varias líneas de investigación:

- **Estudio teórico de la Computación Evolutiva**, es decir, el análisis teórico del grado de convergencia de los modelos evolutivos: EE (Beyer, 1994; Schwefel, 1981); PE (Fogel, 1995) y AG (Rudolph, 1994). Los estudios realizados en este ámbito describen el comportamiento asintótico de determinado tipo de algoritmos bajo diferentes hipótesis. Otra línea de trabajo está siendo el análisis de los distintos problemas donde se pueden aplicar las técnicas evolutivas y las dificultades y condiciones óptimas para la aplicación de cada uno de los paradigmas de la Computación Evolutiva para cada problema (Goldberg, 1989). Por último, el análisis del grado de complejidad y naturaleza de los problemas para la aplicación de las distintas técnicas evolutivas y la realización de estudios de eficiencia de los algoritmos evolutivos. (Goldberg, 1989) (Fogel et al., 1966).
- **Optimización evolutiva** es el campo de la Computación Evolutiva más importante si tenemos en cuenta el número de artículos publicados (Yao, 2002) y de aplicaciones a la resolución de problemas reales. Aunque en origen los algoritmos genéticos y la programación evolutiva no surgieron para resolver problemas de optimización, los investigadores han ido progresivamente aplicando y adaptando los métodos y algoritmos a la resolución de problemas de optimización de funciones y de optimización combinatoria. Hasta el momento, cuando se habla de optimización evolutiva fundamentalmente se habla de

optimización numérica: con restricciones (Michalewicz and Schoenauer, 1996) o sin restricciones (Yao, 2002) y multiobjetivo (Fonseca and Fleming, 1995). Cuando los algoritmos genéticos se aplican a la optimización numérica de funciones, los vectores de números reales son normalmente expresados de diferentes formas en codificación binaria. A pesar de los esfuerzos realizados buscando la codificación binaria más adecuada, no está claro si esta transformación de los números reales en cadenas binarias es necesaria. Las estrategias evolutivas y la programación evolutiva trabajan directamente con los vectores de números reales como individuos de la población sin realizar ningún tipo de codificación.

Además de la aplicación de los algoritmos evolutivos a la optimización numérica, ha sido frecuente el uso de estos algoritmos en la resolución de problemas de optimización combinatoria como los problemas clásicos del problema del viajante (Fogel, 1998; Yao, 1993) o del problema del transporte (Michalewicz, 1992), de asignación de recursos, o por ejemplo en problemas de optimización de circuitos y empaquetado. En estas aplicaciones los algoritmos evolutivos han obtenido resultados interesantes en comparación con los métodos clásicos.

- **Aprendizaje evolutivo** es uno de los campos más prometedores dentro de la Computación Evolutiva y básicamente intenta resolver los problemas típicos de la máquina de aprendizaje utilizando técnicas evolutivas. Dentro de este marco se incluyen los sistemas clasificadores (Holland, 1988) sistemas autoadaptativos y las redes neuronales artificiales evolutivas.

El presente trabajo se centra en el estudio de las redes neuronales evolutivas de unidades producto y su aplicación a la resolución de problemas de

regresión. El siguiente epígrafe tiene como objetivo fundamental describir este campo de trabajo que puede ser visto como una combinación de las redes neuronales artificiales y de los algoritmos evolutivos (Yao, 1991; Yao, 1993a).

3.7 Redes Neuronales Artificiales Evolutivas

Las redes neuronales artificiales evolutivas se refieren a un tipo de redes neuronales artificiales a la que se le aplican técnicas evolutivas dentro del proceso de diseño y aprendizaje de la red.

De forma general se puede afirmar que los algoritmos evolutivos se utilizan en el contexto de las redes neuronales en tres diferentes niveles: estimación de pesos de las conexiones, arquitecturas e implantación de reglas de aprendizaje (Yao, 1999). La estimación de los pesos de las conexiones introduce un método que sustituye o complementa a los métodos clásicos de optimización basados en el gradiente descendente que a menudo suelen quedar atrapados en mínimos locales y que ofrecen algunas dificultades cuando se aplican a redes neuronales recurrentes (Yao, 1999). Por otra parte, el diseño de la estructura de la red permite a la red adaptar su topología a diferentes tareas permitiendo que la red se adapte fácilmente en entornos dinámicos, por lo que la intervención humana es mínima dentro del proceso de aprendizaje. La implantación de las reglas de aprendizaje permite que el propio sistema "aprenda a aprender" y el proceso de búsqueda sea más eficiente y eficaz.

3.7.1 Estimación de los pesos de las conexiones

Dada una estructura de red, el entrenamiento de los pesos de las conexiones se formula en base a la minimización de la función de error. Como ya se ha dicho, los principales algoritmos de entrenamiento se basan en el descenso del gradiente de la función de error: retropropagación (BP) y en el gradiente

conjugado (Hertz et al., 1991). Dichos algoritmos se han utilizado con éxito en distintas aplicaciones y en diferentes áreas (Knerr et al., 1992; Lang et al., 1990). Aunque son muy eficientes, suelen quedar atrapados en mínimos locales dependiendo fuertemente del punto de partida de la búsqueda. Además, estos algoritmos no se pueden utilizar cuando la función de error es multimodal y/o no diferenciable (Hertz et al., 1991). Por ejemplo, en el caso de que la función de error tenga en cuenta tanto el error cometido entre los datos de salida y los estimados como la complejidad de la red.

Por su parte, los algoritmos evolutivos realizan una búsqueda global del óptimo de forma más efectiva y pueden trabajar con superficies de error multimodales y no diferenciables. Al no necesitar ninguna información relacionada con el gradiente de la función de error, resultan métodos bastante útiles cuando esta información no está disponible. Este hecho es el que ha motivado la utilización de redes neuronales evolutivas en la resolución de numerosos problemas reales en los que con frecuencia la función de error es multimodal y presenta problemas de continuidad y de diferenciabilidad. Por su parte, los algoritmos basados en el gradiente (retropropagación y gradiente conjugado) suelen ser más rápidos en la búsqueda del óptimo que el entrenamiento evolutivo. Sin embargo, los métodos evolutivos son en general menos sensibles a las condiciones iniciales del entrenamiento.

Recientemente, ha aparecido una nueva metodología que combina los algoritmos evolutivos y los métodos de búsqueda local (Kinnebrock, 1994; Skinner and Brougthon, 1995). En el contexto de las redes neuronales, esa metodología lleva a cabo un entrenamiento híbrido, combinando la capacidad de buscador global de un algoritmo evolutivo con la capacidad de afinar la solución que tienen los algoritmos de búsqueda local como los basados en el gradiente. El Capítulo 5 estará dedicado a realizar una propuesta de algoritmo híbrido para el diseño y entrenamiento de redes neuronales basadas en unidades producto.

Dentro del proceso de estimación de los pesos de las conexiones de la red se distinguen dos fases fundamentales: Decidir la representación de los pesos de las conexiones y diseñar los operadores de mutación y cruce más adecuados para aplicar sobre los individuos representados en la forma elegida.

3.7.1.1 Representación binaria

Los Algoritmos Genéticos trabajan sobre una cadena de números binarios que representan a cada individuo de la población. A cada cadena se le denomina cromosoma. Existen numerosos trabajos que estiman los pesos de las conexiones de las redes utilizando este tipo de representación (Janson and Frenzel, 1993; Whitley et al., 1990) que codifica cada peso mediante una cadena de bits. El cromosoma que representa a la red es, a su vez, una concatenación de las cadenas de bits de cada peso de las conexiones de la red. Existen diferentes maneras de ordenar las cadenas de bits de cada peso con el fin de concatenarlas de una manera coherente.

Un ejemplo de representación binaria de una red la podemos ver en la siguiente figura:

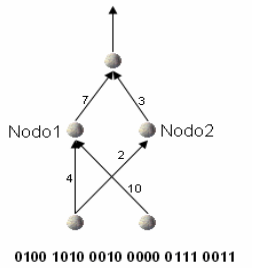


Figura 3-6. Ejemplo de representación binaria de una red[15]

[15] Fuente: Yao, X., 1999. Evolving artificial neural network. Procedings of the IEEE, 9 (87): 1423-1447.

Cada peso de la conexión lo representamos mediante 4 bits. La red completa se representa mediante una cadena de 24 bits donde la cadena 0000 significa que hay ausencia de conexión. Los pesos se codifican de abajo hacia arriba y de izquierda a derecha para cada nodo de la capa oculta.

Existen diferentes métodos de codificación binarios siendo lo más importante la precisión en la representación. Una precisión baja hace que el proceso de entrenamiento sea poco exacto, mientras que una precisión alta provoca grandes cadenas de bits que pueden hacerlo ineficiente.

Las principales ventajas de esta representación estriban en su simplicidad y generalidad, a la vez de que es mucho más fácil su implementación en hardware.

3.7.1.2 Representación real

Por otra parte, otra posibilidad consiste en la representación de los pesos de las conexiones de la red directamente mediante números reales, (Angeline et al., 1994; García-Pedrajas et al., 2002; Yao and Liu, 1997). Esta codificación se denomina codificación real. De esta forma, la red se suele representar mediante un vector de números reales.

3.7.1.3 Operador de mutación

El operador de mutación, como se ha comentado anteriormente, está directamente condicionado por el tipo de representación que se utilice. En el caso de utilizar representación real no se pueden aplicar los operadores de mutación y cruce como se entiende tradicionalmente desde el punto de vista de los algoritmos genéticos.

Los paradigmas de la computación evolutiva más adecuados cuando se utiliza la representación real son la Programación Evolutiva y las Estrategias Evolutivas ya que son especialmente eficientes en la optimización en dominios

continuos. El principal operador de búsqueda es la mutación Gaussiana aunque también se utilizan otros tipos de mutación como puede ser la mutación de Cauchy (Fogel, 1994).

Básicamente, se trata de añadir valores aleatorios a los pesos de las conexiones siguiendo distribuciones Normales o de Cauchy y aceptar los cambios en caso de mejorar la aptitud de la red. No obstante, se utilizan técnicas de enfriamiento simulado[16]. Para permitir con cierta probabilidad aceptar cambios aunque no se mejore la aptitud de la red, evitando así que el algoritmo quede atrapado en mínimos locales (Souto et al., 2002).

3.7.1.4 Operador de cruce

El operador de cruce genera descendientes recombinando el material genético (genotipo) de dos individuos de la población sin tener en cuenta el significado de dicho material.

De esta manera, la aproximación al problema se hace desde una perspectiva dual. Por un lado, las soluciones se muestran mediante cadenas de números y es en este espacio donde se realiza la búsqueda. Por otro lado, la evaluación de los individuos se realiza en el espacio específico del problema. Esta característica, que resulta muy interesante en muchos problemas, puede tener efectos negativos en el problema de la evolución de la arquitectura de una red neuronal.

[16] En dichos sistemas cuando la temperatura es alta las partículas están desordenadas, mientras que conforme se va enfriando las partículas van organizándose de tal manera que buscan estados de energía más bajos y estables. Dada una temperatura T constante los cambios de estado son permitidos siempre que se pase a un estado de menor energía siendo ΔE el incremento de energía. En caso contrario se permite con una probabilidad igual al factor de Boltzmann $\exp(-\Delta E / T)$.

El primer efecto negativo que presentan es la restricción del espacio de búsqueda intrínseca a la representación de la red neuronal como una cadena finita de números. El subconjunto del espacio de búsqueda será más o menos amplio dependiendo de la representación que se elija.

Generalmente, la representación se elige para que el operador de cruce funcione efectivamente. No obstante, existen entornos en los que el operador de cruce no es efectivo. Dichos entornos se denominan engañosos.

Para algunos autores el entorno de evolución de las redes neuronales es un entorno engañoso[17]. Por lo tanto, no es apropiado para la aplicación del operador de cruce. En (Angeline et al., 1994) se argumenta que existen tres formas de engaño en la evolución genética de las redes neuronales.

La primera forma de engaño radica en la existencia de redes que comparten su estructura y pesos y sin embargo su forma de representación es diferente. La función de interpretación no es biyectiva, sino que es de muchos a uno. El cruce de dos redes iguales con diferente representación tiende a repetir componentes en vez de complementar. Este problema se conoce como el problema de la permutación. No obstante, aunque la mayoría de los estudios apuntan la gravedad del problema, otros no lo consideran tan grave (Hancock, 1992). Incluso se han desarrollado esquemas de codificación que evitan dicho problema.

En la Figura 3-7 se puede observar la misma red que en la Figura 3-6 pero con diferente cadena de representación.

[17] En inglés, deceptive.

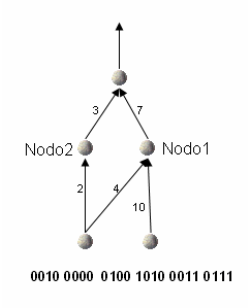


Figura 3-7. Esquema y representación de la misma red que la Figura 3-6 con los nodos ocultos en distinto orden y distinta representación[18].

La segunda forma de engaño viene dada por la existencia de redes con la misma topología pero pesos diferentes. Para una tarea dada, cada topología de red puede implementar soluciones múltiples, cada una correspondiente a una representación distribuida diferente a lo largo de los nodos ocultos (Hinton et al., 1986). Aunque se pueden eliminar nodos sin alterar drásticamente el rendimiento de la red, el papel que juega cada nodo en la representación global está determinado únicamente por el peso de sus interconexiones. Es decir, una unidad oculta no se puede considerar aislada y ser intercambiada libremente, porque su rendimiento depende tanto de sus propios pesos, como de las interconexiones con otras unidades de la red en la que se encuentra.

La tercera fuente de engaño aparece cuando se tiene redes topológicamente diferentes. Los tipos de representación distribuida que pueden adoptar redes diferentes topológicamente varían enormemente. Es probable que estas representaciones sean incompatibles reduciendo mucho la probabilidad de obtener individuos viables si se cruzan. Por ello, la evolución ha de evitar

[18] Fuente: Yao, X., 1999. Evolving artificial neural network. Procedings of the IEEE, 9 (87): 1423-1447.

diversificar demasiado el tipo de redes de la población. Esto reduce de forma drástica la variedad de topologías que se pueden considerar en la búsqueda.

3.7.2 Diseño de la arquitectura de la red

En la sección anterior hemos supuesto que disponíamos de una arquitectura de red fija y óptima. A partir de dicha arquitectura se intenta ajustar los pesos de las conexiones de la red. No obstante, otro elemento importante dentro del entrenamiento de las redes es el diseño de la estructura de la red y la definición de las funciones de transferencia de los nodos. Esta tarea tradicionalmente la ha realizado el experto, mediante su experiencia y ensayando con diferentes estructuras hasta encontrar una adecuada. Esta tarea aparte de ser tediosa, no es sistemática, y por tanto, está abocada a caer en mínimos locales; redes demasiado pequeñas con poca capacidad de aprendizaje, o redes demasiado complejas que sobreentrenan.

Se han realizado diferentes esfuerzos para automatizar esta tarea y hacer el proceso de aprendizaje menos supervisado de lo que es. Una alternativa ha sido el diseño de algoritmos constructivos que intentan, a partir de una red sencilla, añadir nodos en las capas ocultas y conexiones para ir mejorando la red (Frean, 1990). Otro enfoque ha sido justo el contrario, es decir, algoritmos destructivos que, a partir de redes complejas, van quitando nodos y conexiones que sean innecesarios hasta llegar a una red más simple y con capacidad de aprendizaje. No obstante Angeline y colaboradores afirma que estos métodos están abocados a caer en mínimos locales, entre otras cosas porque al partir de una red determinada se reduce el espacio de búsqueda (Angeline et al., 1994).

Desde el punto de vista evolutivo, el diseño óptimo de la arquitectura de la red puede ser abordado como un proceso de búsqueda en el espacio de las arquitecturas en el que cada punto se refiere a una estructura distinta, y dado un determinado criterio, como el de la complejidad y/o error de entrenamiento

formar una superficie de error. Por tanto, buscar la mejor estructura será buscar el mejor punto de la superficie. Miller y colaboradores describen las características de esta superficie de error (Miller et al., 1989).

En primer lugar, la superficie es infinita ya que el número de nodos y conexiones posibles en la red no es finito. La superficie no es continua y por tanto no derivable ya que los cambios en la estructura son discretos. La superficie es compleja, ya que la relación entre la estructura y su aptitud es un proceso complejo y depende del método de evaluación, entrenamiento utilizado y de las condiciones iniciales para el proceso de entrenamiento. El entorno es engañoso ya que arquitecturas parecidas pueden tener distinta aptitud. Además, la superficie de error es multimodal ya que estructuras distintas pueden tener aptitud parecida.

Como en el caso de la estimación de pesos, se nos presentan dos problemas a la hora de diseñar la estructura de una red neuronal. Por una parte, la representación elegida y por otra los operadores de mutación y/o de cruce que se van a utilizar. En el caso de la representación de la estructura hay que decidir la información a codificar en el cromosoma. Es posible codificar todos los detalles de la red, en tal caso hablaríamos de codificación directa o, sin embargo, codificar ciertos parámetros de la red como puede ser el número de conexiones, número de capas ocultas, número de nodos en la capa oculta, etc. En tal caso hablaríamos de codificación indirecta.

3.7.2.1 Codificación directa

Numerosos trabajos utilizan codificación directa para diseñar arquitecturas de red (Angeline et al., 1994; García-Pedrajas et al., 2002; Yao and Liu, 1997). Este diseño se puede realizar conjuntamente con la estimación o aprendizaje de los pesos, o hacerlo independientemente. La codificación utilizada para representar la topología de una red se hace mediante matrices binarias.

Supongamos que tenemos una red de N nodos, la matriz $C = (c_{ij})_{N \times N}$ representaría la topología de la red de tal manera que si $c_{ij} = 1$ indica que existe una conexión del nodo i al nodo j. En caso de que $c_{ij} = 0$ indica la ausencia de conexión del nodo i al nodo j. Para los casos en que se determina la estructura y los pesos de las conexiones de manera conjunta, es posible codificar el valor de los pesos directamente en la matriz. De esta manera $c_{ij} = \alpha$ donde $\alpha \in \mathbb{R}$, es el valor del peso de la conexión. Si $\alpha = 0$ no existe conexión del nodo i al nodo j.

Cada matriz C representa biunívocamente la estructura de una red. Además desde el punto de vista matricial es fácil implementar restricciones en la topología de la red. Por ejemplo, una red neuronal no recurrente debe tener a cero todos los valores por debajo de la diagonal principal.

Este tipo de codificación es bastante fácil de implementar, y permite definir con facilidad operadores de cruce y de mutación: eliminar y añadir conexiones, crear nuevos nodos…

Sin embargo, el problema de la codificación directa es que en el caso de que las estructuras sean grandes, también crece su representación lo que produce un algoritmo poco eficiente. No obstante, existen posibilidades de reducir el tamaño de esta matriz, si tenemos en cuenta algunas de las restricciones propias de la red neuronal como puede ser que no existen conexiones entre los nodos de entrada.

Este tipo de representación no evita el problema de la permutación, por tanto se evita el uso del cruce a la hora de evolucionar redes codificadas mediante este método (Angeline et al., 1994). No obstante, Hancock asegura que el problema de la permutación no es tan importante como pudiera parecer, puesto que a lo más produce una cierta ineficiencia (Hancock, 1992). Existen algunos trabajos que intentan evitar este problema aunque no se han llegado a aplicar a problemas reales (Thierens, 1996).

3.7.2.2 Codificación indirecta

Con el objetivo de minimizar la longitud del cromosoma que representa a la estructura de la red, se han utilizado otros métodos de codificación denominados indirectos ya que su objetivo no es representar fielmente la estructura de la red sino algunas características importantes que la identifiquen.

Un tipo de representación indirecta puede ser la representación paramétrica, donde una red se puede representar como un conjunto de parámetros: número de capas ocultas, número de nodos en cada capa, número de conexiones entre capas, etc. Un ejemplo de este tipo de representación lo podemos en encontrar en (Harp et al., 1989). Otro tipo de representación indirecta puede ser mediante gramáticas, compuestas por reglas de producción para construir arquitecturas. En (Kitano, 1990) podemos ver un ejemplo de este tipo de representación donde se utiliza un operador de cruce, que minimiza el problema de la permutación.

3.7.3 Evolución de las funciones de transferencia de los nodos

Hasta ahora, hemos considerado fijo el comportamiento de los nodos de la red, dejando al experto la elección de la función de transferencia a utilizar en cada nodo o en cada capa de la red. Sin embargo, es posible evolucionar igualmente el comportamiento de cada nodo de la red. En (White and Ligomenides, 1993) se pueden observar métodos donde se evoluciona tanto la estructura de la red como las funciones de transferencia de los nodos de la red. De esta forma, para cada red se establece inicialmente un 80% de nodos que utilizan la función de transferencia sigmoide, y un 20% que utilizan función de transferencia gaussiana. Durante la evolución se van generando redes con mezcla de nodos. En (Cohen and Intrator, 2002), se introduce una metodología de entrenamiento para una red híbrida de redes MLP/RBF, donde se utiliza una

aproximación de selección de modelos utilizando el principio MDL[19]. Otros autores mediante coevolución también han coevolucionado tanto estructuras, como pesos de conexiones y funciones de transferencia (García-Pedrajas et al., 2002).

3.8 Entrenamiento de redes neuronales basadas en unidades producto

Las redes neuronales basadas en unidades producto han demostrado buenos resultados en el modelado de datos en los que existen interacciones de diferentes órdenes entre las variables independientes del problema.

Esta capacidad de representar interacciones entre variables de las redes neuronales basadas en unidades producto se ve de alguna forma ensombrecida por la mayor dificultad que presenta su entrenamiento. En numerosas ocasiones, determinar la estructura óptima de la correspondiente red neuronal y el valor de los pesos no es una tarea sencilla debido a la complejidad de la superficie de error. Esta superficie presenta con frecuencia una geometría bastante rugosa en la que son numerosos los óptimos locales y en la que es posible encontrarse con amplias mesetas (Engelbrecht et al., 1999; Engelbrecht and Ismail, 1999). La mayor complejidad de la superficie de error está lógicamente motivada por la estructura potencial de cada nodo de la capa intermedia. Es fácil ver que para una función potencial, una pequeña variación en alguno de los exponentes de una variable tiene un efecto considerable en el valor de la función y por tanto en la correspondiente superficie de error. Por ejemplo, podemos comprobar a partir de las propiedades indicadas anteriormente de las unidades de tipo producto, que la función $f(x,y) = x^{0,49}y^{0,49}$ es estrictamente cóncava y sin embargo la función

[19] Minimum Description Length

$f(x,y) = x^{0,51}y^{0,49}$ obtenida realizando una ligera variación en los exponentes es convexa en una parte del dominio y cóncava en otra.

A continuación se exponen de manera breve diferentes algoritmos de optimización usados hasta el momento para el aprendizaje y entrenamiento de una red neuronal basada en unidades producto, así como una descripción de los resultados obtenidos.

3.8.1 Algoritmos basados en el gradiente

La estructura y complejidad de la superficie de error de una red neuronal basada en unidades producto es la responsable de que el clásico algoritmo de retropropagación, adaptado a este tipo de redes, (Durbin and Rumelhart, 1989; Engelbrecht et al., 1999) y los algoritmos de tipo gradiente, en general, no tengan buenos resultados y queden con frecuencia atrapados en óptimos locales o queden paralizados cuando encuentran regiones llanas (Janson and Frenzel, 1993; Leerink et al., 1995).

Estas dificultades del clásico algoritmo de retropropagación son estudiadas con detalle por Leerink y colaboradores (Leerink et al., 1995). Concretamente, prueban que el problema de la paridad 6-bit no puede ser entrenado usando el algoritmo de retropropagación estándar en modelos de unidades producto. La presencia de numerosos mínimos locales y la inicialización de los pesos son los motivos que dificultan el entrenamiento. La inicialización de los pesos, que usualmente se realiza de forma aleatoria, sólo obtiene buenos resultados cuando se realiza cerca de los valores óptimos que generalmente son desconocidos. Otro hecho a destacar del estudio realizado por Leerink es que para problemas pequeños, con menos de 3 variables independientes, el método de retropropagación funciona relativamente bien. Sin embargo, cuando crece al número de variables el entrenamiento con retropropagación no proporciona

buenos resultados, quedando atrapado con frecuencia en mínimos locales, aún cuando se realice una inicialización de los pesos cerca de los valores óptimos.

Otro intento para entrenar redes neuronales basadas en unidades producto a través de un algoritmo de tipo gradiente podemos encontrarlo en los trabajos de Saito y Nakano (Saito and Nakano, 1997a; Saito and Nakano, 1997b; Saito and Nakano, 2002). En este caso, se utiliza un algoritmo de aprendizaje denominado RF5, basado en un método quasi-Newtoniano denominado BPQ, junto con un criterio de selección de los mejores modelos basado en el procedimiento MDL. El procedimiento usado calcula la dirección del gradiente descendente a partir del método BFGS (Broydon-Fletcher-Goldfarb-Shanno) (Saito and Nakano, 1997c) y ajusta la longitud del tamaño de paso λ como el mínimo a partir de una aproximación de segundo orden de la función de error. La selección del mejor modelo se realiza minimizando una función que considera el error cometido en la aproximación y el tamaño del modelo determinado por el número de pesos de la red a partir del método (MDL) (Rissanen, 1989; Saito and Nakano, 1997d). El procedimiento obtiene mejores resultados que el algoritmo de retropropagación, aunque se ha aplicado a problemas con datos artificiales simulados en los que la complejidad de la función a modelar es limitada, por ejemplo, la función

$$y = 2 + 3x_1^{-1}x_2^3 + 4x_3x_4^{1/2}x_5^{-1/3}$$

o bien, a los problemas reales de una y dos variables construidos a partir de la ley de Boyle, la ley de Kepler y la ley de Hagen-Rubens (Saito and Nakano, 1997a).

Una ligera modificación del método RF5, denominada RF6, se ha aplicado al modelado de datos reales financieros, en donde los valores de la variable dependiente son numéricos y nominales (Saito et al., 2000) y a la extracción de reglas de regresión (Saito and Nakano, 2002; Saito et al., 2000).

Por último, E. M. Oost y colaboradores, realizan una modificación del algoritmo RF5 para poder resolver problemas con datos reales de mayor

complejidad. La modificación consiste en sustituir el algoritmo BPQ por el algoritmo de Levenberg-Marquardt (Oost et al., 2002).

En general, se puede afirmar que la eficacia de los algoritmos basados en el gradiente está determinada por la complejidad de la superficie de error, y por las condiciones iniciales que se consideren para la búsqueda a partir del gradiente de la función.

3.8.2 Búsqueda aleatoria

Este es el método más sencillo de entrenamiento. Se parte de unos pesos determinados aleatoriamente dentro del rango de búsqueda, cada peso se modifica a partir de un ruido uniforme y se computa el error cuadrático medio. El proceso se detiene cuando se encuentra un valor del error por debajo del umbral señalado o bien se llega a un número de generaciones determinado. Con frecuencia, resulta ineficiente ya que el tiempo de computación puede ser muy grande hasta que se obtiene una solución aceptable (Engelbrecht and Ismail, 1999).

3.8.3 Algoritmo genético (AG)

Se aplica un algoritmo genético con los operadores básicos de selección, reproducción, mutación y cruce en cada generación en búsqueda del mejor individuo de la población. En un algoritmo genético, los individuos de la población compiten para sobrevivir. Los individuos están formados en este caso por el vector de pesos de la red, la función de aptitud de cada individuo se establece a partir del error cuadrático medio obtenido por la red en el conjunto de entrenamiento. Diferentes aplicaciones de los algoritmos genéticos al diseño y entrenamiento de redes neuronales de unidades producto pueden verse en (Engelbrecht and Ismail, 1999; Janson and Frenzel, 1993).

3.8.4 Optimización de colonias o enjambres de partículas (Particle swarm optimization) (PSO)

Se trata de un algoritmo de optimización global basado en poblaciones. Los individuos de la población, denominados partículas, se agrupan en colonias o enjambres. Cada partícula representa una posible solución del problema de optimización. Cada partícula se mueve a través del espacio multidimensional, ajustando su posición en el espacio de búsqueda en función de su posición y de la de las partículas vecinas. El efecto es que las partículas "vuelan" hacia el mínimo, realizando la búsqueda alrededor de una amplia área en un entorno óptimo. La proximidad de cada partícula al óptimo se mide según los valores de una función de aptitud previamente fijada y que dependerá de la naturaleza del problema. En el problema que nos ocupa, el entrenamiento de una red neuronal, cada partícula representa el vector de pesos de la red y la función de aptitud se determina a partir del error cuadrático medio de la red sobre el conjunto de entrenamiento.

Más detalles sobre este método de optimización pueden encontrarse en (Corne et al., 1999; Eberhart et al., 1996).

Una variante del método anterior (CPSO, optimización cooperativa de enjambres de partículas) ha sido diseñado para el entrenamiento de redes de pequeño tamaño basadas en unidades producto (Ismail and Engelbrecht, 1999; Van den Bergh and Engelbrecht, 2001).

3.8.5 LeapFrog

LeapFrog es un método de optimización inspirado en el problema físico del movimiento de una partícula de masa unitaria en un campo de fuerzas conservativo n dimensional.

La energía potencial de la partícula se representa a través de la función que hay que minimizar, en el caso de una red neuronal, el error cuadrático medio. El objetivo es conservar la energía total de la partícula (energía cinética más energía potencial) en el campo de fuerza. El método de optimización consiste en simular el movimiento de la partícula en el campo de fuerza, supervisando la energía cinética y aplicando estrategias adecuadas para reducir la energía potencial. Más información acerca de este método de optimización puede verse en (Snyman, 1982; Snyman, 1983). Este método ha sido usado con éxito en redes neuronales basadas en unidades producto, aunque de tamaño reducido (Engelbrecht and Ismail, 1999).

Como se ha podido constatar existen pocos trabajos que versen sobre el entrenamiento y diseño de redes neuronales de unidades producto. No obstante, entre los diferentes trabajos que abordan el diseño y el entrenamiento de redes neuronales basadas en unidades producto, destacamos el trabajo de Ismail y Engelbrech (Engelbrecht and Ismail, 1999). En este trabajo, los autores llevan a cabo el entrenamiento de redes neuronales de tipo producto aplicando diferentes métodos de optimización global: algoritmo genético estándar, método de optimización con enjambres de partículas (PSO) y LEAPFROG. Los resultados obtenidos muestran que estos algoritmos son eficientes en el entrenamiento de redes basadas en unidades producto, obteniéndose los mejores resultados con el algoritmo genético y con PSO. Es importante señalar, sin embargo, que las funciones usadas en la experimentación corresponden a casos sencillos, funciones polinómicas de una sola variable independiente y de grados 2 y 3. Concretamente, las funciones consideradas en el estudio son $f(x) = x^2$ y $f(x) = x^3 - 0,04x^2$. También se realiza el estudio con la función de Henon definida por la serie temporal $x_t = 1 + 0,03x_{t-2} - 1,4x_{t-1}^2$. En este mismo trabajo se prueba la ineficiencia de los algoritmos de tipo gradiente (gradiente descendente y gradiente conjugado) y del algoritmo de búsqueda aleatoria para el entrenamiento de este tipo de redes. Además, el estudio comparado aplicando unidades de tipo producto

y unidades aditivas muestra que con las redes con unidades de tipo producto se obtienen mejores resultados tanto en el ajuste como en el tamaño de la red.

3.9 Algoritmo evolutivo para el entrenamiento y diseño de redes neuronales de unidades producto

A continuación, se describe el algoritmo evolutivo propuesto para el entrenamiento y diseño de redes neuronales de unidades producto.

3.9.1 Representación funcional de la red neuronal de unidades producto

El algoritmo propuesto pretende modelar un sistema de k variables independientes y una dependiente mediante una función f perteneciente a la siguiente familia de funciones:

$$\mathbb{F} = \left\{ \begin{array}{l} f : A \subset \mathbb{R}^k \rightarrow \mathbb{R}: \\ \\ f(x_1, x_2, ..., x_k) = \sum_{j=1}^{m} \beta_j (\prod_{i=1}^{k} x_i^{w_{ji}}) + t \end{array} \right\} \tag{3.1}$$

donde

$$\beta_j, t \in [-M, M] \subset \mathbb{R},\ w_{ji} \in [-L, L] \subset \mathbb{R},$$
$$i = 1, 2, ..., k,\ j = 1, 2, ..., m$$

siendo el dominio de definición de f el subconjunto A de $\mathbb{R}^k$ dado por

$$A = \left\{ (x_1, x_2, ..., x_k) \in \mathbb{R}^k : 0 < x_i \leq K \right\}$$

La representación de la función la haremos mediante una red neuronal de unidades producto con la estructura mostrada en la Figura 2-6.

El algoritmo intentará encontrar una red neuronal de unidades producto capaz de modelar el sistema.

3.9.2 Descripción general del algoritmo evolutivo propuesto

Mediante técnicas evolutivas, el algoritmo propuesto está basado en una población de individuos, en este caso redes neuronales de unidades producto, e intenta diseñar la estructura de la red y estimar los valores de los pesos de las conexiones. El proceso de búsqueda comienza con una población inicial de redes generada aleatoriamente tanto en estructura como en pesos de las conexiones. A partir de ese momento, la población es modificada generación a generación, utilizando los operadores de mutación y replicación.

Cabe destacar que no se utiliza el operador de cruce ya que, como se ha resaltado anteriormente, hay estudios que afirman que dicho operador no es útil en la evolución de redes neuronales. (Angeline et al., 1994)

3.9.3 Estructura general del algoritmo

En primer lugar construímos la población inicial $B^{(0)}$ de tamaño N_R. A partir de la población inicial $B^{(0)}$, el algoritmo se estructura, siguiendo la metodología propuesta por Deb (Deb, 2003), de la siguiente forma:

Paso 1. Plan de Selección. Seleccionamos el $r\%$ $(r = 90)$ de individuos con mejor aptitud de la población $B^* = B - \left\{ \textit{mejor individuo de } B \right\}$ de cardinal ${N_R}^* = N_R - 1$ y formamos una población P de tamaño $r{N_R}^* / 100$.

Paso 2. Plan de Generación. Aplicamos mutación estructural a cada elemento de la población P y mutación paramétrica sólo a los $(100 - r)N_R^* / 100$ mejores individuos de P (los individuos a los que se les aplica la mutación paramétrica se corresponden en realidad con el $(100 - r)\%$ de mejores individuos de la población B^*). La población obtenida después del plan de generación la denominaremos C. Es fácil ver que el cardinal de C es igual a ${N_R}^* = N_R - 1$.

Paso 3. Plan de Reemplazamiento. Elegimos todos los elementos de B^* para ser reemplazados.

Paso 4. Plan de Sustitución. Los elementos de B^* son reemplazados por los elementos de C. Por tanto, la nueva población es $B = \left\{ Mejores\ de\ B \right\} \cup C$. De esta forma, el tamaño de la población se mantiene constante durante la evolución. Se puede observar, por último, que se trata de un algoritmo elitista ya que el mejor individuo de B se mantiene para la siguiente generación.

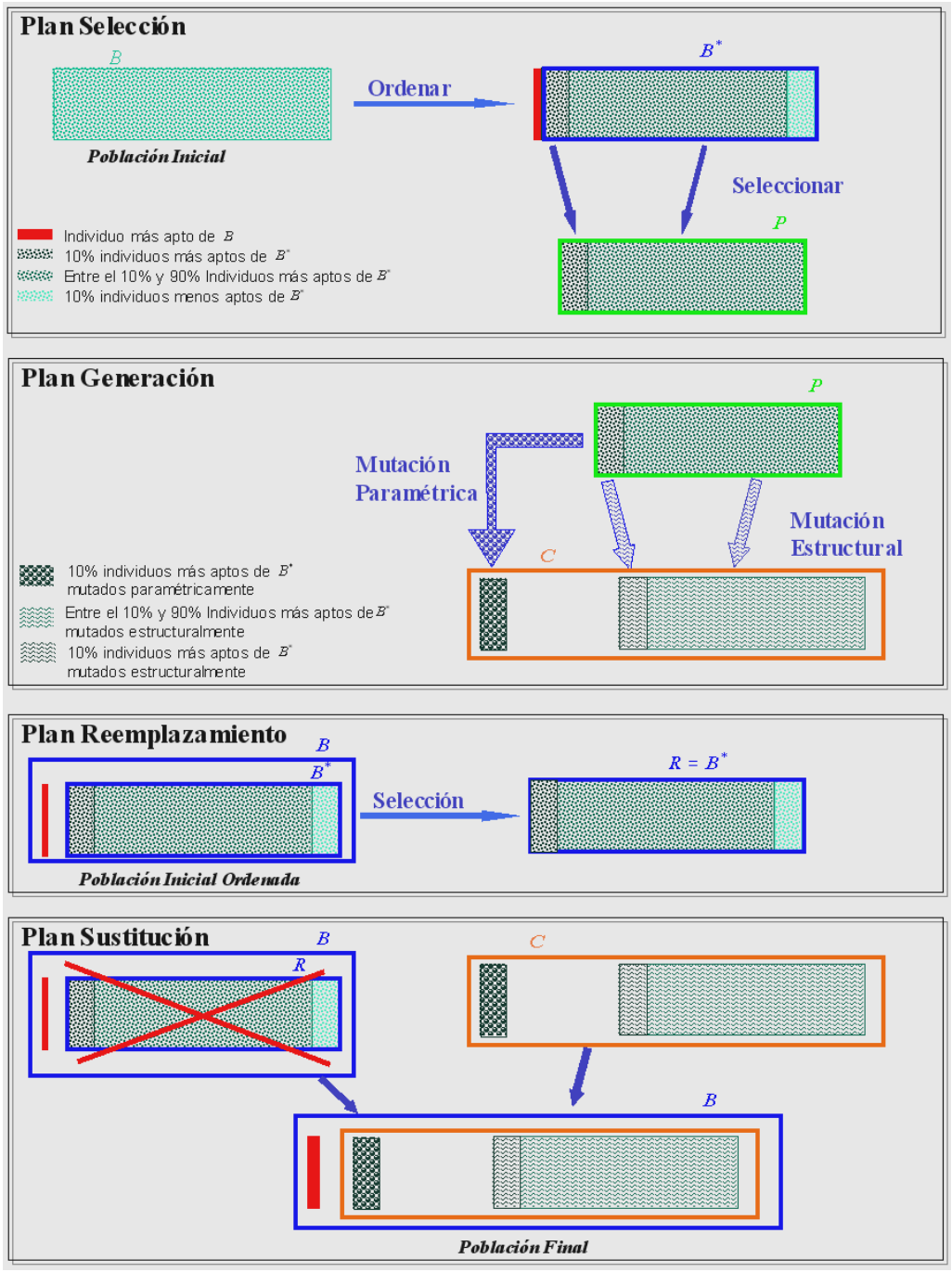


Figura 3-8. Descripción Gráfica del Algoritmo según la metodología de Deb.

3.9.4 Representación de los individuos

La codificación elegida, tanto para representar los pesos de las conexiones como la topología de la red, ha sido directa. Es decir, los pesos de las conexiones y la topología se representan mediante un vector de números reales, aprovechando que se va a evolucionar conjuntamente tanto los pesos de las conexiones como la estructura de la red.

Debido a las características de los individuos de la población, (redes neuronales de unidades producto, ver Figura 2-6) que va a evolucionar el algoritmo hemos realizado una simplificación en el modelo de representación de la red con el objetivo de agilizar el algoritmo. Recordemos en primer lugar, que la red consta de tres capas, donde la capa de entrada y la capa de salida son fijadas de antemano, ya que la capa de salida solo tendrá una salida (un nodo) y la capa de entrada tantos nodos como variables independientes tenga el sistema a modelar. Además la red incluye forzosamente sesgo. Por tanto, la parte variable será tanto el número de nodos en la capa oculta como el número de conexiones entre los nodos de la capa de entrada y oculta. Por su parte, un nodo de la capa oculta siempre estará conectado con el nodo de la capa de salida, y no existirán conexiones entre nodos de la misma capa y conexiones entre la capa de entrada y la de salida.

	0	**1**	**2**	**.....**	$k-1$	k
Nodo 1- P_1	β_1	w_{11}	w_{12}	.....	w_{1k-1}	w_{1k}
Nodo 2- P_2	β_2	w_{21}	w_{22}	.....	w_{2k-1}	w_{2k}
$\vdots$	$\vdots$	$\vdots$	$\vdots$	.....	$\vdots$	$\vdots$
Nodo m **-** P_m	β_m	w_{m1}	w_{m2}	.....	w_{mk-1}	w_{mk}

Figura 3-9. Estructura de datos asociada a la red neuronal de unidades producto representada en la Figura 2-6

Por tanto, la red se representará mediante una lista de m nodos ocultos. Además, almacenaremos el sesgo t como un parámetro más de la red neuronal.

El nodo de la capa oculta j constará de un vector P_j de pesos de dimensión $k+1$. Dicha dimensión es fija durante el proceso de evolución ya que se corresponde con el número de variables independientes. Sin embargo, el número de nodos de la red en la capa oculta es variable durante el proceso de evolución.

Según se observa en la Figura 3-9, el vector de pesos $P_j[i]$ representará el valor de los pesos de las conexiones entre el nodo j de la capa oculta y el nodo de la capa de salida si $i = 0$; mientras que si $i \neq 0$ será con los nodos de la capa de entrada. Si $P_j[i]$ es un valor w_{ji} distinto de 0 indicará que existe una conexión entre el nodo i de la capa de entrada y el nodo j de la capa oculta y será el peso de dicha conexión; en otro caso, significará que no existe la conexión. $P_j[0] = \beta_j$ representa el valor del peso de la conexión entre el nodo j de la capa oculta y el nodo de la capa de salida. Obsérvese que, por las restricciones descritas anteriormente, se cumple que $P_j[0] \neq 0$.

3.9.5 Generación de la población inicial

El algoritmo se inicia generando aleatoriamente un número mayor de redes que el usado posteriormente en el proceso de evolución, concretamente generamos $10 * N_R$ redes, siendo N_R el número de redes que tendrá la población durante el proceso de evolución.

Para la generación aleatoria de una red, se comienza eligiendo el número de nodos ocultos a partir de una distribución uniforme en el intervalo $(0, m/2]$, donde m corresponde al número máximo de nodos ocultos de cada una de las redes de la población. De esta forma formamos la población inicial con modelos más sencillos. El número de conexiones entre cada nodo de la capa oculta y los

nodos de la capa de entrada queda determinado a partir de una distribución uniforme en el intervalo $(0,k]$, siendo k el número de variables independientes. Siempre se añade una conexión entre el nodo de la capa oculta y el nodo de la capa de salida. Definida la topología de la red, se asigna a cada conexión un peso, a partir de una distribución uniforme en el intervalo $[-L,L]$ para los pesos entre la capa de entrada y la oculta y $[-M,M]$ para los pesos entre la capa oculta y la de salida y el sesgo. Por último, la población inicial que constituye la solución base B se forma seleccionando las N_R redes mejores (con mejor aptitud) entre las $10 * N_R$ generadas.

3.9.6 Función de aptitud

Sea $D = \left\{(\mathbf{x_i}, \hat{y}_i) : i = 1,2,...,n\right\}$ el conjunto de datos de entrenamiento, donde n es el número de ejemplos. Consideraremos el siguiente error:

- Error medio de la suma de residuos al cuadrado $MSE(g)$ de cada individuo g de la población como:

$$MSE(g) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \tag{3.2}$$

donde $\hat{y}_i$ son los valores esperados.

Con el objeto de obtener modelos más sencillos, se define un término de regularización que hace referencia a la simplicidad de la red $S_{estruct}$.

$$S_{estruct}(g) = \gamma\left(1 - \frac{n_n(g)}{m}\right) + (1-\gamma)\left(1 - \frac{n_c(g)}{n_{cMax}}\right), \quad \gamma \in [0,1] \tag{3.3}$$

Éste término tiene en cuenta el número de conexiones con los que cuenta la red g, $(n_c(g))$, con respecto al número máximo de conexiones permitidas, $(n_{eMax} = m(k+1))$, así como el número de nodos que tiene la red g en la capa oculta, $(n_n(g))$ con respecto al número máximo de nodos m que puede tener la red g. El parámetro γ indica el factor que más se favorece dentro de la

estructura de la red, si $\gamma < 0.5$ se premiará el que la red tenga menos conexiones, si por el contrario $\gamma > 0.5$ se favorecerá que la red tenga menos nodos.

Por último, se define la función de aptitud $A(g)$ como una combinación lineal de una transformación estrictamente decreciente de la función de error (3.2) considerada anteriormente y del factor de simplicidad del modelo (3.3).

$$A(g) = \alpha\left(\frac{1}{1 + MSE(g)}\right) + (1-\alpha)S_{estruct}(g), \quad \alpha \in [0,1] \tag{3.4}$$

De tal manera que si $\alpha > 0.5$ le damos mayor importancia al error de predicción que a la complejidad de la red, y viceversa. Obsérvese que cuando $\alpha = 1$ no se tiene en cuenta la complejidad de la red en la función de aptitud. La determinación óptima de los valores de α, γ no es fácil. En nuestro caso, hemos utilizado siempre $\gamma = 0.75$ dándole mayor importancia al número de sumandos de la función y $\alpha = 0.9$ en el caso de querer obtener modelos más interpretables y $\alpha = 1$ en el caso de querer ajustar lo máximo posible a los datos de predicción, desestimando, de esta forma, el término de regularización.

3.9.7 Mutación paramétrica

La mutación paramétrica es un operador basado en técnicas de enfriamiento simulado (Otten and van Ginneken, 1989). La severidad de la mutación que se le aplica a un individuo g se basa en la función de temperatura $T(g)$ dada por:

$$T(g) = 1 - A(g) \quad \text{, donde} \quad 0 \leq T(g) < 1 \tag{3.5}$$

De esta manera, la temperatura está determinada por lo cerca o lejos que nos encontremos de la solución del problema. Las mutaciones paramétricas se aplican a cada parámetro w_{ji} o β_j de la red, añadiéndoles una variable aleatoria normal con media 0, y donde la varianza de la distribución depende de la temperatura. Así, redes con temperatura alta (aptitud baja) son mutadas

severamente mientras que redes con temperatura baja (aptitud alta) sufrirán mutaciones suaves. Esto permite una búsqueda menos afinada inicialmente, pero conforme nos acercamos a la solución la búsqueda será más afinada. Particularmente, los exponentes w_{ji} de las unidades producto que están representados con los pesos de las conexiones entre los nodos de la capa de entrada y los nodos de la capa oculta son modificados de la siguiente manera:

$$\begin{aligned} &w_{ji}(t+1) = w_{ji}(t) + \xi_1(t), \\ &i = 1,...,k,\ j = 1,...,m \end{aligned} \tag{3.6}$$

donde $\xi_1(t) \sim N(0, \alpha_1(t)T(g))$ representa una variable aleatoria con distribución normal de media 0 y varianza $\alpha_1(t)T(g)$.

Mientras que los coeficientes β_j de la función, que representan los pesos de las conexiones entre los nodos de la capa oculta y el nodo de la capa de salida y el sesgo, se modifican como sigue:

$$\begin{aligned} &\beta_j(t+1) = \beta_j(t) + \xi_2(t), \\ &j = 1,...,m \end{aligned} \tag{3.7}$$

donde $\xi_2(t) \sim N(0, \alpha_2(t)T(g))$ representa una variable aleatoria con distribución normal de media 0 y varianza $\alpha_2(t)T(g)$.

2. Repetir para cada nodo oculto j de la red;
   a. Repetir para todos las conexiones existentes entre el nodo oculto y los nodos de entrada i;
      i. Calcular el incremento de peso para el enlace, según la siguiente expresión: $\Delta w_{ji} = N(0, \alpha_1 T(g))$;
      ii. Calcular el nuevo $w_{ji} = \Delta w_{ji} + w_{ji}$;
   b. Fin Repetir;
   c. Calcular el incremento de peso para la conexión con el nodo de salida, según la siguiente expresión: $\Delta\beta_j = N(0, \alpha_2 T(g))$;
   d. Calcular el nuevo $\beta_j = \Delta\beta_j + \beta_j$;
   e. Calcular la Aptitud de la nueva red;
   f. Si $\Delta A > 0$, entonces Aceptar los cambios;
   g. En otro caso, Si $e^{\frac{\Delta A}{T}} < U(0,1)$ se aceptan, si no se rechazan;
3. Fin Repetir;

Figura 3-10. Descripción en Pseudo-Código de la Mutación Paramétrica

El sesgo se considera a efecto de la mutación paramétrica como un nodo en la capa oculta que tiene como salida 1.

Como se puede observar en la Figura 3-10, las mutaciones son realizadas nodo a nodo de la capa oculta, modificándose los pesos de las conexiones que entran o salen de dicho nodo. La aptitud del individuo g se recalcula[20] (en la Figura 3-10 paso e) y se aplica la técnica de enfriamiento simulado. Siendo ΔA el incremento de la función de aptitud antes y después de la mutación: Si $\Delta A \geq 0$ se acepta el cambio y si $\Delta A < 0$ se acepta con una probabilidad igual a $\exp(\Delta A / T(g))$.

3.9.7.1 Evolución de los parámetros

Obsérvese que la severidad de la modificación de los exponentes w_{ji} ha de ser diferente a la de los coeficientes β_i, por tanto los parámetros que afectan a las varianzas de las distribuciones normales deben verificar que $\alpha_1(t) \ll \alpha_2(t), \quad \forall t \in \mathbb{N}$, los cuales cambian de forma adaptativa en cada generación, de acuerdo a una regla prefijada. Los parámetros α_1 y α_2 que, junto a la temperatura, determinan las varianzas de la distribuciones normales van cambiando durante el proceso de evolución, por lo que se realiza un aprendizaje adaptativo, a diferencia de lo que ocurre en otros trabajos (Angeline et al., 1994; García-Pedrajas et al., 2002) donde estos parámetros permanecen fijos durante toda la evolución.

[20] Es claro que, como solo se modifican los pesos de las conexiones de un nodo oculto, solo habrá que recalcular parte de la red, pues el resto de la red permanece constante, ya que las salidas para cada ejemplo de entrenamiento se almacenan en memoria para cada nodo de la capa oculta. Por tanto, al final de la mutación paramétrica se habrá recalculado la red completa una sola vez, aunque realmente se haya realizado parte a parte.

Durante la experimentación del modelo utilizando los parámetros α_1 y α_2 fijos, hemos observado que cuando nos acercamos a la solución, el proceso de evolución se para puesto que los valores de α_1 y α_2 son demasiado grandes para las variaciones que se deben realizar. En cambio, si α_1 y α_2 se fijan con valores pequeños desde el principio, el proceso de evolución es demasiado lento y con frecuencia queda atrapado en mínimos locales. Asimismo, observamos que si los valores de α_1 y α_2 toman valores pequeños en un momento dado, si el proceso de búsqueda salta a otra zona donde no es necesario que α_1 y α_2 sean tan pequeños, este proceso se ralentiza sin motivo.

Por tanto, adaptando los valores de α_1 y α_2 al proceso de evolución, evitamos quedar atrapados en mínimos locales y aceleramos el proceso de evolución cuando las condiciones del proceso lo permitan. Concretamente, la evolución de los parámetros α_1 y α_2 viene dada por:

$$\alpha_i(t+1) = \begin{cases} (1+\beta)\alpha_i(t), & \text{si } A(g_s) > A(g_{s-1}), \quad \forall s \in \{t, t-1, ..., t-\rho\} \\ (1-\beta)\alpha_i(t), & \text{si } A(g_s) = A(g_{s-1}), \quad \forall s \in \{t, t-1, ..., t-\rho\} \\ \alpha_i(t) & \text{en el resto de casos} \end{cases} \quad (3.8)$$

$i = 1,2$, donde $A(g_s)$ representa la aptitud de la red con aptitud máxima en la generación s .

Los valores considerados para los parámetros $\alpha_1, \alpha_2, \beta$ y ρ en todos los experimentos realizados han sido: $\alpha_1(0) = 1, \alpha_2(0) = 5, \beta = 0.1, \rho = 10$, aunque el algoritmo es bastante robusto con respecto a los valores de $\alpha_1(0)$ y $\alpha_2(0)$.

Teniendo en cuenta que una generación es satisfactoria si el mejor individuo de la población es mejor que el mejor de la generación anterior. Si observamos varias generaciones satisfactorias consecutivamente, es indicativo de que los mejores de la población se encuentran en una buena región dentro del espacio de búsqueda. En este caso, incrementamos la varianza de la mutación esperando encontrar mejores soluciones alrededor de la solución óptima. Si la

aptitud del mejor individuo es constante durante varias generaciones consecutivas decrementamos la varianza de la mutación. En otros casos, mantenemos constante dicha varianza.

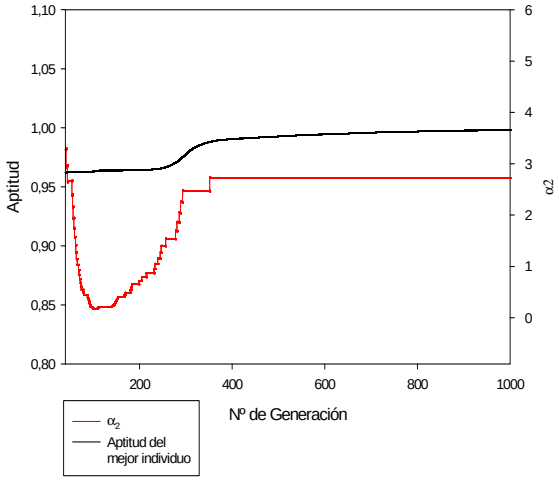


Figura 3-11. Ejemplo de evolución de la aptitud del mejor de la población y valor de α_2

Podemos observar en la Figura 3-11, un ejemplo de cómo α_2 disminuye de valor cuando la aptitud del mejor individuo no mejora, es decir, cuando el algoritmo queda atrapado en un mínimo local. El valor de α_2 va disminuyendo hasta que el proceso sale del mínimo local, por el efecto de las mutaciones estructurales. Se puede observar cómo la pendiente de la curva de aptitud del mejor individuo aumenta conforme aumenta de nuevo el valor de α_2; puesto que otra vez debe tomar fuerza la mutación paramétrica, hasta que llega un momento que se estanca el valor de α_2 para continuar con la evolución.

3.9.8 Mutación estructural

La mutación estructural es más compleja, pues implica una modificación en la estructura de la red neuronal. La mutación estructural permite realizar la

exploración de diferentes regiones en el espacio de búsqueda y mantener la diversidad de la población. Utilizamos cinco mutaciones estructurales diferentes: añadir nodos (AN), eliminar nodos (EN), añadir conexiones (AC), eliminar conexiones (EC) y unir nodos (UN). Las cuatro primeras son similares a las utilizadas en el modelo GNARL (Angeline et al., 1994).

Para cada mutación (excepto en la mutación unir nodos) hay un valor mínimo, Δ_{MIN} y un valor máximo Δ_{MAX}. El número de elementos (nodos y conexiones) involucrados en la mutación se calcula según la siguiente expresión:

$$\Delta_{MIN} + \lfloor uT(g)(\Delta_{MAX} - \Delta_{MIN}) \rfloor$$

donde u es una variable aleatoria uniformemente distribuida en [0,1].

En GNARL, se aplican todas las mutaciones estructurales de manera secuencial sobre cada individuo sujeto a este tipo de mutación. En nuestro modelo, con el objetivo de no modificar en exceso la estructura de la red, intentando no modificar sustancialmente el comportamiento de la red; la probabilidad de aplicar cada tipo de mutación es igual $T(g)$ en la misma generación y para cada red g. Es decir, al principio de la evolución se permitirán grandes cambios, y conforme vaya progresando el algoritmo los cambios serán menores. Se forzará a que, al menos, se le aplique una mutación estructural a la red g. La mutación se elegirá aleatoriamente, en el caso de que no se haya seleccionado ninguna mutación estructural porque tenga una temperatura tan baja que por probabilidad no se haya elegido ninguna mutación estructural para aplicar.

3.9.8.1 Añadir nodo - AN

El nodo de la capa oculta se añade con conexiones entre el nodo de la capa oculta y el nodo de la capa de salida con pesos aleatorios y conexiones aleatorias con los nodos de la capa de entrada. Los valores que tomarán los pesos

y el número de conexiones se tomarán según el rango inicial considerado para la generación inicial de individuos.

3.9.8.2 Eliminar nodo – EN

Seleccionamos un nodo de la capa oculta al azar y lo eliminamos junto a sus conexiones.

3.9.8.3 Añadir conexiones – AC

Elegimos aleatoriamente un nodo de la capa de entrada y de la capa oculta y establecemos una conexión entre ellos con peso aleatorio. Al igual que en el caso de Añadir nodo, el peso se elige en el rango definido para la generación de los individuos de la población inicial.

3.9.8.4 Eliminar conexión - EC

Elegimos aleatoriamente una conexión entre un nodo de la capa de entrada y un nodo de la capa oculta y lo borramos.

3.9.8.5 Unir nodos - UN

La mutación unir nodos consiste en sustituir dos nodos de la capa oculta a y b elegidos aleatoriamente por otro nodo c. Las conexiones comunes se conservan y las no comunes se conservan con una probabilidad de 0.5. Los pesos de las conexiones no comunes se mantienen y los pesos de las conexiones comunes se calculan de la siguiente manera:

$$\beta_c = \beta_a + \beta_b, w_{ic} = \frac{w_{ia} + w_{ib}}{2}$$

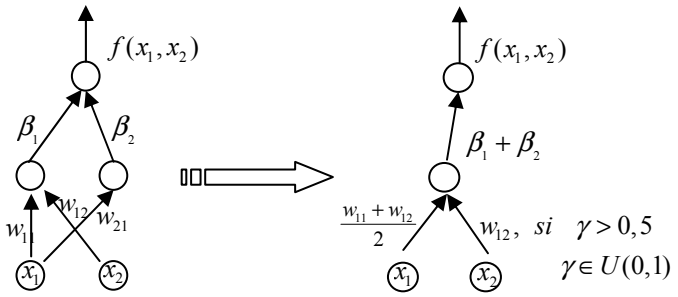


Figura 3-12. Ejemplo gráfico de la mutación Unir Nodos

Como se puede observar en la Figura 3-12, el objetivo fundamental de la mutación Unir Nodos es la de simplificar el modelo eliminando nodos y promediando el nuevo peso a partir de los anteriores. Esta mutación solo afectará a dos nodos de la capa oculta cada vez que se aplique a un individuo.

3.9.9 Criterio de parada

La condición de parada del algoritmo se alcanza cuando se llega a una aptitud previamente marcada, o bien, los valores $\alpha_1(t)$ y $\alpha_2(t)$ llegan a un valor próximo a cero, o bien no mejora, durante un número de generaciones consecutivas ni la media del 10% mejor de la población ni el mejor de la población.

Consideramos que si los valores $\alpha_1(t)$ y $\alpha_2(t)$ llegan a un valor próximo a cero, las modificaciones que se realizarán sobre los pesos de las conexiones mediante las mutaciones paramétricas van a ser mínimos por lo que el modelo no va a mejorar significativamente. Además, si se han llegado a estos valores es porque el mejor individuo no ha mejorado durante muchas generaciones consecutivas, y lo más probable es que no mejore más. En este caso pararemos el algoritmo y devolveremos el individuo más apto.

Por otra parte, debido a las características del algoritmo, el 10% mejor de la población debe ir mejorando generación a generación, no ocurre así con el resto

de la población que puede empeorar debido a la naturaleza de la mutación estructural que puede "mutilar" seriamente a una red haciendo que su aptitud disminuya considerablemente. Por tanto, si no mejoramos la media del 10% mejor de la población ni el mejor individuo durante un número de generaciones consecutivas, al igual que en el caso anterior probablemente no mejoremos el mejor individuo por lo que detendremos el proceso y devolveremos el mejor individuo de la población.

4 Modelos de Regresión a partir de la Evolución de Redes Neuronales de Unidades Producto

En el presente capítulo analizamos la capacidad de aprendizaje del algoritmo evolutivo propuesto en el capítulo anterior, aplicándolo a diferentes tipos de problemas de regresión tanto simulados como reales.

Por una parte, vamos a generar distintos conjuntos de datos obtenidos a partir de funciones de varias variables diseñadas con ese propósito. Comprobaremos si el algoritmo es capaz de encontrar una función que se acerque lo más posible a la que generó los datos. Compararemos, igualmente, dicho algoritmo evolutivo con un algoritmo clásico de optimización como es el de Levenberg-Marquardt.

En una segunda parte, comprobaremos el rendimiento del algoritmo evolutivo a dos problemas típicos benchmark: la función propuesta por Saito y Nakano en (Saito and Nakano, 2002) y la función de Sugeno presentada en (Sugeno and Kang, 1988).

Por último, se analizará un problema real de crecimiento microbiano y se interpretarán los resultados obtenidos.

Comencemos por definir el problema de regresión.

4.1 Problema de regresión

El problema de regresión que pretendemos resolver puede describirse de forma general como sigue. Consideremos el par

$$(y_l, \mathbf{x}_l) = (y_l; x_{l1}, x_{l2}...., x_{lp}) \,, \quad l = 1,2,...,n \tag{3.9}$$

donde los valores y_l han sido generados a partir del modelo

$$y_l = g(\mathbf{x}_l, \hat{\theta}) + \varepsilon_l \quad , l = 1,2,...,n \tag{3.10}$$

siendo $\{\mathbf{x}_i\}$ las variables independientes del problema. La función g es una función desconocida definida en el espacio euclídeo de dimensión p con valores en $\mathbb{R}$:

$$g : \mathbb{R}^p \rightarrow \mathbb{R}$$

Mientras que $\{\varepsilon_l\}$, para $l = 1,2,...,n$, son variables aleatorias independientes entre sí e independientes de $\{\mathbf{x}_i\}$, de media cero. El objetivo del problema de regresión es construir el estimador $\hat{g}$ en función de los datos $(y_l, \mathbf{x}_l)$, para aproximar la función desconocida g a una nube de puntos o conjunto de datos, y usar esta estimación para predecir el valor de y dado un nuevo valor de $\mathbf{x}$:

$$\mathbf{y} = \hat{g}(\mathbf{x}, \hat{\boldsymbol{\theta}}) \tag{3.11}$$

Para resolver el problema de regresión consideraremos la familia de funciones definidas a partir de redes neuronales basadas en unidades producto. Concretamente, consideraremos la familia de funciones F definidas por:

$$F = \left\{ f : A \subset \mathbb{R}^k \rightarrow \mathbb{R} : f(\mathbf{x}, \boldsymbol{\theta}) = \sum_{j=1}^{m} \beta_j \left(\prod_{i=1}^{k} x_i^{w_{ji}} \right) \right\}$$

donde $\mathbf{x} = (x_1, x_2, ..., x_k)$, $\boldsymbol{\theta} = (\beta_j, w_{ij})_{i,j}$, $\beta_j \in [-M, M] \subset \mathbb{R}$, $w_{ij} \in [-L, L] \subset \mathbb{R}$, $i = 1,2,...,k$, $j = 1,2...,m$ y $m \in \mathbb{N}$.

4.2 Reconocimiento de Funciones. Datos Simulados

4.2.1 Metodología

Se han diseñado dos funciones de tres y cuatro variables, f_1 y f_2, respectivamente. Los modelos se han diseñado para que los sumandos de las funciones de tipo potencial tengan exponentes positivos y negativos y los valores sean reales. Además en la función f_2 se ha incorporado un término independiente, para comprobar que el algoritmo es capaz de reconocer funciones con este término, al igual que la función f_1 se ha definido sin él.

$$f_1(x_1, x_2, x_3) = x_1^2 \sqrt{x_2} \sqrt[3]{x_3^2} - \frac{2x_2^2 x_3}{x_1} + 2x_1^2 x_2^2$$

$$f_2(x_1, x_2, x_3, x_4) = 3 - \frac{2x_1^2 x_3}{\sqrt{x_2}} + 3x_3^3 x_4^{2,1} \sqrt[3]{\frac{x_2}{x_1}}$$

Se han generado 100 ejemplos de forma aleatoria, de los cuales los 75 primeros se utilizaron para entrenamiento y los 25 restantes para generalización.

Los valores de las variables independientes se han elegido aleatoriamente en el intervalo [0,1]. No se ha escalado el valor de la variable dependiente con el objeto de obtener el mismo modelo.

Los valores de los parámetros del algoritmo evolutivo para estos modelos han sido los mostrados en la Tabla 4-1.

Parámetro	Valor
Parámetros de la Población	
Tamaño de la Población	1000
Máximo número de nodos en la capa oculta m	5
Número de variables independientes k	3,4[21]
Intervalo de los exponentes	[0,5]
Intervalo de los coeficientes	[-5,5]
Parámetros del Algoritmo Evolutivo	
α	0,9
γ	0,75
r	90
Parámetros de la Mutación Paramétrica	
$\alpha_1(0)$	1
$\alpha_2(0)$	5
β	0,1
ρ	10
Parámetros de las Mutaciones Estructurales: Intervalos $[\Delta_{MIN}, \Delta_{MAX}]$	
AN	[1,2]
EN	[1,3]
AC	[1,10]
EC	[1,10]
Cóndición de Parada	
Nº Máx. de Generaciones	6000
Nº Máx. de Generaciones sin mejorar media 10% ni el mejor	20

Tabla 4-1. Parámetros utilizados en el algoritmo evolutivo para el modelado de las funciones simuladas f_1 y f_2

[21] 3 para el caso de f_1, 4 para el caso de f_2.

Debido a las características aleatorias propias de los algoritmos evolutivos y con el fin de poder realizar análisis estadísticos, se ha ejecutado 30 veces el algoritmo.

4.2.2 Resultados obtenidos

A continuación se muestran los valores estadísticos del *MSE* para cada una de las funciones: Media, desviación típica, mejor y peor valores tanto para el conjunto de entrenamiento como para el de generalización.

El *MSE* se define como:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(f(\mathbf{x}_i) - y_i)^2$$

donde n es el número de patrones, $f(\mathbf{x}_i)$ es la salida teórica para el patrón i y y_i es la salida del modelo para el mismo patrón.

Función	Conjunto	Media	Desv,	Mejor	Peor
f_1	Entrenamiento	0,0094	0,0225	0	0,1268
	Generalización	0,0122	0,0179	0	0,0225
f_2	Entrenamiento	0,0015	0,0011	0	0,0024
	Generalización	0,0028	0,0022	0	0,0046

Tabla 4-2. Valores del *MSE* obtenidos a partir del algoritmo evolutivo para las funciones simuladas f_1 y f_2, para los conjuntos de entrenamiento y generalización

Hay que destacar que el algoritmo evolutivo fue capaz de encontrar exactamente la función en 15 ocasiones para la función f_1 y en 7 ocasiones para la función f_2.

4.2.3 Comparativa con algoritmo de búsqueda local Levenberg-Marquardt

Para comparar el algoritmo evolutivo con el algoritmo de Levenberg-Marquardt (L-M) (Levenberg, 1944; Marquardt, 1963), se ha aplicado el algoritmo de búsqueda local sobre los 1000 individuos de la población inicial del algoritmo evolutivo y 30 veces consecutivas, al igual que el algoritmo evolutivo. Es decir, se ha ejecutado el algoritmo desde 1000 puntos distintos de inicio. El método y los parámetros utilizados para generar los puntos de partida del algoritmo (L-M) han sido los mismos que para el algoritmo evolutivo.

Función	Conjunto	Media	Desv,	Mejor	Peor
f_1	Entrenamiento	3,8789	0,5941	1,9357	4,4485
	Generalización	222,9383	917,5231	10,6113	5059,9850
f_2	Entrenamiento	0,5967	0,1073	0,3594	0,9413
	Generalización	0,9226	0,3087	0,6288	10,4865

Tabla 4-3. Valores del MSE obtenidos a partir del algoritmo de Levenberg-Marquardt para las funciones simuladas f_1 y f_2, para los conjuntos de entrenamiento y generalización en 30 ejecuciones

Los valores de la Tabla 4-3 se muestra los valores medios, varianzas, mejor y peor modelo de los mejores individuos obtenidos en cada una de las 30 generaciones. Se puede observar que el algoritmo evolutivo ofrece mejores resultados que el algoritmo de búsqueda local en todos los aspectos. Además el

algoritmo de Levenberg-Marquardt no ha encontrado ni una sola vez los modelos propuestos[22].

Este hecho nos hace pensar en la dificultad de la superficie de error de este tipo de funciones y la incapacidad que tiene el algoritmo de Levenberg-Marquardt para el entrenamiento de este tipo de redes por sí solo.

4.3 Problemas de prueba

En esta sección se va a comparar el algoritmo evolutivo con los resultados obtenidos por otros algoritmos de modelado sobre problemas de prueba o benchmark. Más concretamente, los propuestos por (Saito and Nakano, 2002) y (Jankowsky, 1999).

4.3.1 Función de Saito-Nakano

La función de Saito-Nakano es una versión modificada de la función propuesta por Sutton y Matheus en (Sutton and Matheus, 1991). La función original era

$$y = 2 + 3x_1x_2 + 4x_3x_4x_5$$

la modificación realizada por Saito y Nakano es:

$$y_{S-N} = 2 + 3x_1^{-1}x_2^3 + 4x_3x_4^{1/2}x_5^{-1/3}$$

Cada ejemplo es generado de la siguiente forma: cada valor de las variables $x_1,...,x_5$ son generadas aleatoriamente en el intervalo [0,1]. El valor de la variable dependiente y se calcula mediante la ecuación de la función modificada por Saito y Nakano. Además para el entrenamiento se añaden las

[22] Si se le fija previamente el número de sumandos y las variables que intervienen en cada sumando sí la encuentra en algunas ocasiones.

variables independientes irrelevantes $x_6,...,x_9$, con valores generados aleatoriamente en el intervalo [0,1]. El número de ejemplos que se generan es de 200. Saito Y Nakano no utilizan conjunto de generalización, en nuestro caso, utilizamos el mismo conjunto de ejemplos para entrenar y generalizar.

El algoritmo propuesto por Saito y Nakano denominado RF5, como se ha visto en la sección 3.8.1, consiste en un método de aprendizaje basado en el gradiente conjugando métodos de poda utilizando el método MDL.

Saito y Nakano realizan el entrenamiento fijando de antemano el número de sumandos que va a tener la función (2,3,4) y utilizan el *MSE* como medida de error. En nuestro caso, fijamos como máximo 5 sumandos.

Los parámetros del algoritmo evolutivo utilizados son los mismos que los considerados para las funciones f_1 y f_2 (ver Tabla 4-1), a excepción del número de variables independientes k que será 9.

Saito y Nakano repiten el experimento 200 veces, en nuestro caso lo hacemos 30. Los resultados obtenidos son los siguientes:

Algoritmo	Nº Sumandos	Media	Desv, Standard	Mejor
Saito-Nakano	2	1,3210	0	1,3170
	3	0,0330	0,1720	0
	4	0	0	0
Algoritmo Evolutivo	Máximo 5	0,0005	0,0030	0

Tabla 4-4 Comparativa de resultados entre el algoritmo RF5 propuesto por Saito-Nakano y el algoritmo evolutivo

Cabe destacar, que nuestro modelo no necesita fijar de antemano el número de sumandos y que en la gran mayoría de las ejecuciones se encontró el modelo propuesto.

Además el algoritmo evolutivo ha sido capaz de descartar las variables que no afectaban a la variable dependiente, quitándolas del modelo y encontrando, de esta forma, la misma función que generó los ejemplos.

4.3.2 Función de Sugeno

El segundo problema de prueba es la función de Sugeno propuesta en (Sugeno and Kang, 1988), y se define como sigue:

$$f(x_1, x_2, x_3) = \left(1 + \sqrt{x_1} + \frac{1}{x_2} + \frac{1}{\sqrt{x_3^3}}\right)^2$$

Se trata de un problema utilizado con frecuencia para probar la capacidad de aproximación de sistemas adaptativos de aprendizaje.

Los parámetros son análogos a los utilizados para el reconocimiento de la función f_1, a excepción del número de sumandos m, que será 11, el número máximo de generaciones será 12000, y el valor de $\alpha = 1$, descartando de esta forma el término de regularización $S_{estruct}$ en la función de aptitud.

Los datos de entrenamiento se generan de la siguiente manera: Se generan 216 ejemplos para entrenamiento y 125 para generalización. Los valores de las variables independientes para el conjunto de entrenamiento se eligen de manera aleatoria en el intervalo [1,6] y los del conjunto de generalización de la misma forma en el intervalo [1.5,5.5].

Los resultados obtenidos mediante el algoritmo evolutivo se comparan con los resultados obtenidos en los trabajos de (Horikawa et al., 1992; Jankowsky, 1999; Kosinski and Weigl, 1995; Sugeno and Kang, 1988)

La medida de comparación utilizada es el porcentaje medio de error (APE)[23]

$$APE = \frac{1}{n}\sum_{i=1}^{n}\left|\frac{f(\mathbf{x}_i) - y_i}{y_i}\right| \times 100\%$$

donde n es el número de ejemplos, y_i la salida del modelo e $f(\mathbf{x}_i)$ es la salida teórica para el ejemplo i.

Según los resultados mostrados en la Tabla 4-5 (Jankowsky, 1999), podemos concluir que los resultados medios de generalización obtenidos tras 30 ejecuciones mediante el algoritmo evolutivo, solo es mejorado por el método IncNet Rot[24]. En el caso del conjunto de entrenamiento también es superado por el método IncNet.

En el Capítulo 5, se propone un algoritmo híbrido que mejorará los resultados obtenidos por el método IncNet Rot.

[23] Average Porcentaje Error

[24] IncNet, Redes neuronales Incrementales de base biradial. IncNet Rot, Redes neuronales Incrementales de base biradial con rotación.

Modelo	APE_E	APE_G
GMDS model Kongo	4,7000	5,7000
Fuzzy model 1 Sugeno	1,5000	2,1000
Fuzzy model 2 Sugeno	0,5900	3,4000
FNN Type 1 Horikawa	0,8400	1,2200
FNN Type 2 Horikawa	0,7300	1,2800
FNN Type 3 Horikawa	0,6300	1,2500
M-Delta model	0,7200	0,7400
Fuzzy INET	0,1800	0,2400
Fuzzy VINET	0,0760	0,1800
IncNet	0,1190	0,1220
Algoritmo Evolutivo	**0,1208**	**0,1035**
IncNet Rot	0,0530	0,0610

Tabla 4-5. Resultados obtenidos por los diferentes modelos propuestos tanto en entrenamiento APE_E como en generalización APE_G en la aproximación de la función de Sugeno bajo las mismas condiciones iniciales y con 11 nodos en la capa oculta como máximo.

4.4 Modelos de Crecimiento Microbiano

4.4.1 Descripción del problema

Las bacterias ácido lácticas (BAL) son consideradas como los principales microorganismos responsables del deterioro de productos cárnicos cocidos envasados, debido a la formación de ácido láctico, limo y CO_2 que provocan la aparición de olores y sabores extraños, afectando al grado de aceptación del producto. La BAL *Leuconostoc mesenteroides* ha sido frecuentemente aislada como el microorganismo responsable de la alteración de diversos tipos de productos cárnicos (Björkroth and Korkeala, 1996).

Para evitar las grandes pérdidas económicas que se producen en las industrias cárnicas debido al deterioro de los productos, es muy importante la aplicación de la Microbiología Predictiva en este ámbito y de este modo desarrollar modelos matemáticos que estimen el crecimiento y evolución de los microorganismos alterantes, así como la vida comercial de este tipo de productos cárnicos.

Para la obtención de datos de absorbancia (a partir de los cuales se obtendrán los parámetros de las curvas de crecimiento) se elaboró[25] un Diseño Central Compuesto (DCC), que sirvió para calcular cuantitativamente los efectos de los principales factores que afectan a la estabilidad microbiana en los productos cárnicos. De esta forma se seleccionaron como factores más importantes la temperatura (T °C), pH, concentración de cloruro sódico (NaCl %), concentración de nitrito sódico ($NaNO_2$ ppm) y la condición atmosférica (aeróbica/anaeróbica), para estudiar el efecto combinado de las mismas en el crecimiento de *Leuconostoc mesenteroides* en un medio de cultivo TSB (Medio Triptona Soja).

Los rangos de los factores estudiados fueron escogidos teniendo en cuenta los procesos seguidos por la industria cárnica, y comprendieron los siguientes valores:

- Para la temperatura se han tomado valores de 10.5, 14, 17.5, 21 y 24°C.
- Para el pH se han tomado valores de 5.5, 6, 6.5, 7 y 7.5.
- Para las concentraciones de cloruro sódico los valores han sido 0.25, 1.75, 3.25, 4.75 y 6.25 %.

[25] Los datos experimentales utilizados nos han sido suministrados por el grupo de investigación HIBRO de la Universidad de Córdoba al cual agradecemos su colaboración.

- Por último el nitrito sódico ha variado según 0, 50, 100, 150 y 200 ppm.
- Condiciones de la atmósfera: aerobiosis y anaerobiosis.

Una vez diseñado el experimento se prepararon los cultivos y mediante un espectrofotómetro se realizaron lecturas de absorbancia, ya que el crecimiento del microorganismo se manifiesta con un incremento de la turbidez del medio.

Se obtienen, por tanto, siete curvas de crecimiento (relaciones señal-tiempo) para cada una de las 30 combinaciones de los parámetros, para cada modelo de crecimiento (aeróbico y anaeróbico). Un total de 420 curvas de crecimiento (210 de crecimiento anaerobio y 210 para crecimiento aerobio).

Los valores de densidad óptica obtenidos se transformaron a logaritmo neperiano para homogeneizar la variabilidad. A continuación estos valores de absorbancia resultantes, frente al tiempo, se ajustaron al modelo primario de Baranyi y Roberts (Baranyi and Roberts, 1994) con la ayuda del programa DMFit 1.0, que permite modelar la variación del logaritmo de la concentración celular de cultivos bacterianos a lo largo del tiempo. Como resultado se obtienen los principales parámetros cinéticos de crecimiento (tasa o velocidad de crecimiento, punto de despegue, densidad máxima, etc.) del microorganismo para las condiciones experimentales estudiadas.

El resultado fueron 210 tuplas por cada tipo de crecimiento, 150 fueron usadas para el conjunto de entrenamiento durante la evolución de las poblaciones y las 60 restantes constituyen el conjunto de generalización. Cada tupla consta de cuatro variables de predicción (variables independientes): T, pH, $NaCl$ y $NaNO_2$; y de tres variables de respuesta (variables dependientes): tasa de crecimiento (Growth rate), punto de despegue (Lag) y densidad máxima (Yend). Estos tres valores se muestran en la Figura 4-1 en una típica curva de crecimiento microbiano.

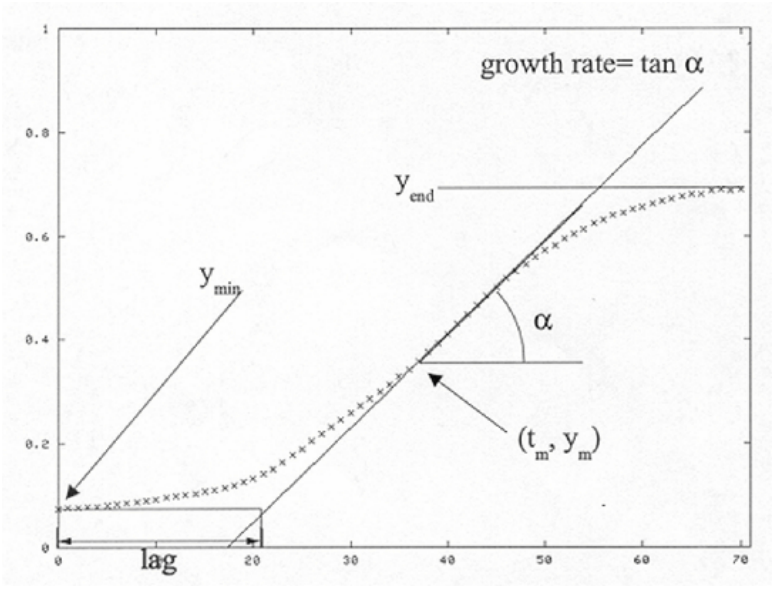


Figura 4-1. Curva de Crecimiento Microbiano

Por lo tanto se platean tres subproblemas, modelar cada una de las tres variables de predicción:

• El problema "Grate" trata de modelar la tasa de crecimiento en función de las cuatro variables de predicción.

• El problema "lnLag" modelará el punto de despegue (procesado mediante un logaritmo neperiano, puesto que se observó experimentalmente que los resultados mejoraban en función de las mismas variables de predicción).

• Y por último, el problema "Yend" intentará obtener un modelo que describa la densidad máxima como respuesta a dichas variables de predicción.

A continuación se mostrará una tabla con cada una de las condiciones medioambientales, para las cuales se estimaron las variables de predicción de las curvas de crecimiento.

Condiciones Medioambientales

Caso	T	pH	NaCl	NaNO2	Caso	T	pH	NaCl%	NaNO2
1	10,5	6,5	3,25	100	16	17,5	6,5	3,25	100
2	14	6	1,75	50	17	17,5	6,5	3,25	100
3	14	6	1,75	150	18	17,5	6,5	3,25	100
4	14	6	4,75	50	19	17,5	6,5	3,25	100
5	14	6	4,75	150	20	17,5	6,5	6,25	100
6	14	7	1,75	50	21	17,5	6,5	0,25	100
7	14	7	1,75	150	22	21	6	1,75	50
8	14	7	4,75	50	23	21	6	1,75	150
9	14	7	4,75	150	24	21	6	4,75	50
10	17,5	5,5	3,25	100	25	21	6	4,75	150
11	17,5	7,5	3,25	100	26	21	7	1,75	50
12	17,5	6,5	3,25	0	27	21	7	1,75	150
13	17,5	6,5	3,25	200	28	21	7	4,75	50
14	17,5	6,5	3,25	100	29	21	7	4,75	150
15	17,5	6,5	3,25	100	30	24,5	6,5	3,25	100

Tabla 4-6 Condiciones medioambientales de los experimentos

4.4.2 Metodología

Para comprobar la eficiencia del algoritmo evolutivo y puesto que se trata de un algoritmo de búsqueda global no determinista se han realizado 30 ejecuciones consecutivas del algoritmo[26] para analizar la eficiencia real del algoritmo y contar con el suficiente número de pruebas para realizar análisis estadísticos.

[26] Las simulaciones consecutivas se realizan para asegurar la total aleatoriedad ya que cada ejecución comienza con una semilla distinta.

Los resultados se han comparado con los resultados obtenidos en (Hervás et al., 2001) tras evolucionar redes neuronales de unidades sigmoides mediante un algoritmo genético. En dicho trabajo, las redes son entrenadas mediante retropropagación con la regla EDBD (Minai and Williams, 1990), y la estructura es evolucionada mediante un algoritmo genético propuesto en (Beasly et al., 1993)

Para la comparación de los modelos obtenidos se utilizará el *SEP* [27].

$$SEP = \frac{100}{|\overline{y}|}\sqrt{\frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{n}}$$

donde y_i es el valor observado, $\hat{y}_i$ es el valor predicho y n es el número de ejemplos del conjunto de entrenamiento. Compararemos tanto los valores del *SEP* para los ejemplos de entrenamiento SEP_E, como el de los ejemplos de generalización SEP_G.

Igualmente, compararemos el número de conexiones de la red resultado. Analizaremos si existen diferencias significativas entre los modelos.

Por último, sobre el mejor modelo de cada variable a predecir, intentaremos obtener algunas reglas que simplifiquen el modelo obtenido, y que sirvan para ayudar a interpretar el modelo.

4.4.3 Parámetros del Algoritmo Evolutivo

Los parámetros utilizados en el algoritmo han sido los mostrados en la Tabla 4-7.

Los valores de las variables de entrada y salida se han escalado en el intervalo [0.1,0.9] en el caso del algoritmo genético con redes neuronales de

[27] Error estándar de predicción. *Standard Error of Prediction.*

unidades sigmoide para evitar el problema de saturación de señal. Para el caso de las redes neuronales de unidades producto se han escalado las variables de entrada en el intervalo [0.1,1.1] ya que cada variable de entrada se mueve en un dominio distinto y los valores de algunas variables como el ($NaNO2$ ppm) son muy altos para una función potencial. Para estos valores de las variables de entrada hemos comprobado que el mejor intervalo de salida es [1,2] por lo que también se ha escalado los valores de las variables a predecir a dicho intervalo.

Se ha limitado el número de nodos de la capa oculta a 4 con el fin de hacer una comparación real con los modelos obtenidos con redes neuronales de unidades sigmoide obtenidos en (Hervás et al., 2001).

Parámetro	Valor
Parámetros de la Población	
Tamaño de la Población	1000
Máximo número de nodos en la capa oculta m	4
Número de variables independientes k	4
Intervalo de los exponentes	[-5,5]
Intervalo de los coeficientes	[-5,5]
Parámetros del Algoritmo Evolutivo	
α	1
r	90
Parámetros de la Mutación Paramétrica	
$\alpha_1(0)$	1
$\alpha_2(0)$	5
β	0.1
ρ	10
Parámetros de las Mutaciones Estructurales: Intervalos $[\Delta_{MIN}, \Delta_{MAX}]$	
AN	[1,2]
EN	[1,3]
AC	[1,16]
EC	[1,16]
Cóndición de Parada	
Nº Máx. de Generaciones	12000
Nº Máx. de Generaciones sin mejorar media 10% ni el mejor	20

Tabla 4-7. Parámetros utilizados en el algoritmo evolutivo para el modelado de crecimiento microbiano

4.4.4 Resultados

A continuación se detallan los resultados obtenidos tras haber realizado 30 ejecuciones de ambos algoritmos. Los resultados obtenidos se refieren a los experimentos en anaerobiosis. En la Tabla 4-8, se muestran los valores del error de SEP_E y SEP_G en media, desviación típica el mejor y peor modelo. Por otra parte, en la Tabla 4-9 se indica el número medio y desviación típica del número de conexiones de los distintos modelos.

Variable	Tipo de Red [28]	Media	Desv, Standard	Mejor	Peor
Ln(lag)	US (4:4:1)	5,49	0,18	5,30	6,22
	UP (4:4:1)	4,96	0,44	4,31	6,00
Growth rate	US (4:4:1)	4,73	0,88	3,86	7,42
	UP (4:4:1)	4,42	0,87	3,39	6,69
Yend	US (4:4:1)	22,00	4,93	17,00	33,00
	UP (4:4:1)	18,20	1,83	15,67	21,78

Tabla 4-8. Valores del error de SEP_E para las tres variables a predecir

Variable	Tipo de Red	Media	Desv, Standard	Mejor	Peor
Ln(lag)	US (4:4:1)	6,35	0,12	6,13	6,69
	UP (4:4:1)	5,82	0,28	5,36	6,69
Grate	US (4:4:1)	4,31	0,84	3,51	6,94
	UP (4:4:1)	4,55	0,90	3,62	6,89
Yend	US (4:4:1)	23,33	1,89	20,97	28,00
	UP (4:4:1)	21,37	1,29	19,22	25,28

Tabla 4-9. Valores del error de SEP_G para las tres variables a predecir

[28] US significa Red Neuronal de Unidades Sigmoides. UP significa Red Neuronal de Unidades Producto. (n¡:n¡:n¡) n¡ nodos de la capa de entrada, n¡ nodos de la capa oculta, n¡ nodos de la capa de salida.

Variable	Tipo de Red	Media	Desv, Standard
Ln(lag)	US (4:4:1)	15,6	4,2
	UP (4:4:1)	19,2	1,0
Growth rate	US (4:4:1)	17,9	4,0
	UP (4:4:1)	18,5	1,3
Yend	US (4:4:1)	16,1	4,7
	UP (4:4:1)	19,8	1,1

Tabla 4-10. Número de Conexiones de las redes obtenidas

En orden a analizar la diferencia en el rendimiento de los dos modelos de redes neuronales (redes MLP o de unidades sigmoides US, versus redes de unidades producto UP) realizaremos tres tipos de contrastes de hipótesis. Todos estos contrastes fueron realizados con un coeficiente de confianza $\alpha = 0.05$; y utilizando el software estadístico SPSS (SPSS, 1999). El primer contrate consiste en analizar si las distribuciones de los errores de SEP son normales, para ello realizamos un contraste de Kolmogorov-Smirnov (Conover, 1971). Los resultados de los contrastes se muestran en la Tabla 4-11.

En segundo lugar contrastaremos si los errores cometidos con ambos modelos de redes neuronales tienen varianzas iguales frente a la hipótesis alternativa de que no lo son; para ello realizamos un test de Levene (Anderson, 1984). Por último, realizamos contrastes t de student que nos permitan contrastar las hipótesis de que las medias de los errores cometidos con ambos modelos son iguales frente a la alternativa de que no los son. Esto es, si existen diferencias significativas en media. Para comentar estos contrastes lo haremos a través de los Intervalos de Confianza del cociente de varianzas y de la diferencia de medias por ser más ilustrativos. Los resultados de los contrastes se muestran en la Tabla 4-12.

test de Kolmogorov-Smirnov							
		Lnlag		Grate		Yend	
	Parámetros	4:4:1	4:4:1	4:4:1	4:4:1	4:4:1	4:4:1
	Estadísticos	US	UP	US	UP	US	UP
SEP_G	Z	0,760	0,886	1,536	1,082	1,349	0,800
	p	0,610	0,412	0,018	0,192	0,052	0,544
n	Z	1,025	1,050	0,682	0,999	0,573	1,998
	p	0,244	0,220	0,742	0,271	0,898	0,001

Tabla 4-11.Contrastes estadísticos y valores críticos p para el error (SEP_G) y el número de conexiones (n) para modelos ANN con unidades sigmoides, US, y producto, UP.

4:4:1 US *vs.* 4:4:1 UP	SEP_G				número de conexiones			
	IC cociente de varianzas		IC diferencia de medias		IC cociente de varianzas		IC diferencia de medias	
	EI	ES	EI	ES	EI	ES	EI	ES
Lnlag	0,086	0,382	0,406	0,637	8,395	37,006	-5,205	-1,995
Grate	0,420	1,854	-0,697	0,217	4,506	19,894	-2,163	0,963
Yend	1,018	4,494	1,105	2,815	8,688	38,361	-5,494	-1.906

Tabla 4-12.Intervalos de confianza para el cociente de varianzas y la diferencias de medias para un coeficiente de confianza del 95%, EI = Extremo Inferior, ES= Extremo Superior

Los resultados de los test de normalidad indican que la mayoría de las distribuciones son normales para valores del nivel de significación α del 5%; esto es, $\alpha < p$

Por otra parte, los intervalos de confianza muestran que las varianzas de los valores del SEP para ambos modelos de redes son diferentes en Lnlag e Yend y son iguales en Grate, dado que el valor unidad está contenido en el intervalo; mientras que los IC para la diferencia de medias muestran que existen diferencias significativas en los valores medios del SEP_G para Lnlag e Yend a favor de los modelos de UP; mientras que no existen diferencias en los valores del Grate, dado que el valor cero se encuentra dentro de este intervalo de confianza.

Si analizamos de forma similar estos test relacionados con el número de conexiones de ambos modelos de redes observamos que en todos los casos las varianzas son significativamente diferentes a favor de los modelos de UP, mientras que para las medias existen diferencias significativas en el número medio de conexiones para Lnlag y para Yend en este caso a favor de los modelos con US; no existiendo diferencias para Grate.

4.4.5 Ejemplo de interpretación de los modelos

Con el objeto de mejorar la interpretabilidad, se ha establecido la restricción de que los exponentes del modelo sean positivos para evitar ratios. A cambio, se ha relajado la restricción de número de nodos en la capa oculta a 5 y no se fuerza a que los modelos obtenidos tengan sesgo. Se realizó igualmente 30 experimentos para cada variable y se obtubieron resultados análogos a los obtenidos con los modelos con 4 nodos en la capa oculta más sesgo y exponentes positivos y negativos.

A partir del mejor modelo obtenido para cada variable vamos a intentar extraer reglas que simplifiquen cada uno de los modelos. La metodología que se va a seguir va a ser la siguiente: Quitar los sumandos de la función potencial cuyo valor absoluto sea menor de 0.01. Según se ha indicado antes, los valores de las variables de salida se han escalado en el rango de salida [1,2], por tanto se eliminan los términos que afecten menos del 1% al modelo.

A partir de una determinada región de las variables de entrada, vamos a extraer reglas y explicar el comportamiento del crecimiento microbiano bajo ciertas condiciones marcadas por la región de estudio.

4.4.5.1 LnLag

El mejor modelo obtenido con el algoritmo evolutivo ha sido una red neuronal de unidades producto con un SEP_G de 5.77 y 18 conexiones.

El modelo obtenido ha sido el siguiente:

$$\begin{aligned}\ln lag^* &= 2.1146(NaCl^*)^{0.193} \\ &+ 1.31187(NaCl^*)^{0.1792}(NaNO_2^*)^{2.098} \\ &- 1.5026(T^*)^{0.750}(pH^*)^{0.081}(NaCl^*)^{0.578} \\ &+ 2.6521(T^*)^{0.796}(NaCl^*)^{2.652}(NaNO_2^*)^{0.578} \\ &- 3.6252(T^*)^{0.394}(pH^*)^{0.280}(NaCl^*)^{2.660}(NaNO_2^*)^{1.463}\end{aligned}$$

Siguiendo la metodología expuesta anteriormente, hemos obtenido dos casos en los que el modelo puede ser simplificado. Además hemos detectado dos términos que son los más importantes dentro del modelo.

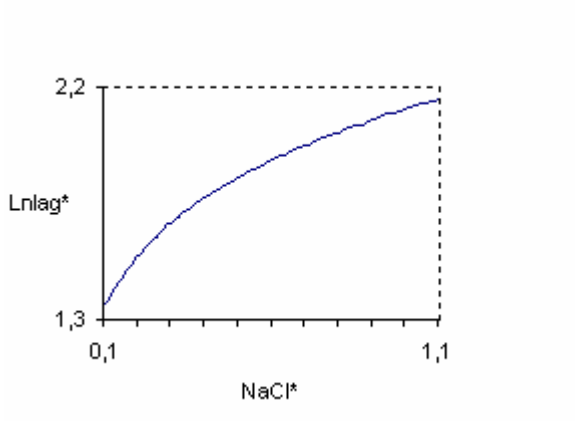
Modelo	Condiciones	Modelo Simplificado
1	$NaCl^* \leq 0.1$	$\ln lag^* = 2.1146(NaCl^*)^{0.193} - 1.5026(T^*)^{0.750}(pH^*)^{0.081}(NaCl^*)^{0.578}$
2	$NaNO_2^* \leq 0.1$	$\ln lag^* = 2.1146(NaCl^*)^{0.193} - 1.5026(T^*)^{0.750}(pH^*)^{0.081}(NaCl^*)^{0.578} + 2.6521(T^*)^{0.796}(NaCl^*)^{2.652}(NaNO_2^*)^{0.578} - 3.6252(T^*)^{0.394}(pH^*)^{0.280}(NaCl^*)^{2.660}(NaNO_2^*)^{1.463}$
Térm. Básicos		$S_1 = 2.1146(NaCl^*)^{0.193}$ $S_3 = -1.5026(T^*)^{0.750}(pH^*)^{0.081}(NaCl^*)^{0.578}$

Tabla 4-13. Modelos simplificados para *lnlag*

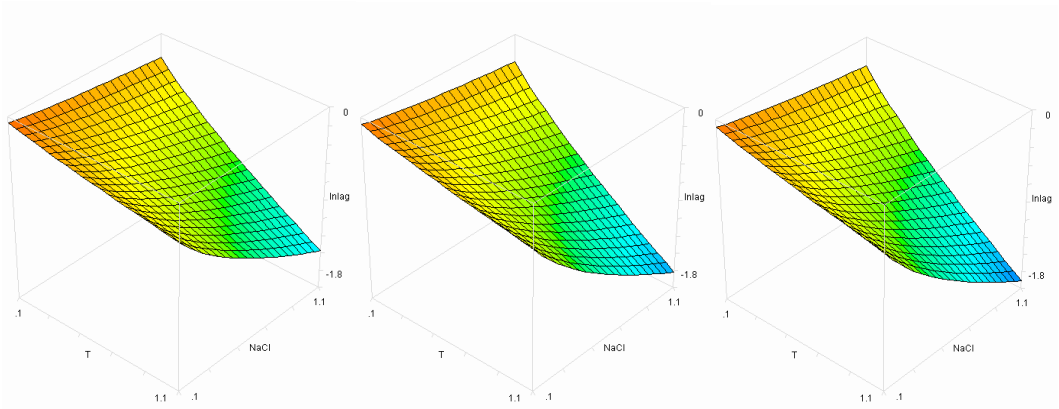
De los modelos obtenidos se pueden extraer las siguientes conclusiones sobre el comportamiento del *lnlag* a partir de los valores de las variables de entrada.

1. El valor del *lnlag* depende fundamentalmente de la concentración de *NaCl*, temperatura y pH. La influencia de la concentración de $NaNO_2$ es menos importante.
2. Si la concentración de *NaCl* es baja, el *lnlag* depende solo de la concentración de *NaCl* y la interacción entre la concentración de *NaCl*, la temperatura y el pH.
3. Si la concentración de $NaNO_2$ el *lnlag* depende de la concentración de *NaCl*, la interacción entre las concentraciones de $NaNO_2$ y *NaCl*, la interacción entre las concentraciones de $NaNO_2$ y *NaCl* y la temperatura, y la interacción de la temperatura, pH y las concentraciones de $NaNO_2$ y *NaCl*.

En la Figura 4-2, se pueden observar gráficamente los términos importantes del modelo para distintos valores del pH.



$S_{\cdot}$



S_3(pH=0.1) S_3 (pH=0.6) S_3 (pH=1.1)

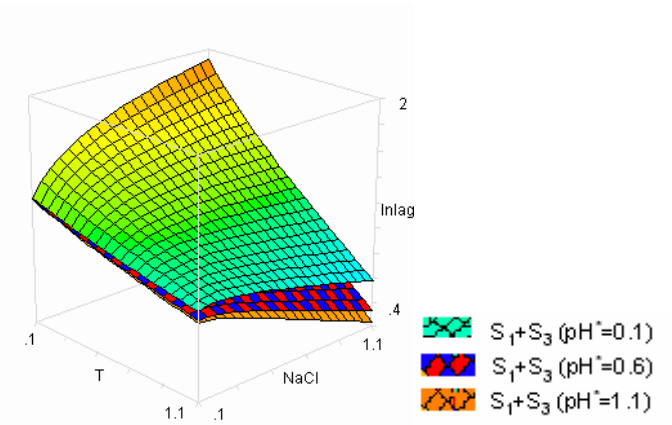


Figura 4-2. Representación gráfica de los términos básicos del modelo Lnlag, separados y conjuntos, para distintos valores del pH.

4.4.5.2 Growth rate

El mejor modelo obtenido con el algoritmo evolutivo ha sido una red neuronal de unidades producto con un SEP_G de 3.06 y 17 conexiones.

El modelo obtenido ha sido el siguiente:

$$\begin{aligned} Grate^* &= 2.9118(T^*)^{1.912} \\ &+ 6.0509(T^*)^{6.548}(NaCl^*)^{5.246} \\ &+ 1.8178(T^*)^{0.167}(pH^*)^{0.127}(NaCl^*)^{0.112} \\ &- 0.2551(T^*)^{0.107}(pH^*)^{1.548}(NaCl^*)^{0.398} \\ &- 4.3718(T^*)^{1.817}(pH^*)^{0.555}(NaNO_2^*)^{0.042} \end{aligned}$$

Siguiendo la metodología expuesta anteriormente, hemos obtenido dos casos en los que el modelo puede ser simplificado. Además hemos detectado un término que es el más importante dentro del modelo.

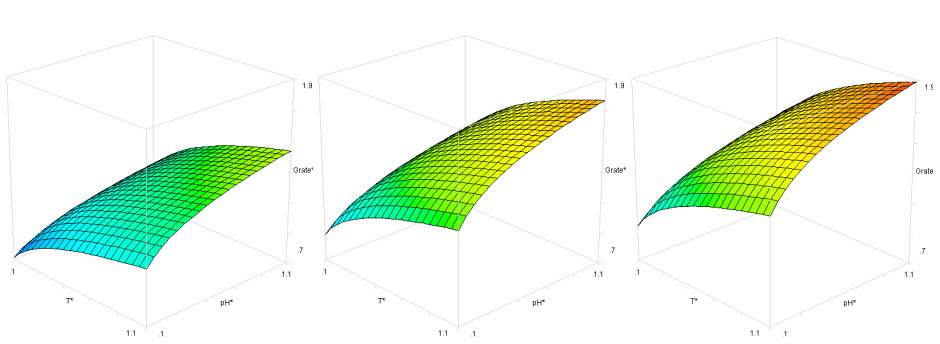
Modelo	Condiciones	Modelo Simplificado
1	$T^* \leq 0.35 \wedge NaCl^* \leq 0.35$	$Grate^* = 2.9118(T^*)^{1.912} + 1.8178(T^*)^{0.167}(pH^*)^{0.127}(NaCl^*)^{0.112} - 0.2551(T^*)^{0.107}(pH^*)^{1.548}(NaCl^*)^{0.398} - 4.3718(T^*)^{1.817}(pH^*)^{0.555}(NaNO_2^*)^{0.042}$
2	$pH^* \leq 0.1$	$Grate^* = 2.9118(T^*)^{1.912} + 6.0509(T^*)^{6.548}(NaCl^*)^{5.246} + 1.8178(T^*)^{0.167}(pH^*)^{0.127}(NaCl^*)^{0.112} - 4.3718(T^*)^{1.817}(pH^*)^{0.555}(NaNO_2^*)^{0.042}$
Térm. Básico		$S_3 = 1.8178(T^*)^{0.167}(pH^*)^{0.127}(NaCl^*)^{0.112}$

Tabla 4-14. Modelos simplificados para *Grate*

De los modelos obtenidos se pueden extraer las siguientes conclusiones sobre el comportamiento del *Grate* a partir de los valores de las variables de entrada.

1. El valor del Growth rate depende básicamente de la interacción entre la temperatura, pH, y concentración de *NaCl*.
2. Si el valor de la temperatura o la concentración de *NaCl* es baja, el Growth rate depende de la temperatura, la interacción entre temperatura, pH y concentración de *NaCl* y entre la temperatura, pH y concentración de $NaNO_2$.
3. Si el pH es bajo, el valor del Growth rate depende de los mismos términos que en el caso anterior y de la interacción entre la temperatura y la concentración de *NaCl*.

En la Tabla 4-14 se puede observar que ha sido señalado el tercer término como el principal del modelo. En la siguiente figura podemos observar dicho término para distintos valores de la concentración de *NaCl.*



S_3 (NaCl*=0.1) S_3 (NaCl*=0.6) S_3 (NaCl*=1.1)

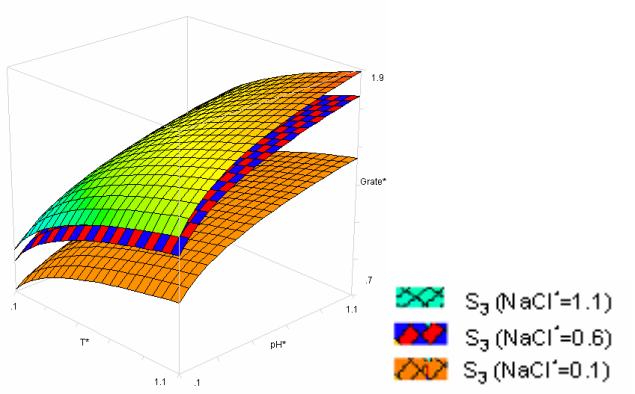


Figura 4-3. Representación gráfica del término básico S_3 del modelo Grate*, para distintos valores del NaCl.

4.4.5.3 Y_{end}

El mejor modelo obtenido con el algoritmo evolutivo ha sido una red neuronal de unidades producto con un SEP_G de 20.21 y 25 conexiones.

El modelo obtenido ha sido el siguiente:

$$
\begin{aligned}
Y^*_{end} = {} & 7.1439(T^*)^{1.191}(pH^*)^{2.166}(NaCl^*)^{0.891}(NaNO_2^*)^{3.645} \\
& - 0.8340(T^*)^{0.074}(pH^*)^{0.188}(NaCl^*)^{1.779} \\
& + 2.0445(NaCl^*)^{0.033} \\
& - 2.2889(T^*)^{0.231}(pH^*)^{0.514}(NaCl^*)^{0.540}(NaNO_2^*)^{2.999} \\
& + 2.8170(T^*)^{1.282}(pH^*)^{2.186}(NaCl^*)^{1.309} \\
& - 11.3193(T^*)^{3.890}(pH^*)^{4.8216}(NaCl^*)^{1.266}(NaNO_2^*)^{0.903}
\end{aligned}
$$

Siguiendo la metodología expuesta anteriormente, hemos obtenido dos casos en los que el modelo puede ser simplificado. Además hemos detectado dos términos que son los más importantes dentro del modelo.

Modelo	Condiciones	Modelo Simplificado
1	$pH^* \leq 0.1$	$Y^*_{end} = -0.8340(T^*)^{0.074}(pH^*)^{0.188}(NaCl^*)^{1.779} + 2.0445(NaCl^*)^{0.033} - 2.2889(T^*)^{0.231}(pH^*)^{0.514}(NaCl^*)^{0.540}(NaNO_2^*)^{2.999}$
2	$NaNO_2^* \leq 0.1$	$Y^*_{end} = -0.8340(T^*)^{0.074}(pH^*)^{0.188}(NaCl^*)^{1.779} + 2.0445(NaCl^*)^{0.033} + 2.8170(T^*)^{1.282}(pH^*)^{2.186}(NaCl^*)^{1.309}$
Términos Básicos		$S_2 = -0.8340(T^*)^{0.074}(pH^*)^{0.188}(NaCl^*)^{1.779}$ $S_3 = 2.0445(NaCl^*)^{0.033}$

Tabla 4-15.Modelos simplificados para Y_{end}

De los modelos obtenidos se pueden extraer las siguientes conclusiones sobre el comportamiento del Y_{end} a partir de los valores de las variables de entrada.

1. El valor de Y_{end} depende fundamentalmente de la concentración de $NaCl$ y de la interacción entre la temperatura y el pH. La

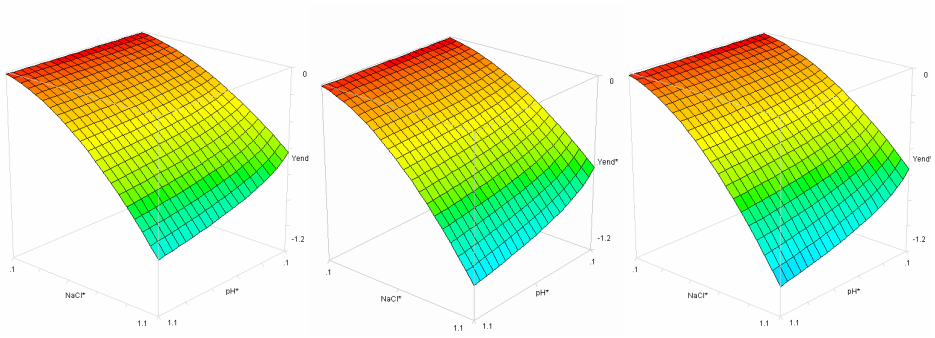
concentración de $NaNO_2$ también tiene influencia sobre el valor de y_{end} pero en mucho menor grado.

2. Si el valor de pH es bajo, Y_{end} depende de la concentración de $NaCl$, la interacción de la concentración de $NaCl$, temperatura y pH, y la interacción de la concentración de $NaCl$, temperatura, pH y concentración de $NaNO_2$.

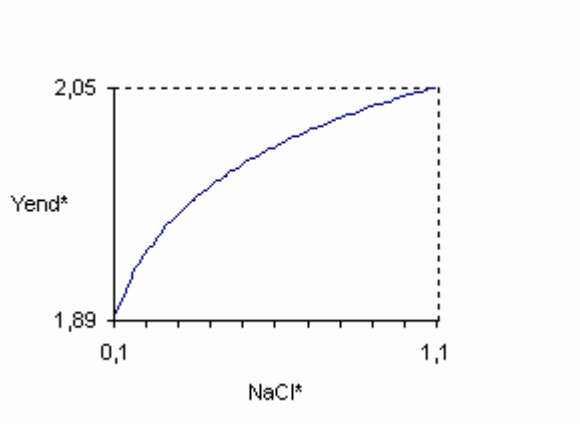
3. Si la concentración de $NaNO_2$ es baja, Y_{end} depende de la concentración de $NaCl$ y la interacción de la concentración de $NaCl$, temperatura y pH.

4. Si la temperatura o la concentración de $NaCl$ son bajas, o si la concentración de $NaCl$ es baja y el valor de pH es moderado, el valor de Y_{end} depende de la concentración de $NaCl$ y la interacción de la concentración de $NaCl$, temperatura y pH además de la interacción entre las cuatro variables de entrada.

En la figura, se pueden observar gráficamente los términos importantes del modelo para distintos valores de la temperatura.



S_2 (T*=0.1) S_2 (T*=0.6) S_2 (T*=1.1)



S_3

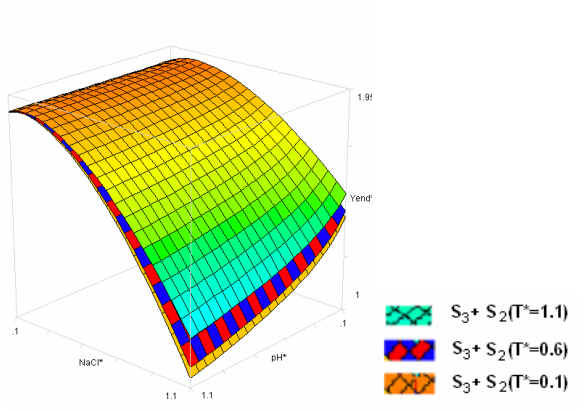


Figura 4-4. Representación gráfica de los términos básicos del modelo Yend, separados y conjuntos, para distintos valores de la temperatura

5 Algoritmo de Optimización Híbrido

5.1 Introducción

Los algoritmos evolutivos (AEs) son suficientemente eficientes explorando el espacio de búsqueda en un problema de optimización, sin embargo, está probada su ineficiencia para encontrar el óptimo local dentro de la región del espacio en la que el algoritmo converge. Numerosos investigadores (Houck et al., 1996; Houck et al., 1997; Joines et al., 2000; Michalewicz and Schoenauer, 1996), han demostrado que los algoritmos evolutivos son una herramienta adecuada para la búsqueda global, ya que son capaces de hallar rápidamente regiones del espacio de búsqueda en las que se encuentran los óptimos locales del problema o bien la solución global, mientras que la convergencia hacia el óptimo correspondiente es con frecuencia muy lenta. Algunos estudios realizados sobre el proceso de convergencia de un algoritmo genético en un problema concreto de optimización

muestran como el algoritmo genético encuentra rápidamente buenas soluciones del problema, sin embargo, necesita muchas generaciones para alcanzar con precisión la solución óptima. Concretamente, sobre el problema de división de celdas (100×40 manufacturing cell formation) la aproximación en media al óptimo de un algoritmo genético es del 90% con sólo el 65% del tiempo total de computación, dedicando el 35% de tiempo de computación restante a alcanzar el óptimo con exactitud (Joines and Kay, 2002).

Por otra parte, es conocido que determinados procedimientos, conocidos como procedimientos de mejora local (local procedures, LIPs), son capaces de encontrar con precisión el óptimo local cuando la búsqueda se realiza en una pequeña región del espacio. En los últimos diez años se han desarrollado este tipo de métodos para mejorar la falta de precisión de los algoritmos evolutivos. De esta forma, procedimientos del tipo "ascensión de colinas" (métodos cuasi-Newtonianos, gradiente conjugado, SQP) permiten refinar eficientemente la solución local alcanzada por el algoritmo evolutivo.

Las nuevas metodologías basadas en la combinación de los algoritmos evolutivos como buscadores globales y los procedimientos de búsqueda local (BL) se denominan comúnmente algoritmos híbridos. Numerosas investigaciones utilizan los algoritmos híbridos para realizar esta combinación entre la búsqueda global y local permitiendo así mejorar la precisión de los algoritmos evolutivos clásicos. La combinación de las estrategias de búsqueda global con los procedimientos de búsqueda o mejora local es la clave para encontrar de manera eficiente soluciones de calidad. Es necesario, por tanto, encontrar un equilibrio entre los procesos de exploración del espacio de soluciones y la explotación de las regiones más prometedoras que establezca un compromiso entre las búsquedas globales y locales.

Desde un enfoque más amplio, los algoritmos que incorporan procedimientos de mejora local pueden considerarse pertenecientes a la clase de

metaheurísticas denominadas algoritmos meméticos (MAs). Los algoritmos meméticos (Moscato, 1989; Moscato, 1999) son algoritmos evolutivos que incluyen dentro de la estrategia de búsqueda un proceso de optimización o aprendizaje de cada individuo. La diferencia de un algoritmo memético con respecto a un algoritmo genético estándar reside en el uso de la búsqueda local para mejorar los individuos creados. Mientras que los algoritmos genéticos están inspirados en el concepto de evolución biológica, los algoritmos meméticos se basan en el concepto de evolución cultural. Los algoritmos meméticos[29] están inspirados por modelos de adaptación de sistemas naturales en los que se combinan la adaptación evolutiva de poblaciones de individuos con el aprendizaje individual de cada individuo dentro de su periodo de vida.

Los algoritmos meméticos han sido usados con éxito en la resolución de problemas de optimización en diferentes áreas, especialmente en la resolución de problemas de optimización combinatoria pertenecientes a la clase NP, en donde los clásicos algoritmos genéticos han resultado ineficientes (Moscato, 1992).

Un estudio detallado de los algoritmos meméticos puede verse en (Krasnogor, 2002; Moscato, 1999).

Admitida la necesidad de realizar la combinación de búsquedas globales y locales dentro del proceso de optimización y teniendo en cuenta el coste computacional que tienen los procesos de búsqueda local, se plantean numerosos interrogantes (Hart, 1994) sobre la forma concreta en la que se debe realizar esta hibridación para que resulte más eficiente:

[29] La denominación de memético surge del término inglés "meme", acuñado por R. Dawkins como análogo del concepto de gen en el contexto de la evolución cultural Dawkinss, R., 1976. The Selfish Gene. Oxford University Press, Oxford, UK.. Según este paralelismo, mientras los genes no son modificados durante la vida del individuo en la evolución biológica, los "memes" sí lo son.

- ¿Con qué frecuencia debe realizarse la búsqueda local?
- ¿En qué momento del algoritmo evolutivo es más adecuado realizar la búsqueda local?
- ¿Sobre qué soluciones debe aplicarse la búsqueda local?
- ¿Cuánto debe durar el proceso de mejora local de una solución?
- ¿Cuál debe de ser la eficiencia de la búsqueda local de manera que exista un compromiso entre la eficiencia y la eficacia del algoritmo híbrido?

A partir de las cuestiones anteriores parece lógico que existan multitud de formas de incorporar un procedimiento de mejora local en un algoritmo evolutivo. A continuación señalamos algunos de los enfoques más frecuentes en la práctica:

1. El algoritmo evolutivo se desarrolla sin realizar la búsqueda local, aplicándose ésta sólamente sobre el individuo obtenido como solución final del algoritmo evolutivo.
2. El procedimiento de búsqueda local interviene en la función de evaluación del algoritmo evolutivo. La búsqueda local evalúa cada individuo de la población determinando el mejor valor objetivo a partir de la localización considerada.
3. Aplicando el proceso de búsqueda local como un operador genético de forma similar a cualquier operador de mutación.

Existen diferentes tipos de metaheurísticas, no siempre incluidas bajo la denominación de algoritmos meméticos[30], que combinan la búsqueda global con la

[30] Con frecuencia este tipo de metaheurísticas no se presentan en la literatura como algoritmos meméticos sino con otras denominaciones como Algoritmos Genéticos Híbridos, Buscadores Genéticos Locales, Algoritmos Genéticos Lamarkianos, Algoritmos Genéticos Balwinianos, etc.

local. A continuación, señalamos de forma breve las técnicas más importantes (Joines and Kay, 2002; Kollen and Pesch, 1994; Ulder et al., 1991).

- **Técnica multiarranque**[31]

Consiste en una metaheurística muy simple que se inicia generando aleatoriamente diferentes puntos del espacio de soluciones que sirven como puntos iniciales para realizar una búsqueda local, tras la cual se guarda la mejor solución obtenida. Así el proceso alterna una fase de generación de soluciones con otra de mejora de las mismas. El proceso se repite hasta que se cumpla la condición de parada. Esta técnica ha sido usada con frecuencia en la resolución de problemas de optimización no lineales y en problemas combinatorios, aunque resulta poco eficiente en problemas complejos al ser una búsqueda "a ciegas" en la que no se tiene en cuenta la información obtenida con anterioridad en el algoritmo (Martí and Moreno, 2003).

- **Efecto Baldwin y Evolución Lamarkiana**

Cuando se incorporan los resultados de la búsqueda local a la función de evaluación aparecen los conceptos de Efecto Baldwin y de Evolución Lamarkiana. El Efecto Baldwin permite cambiar el fitness de un individuo (phenotipo) a partir del aprendizaje, es decir, de la mejora local realizada (Hinton and Nolan, 1987). Mientras que la evolución Lamarckiana, además de usar el aprendizaje para determinar la aptitud de un individuo, cambia su estructura genética para reflejar el resultado del aprendizaje (Turney, 1996; Whitley et al., 1994).

Estas dos metaheurísticas no aparecen siempre de forma separada. Con frecuencia se realiza una mezcla de ellas dando lugar al denominado Lamarkianismo Parcial, en donde a un determinado porcentaje de individuos (por

[31] Multistart technique

ejemplo el 50%) se les aplica un cambio en el genotipo y al resto en el fenotipo (Houck et al., 1997; Joines et al., 2000).

- **Enlace Aleatorio (RL)**[32]

Este método es una estrategia de búsqueda para la resolución de problemas de optimización global (Locatelli and Schoen, 1999). El algoritmo funciona generando aleatoriamente, con una distribución uniforme, diferentes puntos del espacio de búsqueda. Aplicando un criterio de aceptación o rechazo para cada uno de los puntos obtenidos, se realiza o no el proceso de búsqueda local. Esta metaheurística, al partir de localizaciones obtenidas aleatoriamente, no asegura que el algoritmo sea eficiente para explorar globalmente el espacio de soluciones, y por tanto, para determinar el óptimo global. Si bien, por otra parte, desde un punto de vista teórico el algoritmo de enlace aleatorio posee interesantes propiedades de convergencia

- **Algoritmos evolutivos con enlace aleatorio (EARL)**[33]

La combinación de un algoritmo evolutivo con el procedimiento de enlace aleatorio tiene como objetivo evitar la repetición de búsquedas locales y permitir que se realice una exploración global del espacio de soluciones (Houck et al., 1997). En este algoritmo, los diferentes puntos del espacio de búsqueda no pertenecen a una distribución uniforme, sino que son determinados por el algoritmo evolutivo y, por tanto, pueden pertenecer a cualquier distribución. Por este motivo, no se conservan las propiedades teóricas de convergencia del algoritmo enlace aleatorio. Sin embargo, los estudios experimentales realizados muestran que no sólo encuentra mejores soluciones que un algoritmo evolutivo

[32] Random linkage

[33] Evolutionary algorithms with random linkage

clásico, sino que realiza la búsqueda de manera más eficiente (Bersini and Renders, 1994; Merz, 2001; Moscato, 1989; Moscato, 1999).

Un estudio más detallado de estos métodos híbridos y un estudio comparativo puede encontrarse en (Joines and Kay, 2002).

- **Algoritmos meméticos con técnicas de ascensión de colinas basadas en el cruce**

En los últimos años han merecido especial atención el desarrollo de metaheurísticas híbridas en el contexto de los algoritmos genéticos con codificación real. Para obtener soluciones en problemas de optimización con un alto valor de precisión, han ido apareciendo diversos operadores de cruce que permiten un refinamiento local de las soluciones (Herrera et al., 1998).

Entre estas metaheurísticas, merece destacar por su importancia la desarrollada en (Lozano et al., 2004) con la idea básica de utilizar un algoritmo de ascensión de colinas como mecanismo de búsqueda, y el cruce como operador de movimiento para afinar la precisión de los individuos obtenidos por los operadores genéticos. El algoritmo propuesto ajusta la búsqueda global y local según el problema considerado, llevando a cabo un equilibrio entre la exploración del espacio de búsqueda y la explotación de las regiones más prometedoras.

- **Procedimiento de búsqueda mediante gradiente evolutivo**[34]

Este método desarrollado en (Salomon, 1998) responde a un tipo de hibridación diferente a las consideradas con anterioridad. Concretamente, se trata de usar una estructura evolutiva para realizar una estimación del gradiente y posteriormente aplicar el procedimiento del gradiente descendente, junto con la autoadaptación del tamaño de paso. De esta forma, el procedimiento no es en

[34] Evolutionary gradient search procedure

sentido estricto ni un algoritmo evolutivo ni un método del tipo graciente descendente. El autor prueba que puede ser usado con garantía en la resolución de problemas de optimización continua.

5.2 Algoritmos híbridos

La propuesta de algoritmo evolutivo híbrido que presentamos aquí difiere significativamente de las presentadas con anterioridad. En primer lugar, hay que señalar que la estructura del algoritmo está enfocada a la resolución de problemas de regresión mediante el diseño y el aprendizaje de redes neuronales basadas en unidades producto. Sin embargo, la estructura básica del algoritmo propuesto puede ser utilizada sin demasiados cambios en otros contextos, como en problemas de clasificación o en problemas de regresión con redes neuronales de otro tipo. Antes de pasar al siguiente epígrafe en el que se realizará la descripción detallada del algoritmo, veamos de forma breve cuáles son las características fundamentales de la propuesta híbrida que realizamos:

- Presentamos dos versiones del algoritmo evolutivo híbrido. En la primera, los procedimientos de búsqueda local se realizan cuando el algoritmo evolutivo finaliza, mientras que en la segunda versión, denominada versión dinámica, los procedimientos de búsqueda local se realizan de forma periódica, cada determinado número de generaciones, durante el proceso evolutivo.
- El tamaño de la población de individuos, en especial en un problema de modelado, hace que sea ineficiente la aplicación del proceso de búsqueda local a cada individuo de la población, como puede ocurrir en un algoritmo memético estándar. Por este motivo, sólo se realizará el proceso de búsqueda local sobre un reducido número de individuos pertenecientes a un porcentaje de los mejores individuos de la población.
- Con el fin de reducir el tiempo de computación dedicado a la búsqueda local, la selección de los individuos a los que se les aplica dicha búsqueda, se realiza

a partir de un proceso de agrupación que agrupa previamente un porcentaje de los mejores individuos en diferentes clases y elige posteriormente el individuo de cada clase con mejor aptitud. Este proceso de agrupamiento permite construir diversos nichos en la subpoblación de los mejores individuos donde llevar a cabo posteriormente la búsqueda local. Es éste un conjunto de ideas absolutamente novedosas en el sentido de dar protagonismo a la subpoblación de los mejores individuos no solo en la generación final sino de forma dinámica en diferentes instantes de la evolución. La efectividad de estos procedimientos viene avalada como veremos por los resultados experimentales, tanto en el algoritmo de hibridación aplicado en la generación final como en el algoritmo dinámico de hibridación.

- El algoritmo de búsqueda local aplicado es el algoritmo de Levenberg-Marquardt (L-M). Este algoritmo está indicado especialmente para resolver problemas de regresión en los que se considera la suma de residuos al cuadrado como función de error (Bishop, 1995; Levenberg, 1944; Marquardt, 1963). Esta clase de algoritmos de tipo gradiente tiene la ventaja de que son fáciles de implementar y que con su utilización se obtienen modelos muy buenos en entrenamiento, aunque a veces esta bondad en entrenamiento reduzca su capacidad de generalización.
- Dada la tipología de los modelos de redes neuronales de unidades producto propuestos es bastante factible que el algoritmo quede atrapado en un óptimo local. Recordemos que pequeños cambios en los coeficientes del modelo, sobre todo los asociados a los exponentes de las variables independientes, producen grandes diferencias en la función de aptitud, función que por otra parte es altamente multimodal puesto que diferentes modelos tienen aptitudes muy similares; y además epistática, puesto que tratamos de reconocer las relaciones causa efecto entre diferentes variables independientes con una alta interacción entre ellas, y una variable dependiente, función no lineal de las anteriores. Es

por ello que nos hemos decantado por realizar, en cada generación, la hibridación sobre los mejores individuos obtenidos por el algoritmo evolutivo una vez que el algoritmo ha evolucionado y que los modelos obtenidos mediante esta hibridación sean considerados como soluciones locales (o globales) y no participen en las generaciones siguientes del algoritmo. Es importante señalar que partimos de la hipótesis de que mutar individuos óptimos locales, (recordemos que en nuestro algoritmo no definimos un operador de cruce), tan sólo puede producir una progenie de individuos peores en media a los de la generación de los padres.

5.2.1 Descripción del algoritmo híbrido

En este epígrafe describiremos el algoritmo híbrido, expresado con diferentes variantes, que proponemos para la resolución de problemas regresión usando redes de neuronales basadas en unidades producto. El algoritmo no responde al esquema clásico de algoritmos híbridos descritos con anterioridad, pues no corresponde a ningún tipo de algoritmo memético ni responde a los esquemas de Lamarkianismo ni Baldwinnismo. El algoritmo evolutivo que proponemos presenta en sus diferentes variantes dos elementos básicos: por una parte, un proceso de agrupamiento que permite aglutina los individuos y por otra un proceso de búsqueda local realizado con el método de Levenberg-Marquardt.

Antes de realizar la descripción del algoritmo híbrido presentamos la descripciones del proceso de agrupación y del método de búsqueda local utilizado.

5.2.2 Proceso de agrupación

En este epígrafe presentamos el proceso de agrupación que utilizaremos posteriormente en la construcción del algoritmo híbrido. Generalmente, la técnica de agrupación se aplica en el espacio euclídeo usual donde los elementos son vectores y la distancia es la distancia euclídea. En el problema que pretendemos

resolver, se necesita realizar el proceso de agrupación dentro del espacio F de las funciones, lo que obliga a comenzar definiendo una distancia en dicho espacio.

Sea $D = \{(\mathbf{x}_l, y_l) : l = 1,2,..,n\}$ el conjunto de datos de entrenamiento, donde el número de patrones es n. Comenzamos definiendo la siguiente aplicación de la familia de funciones F en el espacio euclídeo $\mathbb{R}^n$:

$$\begin{aligned} &H : F \to \mathbb{R}^n \\ &f(\mathbf{x},\boldsymbol{\theta}) \to H(f(\mathbf{x},\boldsymbol{\theta})) = \hat{y}_f = (f(\mathbf{x}_l,\boldsymbol{\theta}))_{l=1,2,...,n} \end{aligned} \tag{5.1}$$

Concretamente, la aplicación asigna a cada función de la familia el vector del espacio euclídeo construido a partir de los valores de la función sobre el conjunto de datos de entrenamiento asociados al problema de regresión. Esta asignación permite definir una distancia en el espacio de las funciones de la familia F. La distancia entre dos funciones se calcula como la distancia euclídea entre los correspondientes vectores asociados:

$$d(f,g) = \|\hat{y}_f - \hat{y}_g\| = \left[\sum_{l=1}^{n} |f(\mathbf{x}_l,\boldsymbol{\theta}) - g(\mathbf{x}_l,\boldsymbol{\theta})|^2 \right]^{1/2} \tag{5.2}$$

Si consideramos el espacio de las funciones F restringidas al conjunto de datos de entrenamiento D, $F|_D$, es fácil comprobar que d es una distancia sobre $F|_D$, ya que se cumplen las propiedades de no negatividad, simetría y la desigualdad triangular. Realmente, la proximidad entre dos funciones está determinada por el comportamiento de las funciones frente al problema de regresión considerado. Así, según la distancia definida, dos funciones están próximas si y sólo si tienen valores similares sobre el conjunto de datos de entrenamiento asociado al problema de regresión.

Tras definir la distancia en el espacio de funciones, pasamos a describir el proceso de agrupación.

Consideremos el conjunto de funciones $\{f_1, f_2, ..., f_M\}$ de la familia F y el correspondiente conjunto de vectores $\{\hat{y}_{f_1}, \hat{y}_{f_2}, ..., \hat{y}_{f_M}\}$ de $\mathbb{R}^n$ asociados por la

aplicación H. El problema de agrupación consiste en determinar una partición $P = \{C_1, C_2, ..., C_K\}$ del conjunto de vectores $\{\hat{y}_{f_1}, \hat{y}_{f_2}, ..., \hat{y}_{f_M}\}$ en K subconjuntos disjuntos (clases) tal que la suma de distancias al cuadrado de cada punto de la clase a un punto concreto de la misma, denominado centroide, sea mínima. En general este punto es la media o la mediana de la clase.

De forma esquemática, el algoritmo se lleva a cabo de la siguiente forma:

1. Asignación inicial aleatoria de los elementos de la poblacion en K clases o subconjuntos.
2. Se calcula el centroide de cada clase.
3. Se asigna cada elemento de la población a la clase que esté más próxima.
4. Se vuelven a calcular los centroides de cada clase y se repite el proceso de asignación

Figura 5-1. Pseudocódigo del algoritmo K-medias

Hemos usado el algoritmo K-medias (Fukunaga, 1990) en el que el centroide de cada clase se calcula como la media aritmética de los elementos que pertenecen a la clase. Existen diversos algoritmos de agrupación, con una estructura de funcionamiento similar a la descrita en el algoritmo de K-medias, y en los que la diferencia consiste en la elección del estadístico de localización de cada clase. Así, en el algoritmo K-medianas, el centroide es la mediana de la clase y en el algoritmo k-medioides, el centroide es el medioide de la clase.

Formalmente, el problema puede expresarse de la siguiente forma. Se trata de minimizar la expresión:

$$\min_{P \in P_K} \sum_{j=1}^{K} \sum_{\hat{y}_l \in C_j} \left\| \hat{y}_l - \overline{y}_j \right\|^2 \tag{5.3}$$

donde $\overline{y}_j = \frac{1}{N_j} \sum_{\hat{y}_j \in C_j} \hat{y}_j$, $N_j = \left| C_j \right|$, es el centroide de la clase C_j y P_K representa el conjunto de todas las particiones del conjunto de vectores

$\{\hat{y}_{f_1}, \hat{y}_{f_2}, ..., \hat{y}_{f_M}\}$ en K subconjuntos. Es importante señalar que el centroide $\overline{y}_j$ no se corresponde necesariamente con ninguna función concreta de la familia F. El centroide en este caso actúa solamente como un elemento del algoritmo.

La clasificación inicial se realiza de forma aleatoria y el número K de particiones debe ser definido previamente. La determinación de este parámetro no es fácil y especialmente cuando se usa dentro de un algoritmo evolutivo en el que las características de la población van cambiando mientras evoluciona el algoritmo. En este contexto, y como una nueva aportación de este trabajo, veremos en la posterior descripción del algoritmo que, dado que al principio de la evolución hay una mayor diversidad de la población, el valor del parámetro K será mayor que al final del proceso de evolución en el que la diversidad de la población es menor. Así, el valor de K decrecerá durante la evolución con valores obtenidos mediante una función monótona decreciente del nº de generaciones; con valores inicial y final de K obtenidos heurísticamente.

El proceso de partición que acabamos de presentar tiene como objetivo fundamental agrupar las funciones de la familia, o una parte de ellas, en diferentes subconjuntos para realizar posteriormente la búsqueda local en el elemento de cada partición que tenga mejor aptitud. Es importante señalar que debido a la definición de distancia realizada en el espacio de las funciones, las funciones que pertenecen a una partición determinada tienen valores similares en el conjunto de datos de entrenamiento.

5.2.3 Algoritmo de Levenberg-Marquardt

En este epígrafe se expone el algoritmo de búsqueda local que vamos a utilizar dentro del algoritmo híbrido, justificando su uso en el contexto de la resolución de un problema de regresión mediante redes neuronales basadas en unidades producto.

Tal y como se detalló en el Capítulo 2, el problema de aprendizaje en redes neuronales se formula en términos de la minimización de una superficie de error. La superficie de error es una función de los parámetros $\boldsymbol{\theta} = (\beta_j, w_{ij})_{i,j}$ de la red y por tanto tiene dimensión $m(k+1)$. Para una red neuronal basada en unidades producto, la función de error que definimos es el error cuadrático medio MSE, función continua y diferenciable, por lo que es posible aplicar los algoritmos basados en el gradiente de la función. Existe una gran variedad de algoritmos que utilizan el gradiente de la función objetivo para la optimización de la correspondiente función. Se sabe que la naturaleza de la superficie de error asociada a una red neuronal tiene numerosos mínimos locales, lo que hace que los métodos basados en el gradiente no sean métodos eficientes para encontrar el óptimo global de la función. Este hecho aparece especialmente en las superficies de error asociadas a redes neuronales basadas en unidades producto (Leerink et al., 1995; Schmitt, 2001). Sin embargo, los métodos basados en el gradiente tienen un buen funcionamiento cuando se trata de realizar una optimización local (Saito and Nakano, 1997a; Saito and Nakano, 1997c).

Los métodos de búsqueda local basados en el gradiente de la función más usados en la resolución de problemas de optimización son: el método del gradiente descendente, el método del gradiente conjugado, los métodos Newtonianos y cuasiNewtonianos y el algoritmo de Levenberg-Marquardt. Para una descripción detallada de estos métodos de optimización, realizada en el contexto del entrenamiento de redes neuronales, puede consultarse (Bishop, 1995).

El método elegido para realizar la búsqueda local dentro del algoritmo híbrido ha sido el algoritmo de Levenberg-Marquardt (Marquardt, 1963). Son varias las razones que nos han llevado a decidirnos por este método. Por una parte, parece un algoritmo más apropiado para el entrenamiento de redes basadas en unidades producto (Oost et al., 2002) que los algoritmos quasi-Newtonianos, por otra, su eficacia demostrada en numerosos problemas de optimización y

especialmente porque está diseñado específicamente para la minimización de la suma de residuos al cuadrado, que es precisamente la función de error que usamos en la resolución del problema de regresión.

A continuación realizamos una descripción esquemática del algoritmo en el contexto del problema de regresión que estamos tratando y de las redes neuronales basadas en unidades producto.

Consideremos el conjunto de datos de entrenamiento dado por $D = \{(y_l, \mathbf{x}_l) : l = 1,2,..,n\}$. El error de estimación asociado a un individuo de la población $f(\mathbf{x}, \boldsymbol{\theta})$ está dado por la suma de residuos al cuadrado de la forma:

$$J(\boldsymbol{\theta}) = \frac{1}{2}\sum_{l=1}^{n}(y_l - f(\mathbf{x}_l, \boldsymbol{\theta}))^2 = \frac{1}{2}\sum_{l=1}^{n}(e_l)^2 = \frac{1}{2}\|\mathbf{y} - \mathbf{f}(\mathbf{x}_l, \boldsymbol{\theta})\|^2 = \frac{1}{2}\|e(\boldsymbol{\theta})\|^2 \quad (5.4)$$

donde e_l es el error asociado al patrón l del conjunto de entrenamiento D.

Sea un punto θ_o de la superficie de error y sea θ_n un punto situado en un entorno de θ_o. Si la distancia entre los dos puntos $\theta_n - \theta_o$ es pequeña, entonces es posible aplicar el desarrollo de Taylor de primer orden de la función de error en un entorno del punto inicial θ_o. De esta forma si linealizamos la función de error en las cercanías del punto de partida, obtenemos que:

$$e(\boldsymbol{\theta}_n) = e(\boldsymbol{\theta}_o) + \mathbf{Z}(\boldsymbol{\theta}_n - \boldsymbol{\theta}_o)$$

donde Z representa la matriz Jacobiana cuyos elementos está dados por las derivadas parciales $(Z_{ij}) = \dfrac{\partial e_i}{\partial \theta_j}$ de las funciones componentes de la función de error respecto de cada parámetro.

A partir de esta aproximación, la función de error (5.4), puede escribirse de forma equivalente como:

$$\frac{1}{2}\|e(\boldsymbol{\theta}_o) + \mathbf{Z}(\boldsymbol{\theta}_n - \boldsymbol{\theta}_o)\|^2$$

Si minimizamos este error con respecto al nuevo valor de los pesos θ_n se obtiene que:

$$\mathbf{\theta}_n = \mathbf{\theta}_o - (\mathbf{Z}^T\mathbf{Z})^{-1}\mathbf{Z}^T e(\mathbf{\theta}_o)$$

En principio la fórmula anterior puede aplicarse de forma iterativa hasta encontrar el mínimo de la función de error. Sin embargo, este planteamiento presenta un problema relacionado con el tamaño de paso realizado en cada iteración. Así, si el tamaño de paso es demasiado grande, puede ocurrir que la aproximación local realizada no sea válida y, por tanto, el método no convergerá al óptimo local de la función de error. Para resolver este problema, se intenta minimizar la función de error a la vez que se minimiza el tamaño de paso, de forma que la aproximación lineal siga siendo válida en cada paso del proceso de optimización. Este objetivo se lleva a cabo realizando una modificación de la función de error, añadiendole un término de regularización que considera el tamaño de paso en cada iteración:

$$\tilde{J}(\mathbf{\theta}) = \frac{1}{2}\|e(\mathbf{\theta}_o) + \mathbf{Z}(\mathbf{\theta}_n - \mathbf{\theta}_o)\|^2 + \lambda\|(\mathbf{\theta}_n - \mathbf{\theta}_o)\|^2 \tag{5.5}$$

donde el parámetro $\lambda > 0$ dirige el tamaño de paso en cada iteración.

Si ahora minimizamos la función de error modificada (5.5) con rspecto a θ_n, se obtiene que:

$$\mathbf{\theta}_n = \mathbf{\theta}_o - (\mathbf{Z}^T\mathbf{Z} + \lambda\mathbf{I})^{-1}\mathbf{Z}^T\mathbf{e}(\mathbf{\theta}_o)$$

donde I representa la matriz identidad. El valor del parámetro λ se ha tomado igual a 0,1 de forma tal que el tamaño de paso sea pequeño. Es importante señalar, como veremos en detalle en el siguiente epígrafe, que este algoritmo de búsqueda local se realiza dentro del algoritmo evolutivo en cada clase, después de llevar a cabo el proceso de partición de la población. El algoritmo se detiene cuando el cambio relativo entre dos iteraciones consecutivas es inferior a una cantidad previamente fijada, o bien se alcanza un número máximo de iteraciones.

La aplicación del algoritmo de L-M se puede realizar de diferentes maneras:

i) Inicializando los valores de los coeficientes del modelo y ejecutando el algoritmo una sola vez con esta condición inicial.

ii) Ejecutando varias veces el algoritmo con diferentes condiciones iniciales y eligiendo el valor más pequeño obtenido (Rivals and Personnaz, 2003). Así se aumenta la probabilidad de obtener un mínimo global de la función de error $\tilde{J}(\boldsymbol{\theta})$.

En nuestro caso, la aplicación del algoritmo de Levenberg-Marquardt hay que considerarla en el contexto del algoritmo evolutivo, por tanto, aplicaremos el método i). Concretamente, el algoritmo evolutivo guía la búsqueda de las localizaciones más prometedoras, estas localizaciones son particionadas mediante el algoritmo K-medias y los valores de los coeficientes de los mejores modelos en cada clase se considerarán como los valores iniciales para aplicar el algoritmo de Levenberg-Marquardt.

5.2.4 Algoritmos híbridos propuestos

5.2.4.1 Introducción

En este trabajo de tesis proponemos tres modelos de algoritmos de programación evolutiva híbrida, basados en métodos de optimización utilizando poblaciones de funciones y en un afinado de las mejores soluciones mediante algoritmos de gradiente y todo ello, además, simplificado utilizando análisis de agrupamiento de las funciones de ajuste para determinar zonas o nichos de similares caracteristicas en cuanto a su bondad de ajuste. Estos métodos los hemos acoplado al diseño de la estructura y a la estimación de los pesos de modelos de redes neuronales basadas en unidades producto y que utilizamos para problemas de regresión. En concreto, esta metodología se basa en la combinacion

de un algoritmo evolutivo (explorador global), un proceso de agrupamiento utilizando un algoritmo K-medias y un procedimiento de mejora de las soluciones a través de un algoritmo de búsqueda local de Levenberg-Marquardt, L-M.

El algoritmo general empieza la búsqueda de la mejor función asociada a un problema de regresión no lineal definiendo una población inicial elegida al azar y, de ella, elegimos de forma determinista un número determinado de mejores funciones. En cada iteracción esta población evoluciona utilizando operadores de selección y de mutación. Esto es, utilizamos operadores de replicación y dos tipos de operadores de mutación: estructural y paramétrica. La mutación estructural implica un cambio de la estructura de la función (cambio en el número de sumandos de la función y/o en el número de parámetros de cada sumando); esto nos permitirá una exploración de diferentes subespacios del espacio de búsqueda. La mutación paramétrica implica la modificación de los coeficientes del modelo utilizando un algoritmo de enfriamiento simulado de tipo adaptativo. El algoritmo de búsqueda local es el de L-M, diseñado especificamente para minimizar la suma de residuos al cuadrado cuando la función de error es alisada. Si además queremos utilizar eficientemente el algoritmo híbrido tenemos que reducir el tiempo de computación utilizado por la búsqueda local, es por ello, por lo que, en este caso, no tiene sentido utilizar un algoritmo memético dado el gran número de elementos con que cuenta la población del algoritmo evolutivo. Por esta razón, utilizamos como alternativa un procedimiento de agrupamiento de los mejores individuos de la población, de tal manera que formemos clases disjuntas de mejores funciones y que apliquemos el algoritmo de L-M sobre el mejor individuo de cada una de estas clases.

Existen diferentes metodologías de aplicación de la hibridación del algoritmo según el momento del algoritmo evolutivo en el que realice el proceso de agrupamiento y la búsqueda local y según el número de individuos a los que se le aplique. Así, aplicaremos el algoritmo de L-M sobre el mejor individuo de la

población, o sobre los mejores individuos situados en las K clases obtenidas por el algoritmo K-medias utilizado. Según la metodología utilizada, la búsqueda local se realizará al final del algoritmo evolutivo o pediódicamente en diversas generaciones del proceso de evolución.

Es importante destacar que el método híbrido que proponemos no corresponde a los esquemas de Lamarkianismo ni de Baldwinismo. Debido a las características específicas de nuestro problema de regresión, pensamos que no tiene sentido introducir en el AE para ser mutados los óptimos locales, obtenidos mediante los procesos de agrupación y búsqueda local mediante L-M. Por otra parte, utilizar un algoritmo memético puro es inviable desde el punto de vista del coste computacional, ya que implicaría la búsqueda local sobre todos los elementos de la población.

5.2.4.2 Descripción de los algoritmos híbridos propuestos

La metodología presentada trata de analizar la evolución de la población de los mejores modelos obtenidos mediante el algoritmo evolutivo y construir localizaciones a lo largo del proceso evolutivo, donde podamos encontrar modelos cuyos parámetros asociados minimicen globalmente la suma de residuos al cuadrado. Para ello, analizamos periódicamente un número determinado de generaciones, donde estudiamos los mejores modelos obtenidos mediante los métodos antes descritos. Así, en cada generación, empezando por la inicial, o generación 0, y terminando por la última o generación End, obtenemos el mejor modelo: Primero utilizando sólo el algoritmo evolutivo, EP, excepto en la primera generación donde la población se genera al azar; en segundo lugar, utilizando además el algoritmo de L-M donde los coeficientes iniciales son los del mejor modelo obtenido con EP; y por último, usando un algoritmo de agrupamiento del tipo K-medias, y considerando en cada clase como coeficientes del algoritmo de L-M los del mejor modelo perteneciente a cada clase. Para poder analizar el comportamiento de cada uno de los algoritmos a lo largo del proceso evolutivo

obtendremos los siguientes estadísticos en alguna de las generaciones: valores medios, varianzas, mejores y peores individuos de cada método a lo largo de la evolución.

En resumen, tendremos diferentes versiones de un algoritmo evolutivo híbrido dependiendo del momento en el proceso de evolución en el que apliquemos la búsqueda local y la partición en clases del espacio de las mejores funciones obtenidas en algunas generaciones; de esta forma denominaremos como sigue a los algoritmos:

- Al algoritmo evolutivo sin búsqueda local y sin procedimiento de agrupación lo denominamos, **(EP)**[35]. Este algoritmo coincide con el presentado en el capítulo 3.
- En el algoritmo evolutivo híbrido **(HEP)**[36] ejecutamos el algoritmo EP sin aplicar la búsqueda local a lo largo de la evolución, aplicándola sobre al mejor individuo obtenido, con este algoritmo, en la última generación. Esta metodología nos permitirá precisar el optimo local o global en un entorno de la solución final encontrada por el algoritmo evolucionario. (Vease la Figura 5-2)

[35] Evolutionary Programming

[36] Hybrid evolutionary programming

1. Generar la Población Inicial B de soluciones
2. Mientras no se cumpla la condición de parada
   a. Seleccionar $B' \subset B$
   b. Aplicar mutación estructural a los individuos de B' para generar B'_{estruc}
   c. Aplicar mutación paramétrica a los mejores s individuos de B' para generar B'_{param}
   d. $B'' = B'_{estruc} \bigcup B'_{param}$
   e. $B = B'' \bigcup \left\{ mejores\ de\ B \right\}$
3. Fin mientras
4. Aplicar L-M sobre el mejor individuo obtenido
5. Devolver el mejor individuo obtenido

Figura 5-2. Pseudocódigo del algoritmo HEP

- En el algoritmo evolutivo híbrido con agrupación (**HEPC**)[37] aplicamos un procedimiento de agrupación sobre un subconjunto suficientemente grande de los mejores individuos de la última generación. En este método es fundamental determinar el número de mejores individuos y el número de clases K a considerar. A continuación, aplicamos el algoritmo de Levenberg-Marquardt al mejor individuo de cada una de las clases. (Veáse la Figura 5-3).

[37] Hybrid clustering evolutionary programming

1. Generar la Población Inicial B de soluciones
2. **Mientras** no se cumpla la condición de parada
   a. Seleccionar $B' \subset B$
   b. Aplicar mutación estructural a los individuos de B' para generar B'_{estruc}
   c. Aplicar mutación paramétrica a los mejores s individuos de B' para generar B'_{param}
   d. $B'' = B'_{estruc} \bigcup B'_{param}$
   e. $B = B'' \bigcup \left\{ mejores\ de\ B \right\}$
3. Fin mientras
4. Aplicar proceso de agrupamiento a parte de B, aplicar L-M sobre el mejor individuo obtenido de cada clase y guardar en C.
5. Devolver el mejor individuo obtenido de C

Figura 5-3. Pseudocodigo del algoritmo HEPC

- Por último, desarrollamos el algoritmo **HEPC dinámico**[38] aplicando el procedimiento de agrupación y la búsqueda local mediante L-M cada G_0 generaciones y en la población final. La solución final vendrá determinada a partir de la utilización del algoritmo HEPC en las diferentes generaciones; y será el mejor individuo obtenido de entre los mejores encontrados en el proceso de evolución, esto es, será el mejor de los mejores individuos de cada una de las generaciones en las que utilizamos HEPC. (Véase la Figura 5-4 y Figura 5-5).

[38] Dynamic hybrid clustering evolutionary programming

1. Generar la Población Inicial B de soluciones
2. Mientras no se cumpla la condición de parada
   a. Seleccionar $B' \subset B$
   b. Aplicar mutación estructural a los individuos de B' para generar B'_{estruc}
   c. Aplicar mutación paramétrica a los mejores s individuos de B' para generar B'_{param}
   d. $B'' = B'_{estruc} \bigcup B'_{param}$
   e. $B = B'' \bigcup \{mejores\ de\ B\}$
   f. Aplicar proceso de agrupamiento y L-M sobre el mejor individuo de cada clase cada G_0 generaciones y guardar en C
3. Fin mientras
4. Aplicar proceso de agrupamiento a parte de B, aplicar L-M sobre el mejor individuo obtenido de cada clase y guardar en C.
5. Devolver el mejor individuo obtenido de C

Figura 5-4. Pseudocódigo del algoritmo HEPC dinámico.

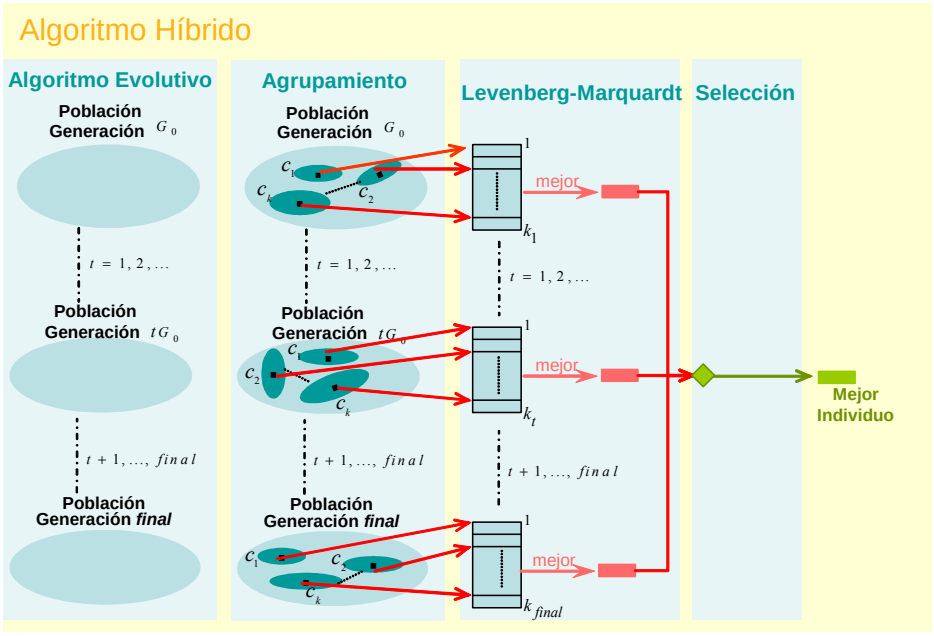


Figura 5-5. Gráfico algoritmo HEPC dinámico.

A continuación se detallan cada uno de los pasos realizados en el algoritmo evolutivo EP.

- **Construcción de la población inicial**

El algoritmo se inicia generando aleatoriamente un número mayor de redes que el usado posteriormente en el proceso de evolución, concretamente generamos $10 * N_R$ redes, siendo N_R el número de redes que tendrá la población durante el proceso de evolución. Por último, la población inicial que constituye la solución base B se forma seleccionando las N_R redes mejores (con mejor aptitud) entre las $10 * N_R$ generadas.

- **Plan de Seleccion**

Seleccionamos el $r\%$ $(r = 90)$ de individuos con mejor aptitud de la población $B^* = B - \left\{ mejor\ individuo\ de\ B \right\}$ de cardinal ${N_R}^* = N_R - 1$ y formamos una población B' de tamaño $r{N_R}^* / 100$.

- **Plan de Generación**

Se aplica mutación estructural a todos los individios de B' obteniendo la población B'_{estruc}. La mutación paramétrica se aplica únicamente a los mejores $s = (100 - r)N_P^* / 100$ individuos de B' obteniendo la población B'_{param}. Se construye la población $B'' = B'_{estruc} \bigcup B'_{param}$ donde el cardinal de B'' es $N_R^* = N_R - 1$.

- **Plan de Reemplazamiento**

Elegimos todos los elementos de B^* para ser reemplazados.

- **Plan de Sustitución**

La nueva población será $B = \{\text{Mejores de } B\} \cup B''$ cuyo cardinal es N_R.

- **Agrupamiento y búsqueda local**

Tenemos tres algoritmos diferentes según el momento en el cual se realiza la búsqueda local y el proceso de agrupamiento.

Caso 1. Se aplica el algoritmo L-M sobre el mejor individuo obtenido a partir del algoritmo evolutivo al final de la evolución. Como ya se ha dicho se trata del algoritmo **(HEP)**.

Caso 2. Se aplica el proceso de agrupamiento sobre los mejores $\tilde{s}\,N_P$ individuos de la población final B, la cual se ha dividido en K clases $C_1, C_2, ..., C_K$. Tras esto, se aplica algoritmo L-M sobre el mejor individuo de cada clase. Los individuos obtenidos mediante la búsqueda local en cada clase se guardan en un conjunto C (Conjunto de óptimos locales). Se trata del algoritmo **(HEPC)**

Caso 3. Se aplica el proceso de agrupamiento y algoritmo L-M sobre la población cada G_0 generaciones y sobre la población final. El proceso de agrupamiento se aplica sobre los mejores $\tilde{s}\,N_P$ individuos de la población en curso. Los individuos obtenidos mediante la búsqueda local en cada clase se guardan en un conjunto C (Conjunto de óptimos locales). En este caso, se trata del algoritmo **(HEPC dinámico).**

En los casos 2 y 3, la solución final es la mejor solución entre las soluciones locales del conjunto C.

5.2.4.3 Función de Aptitud

Sea $D = \{(\mathrm{y}_l, \mathbf{x}_l) : l = 1,2,..,n\}$ el conjunto de datos de entrenamiento. Si definimos la suma de errores cuadráticos obtenidos por un individuo, $f(\mathbf{x}, \boldsymbol{\theta})$, de la población del algoritmo evolutivo, al intentar ajustarse a la nube de puntos determinada por D, en la forma:

$$J(\boldsymbol{\theta}) = \frac{1}{2}\sum_{l=1}^{n}(y_l - f(\mathbf{x}_l, \boldsymbol{\theta}))^2$$

podremos definir una función de aptitud asociada a dicho individuo $A(f(\mathbf{x}, \boldsymbol{\theta}))$ como una función mónotona decreciente de ese error:

$$A(f(\mathbf{x}, \boldsymbol{\theta})) = \frac{1}{1 + J(\boldsymbol{\theta})}$$

Por tanto, esta función de aptitud tomará valores entre 0 y 1, y la utilizaremos además de para poder ordenar los individuos de la población en función de su valor de aptitud, para definir el grado de transformación que puede tener un determinado gen del cromosoma asociado a la representación de una función de la población, mediante un operador de mutación paramétrica.

5.2.4.4 Mutación paramétrica

El operador de mutación paramétrica que vamos a definir consiste en añadir, a un gen determinado de un cromosoma elegido al azar, una variable aleatoria con distribución normal de media cero y donde la varianza va a depender de un parámetro adaptativo y de un parámetro de temperatura. De esta forma, la severidad de la mutación de un individuo $f(\mathbf{x}, \boldsymbol{\theta})$ está determinada por la temperatura, $T(f(\mathbf{x}, \boldsymbol{\theta}))$. La definición de esta temperatura está asociada a un algoritmo de enfriamiento simulado (Kirkpatric et al., 1983; Otten and van Ginneken, 1989), en la forma:

$$T(f(\mathbf{x}, \boldsymbol{\theta})) = 1 - A(f(\mathbf{x}, \boldsymbol{\theta})),\ \ 0 \leq T(f(\mathbf{x}, \boldsymbol{\theta})) < 1$$

La mutación paramétrica puede incidir, por tanto, en cada uno de los coeficientes w_{ji}, β_j del modelo de red o función, asociándoles un ruido gaussiano:

$$w_{ji}(t+1) = w_{ji}(t) + \xi_1(t) \quad \text{y} \quad \beta_j(t+1) = \beta_j(t) + \xi_2(t)$$

donde $\xi_k(t) \in N(0, \alpha_k(t)T(f(\mathbf{x}, \boldsymbol{\theta})))$, $k = 1,2$ representa una variable aleatoria unidimensional con distribución normal de media 0 y varianza $\alpha_k(t)T(f(\mathbf{x}, \boldsymbol{\theta}))$.

Los parámetros $\alpha_k(t)$, $k = 1,2$, permiten la adaptación al proceso de aprendizaje y cambio a lo largo de la evolución:

$$\alpha_k(t+1) = \begin{cases} (1+\beta)\alpha_k(t) & Si \;\; A(g_s) > A(g_{s-1}), \;\; \forall s \in \{t, t-1, ..., t-\rho\} \\ (1-\beta)\alpha_k(t), & Si \;\; A(g_s) = A(g_{s-1}), \;\; \forall s \in \{t, t-1, ..., t-\rho\} \\ \alpha_k(t) & \text{en otro caso} \end{cases}$$

para $k = 1,2$, donde $A(g_s)$ es la aptitud del mejor individuo g_s o modelo de red en la generación s, y ρ es un parámetro definido heurísticamente. En todos nuestros experimentos hemos considerado los siguientes valores para estos parámetros $\alpha_1(0) = 1, \alpha_2(0) = 5$, $\beta = 0{,}1$ y $\rho = 10$. Si observamos estos valores, nos daremos cuenta de que la posible modificación de los exponentes asociados a las funciones potenciales w_{ji} es mucho menor que la modificación de los coeficientes asociados a cada sumando del modelo de red β_j, $\alpha_1(t) \ll \alpha_2(t), \;\; \forall t \in \mathbb{N}$. Esto es así, debido a que pequeños cambios en los valores de los w_{ji}, pueden dar lugar a modelos de red muy diferentes.

5.2.4.5 Mutación estructural

Esta mutación implica una modificación de la estructura de la función o modelo de red en base a una modificación en el número de nodos ocultos o sumandos del modelo y/o del número de potencias (conexiones) implicadas en cada uno de los sumandos; lo que nos permite la exploración de diferentes regiones del espacio de búsqueda y nos permite introducir diversidad en la población de soluciones o modelos. Definiremos cinco tipos diferentes de mutaciones estructurales: Añadir nodo (AN), Eliminar nodo (EN), Añadir conexión (AC), Eliminar conexión (EC) y Unir nodo (UN). Las cuatro primeras son similares al modelos de mutaciones GNARL de Angeline (Angeline et al., 1994). Respecto a la última (UN), se seleccionan dos nodos al azar a y b, y se reemplazan por un nuevo nodo c, el cual es una combinación lineal de ambos.

Las conexiones que son comunes a los dos nodos de partida se combinan, con pesos definidos en la forma:

$$\beta_c = \beta_a + \beta_b\,, \quad w_{ic} = \frac{w_{ia} + w_{ib}}{2}$$

Las conexiones que no estan compartidas por los nodos, las heredará el nodo c con probabilidad 0.5 y su peso asociado no cambiará. Para cada mutación (a excepción de la unión de nodos) definimos un valor mínimo, Δ_{MIN}, y un valor máximo, Δ_{MAX}, y el número de elementos (contando nodos y conexiones) que están involucrados en este tipo de mutación se calcula mediante:

$$\Delta_{MIN} + \left\lfloor uT(f(\mathbf{x},\boldsymbol{\theta}))\,(\Delta_{MAX} - \Delta_{MIN})\right\rfloor$$

donde u es una variable aleatoria definida en el intervalo $[0,1]$. Los cinco tipos de mutaciones se ejecutan secuencialmente, en el orden en el que las hemos definido anteriormente, con una probabilidad $T(f(\mathbf{x},\boldsymbol{\theta}))$, en la misma generación y para el mismo modelo de red. Si al tomar probabilidades no se selecciona ninguna mutación, se elige una mutación al azar y la aplicamos al modelo de red.

5.2.4.6 Parámetros usados en las diferentes metodologías de los algoritmos híbridos

Los parámetros que intervienen en los diferentes algoritmos implementados son:

- Los exponentes w_{ji} han variado en el intervalo $[-5,5]$ y los coeficientes β_j de cada uno de los monomios han variado en el intervalo $[-10,10]$.
- El número máximo de monomios considerados ha sido $m = 8$.
- El tamaño de la población N_P ha sido de 1000 redes.
- El número de generaciones máximo es de 6000 generaciones.

- El número de nodos a añadir en la mutación estructural se ha limitado en el intervalo [1,2], al igual que el número de nodos a eliminar. El número de enlaces a añadir y eliminar se ha limitado según el intervalo [1,6].
- Para las mutaciones paramétricas se han ampliado los intervalos con un coeficiente de amplitud de 5, de manera que tanto los exponentes como coeficientes del modelo lineal serán mutados dentro del intervalo [-25, 25].
- El único parámetro del algoritmo de Levenberg Marquardt fijado es la diferencia de error utilizada para detener el algoritmo o tolerancia cuyo valor es 0,01.
- Los parámetros del análisis K-Medias utilizados para las búsquedas locales son : El algoritmo se aplicará al 25% ($\tilde{s} = 0,25$) de mejores individuos de la población, es decir, a 250 redes. El número de clases en las que se dividirá la población es de 4 clases en la versión estática de la metodología HEPC. Sin embargo en la versión dinámica, hemos considerado que el número de clases debe ser un parámetro que varíe a lo largo de la evolución. De este modo, comenzaremos con un número máximo de 6 clases en la primera generación en que se aplique la búsqueda y este parámetro irá disminuyendo linealmente hasta llegar a un mínimo de 4 clases en la última generación en que se aplica. Estos valores aunque determinados de forma heurística son bastante robustos cuando se modifica ligeramente su valor.
- El criterio de parada del algoritmo se alcanza cuando se cumple una de las siguientes condiciones:
    a. El algorimo alcanza una media de aptitud determinada en una generación
    b. Los valores de $\alpha_1(t)$ y $\alpha_2(t)$ alcanzan un valor prefijado próximo a cero.

c. No existe mejora, durante un determinado nº de generaciones, ni en las aptitudes medias del 20% de los mejores individuos de la población, ni en la aptitud del mejor individuo. Este número de generaciones, de nuevo, se obtiene de forma heurística, pero el valor habitual de nuestros experimentos, 20 generaciones, es también bastante robusto.

5.3 Experimentos realizados

Con el objeto de probar la eficacia del algoritmo propuesto se han seleccionado dos problemas de prueba o "benchmark" y un problema real dentro del ámbito de la microbiología predictiva para la estimación de los parámetros de segundo orden que marcan el crecimiento microbiano. Se realiza una comparación de los algoritmos entre sí y de cada uno de ellos con varios métodos aplicados a la resolución de problemas de regresión mediante redes neuronales (Rivals and Personnaz, 2003). Los resultados muestran unos buenos resultados de predicción y un aceptable grado de interpretabilidad de los modelos. Es interesante señalar que el modelo de red neuronal basada en unidades producto y el algoritmo evolutivo utilizado para resolver el problema de regresión, obtiene mejores resultados cuando el número de variable independientes es elevado ($n \geq 4$).

El diseño experimental se basa en analizar los resultados de los errores obtenidos por los mejores modelos en 30 ejecuciones de los algoritmos utilizados sobre el conjunto de generalización. Los estadísticos de error utilizados son $\mathrm{SEP_G}$, $\mathrm{MSE_G}$ y $\mathrm{APE_G}$ cuyas expresiones están dadas por:

$$SEP_G = 100 \frac{\sqrt{\sum_{i=1}^{N} (y_i - \hat{y}_i)^2}}{\overline{y}} ,$$

$$MSE_G = \frac{\sum_{i=1}^{N}(y_i - \hat{y}_i)^2}{N},$$

$$APE_G = \frac{1}{n}\sum_{i=1}^{n}\left|\frac{\hat{y}_i - y_i}{y_i}\right| \times 100\%$$

siendo y_i el valor de la variable dependiente para los datos del conjunto de generalización e $\hat{y}_i$ el valor obtenido utilizando el mejor modelo obtenido con cada metodología sobre el conjunto de entrenamiento. El estadístico de error utilizado en el problema de crecimiento microbiano es el *SEP*, ya que es adimensional y sirve para poder comparar resultados de error para variables dependientes con diferente escala de medida[39]. Por otra parte, en el caso de la función de Friedman#1, también utilizaremos el valor del error cuadrático medio, *MSE*, para poder hacer comparaciones con otros algoritmos que se presentan en diferentes trabajos de regresión presentes en la literatura. Por último, para la función de Sugeno se utiliza como medida del error el porcentaje medio del error *APE* con el mismo fin que con la anterior función.

Para el caso de la función de Sugeno mostremos escuetamente los resultados obtenidos comparándolos con los resultados de otros métodos que han intentado modelar dicha función. Sin embargo, para el caso de la función de Friedman#1 y el problema del crecimiento microbiano, el análisis de los resultados se hará para todos los algoritmos en todas las generaciones consideradas y de forma más específica en la última generación. Para ello haremos tests previos de normalidad y de independencia de los resultados obtenidos para determinar los contrastes paramétricos y no paramétricos más adecuados para hacer comparaciones de varianzas y de medias o medianas. La información que obtendremos será acerca de:

[39] Esta medida del error es usada con frecuencia en los trabajos de microbiología predictiva.

a) Si existen diferencias en media entre los tres métodos empleados en esas generaciones, EP, HEP and HEPC, en función de las medias de los diferentes estadísticos de error utilizados.

b) Si existen diferencias entre los valores medios de error obtenidos a lo largo de las siete generaciones analizadas (desde la generación inicial 0 hasta la final End). Esto nos indicará si el algoritmo EP evoluciona y permite a su vez la evolución de los métodos HEP y HEPC.

c) Si existen diferencias significativas entre los valores medios de error obtenidos en la última generación por los diferentes algoritmos.

d) Obtener el mejor modelo de previsión de entre los 21 obtenidos en las 7 generaciones analizadas, determinando bajo que metodología y en que generación se ha obtenido.

5.3.1 Función Friedman#1

Este es un problema de prueba o "benchmark" propuesto por (Friedman, 1991) y ampliamente utilizado en la bibliografía de modelado (Carney and Cunningham, 2000; Lee et al., 2004). El conjunto de entrenamiento $D = \{(y_l, \mathbf{x}_l) : l = 1,2,...,200\}$ está formado por 200 patrones, cada uno formado por cinco variables independientes $\mathbf{x} = (x_1, x_2, ..., x_5)$ y una variable dependiente; mientras que el conjunto de generalización esta formado por 1000 patrones con las mismas variables. Los patrones se obtienen a través de la simulación de la función cuya expresión es:

$$f(\mathbf{x}) = 10\text{sen}(\pi x_1 x_2) + 20(x_3 - 0,5)^2 \ + 10x_4 + 5x_5 + \varepsilon$$

donde ε es una variable aleatoria Normal, $N(0,1)$, y las variables de entrada están distribuidas uniformemente en el (0,1].

5.3.1.1 Primer experimento

Con el fin de poder comparar nuestros resultados con los obtenidos en (Lee et al., 2004), se crearon 1200 patrones al azar, y de ellos se extrajeron aleatoriamente 200 para formar el conjunto de entrenamiento y los 1000 restantes para formar el conjunto de test o generalización.

Para comparar resultados hemos utilizado los valores que aparecen en la (Lee et al., 2004) donde se puede ver los resultados de predicción de GRNNFA y de otros métodos (NBAG, Bench y Simple) mostrados en (Carney and Cunningham, 2000).

Métodos	Media	SD
NBAG	4,502	0,268
Bench	5,372	0,646
Simple	4,948	0,589
GRNNFA	4,563	0,195

Tabla 5-1. Valores medios y desviación típica del error MSE_G tras 20 ejecuciones para el problema Friedman#1, obtenidos mediante distintos métodos.

En esta tabla, hemos puesto el estadístico asociado al error sobre el conjunto de generalización, MSE_G, el cual fue obtenido por los autores promediando los resultados obtenidos sobre 20 ejecuciones de sus algoritmos; mientras que en nuestro trabajo realizamos 30 ejecuciones utilizando nuestros algoritmos para reforzar los tests de comparaciones. Los mejores resultados en media y en desviaciones típicas, SD, se han obtenido en (Carney and Cunningham, 2000; Lee et al., 2004) con los algoritmos NBAG (media=4,502, SD=0,268) y GRNNA (media=4,563, SD=0,195). Los mejores resultados, en media, para este ejemplo se han obtenido con el algoritmo HEPC; pero no en la última generación lo que nos indica que podemos parar el entrenamiento de las

redes en torno a la generación 1200, aunque la desviación típica es menor en la generación 2000, o generación final en este caso.

Nº gen.	400		800		1200		2000	
Algoritmo	Media	SD	Media	SD	Media	SD	Media	SD
EP	3,68	0,79	3,13	0,50	2,99	0,53	2,72	*0,50*
HEP	2,50	0,82	1,91	0,63	2,04	1,01	1,80	*0,57*
HEPC	2,31	0,72	1,60	0,46	1,58	0,46	1,61	*0,39*
	Mejor	Nº con	Mejor	nº con	Mejor	nº con	Mejor	nº con
EP	2,59	23	2,19	32	2,08	31	1,86	33
HEP	1,36	27	1,22	24	1,23	31	1,27	34
HEPC	1,29	25	1,22	24	1,23	31	1,25	30

Tabla 5-2. Valores de estadisticos del error MSE_G para Friedman#1 con una arquitectura (5:6:1) en diferentes generaciones y en 30 ejecuciones

Algoritmo	MSE				Nº de conexiones			
	Media	SD	Mejor	Peor	Media	SD	Mejor	Peor
HEPCD	1,40	0,22	1,22	2,36	29,53	3,45	24	37

Tabla 5-3. Valores de estadisticos del error MSE_G y nº de conexiones para Friedman#1 con una arquitectura (5:6:1) para HEPCD en 30 ejecuciones

Para la metodología HEPCD los valores, de la media y de la desviación típica de MSE_G a lo largo de todas las generaciones de la evolución, muestran unos resultados muy precisos y homogéneos (ver fila HEPC de la Tabla 5-2), Además, el peor valor obtenido con esta metodología es inferior al valor medio obtenido con EP en la última generación 2,36 frente a 2,72 (ver Tabla 5-3), lo que da idea de la robustez del método. Por otra parte, el tamaño de la red propuesta como óptima, nº con.=24, se acerca a los valores de los modelos obtenidos en la generación 800 lo que puede indicar un cierto sobreentrenamiento en los mejores modelos obtenidos pasada esta generación.

Por último, para reforzar las conclusiones antes expuestas, vamos a exponer un gráfico donde se pueda observar la evolución que sufre la media y la desviación típica del MSE_G de los mejores modelos a lo largo de las generaciones.

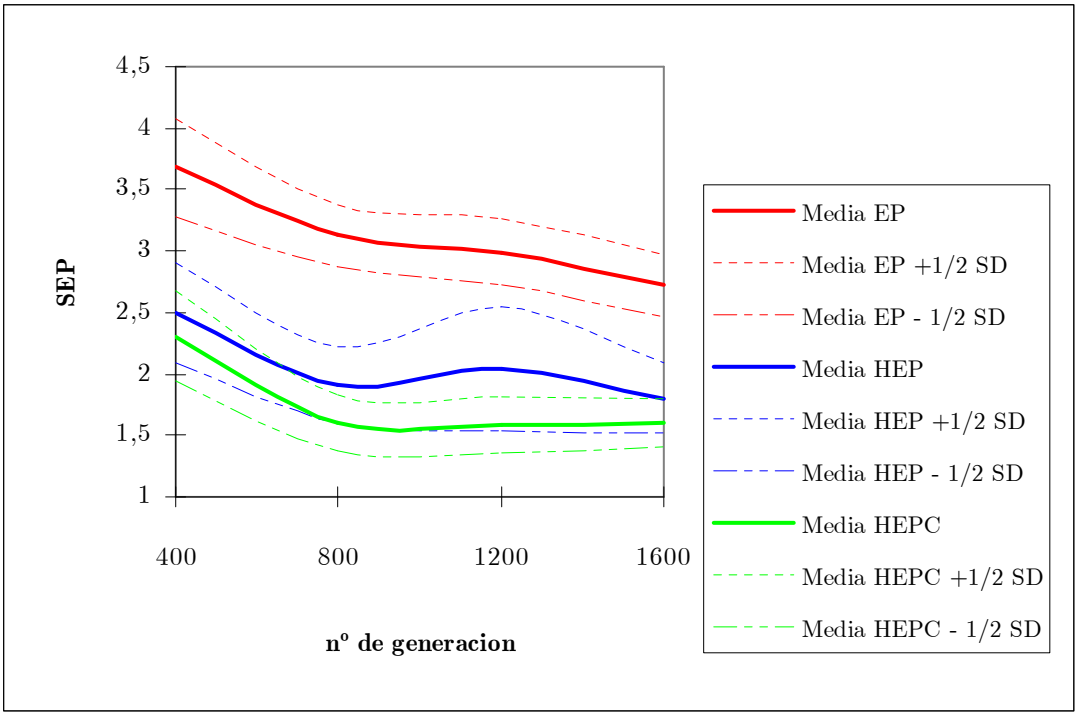


Figura 5-6. Evolución de la media y la desviación típica (SD) del mejor modelo para las metodologías EP, HEP y HEPC en el problema Friedman#1 para 30 ejecuciones.

El mejor modelo se obtiene en la generación 800 con las metodologías HEP y HEPC (MSE_G= 1,22, con 24 conexiones y 6 sumandos) y es de la forma:

$$y = 0,001 + 0,823\frac{x_2^{0,003}x_4^{0,002}}{x_1^{0,012}x_3^{0,132}} + 0,360x_4^{1,055}x_5^{0,015}$$
$$+ 0,167\frac{x_3^{0,099}x_5^{1,191}}{x_1^{0,055}x_2^{0,017}} + 3,560x_1^{1,575}x_2^{1,378}x_4^{0,004}$$
$$- 3,300x_1^{1,975}x_2^{1,728} + 0,177x_1^{0,038}x_3^{4,385}$$

Observamos que algunos de los coeficientes son muy pequeños y que están situados en los exponentes de las variables independientes. De esta forma, es posible que mediante un algoritmo de poda se pudieran eliminar; aunque tendríamos un modelo más interpretable pero menos preciso. Además, en cierta manera, a través de los operadores de mutación estructural de eliminar nodos y/o conexiones y de unir nodos podemos reducir el número de coeficientes del modelo, lo que se puede considerar como una poda encubierta.

5.3.1.1.1 Test de comparaciones

Los algoritmos propuestos para entrenar y diseñar redes son estocásticos, esto es, siguen una tendencia marcada por los mejores individuos de cada generación, pero tanto la inicializaión de los coeficientes del modelo, como la aplicación y desarrollo de los operadores de mutación siguen distribuciones de probabilidad. Pero debido al gran número de posibles soluciones y a la alta dimensionalidad del espacio de búsqueda no es posible determinar cuales van a ser las distribuciones de probabilidad de las aptitudes de los individuos (modelos de red) a lo largo de la evolución de los algoritmos evolutivos propuestos. Por este motivo, consideraremos las distribuciones asociadas a los resultados obtenidos mediante los estadísticos de error en 30 ejecuciones de nuestros algoritmos. Utilizar el valor 30 es obligado si queremos considerar el Teorema Central del Límite, TCL, para considerar de forma aproximada como normales las distribuciones de probabilidad de los resultados. Aún así, el TCL exige que las distribuciones tengan esperanza matemática y varianzas finitas y en nuestro caso no podemos afirmar si esto es así o no.

Test de Normalidad

Por este motivo, realizaremos tests previos de ajuste de normalidad (test de Kolmogorov-Smirnov, K-S) a los 30 valores muestrales obtenidos. Si aceptamos la hipótesis de normalidad para un valor de significación de un 5%,

utilizaremos tests paramétricos de comparaciones de medias y de varianzas. En otro caso, utilizaremos contrastes no paramétricos de comparación de estadísticos de localización. Esta forma de proceder la aplicamos al experimento anterior. Hemos utilizado el software SPSS 11.0 (SPSS, 1989-2001) para la realización de todos los contrastes de hipótesis.

De esta forma, empezamos haciendo un test de K-S de normalidad de los resultados obtenidos de MSE_G en las 30 pruebas realizadas. Los niveles críticos obtenidos son: p=0,701 para EP, p=0,029 para HEP, p=0,075 para HEPC y p=0,071 HEPCD; por lo que para α=0,01 podemos aceptar la hipótesis de normalidad para los resultados obtenidos con cuatro metodologías de optimización propuestas.

Test de comparación de medias y varianzas

Por otra parte, dado que en las metodologías HEP y HEPC partimos de los resultados obtenidos con EP y que la metodología HEPCD es igual a la HEPC pero en diferentes generaciones de la evolución, es bastante razonable que los resultados obtenidos con estas metodologías sean variables aleatorias dependientes. Los tests de dependecias lineales entre las variables nos indican que existen correlaciones significativas, para $\alpha = 0,2$ entre los 30 resultados obtenidos para MSE_G en la última generación con los cuatro métodos (EP versus HEP, EP versus HEPC, EP versus HEPCD, HEP versus HEPC, HEP versus HEPCD and HEPC versus HEPCD con valores del nivel crítico de 0,080, 0,141, 0,200, 0,000, 0,003 y 0,000 respectivamente). Que las variables sean dependientes nos impide realizar contrastes ANOVA de igualdad de medias. Teniendo en cuenta los resultados de los contrastes anteriores de normalidad y dependencia, realizamos contrastes de comparaciones de medias en distribuciones normales y dependientes. De los resultados de estos contrastes observamos que existen diferencias significativas en las medias de los MSE_G obtenidos (Ver las seis primeras filas de la Tabla 5-4) de HEP frente a EP (p= 0,000), de HEPC frente a

EP (p= 0,000), de HEPCD frente a EP (p= 0,000), de HEPC frente a HEP (p= 0,018), de HEPCD frente a HEP (p= 0,000) y de HEPCD frente a HEPC (p= 0,001). De esta forma, los mejores resultados en cuanto a un menor error medio en los valores de MSE_G se obtienen aplicando la metodología HEPCD.

Por otra parte, para realizar comparaciones entre los resultados obtenidos en la última generación con nuestras tres metodologías y los resultados del algoritmo GRNNFA, supondremos que los resultados de los errores MSE obtenidos con los cuatro algoritmos son normales. De esta manera, aplicamos tets de comparaciones de medias para variables independientes, dado que los resultados obtenidos con nuestros algoritmos son independientes de los obtenidos con GRNNFA (Ver las cuatro últimas filas de la Tabla 5-4). Previamente realizaremos tests de igualdad de varianzas, teniendo en cuenta las medias y desviaciones típicas para 30 ejecuciones de nuestros métodos, y 20 para el método GRNNFA (ver Tabla 5-5).

Los resultados de estos contrastes muestran que:

Como $\frac{\overline{S}^2_{EP}}{\overline{S}^2_{GR}}$ = 0,264/0,040= 6,600$\notin$ ($F_{29,19,\ 0,975}$; $F_{29,19,\ 0,025}$)= (0,405;2,47)

Lo que implica que las varianzas poblacionales son diferentes, esto es, GRNNFA es preferible en varianza a EP. Teniendo en cuenta este resultado compararemos las medias $H_0 \equiv \mu_{EP} = \mu_{GR}$, frente a $H_1 \equiv \mu_{EP} \neq \mu_{GR}$. Como $\overline{x}_{GR} - \overline{x}_{EP}$ = 4,563 - 2,719 = 1,844 no pertenece al intervalo (-0,230, 0,230) se rechaza la hipótesis nula de igualdad de medias. Por otra parte, t = 1,844/ 0,102= 18,078, por lo que mirando en tablas de la t de student con 40 grados de libertad tenemos un valor de p= 0,000 (ver fila siete de la Tabla 5-4), por lo que en media es preferible EP frente a GRNNFA.

	Diferencias					
Pares de variables	Media	SD	SE	T	GL	Valores de p
EP – HEP	0,922	0,630	0,115	8,024	29	0,000
EP – HEPC	1,111	0,546	0,100	11,141	29	0,000
EP-HEPCD	1,317	0,499	0,091	14,447	29	0,000
HEP – HEPC	0,188	0,413	0,075	2,496	29	0,018
HEP – HEPCD	0,395	0,498	0,091	4,339	29	0,000
HEPC – HEPCD	0,206	0,289	0,053	3,911	29	0,001
GRNNFA – EP	1,844	0,541	0,102	18,078	40	0,000
GRNNFA – HEP	2,767	0,606	0,113	24,487	38	0,000
GRNNFA – HEPC	2,955	0,435	0,091	32,473	43	0,000
GRNNFA – HEPCD	3,162	0,292	0,060	52,700	48	0,000

Tabla 5-4. Estadísticos asociados al error MSE para las diferencias entre pares de algoritmos.Valores empíricos de los t tests, grados de libertad , GL, y valores de los niveles críticos, p

Algoritmos	Media	SD	nº ejecuciones
EP	2,719	0,505	30
HEP	1,796	0,574	30
HEPC	1,608	0,389	30
HEPCD	1,401	0,217	30
GRNNFA	4,563	0,195	20

Tabla 5-5. Estadísticos asociados al error MSE_G para diferentes algoritmos

Análogamente para comparar HEP con GRNNFA, es similar mostrar que las varianzas son diferentes, esto es, la varianza de GRNNFA es más pequeña que la de HEP. Para la comparación de medias, como $\overline{x}_{HEP} - \overline{x}_{GR}$= 4,563 - 1,796 =

2,767 no pertenece al intervalo (-0,230; 0,230), se rechaza la hipótesis nula. Por otra parte, t = 2,767/ 0,113= 24,487, por lo que mirando en tablas de la t de student en este caso con aproximadamente 38 grados de libertad, tenemos un valor de p= 0,000 (ver fila ocho de la Tabla 5-4), por lo que es preferible en media HEP frente a GRNNFA.

Para comparar HEPC con GRNNFA, actuando de forma similar a lo anterior observamos que en varianza es preferible GRNNFA a HEPC. Para la comparación de medias como $\overline{x}_{GR} - \overline{x}_{HEPC}$ = 4,563 - 1,608 = 2,955 no pertenece al intervalo (-0,183, 0,183), se rechaza la hipótesis nula. Por otra parte t= 2,955/ 0,091= 32,473, ahora con 43 grados de libertad, valor de p= 0,000 (ver penultima fila de la tabla Tabla 5-4), por lo que es preferible en media HEPC frente a GRNNFA.

Por último, para comparar HEPCD con GRNNFA, primero hay que probar si las varianzas son iguales, como $\frac{\overline{S}^2_{EP}}{\overline{S}^2_{GR}}$ = 0,047/0,04=1,18$\in$ (0,405; 2,47) podemos afirmar que las varianzas son iguales en GRNNFA y en HEPCD. Para la comparación de medias como $\overline{x}_{GR} - \overline{x}_{HEPCD}$ = 4,563 - 1,401 = 3,162 no pertenece al intervalo (-0,1215, 0,1215), se rechaza la hipótesis nula, Por otra parte t = 3,162/0,060=52,700, ahora n= 30+20-2= 48 grados de libertad, y el valor de p es 0,000 (ver última fila de la tabla Tabla 5-4), por lo que es preferible en media HEPCD frente a GRNNFA,

En resumen, podemos afirmar que:

- Los algoritmos EP, HEP, HEPC y HEPCD obtienen mejores resultados en media del error MSE_G que el algoritmo GRNNFA para la función de Friedman#1.
- Sin embargo, el algoritmo GRNNFA ofrece mejores resultados en varianza del error MSEG que los algoritmos EP, HEP, HEPC y un resultado similar comparado con HEPCD.

5.3.1.2 Segundo experimento

En este segundo experimento hemos considerado un diseño basado en el método hold-out, esto es un método de validación cruzada donde tomamos el 75% de los patrones para entrenar y el otro 25% para generalizar (Nadeau and Bengio, 2003). De esta forma, creamos 1000 patrones al azar generados mediante la función Friedman#1, y de ellos elegimos 750 para el conjunto de entrenamiento y los restantes 250 para el conjunto de generalización.. Por otra parte, en este experimento no hacemos ninguna transformación ni de las variables independientes ni de la dependiente. En la Tabla 5-6 se sitúan los resultados estadísticos del MSE_G tanto en entrenamiento como en generalización. Una primera observación de estos resultados nos muestra que la metodología de utilizar agrupamiento en clases para realizar la hibridación en la búsqueda a lo largo de diferentes generaciones, es la que mejores resultados nos depara tanto en media como en desviación típica.

Siguiendo la metodología propuesta con anterioridad, hacemos un contraste de ajuste a distribuciones normales de los resultados (Ver Tabla 5-7). Los valores de los errores MSE_G para las 30 ejecuciones muestran que estos no son normales para ninguna de las metodologías aplicadas, puesto que los valores de p son iguales a cero.

	Aprendizaje				Generalización			
Algoritmo	Media	SD	Mejor	Peor	Media	SD	Mejor	Peor
EP	1,057	0,155	0,949	1,585	1,161	0,127	1,103	1,617
HEP	0,988	0,130	0,903	1,4342	1,103	0,102	1,069	1,607
HEPC	0,988	0,130	0,903	1,4342	1,105	0,102	1,066	1,607
HEPCD	0,913	0,054	0,898	1,1995	1,081	0,040	1,065	1,288

Tabla 5-6. Resultados estadísticos en la última generación (4000) del error MSE para Friedman#1 con topología (5:6:1) para 30 ejecuciones. 750 datos entrenamiento y 250 generalizacion. Sin Normalizar.

De esta forma, utilizaremos contrastes no paramétricos de Wilkoxon (Wilcoxon et al., 1973) de igualdad de medianas para poblaciones de distribuciones no conocidas y dependientes (ver Tabla 5-8). Los tests propuestos entre pares de distribuciones muestran que existen diferencias significativas en mediana entre (EP, HEPC), (EP, HEPCD), (HEP, HEPCD) y (HEPC, HEPCD); mientras que no existen entre (HEP, HEPC) p=0,609, para α=0,01. De estos resultados concluímos que HEPCD es el mejor método propuesto.

Algoritmos	Z de K-S	P
EP	2,521	0,000
HEP	2,596	0,000
HEPC	2,630	0,000
HEPCD	2,230	0,000
Aleatorio[40]	1,212	0,106

Tabla 5-7. Valores de Z y p de los tests de K-S

	HEP	HEPC	HEPCD
EP	0,000	0,000	0,000
HEP	-	0,609	0,000
HEPC	-	-	0,000

Tabla 5-8.Valores de p de los tests de Wilcoxon

Como los test de comparación de medianas muestran que la mejor metodología es la dinámica, mostramos a continuación el mejor modelo obtenido. Este modelo se obtiene en la generación 2400, con seis sumandos y con 17

[40] Método propuesto en el tercer experimento

coeficientes y un error MSE_G= 1,0647. Si ordenamos el modelo en la misma forma que la función de Friedman#1, para poder comparar los sumandos tenemos:

$$\begin{aligned}\hat{y} = {} & 5.571 + 66.402\ x_1^{1.256} x_2^{1.257} - 66.772\ x_1^{1.884} x_2^{1.904} \\ & + 43.928\ x_3^{1.665} - 43.703\ x_3^{1.231} x_4^{-0.004} \\ & + 9.745\ x_2^{0.005} x_4^{1.083} \\ & + 4.908\ x_5^{1.091}\end{aligned}$$

De nuevo observamos que un par de coeficientes tienen un valor muy pequeño y se podrían eliminar mediante un sencillo algoritmo de poda, pero hemos preferido mantener un modelo con una mejor capacidad de generalización. Si desarrollamos la función seno en serie de Taylor hasta la segunda potencia, podemos destacar una cierta similitud. De esta forma no sólo hemos ajustado un modelo de red neuronal a la función de partida sino que la hemos reconocido.

5.3.1.3 Tercer experimento

En orden a contrastar la oportunidad de realizar un método de agrupamiento previo a la hibridación, planteamos dos experimentos adicionales. En el primero, se utiliza la metodología HEPC, pero eliminando el análisis de agrupamiento y en su lugar aplicamos el algoritmo de L-M a un número de individuos elegidos al azar de entre el 25% de mejores modelos de redes de la población igual al número de clases propuestas en la última generación (cuatro en este caso), lo denominaremos método Aleatorio. Los resultados sobre el conjunto de generalización después de 10 ejecuciones de cada una de las metodologías se muestran en la Tabla 5-9. La tabla muestra que los resultados utilizando agrupamiento son mejores tanto en media como en varianza. El contraste de estadísticos de localización utilizado ha sido el de Mann-Whitney, dado que la distribución de resultados mediante HEPC no sigue una distribución normal; mientras que el método Aleatorio si sigue una normal (Ver última fila de la Tabla 5-7) y además ambas variables son independientes. El resultado del test muestra

que existen diferencias significativas en sus estadísticos de localización para α= 0.01, ya que p=0.003.

	Generalización				
Algoritmo	Media	SD	Mejor	Peor	Test U de Mann-Whitney
HEPC	1,105	0,102	1,066	1,607	
Aleatorio	1,408	0,665	1,068	2,981	0,003

Tabla 5-9. Estadísticos asociados al error MSE_G para 10 ejecuciones de los métodos HEPC y Aleatorio

El segundo experimento adicional se realizo para probar la eficacia versus eficiencia de nuestras aproximaciones. El método HEPC lo comparamos con un proceso evolucionario donde en lugar de aplicar el algoritmo de optimización al mejor individuo de cada clase, aplicamos el algoritmo de optimización local a cada uno de los individuos del mejor 25% de la población, lo denominaremos L-M Todos. Los resultados después de 10 ejecuciones de ambas metodologías en valores de MSE_G y tiempo (en segundos) se muestran en la Tabla 5-10. La tabla muestra a través de tests t de student, (dado que las pruebas previas indican normalidad en las distribuciones) que las diferencias en media de MSE no son significativas p= 0,993; mientras que si existen diferencias en el tiempo medio empleado por los algoritmos, p=0,000. De esta forma nuestro método es menos costoso desde un punto de vista computacional.

Concluímos que con nuestra metodología obtenemos resultados tan buenos o mejores, similar media, y mejores resultados en cuanto a SD, mejor y peor individuos y menor tiempo de cómputo.

	MSE Generalización				
Algoritmo	Media	SD	Mejor	Peor	t-test
HEPC	1,081	0,009	1,064	1,093	
L-M Todos	1,081	0,010	1,071	1,098	0.993
	Tiempo				
Algoritmo	Media	SD	Mejor	Peor	t-test
HEPC	5,3	0,67	4	6	
L-M Todos	76,9	8,65	64	91	0.000

Tabla 5-10. Estadísticos asociados al error MSE_G y Tiempo de ejecución (en media de segundos por generación) para 10 ejecuciones de los métodos HEPC y L-M Todos.

Los resultados anteriores consolidan nuestros métodos de agrupamiento previo a la hibridación del algoritmo de optimización. Con el primer experimento, hemos mostrado que la aproximación de agrupar es capaz de mejorar un algoritmo con la misma complejidad pero sin el agrupamiento; mientras que con el segundo experimento adicional hemos mostrado que la aproximación de agrupar alcanza el mismo rendimiento que un algoritmo con un mayor coste computacional, con lo que podemos concluir que nuestra idea de agrupar antes de aplicar la búsqueda local es un muy buen compromiso entre el coste computacional y el rendimiento.

5.3.2 Función de Sugeno

En este apartado completamos los resultados presentados en 4.3.2 para la función de Sugeno. Se ha aplicado las metodologías HEPC y HEPCD sobre los datos simulados de la función de Sugeno.

Los parámetros que se han utilizado han sido los mismos que los utilizados para el algoritmo evolutivo (ver 4.3.2). En las generaciones 600, 1200, 1800, 2400,

3000 y generación final se han aplicado las técnicas de agrupamiento y búsqueda local para el algoritmo HEPCD.

Algoritmo	Conjunto	Media	Desv.	Mejor	Peor
HEPC	Entrenamiento	0,0501	0,0281	0,0108	0,1046
	Generalización	0,0431	0,0276	0,0084	0,1112
HEPCD	Entrenamiento	0,0345	0,0216	0,0030	0,0915
	Generalización	0,0279	0,0190	0,0030	0,0846

Tabla 5-11. Valores medios, desviación típica, mejor y peor resultados del error APE_E y APE_G para las metodologías HEPC y HEPCD para la función de Sugeno tras 30 ejecuciones.

Como se puede observar en la Tabla 5-11, los valores obtenidos para la metodología HEPCD mejora en todos los aspectos a la metodología HEPC.

Con respecto a los resultados mostrados en la Tabla 4-5, podemos concluir que se mejora considerablemente los resultados obtenidos en media y desviación típica tanto por el algoritmo evolutivo como por el método IncNet Rot presentado en (Jankowsky, 1999), único método que mejoraba los resultados con respecto al algoritmo evolutivo propuesto en esta tesis.

5.3.3 Microbiología Predictiva

Aplicaremos nuestros algoritmos y metodologías a un problema real ya descrito en el Capítulo 4. Los datos de los que disponemos son 210 curvas que representan el crecimiento del microrganismo *Leuconostoc mesenteroides* a lo largo del tiempo. Las curvas fueron ajustadas a un modelo exponencial con el programa DMFit 1.0 (Baranyi and Roberts, 1994). Para determinar alguno de los

parámetros específicos de los algoritmos y métodos propuestos en este problema es necesario hacer un estudio previo de los valores obtenidos por los SEP del conjunto de entrenamiento para poder tener una idea previa del número de clases a construir. En función de los resultados de este estudio determinamos que los valores iniciales (generación 0 o generación inicial) de K y $\tilde{s}$ son 6 y 25% respectivamente, mientras que los finales (generación End o generación final) son 4 y 25%. En este experimento, hemos utilizado modelos con topología inicial 4:4:1 para Lnlag , Grate e Yend.

5.3.3.1 Análisis de normalidad

Como ya planteamos en la aplicación a la función de Friedman#1, es necesario en primer lugar, realizar un contraste de ajuste a una distribución normal de los errores de SEP sobre el conjunto de generalización, SEP_G, para las cuatro metodologías aplicadas y para cada uno de los tres parámetros de crecimiento microbiano. De esta forma, compararemos los resultados obtenidos mediante contrastes de diferencias de medias o de parámetros de localización de las distribuciones subyacentes. El test que aplicaremos es el de Kolmogorov-Smirnov con un nivel de significación, α= 0,05. Como en la anterior aplicación hemos utilizado el software SPSS 11.0 (SPSS, 1989-2001) para la realización de todos los contrastes de hipótesis. Los resultados de estos contrastes son los siguientes:

- Para Lnlag, las tres metodologias primeras y en las seis generaciones estudiadas los resultados muestran que en las generaciones observadas (250, 500, 750, 1000, 1250, Fin), los resultados de SEP_G son normales con valores de nivel crítico p que oscilan entre 0,099 para HEPFinal y 0,986 EP750.

- Para Grate en las generaciones observadas (600, 1200, 1800, 2400, 3000, Fin) los resultados son también normales con valores de nivel crítico p que oscilan entre 0,129 para HEPC3000 y 0,838 EP1800.
- Para Yend, en las mismas generaciones observadas que para Grate, los resultados muestran que las distribuciones en las primeras generaciones son normales, pero no en las últimas, estas presentan valores de p que entre otros son: 0,005 para HEPC3000, 0,018 para HEPFinal y 0,015 para HEPCFinal.

En función de los resultados anteriores, para Lnlag y Grate utilizaremos contrastes paramétricos de comparaciones de medias, y para Yend contrastes no paramétricos de comparaciones de medianas.

5.3.3.2 Comparación de medias y medianas

- **Parámetro Lnlag**

Para el parámetro Lnlag, realizamos test de comparaciones de medias para todas las generaciones analizadas ANOVA I, y un test posterior de Tamhane de comparaciones múltiples (Tamhane and Dunlop, 2000), dado que no podemos aceptar la hipótesis de igualdad de varianzas de los errores del SEP_G en cada generación.

Para el algoritmo EP, se observa que existen diferencias significativas en las medias del SEP_G en la generación final y en todas las generaciones anteriores con valores de p desde 0,000 hasta 0,011 lo que indica que el algoritmo EP mejora la media del SEP significativamente a lo largo de la evolución; mientras que para HEP y HEPC sólo existen diferencias significativas entre el SEP_G de la última generación y el de la generacion 250 para valores de p=0,000. Si realizamos t test de comparación de medias de SEP_G en las generaciones última y anterior, la 1250, existen diferencias significativas utilizando EP (p=0,000) y no existen utilizando

HEP (p=0,736) o utilizando HEPC (p=0,187) y como consecuencia podríamos acortar el entrenamiento a la generación 1250.

Por otra parte, si analizamos, para este parámetro, los valores de SEP_G en la última generación vemos que están correlados los valores obtenidos utilizando el método, EP versus HEP (p=0,000), y el método HEP versus HEPC (p=0,033); mientras que no existe correlación entre el método EP y el HEPC (ver Tabla 5-12) y en función de este resultado los contrastes de diferencias de medias para variables dependientes o independientes muestran que existen diferencias significativas entre los pares EP-HEP, EP-HEPC (con valores de p, 0,031, 0,000) y no existen entre el par HEP-HEPC con p= 0,112) (ver Tabla 5-13). Esto nos indica que en media y para la última generación, son preferibles los métodos HEP o HEPC, frente al método EP.

- **Parametro Grate**

Para el parámetro Grate, realizamos también tealizamos un test de comparaciones de medias ANOVA I para las seis generaciones analizadas, y un test posterior de Bonferroni de comparaciones múltiples, dado que podemos aceptar la hipótesis de igualdad de varianzas de los errores del SEP_G en cada generación. Para el algoritmo EP, se observa que existen diferencias significativas en las medias del SEP_G en la generación final comparado con cada una de las generaciones anteriores con valores de p desde 0,000. Para el algoritmo HEP el test posterior es de Tamhane (Tamhane and Dunlop, 2000) dado que en este caso no podemos aceptar la hipótesis de igualdad de varianzas; observamos que existen diferencias significactivas entre los pares de generaciones Final-600 y entre Final-1200 (p=0,009 y 0.011) mientras que no existen entre los demas pares de generaciones. Igual procedimiento seguimos para el algoritmo HEPC con idénticos resultados que para HEP, por lo que sólo existen diferencias significativas tan sólo en las comparaciones entre los pares de generaciones Final-600 y Final-1200 (p=0,012 y 0.031).

Si nos centramos en los valores de SEP_G en la última generación vemos que están correlados para los tres métodos (p=0,000) (ver Tabla 5-12), y teniendo en cuenta este resultado pasamos a realizar test de comparaciones de medias de resultados para las tres metodologías muestran que existen diferencias significativas en los tres pares comparados EP-HEP, EP-HEPC y HEP-HEPC con valores de p iguales a 0,000 (ver Tabla 5-13). Podemos concluir que para este parámetro en media y para la última generación, es preferible el HEPC, seguido de Hep y luego el método EP.

- **Parametro Yend**

Para este parámetro, dado que sobre todo en las últimas generaciones los resultados del SEP_G no siguen distribuciones normales, y puesto que existen correlaciones en los valores de SEP_G para los tres métodos (ver Tabla 5-12), utilizaremos test asintóticos bilaterales de Wilcoxon (Wilcoxon et al., 1973) de medianas de SEP_G en la última generación. Los resultados muestran que existen diferencias significativas entre los pares de métodos comparados EP-HEP, EP-HEPC con valores de p iguales a 0,000 y 0,001 respectivamente; mientras que no existen para el par HEP-HEPC (p= 0,053) (ver Tabla 5-13); igual resultado se observa con la prueba de los signos. Por otra parte, hemos realizado test de Kruskal-Wallis (Kruskal and Wallis, 1952) para cada metodología y en las seis generaciones observadas. Estos contrastes muestran que existen diferencias significativas en las medianas del SEP_G para cada generación para las metodologías EP y HEP, con valores de p iguales a 0,000 y 0,001 respectivamente; mientras que no existen para la metodología HEPC, p=0,612

Por último, en cada generación observada realizamos contrastes de Friedman (Friedman, 1991) de medianas para las tres metodologías utilizadas y los resultados obtenidos muestran que existen diferencias significativas en las medianas (con valores de p iguales a 0,000) para todas las generaciones, pero mientras que en las primeras generaciones el orden de preferencia es HEPC, HEP

y EP; en las últimas el orden es HEP, HEPC, EP; por lo que la metodología preferida para este parámetro es la HEP.

Como podemos observar, todas las metodologías proporcionan unos buenos resultados (tanto en términos de precisión como de homogeneidad) para determinar los valores que toman los tres parámetros de las curvas cinéticas. De esta forma, los valores medios del SEP para el conjunto de generalización en la última generación fluctuan desde 5,70 con HEPC hasta 6,03% con EP para Lnlag; desde 3,37 con HEPC y 4,55 con EP para Grate y desde 20,86% con HEP hasta 21,94% con EP, para Yend; mientras que las desviaciones típicas (SD) varian entre 0,26 con HEPC y 0,59 con HEP para Lnlag; entre 0,52 con HEPCD hasta 0,90 con EP para Grate y entre 1,66 con HEPC y 2,02 con EP para Yend. Estos resultados deben analizarse teniendo en cuenta que el número medio de parámetros a estimar para los mejores modelos de predicción utilizando HEPC es de 18,8 (SD= 1,40) para Lnlag, de 18 (SD= 1,80) para Grate y de 19,8 (SD=0,85) para Yend, lo que hace que estos sean bastante interpretables. En resumen, El método HEPC muestra una mejor eficacia para la determinación de los parámetros Lnlag y Grate, mientras que el método HEP es el más eficiente para la predicción de Yend. (Ver Tabla 5-14, Tabla 5-15, Tabla 5-17).

5.3.3.3 Mejores modelos obtenidos

- **Parametro Lnlag**

Los mejores resultados para lnlag se obtienen tanto en media como en desviación típica en la última generación (ver últimas columnas de la Tabla 5-14), utilizando el método HEPC, pues como ya hemos visto, se obtienen diferencias significativas tanto en media como en desviación típica con respecto a HEP y a EP. Por otra parte, el mejor modelo obtenido de todos los propuestos en las seis generaciones analizadas para Lnlag es: metodología HEP, generación 1250, arquitectura 4:4:1 con sesgo SEP_G= 5,07 y nº de conexiones= 19.

$$\text{Lnlag}^{*}=2{,}452-2{,}275\ \frac{(\text{T}^{*})^{3,89}(\text{pH}^{*})^{0,71}(\text{NaCl}^{*})^{10,42}}{(\text{NaNO}_2{}^{*})^{0,64}}$$
$$-0{,}006\ \frac{(\text{NaNO}_2{}^{*})^{24,44}(\text{NaCl}^{*})^{6,87}}{(\text{pH}^{*})^{8,59}}$$
$$+0{,}669\frac{(\text{T}^{*})^{1,26}(\text{NaCl}^{*})^{7,12}}{(\text{NaNO}_2{}^{*})^{1,02}}$$
$$-1{,}195\frac{(\text{T}^{*})^{0,32}(\text{pH}^{*})^{0,02}}{(\text{NaCl}^{*})^{0,12}(\text{NaNO}_2{}^{*})^{0,07}}$$

- **Parametro Grate**

Análogamente y para Grate los mejores resultados se obtienen tanto en media como en desviación típica en la última generación (ver ultimas columnas de la Tabla 5-15), siendo el algoritmo HEPC el mejor, pues con el, como ya hemos visto, se obtienen diferencias significativas tanto en media como en desviación típica con respecto a HEP y a EP. Por otra parte El mejor modelo obtenido para el parámetro Grate es: metodología HEPC, generación Final, arquitectura 4:4:1, con sesgo SEP_G= 2,72 y nº de conexiones= 19 y su ecuación es:

$$\text{Grate}^{*}=1{,}038+9{,}991\frac{(\text{T}^{*})^{5,84}(\text{pH}^{*})^{0,33}(\text{NaCl}^{*})^{0,50}}{(\text{NaNO}_2{}^{*})^{0,01}}$$
$$-6.729(\text{T}^{*})^{3,24}(\text{pH}^{*})^{0,35}(\text{NaCl}^{*})^{1,14}(\text{NaNO}_2{}^{*})^{0,02}$$
$$-0{,}060\frac{(\text{T}^{*})^{0,91}(\text{NaNO}_2{}^{*})^{1,42}}{(\text{pH}^{*})^{1,18}}$$
$$+2{,}371\frac{(\text{T}^{*})^{2,13}(\text{pH}^{*})^{0,20}}{(\text{NaCl}^{*})^{0,10}}$$

- **Parámetro Yend**

Por ultimo, los mejores resultados para Yend se obtienen tanto en media como en desviación típica en la última generación (ver ultimas columnas de la Tabla 5-16), siendo el algoritmo HEPC el mejor, pues con el, como ya hemos visto, se obtienen diferencias significativas tanto en media como en desviación

típica con respecto a EP, aunque no existen diferencias con HEP. Por otra parte el mejor modelo obtenido para Yend es: metodologia HEP, generacion 2400, arquitectura 4:4:1, SEP_G= 17,58 y nº de conexiones= 20 y su ecuación es:

$$\begin{aligned}\mathrm{Yend}^* &= 1,995 - 3,030\frac{(\mathrm{T}^*)^{13,50}(\mathrm{NaCl}^*)^{10,31}(\mathrm{NaNO_2}^*)^{0,35}}{(\mathrm{pH}^*)^{1,39}} \\ &+1,077\frac{(\mathrm{T}^*)^{0,14}(\mathrm{pH}^*)^{1,20}(\mathrm{NaNO_2}^*)^{1,87}}{(\mathrm{NaCl}^*)^{0,12}} \\ &- 0,036\frac{(\mathrm{NaCl}^*)^{5,98}(\mathrm{NaNO_2}^*)^{0,18}}{(\mathrm{T}^*)^{1,63}(\mathrm{pH}^*)^{1,46}} \\ &- 1,257\frac{(\mathrm{pH}^*)^{0,41}(\mathrm{NaNO_2}^*)^{1,81}}{(\mathrm{NaCl}^*)^{0,04}}\end{aligned}$$

Parametro	Lnlag (4:4:1)		Grate (4:4:1)		Yend (4:4:1)	
Algoritmo	HEP	HEPC	HEP	HEPC	HEP	HEPC
EP	0,766	0,305	0,620	0,642	0,794	0,801
	(0,000)	(0,101)	(0,000)	(0,000)	(0,000)	(0,000)
HEP		0,390		0,732		0,950
		(0,033)		(0,000)		(0,000)

Tabla 5-12. Contrastes de dependencia en la última generación. Correlaciones y valores de p entre paréntesis

Parametro	Lnlag (4:4:1) valor de t (valor de p)		Grate (4:4:1) valor de Z (valor de p)		Yend (4:4:1) valor de t (valor de p)	
Algoritmo	HEP	HEPC	HEP	HEPC	HEP	HEPC
EP	2,262 (0,031)	5,527 (0,000)	5,929 (0,000)	9,405 (0,000)	-3,857 (0,000)	-3,281 (0,000)
HEP		1,638 (0,112)		5,109 (0,000)		-1,932 (0,134)

Tabla 5-13. Test t de student y Z de Wilcoxon para las medias y medianas en la generación final

Lnlag	nº gen.	250		500		750		1000		1250		Final	
(4:4:1)	Algorit	Media	SD	Media	SD	Media	SD	Media	SD	Media	SD	Media	SD
	EP	7,43	0,64	6,87	0,53	6,64	0,45	6,46	0,40	6,34	0,37	6,03	0,29
	HEP	6,35	0,50	5,99	0,39	5,92	0,37	5,96	0,38	5,90	0,39	5,86	0,59
	HEPC	6,22	0,44	5,87	0,36	5,89	0,36	5,82	0,32	5,80	0,38	5,70	0,26
	Algorit	Mejor	Nº Con	Mejor	Nº Con	Mejor	Nº Con	Mejor	Nº Con	Mejor	Nº Con	Mejor	Nº Con
	EP	6,19	17	5,85	16	5,78	17	5,78	19	5,74	19	5,68	18
	HEP	5,34	18	5,15	20	5,44	19	5,56	19	5,07	19	5,07	19
	HEPC	5,34	18	5,15	20	5,45	18	5,33	19	5,07	19	5,07	19

Tabla 5-14.Resultados estadísticos del SEP_G para diferente nº de generaciones en 30 ejecuciones de los métodos EP, HEP y HEPC para Lnlag

Grate	nº gen.	600		1200		1800		2400		3000		Final	
(4:4:1)	Algorit	Media	SD	Media	SD	Media	SD	Media	SD	Media	SD	Media	SD
	EP	8,15	1,00	7,25	0,90	6,68	0,85	6,24	0,83	5,85	0,87	4,55	0,90
	HEP	4,85	1,43	4,50	0,90	4,34	0,92	3,96	0,80	4,02	1,00	3,38	0,65
	HEPC	4,10	0,99	3,86	0,65	3,67	0,62	3,47	0,47	3,56	0,62	3,36	0,52
	Algorit	Mejor	Nº con	Mejor	Nº con	Mejor	Nº con	Mejor	Nº con	Mejor	Nº con	Mejor	Nº con
	EP	6,58	16	5,75	17	5,17	17	4,67	18	4,24	19	3,62	18
	HEP	2,85	16	3,11	20	3,02	18	2,96	19	2,86	18	2,75	17
	HEPC	2,95	20	2,85	16	2,73	19	2,76	18	2,75	20	2,72	19

Tabla 5-15 Resultados estadísticos del SEP_G para diferente nº de generaciones en 30 ejecuciones de los

Yend	nº gen.	600		1200		1800		2400		3000		Final	
(4:4:1)	Algorit	Media	SD	Media	SD	Media	SD	Media	SD	Media	SD	Media	SD
	EP	25,40	2,14	23,85	1,78	23,22	1,69	22,88	1,71	22,55	1,79	21,95	2,01
	HEP	22,45	2,27	21,10	2,00	21,48	1,85	21,09	1,95	21,09	1,95	20,86	1,86
	HEPC	21,87	2,17	21,28	1,92	21,35	1,96	21,36	1,88	21,15	1,86	21,04	1,65
	Algorit	Mejor	Nº Con	Mejor	Nº Con	Mejor	Nº Con	Mejor	Nº Con	Mejor	Nº Con	Mejor	Nº Con
	EP	22,20	18	21,86	18	21,42	17	21,03	19	20,71	19	19,68	21
	HEP	19,00	18	18,14	18	18,20	19	17,57	20	17,59	20	17,81	20
	HEPC	19,00	18	19,56	20	18,75	21	19,87	18	17,65	20	19,31	20

Tabla 5-16. Resultados estadísticos del SEP_G para diferente nº de generaciones en 30 ejecuciones de los métodos EP, HEP y HEPC para Yend

5.3.3.4 Resultados obtenidos mediante la metodología dinámica

Para realizar este análisis consideraremos en cada ejecución del algoritmo los resultados alcanzados mediante la metodología de optimización HEPC, pero tomando como solución final el mejor modelo obtenido en las generaciones analizadas, junto con el número de conexiones de este modelo. De esta forma, tenemos en cada ejecución el mejor de los mejores modelos obtenidos en las 6 generaciones propuestas para este experimento. Los resultados estadísticos obtenidos en 30 ejecuciones para cada parámetro de crecimiento se muestran en la Tabla 5-17.

Alg. HEPCD	SEP_G				nº de conexiones			
Parametro	Media	SD	Mejor	Peor	Media	SD	Mejor	Peor
Lnlag	5,50	0,21	5,07	6,01	18,43	1,25	16	20
Grate	3,12	0,38	2,72	4,44	18,57	1,41	14	21
Yend	20,38	1,73	17,65	26,03	19,13	1,35	16	21

Tabla 5-17. Resultados estadísticos del SEP_G y nº de conexiones en 30 ejecuciones del método HEPCD para los tres parámetros de crecimiento.

5.3.3.4.1 Parametro Lnlag

Utilizando la metodología HEPCD para Lnlag los valores de la media y de la desviación standard muestran unos resultados muy precisos y homogéneos (5,50, 0,21), con un SEP de 4,97 para el mejor modelo. Este valor es pequeño si los comparamos con otros estudios donde el modelado de los datos se hace mediante redes MLP, esto es, de tipo perceptrón multicapa con retropropagación y donde el valor del SEP de mejor modelo es 6,55 con 22 coeficientes para un modelo 4:4s:1l (García-Gimeno et al., 2004). Además, el peor valor obtenido con esta metodología, 6,01, es inferior al valor medio obtenido con EP, 6,03. Además, los valores medios de HEP y HEPC en la última generación, 5,86 y 5,70 son

superiores al valor medio de HEPCD, 5,50 (Tabla 5-14,Tabla 5-17), lo que da idea de la robustez del método. Por otra parte, los tamaños de las redes propuestas como óptimas, 16, son menores a los valores de los modelos obtenidos en la última generación con las tres metodologías propuestas lo que da idea de que no ha habido sobreentrenamiento en los mejores modelos obtenidos a lo largo de la evolución.

5.3.3.4.2 Parametro Grate

Para este parámetro los valores de la media y de la desviación típica muestran unos resultados muy precisos aunque no tan homogéneos como para Lnlag (3,12, 0,38). De nuevo el valor del SEP para el mejor modelo, 2,72, es también inferior al obtenido para este parámetro con modelos ANN, cuyo valor es 3,77 con 19 coeficientes para una arquitectura 4:3s:1l (García-Gimeno et al., 2004). En este caso, el peor valor, 4,44, es también inferior a la media obtenida mediante EP, 4,55, (Tabla 5-15). La media de los tamaños de los mejores modelos, 18,57, muestra que estos se encuentran en torno a la generación Final lo que indica que para los mejores modelos no existe un sobreentrenamiento.

5.3.3.4.3 Parametro Yend

Los valores de este parámetro utilizando esta metodología respecto de la media y de la desviación standard muestran unos resultados muy precisos y muy poco homogéneos (20,38, 1,73). Ahora el SEP del mejor modelo 17,65 es peor que el obtenido mediante ANN cuyo valor es 14,15 con 13 coeficientes asociados a un modelo 4:3s:1l (García-Gimeno et al., 2004). La justificación de este peor resultado está en que el parámetro Yend al ser el valor de la señal a tiempo sufientemente grande la interacción entre las variables de entrada ya no es tan importante para su determinación, y los modelos MLP son preferibles a los PU. Hemos de tener cuenta además que este es el parámetro más difícil de predecir, al menos para este microorganismo y en ambiente de anaerobiosis (García-Gimeno

et al., 2004). Además el peor valor, 26,03, es en este caso muy superior a la media obtenida mediante EP, 21,95, (ver Tabla 5-17) por lo que en este caso no es tan clara la utilización de la metodología HEPC, ni en la generación Final ni a lo largo de las distintas generaciones de la evolución, lo que concuerda con los resultados comentados anteriormente. La media de los tamaños de los mejores modelos, 19,13, muestra que estos se encuentran en torno a la generación 1800 lo que indica que para los mejores modelos existe sobreentrenamiento, aunque al ser la desviación típica grande 1,35, los mejores modelos se pueden obtener tanto en las primeras como en las últimas generaciones analizadas.

Como conclusión, observamos que los mejores resultados se obtienen tanto en media como en varianza con la metodología HEPCD para Lnlag y Grate; mientras que para Yend si comparamos las metodologias HEP en la generación Final y HEPC, vemos que tanto la media como la varianza son menores para esta última metodología por lo que es en general la más apropiada

Los resultados experimentales, tanto en la función de prueba de Friedman#1 como en un problema difícil de previsión de parámetros de crecimiento microbiano en microbiologia predictiva mediante modelos de segundo orden, muestran que nuestras propuestas de hibridación despues de evolucionar la población y de agrupar a los mejores individuos tanto en la generación final como dinámicamente a lo largo del proceso evolutivo son muy adecuados y sobre todo esta última puede competir con otros algoritmos y metodologías propuestas hasta el momento dentro de la comunidad científica de Redes Neuronales y de Computación Evolutiva.

5.3.3.5 Comparación con los valores óptimos del problema de regresión

En el experimento de microbiología predictiva vamos a calcular la cota inferior del error de predicción para los patrones del conjunto de entrenamiento,

SEP_E, para tener una idea del grado de aproximación que el mejor modelo obtenido mediante nuestras cuatro metodologías produce sobre los datos del conjunto de entrenamiento. Para ello, calculamos el SEP de la curva que mejor se ajusta a la nube de puntos, esto es, la curva que pasa por las medias marginales de las distribuciones estadísticas de la muestra de 150 datos del conjunto de entrenamiento, este SEP será el optimo global a alcanzar para el conjunto de entrenamiento. Tenemos por tanto, por una parte los 150 datos del conjunto de entrenamiento, y por otra, los valores de las 30 medias marginales (tantas como las diferentes condiciones medio ambientales existentes), obtenidas para los cinco datos tomados al azar, del valor de Lnlag≡l, de entre los siete obtenidos para cada condición de envasado del alimento. Una vez obtenidos los errores en el conjunto de entrenamiento calculamos el valor del SEP_E en la forma

$$SEP_E = 100 \times \frac{\sqrt{\sum_{i=1}^{150} (l_i - \overline{l}_j)^2}}{\overline{l}}$$

donde l_j = mejor valor de predicción de Lnlag en la j-ésima ejecución; para $j = 1,...,30$.

Si tenemos en cuenta que los mejores modelos para Lnlag utilizando el conjunto de entrenamiento y las metodologías EP, HEP y HEPC, tienen unos valores de SEP_E de 4,49, 3,84 y 3,84 respectivamente frente a 5,51 para ANN, para una arquitectura inicial 4:4:1, vemos que utilizando las metodologías HEP y HEPC obtenemos un modelo que se encuentra muy próximo al modelo ideal, cuyo $SEPLnlag_E$= 3,38.

De igual manera, para el parámetro Grate los mejores modelos utilizando el conjunto de entrenamiento y las mismas metodologías tienen unos valores de SEP de 3,39, 2,84 y 2,44 frente a 4,13 para ANN para una arquitectura 4:4:1. Vemos que utilizando la metodología HEPC obtenemos un modelo que se

encuentra también muy próximo al modelo ideal, para el conjunto de entrenamiento, cuyo $SEPgrate_E$= 1,64.

Por último, para Yend tenemos unos valores de SEP_E para los mejores modelos y para cada método, con una arquitectura inicial 4:4:1 de 16,04, 15,35 y 15,35 frente a 14,60 para ANN. En este caso el valor del SEP del modelo ideal (aquel que pasa por las medias marginales) toma el valor 10,42, por lo que estamos algo más lejos de alcanzar el óptimo global en entrenamiento.

6 Conclusiones y Trabajos Futuros

La presente memoria profundiza en el estudio de las redes neuronales basadas en unidades de tipo produto (PU). Este tipo de redes, poco estudiadas y utilizadas por la comunidad científica dedicada al trabajo con redes neuronales, suponen a nuestro juicio una alternativa interesante a las redes neuronales estándar de base sigmoide, ya que permiten implementar funciones de orden superior en las que existen interacciones entre las variables y además verifican propiedades de densidad similares a las que tienen los otros modelos de redes como las redes de base sigmoide o las redes de base radial. La dificultad para entrenar las redes PU es una de las razones del reducido número de trabajos de investigación que las utilizan.

En este trabajo de tesis introducimos en primer lugar, un algoritmo evolutivo para diseñar la estructura y estimar los pesos de una red neuronal PU. El algoritmo propuesto ha demostrado su habilidad para evitar quedar atrapado

en mínimos locales en una superficie de error extremadamente rugosa y con frecuentes regiones planas. Dicho algoritmo se ha aplicado a la resolución de problemas de regresión.

En segundo lugar, se presentan varios algoritmos evolutivos híbridos para realizar la misma tarea de diseño y entrenamiento de las redes PU. En esencia, la propuesta híbrida incorpora al algoritmo evolutivo un procedimiento de búsqueda local realizada por el algoritmo de Levemberg-Marquart, junto con un proceso previo a la búsqueda local de agrupación que permite reducir de forma eficiente el coste computacional del algoritmo. Los resultados obtenidos sobre diferentes funciones de prueba y sobre bases de datos reales muestran diferencias estadísticas significativas entre las versiones dinámica y estática del algoritmo híbrido y con respecto al algoritmo evolutivo sin hibridación.

El estudio realizado durante este trabajo de tesis permite, por tanto, realizar las siguientes conclusiones:

1. Las redes neuronales basadas en unidades producto son aproximadores universales. Esta propiedad de densidad proporciona una sólida base teórica para considerar esta familia de funciones como instrumento de garantía para la aproximación de funciones y para la resolución de problemas de regresión.

2. El algoritmo evolutivo propuesto permite diseñar y entrenar una red neuronal basada en unidades producto, en donde la superficie de error suele ser muy rugosa y con regiones planas. De esta forma, se proporciona una alternativa a los métodos de entrenamiento basados en el gradiente aplicados hasta el momento en este tipo de redes.

3. Los estudios estadísticos realizados muestran que la incorporación de un procedimiento de búsqueda local al algoritmo evolutivo

mediante las dos versiones del algoritmo híbrido desarrolladas (estática y dinámica) mejoran sustancialmente la eficacia y la eficiencia del algoritmo evolutivo. Así mismo, la versión dinámica de la hibridación obtiene resultados significativamente mejores que la versión estática.

4. La inclusión de un proceso de agrupamiento de los individuos de la población de redes, previo a la búsqueda local, mejora sustancialmente la eficiencia del algoritmo evolutivo.

5. Los algoritmos propuestos han sido aplicados a la resolución de problemas de regresión con datos sintéticos y con datos reales, obteniéndose resultados comparables a los obtenidos por otras técnicas de regresión.

6. Los resultados obtenidos usando redes basadas en unidades producto confirman que estas redes son una alternativa a las redes neuronales estándar, basadas en unidades sigmoide, en la resolución de determinado tipo de problemas en los que existen interacciones entre las variables independientes. Concretamente, en los datos reales de crecimiento microbiano considerados han demostrado un rendimiento superior a las redes neuronales estándar. Sin embargo, para estructuras de datos diferentes el rendimiento ha sido inferior.

7. Los modelos obtenidos en el caso de datos reales de crecimiento microbiano han resultado ser más interpretables a juicio de los investigadores del área de microbiología predictiva que los modelos obtenidos hasta el momento con las redes neuronales estándar.

La investigación realizada abre nuevas líneas de trabajo futuras, algunas de ellas iniciadas ya y con resultados previos prometedores. A continuación

presentamos de forma breve las propuestas surgidas a partir de la memoria presentada.

- Los algoritmos evolutivos propuestos han sido diseñados específicamente para la resolución de problemas de regresión. Una primera línea de trabajo que surge es la adaptación del algoritmo evolutivo para abordar problemas de clasificación.
- En la estructura de red neuronal PU utilizada en este trabajo, siempre se ha tomado la identidad como función de salida de los nodos de la capa oculta y de la salida de la red. En estos momentos estamos estudiando el uso de otras funciones de salida de tipo sigmoide. Este cambio probablemente mejorará la capacidad de aproximación de las redes con unidades producto aunque también aumentará su complejidad.
- Otra línea de trabajo abierta es el diseño y entrenamiento de redes híbridas en las que los nodos de la capa oculta puedan tener diferente naturaleza. La combinación de nodos de tipo PU, con nodos de tipo sigmoide o de base radial pueden ser una forma de encontrar modelos funcionales en donde cada submodelo ajuste una región determinada del espacio.

Relacionada con la línea de trabajo anterior, una posibilidad a estudiar es aplicar técnicas de coevolución para construir redes híbridas, en donde cada submódulo estaría formado por redes de tamaño reducido constituídas por diferentes tipos de unidades: de tipo sigmoide, de base radial o de tipo producto.

FIN

7 Bibliografía

Andersen, R.A., Essick, G.K. and Siegel, R.M., 1985. Encoding of spacial location by posterior parietal neurons. Science, 230: 456-458.

Anderson, T.W., 1984. An introduction to multivariate statistical analysis. John Wiley & Sons, New York.

Angeline, P.J., Saunders, G.M. and Pollack, J.B., 1994. An evolutionary algorithm that constructs recurrent neural networks. IEEE Transactions on Neural Networks, 5 (1): 54-65.

Anzani, A., Ohzawa, I. and Freeman, R.D., 1999. Neural mechanisms for processing binocular information I. Simple cells. Journal of Neurophysiology, 82: 891-908.

Baranyi, J. and Roberts, T.A., 1994. A dynamic approach to predicting bacterial growth in food. Int. J. Food Microbiol., 23: 277-294.

Barbolla, R., Cerdá, E. and P., S., 1991. Optimización Matemática: teoría, ejemplos y contraejemplos. Espasa Calpe.

Beasly, D., Bull, D. and Martín, R., 1993. Part 1, fundamentals, An overview of genetic algorithms. University Computing, pp. 58-69.

Bersini, H. and Renders, B., 1994. Hybridizing genetic algorithms which hill-climbing methods for global optimization: two possible ways, Proc. IEEE Int. Symp. Evolutionary Computation, Orlando, FL, pp. 312-317.

Beyer, H.G., 1994. Towards a theory of evolution strategies: the (μ,λ) theory. Evolutionary Computation, 2(4): 381-407.

Bishop, M., 1995. Neural Networks for Pattern Recognition. Oxford University Press.

Björkroth, K. and Korkeala, H., 1996. Evaluation of lactobacilus sake contamination in vacuum packaged sliced cooked meat products by ribotyping. J. Food Prot., 59: 398-401.

Brooks, S.H., 1958. A discussion of random methods for seeking maxima. Operations Research, 6: 244-253.

Carney, J. and Cunningham, P., 2000. Tuning diversity in bagged ensembles. International Journal of Neural Systems, 10(4): 267-279.

Carroll, S.M. and Dickinsion, B.W., 1989. Construction of neural nets using the Randon transform, Proceedings of the IEEE 1989 International Joint Conference on Neural Networks. IEEE, New York, pp. 607-611.

Cohen, S. and Intrator, N., 2002. Forward and backward selection in regression hybrid network, Third International Workshop on Multiple Classifier Systems.

Conover, W.J., 1971. Practical nonparametric statistics. John Wiley & Sons, New York.

Corne, D., Dorigo, M. and Glover, F., 1999. New Ideas in Optimization. McGraw Hill.

Cybenko, G., 1989. Aproximation by superpositions of a sigmoidal function. Mathematics of Control, Signals and Systems, 2: 302-366.

Dawkinss, R., 1976. The Selfish Gene. Oxford University Press, Oxford, UK.

Deb, K., 2003. A population-based algorithm-generator for real-parameter optimization. Soft Computing, (in press). Springer.

Deb, K., 2005. A population-based algorithm-generator for real-parameter optimization. Soft Computing, (in press). Springer.

Denison, D.G.T., Holmes, C.C., Mallick, B.K. and Smith, A.F.M., 2002. Bayesian methods for nonlinear classification and regression. Jonh Wiley & Song, West Sussex, England.

Do·Carmo, M.P., 1976. Differential Geometry of Curves and Surfaces. Prentice Hall.

Duch, W. and Jankowsky, N., 2001. Transfer functions: hidden possibilities for better neural networks, 9th European Symposium on Artificial Neural Networks (ESANN), Brugge, pp. 81-94.

Durbin, R. and Rumelhart, D., 1989. Products Units: A computationally powerful and biologically plausible extension to backpropagation networks. Neural Computation, 1: 133-142.

Eberhart, R., Dobbins, R. and Simpson, P., 1996. Computational Intelligence PC Tools. Academic Press.

Engelbrecht, A., Fletcher, L. and Cloete, I., 1999. Variance Analysis of Sensitivity Information for Pruning Feedforward Neural Networks, IEEE International Joint Conference on Neural Networks, Washington DC, USA.

Engelbrecht, A.P. and Ismail, A., 1999. Training product unit neural networks. Stability and Control: Theory and Applications, 2(1-2): 59-74.

Fogel, D.B., 1994. An introduction to simulated evolutionary optimization. IEEE Transactions on Neural Networks, 5: 3-14.

Fogel, D.B., 1995. Evolutionary Computation: Toward a New Philosofy of Machine Intelligence. IEEE Press, New York.

Fogel, D.B., 1998. An evolutionary approach to travelling salesman problem. Biological Cybernetics, 60: 139-144.

Fogel, D.B., Owens, A.J. and Wals, M.J., 1966. Artificial Intelligence Throught Simulated Evolution. Wiley, New York.

Fonseca, C.M. and Fleming, P.J., 1995. An overview of evolutionary algorithms in multiobjetive optimization. Evolutionary Computation, 3(1): 1-16.

Fraser, A.S., 1957. Simulation of genetic systems by automatic digital computers. I. Introduction. Australian Journal of Biological Sciencies, 10: 484-491.

Frean, M., 1990. The upstart algorithm: A method for constructing and training feedforward neural networks. Neural Computation, 2(2): 198-209.

Friedman, J., 1991. Multivariate adaptive regression splines (with discussion). Ann. Stat., 19: 1-141.

Friedman, J. and Stuetzle, W., 1981. Projection pursuit regression. Journal of the American Statistical Association, 76(376): 817-823.

Fukunaga, K., 1990. Introduction to Statistical Pattern Recognition. Academic Press.

Funahashi, K., 1989. On the approximate realization of continuous mappings by neural networks. Neural Networks, 2(3): 183-192.

Gabbiani, F., Krapp, H.G. and Laurent, G., 1999. Computation of object aproach by a wide-field, motion-sensitive neuron. Journal of Neuroscience, 19: 1122-1141.

García-Gimeno, R.M., Hervás-Martínez, C., Rodríguez-Pérez, M.R. and Zurera-Cosano, G., 2004. Modelling the growth of Leuconostoc mesenteroides by means artificial neural network. Int. J. Food Microbiol., (in review).

García-Pedrajas, N., Hervás-Martínez, C. and Muñoz-Pérez, J., 2002. Multiobjetive cooperative coevolution of artificial neural networks. Neural Networks, 15(10): 1255-1274.

Ghosh, J. and Shin, Y., 1992. Efficient higher-order neural networks for classification and function approximation. International Journal of Neural Systems, 3(4): 323-350.

Giles, C.L. and Maxwel, T., 1987. Learning, invariance,and generalization in high-order neural networks. Applied Optics, 26(23): 4972-4978.

Goldberg, D.E., 1989. Genetic Algorithms in Search, Optimization, and Machine Learning. Addison-Wesley, Reading, MA.

Hancock, P., 1992. Genetic algorithm and permutation problems: A comparison of recombination operators for neural net structure specification. In: D.y.S. Whitley, J. D. (Editor), Proc. Int. Workshop of Combinations of Genetics Algorithms and Neural Networks (COGANN-92),. IEEE Computer Soc. Press., Los Alamitos, CA, pp. 108-122.

Harp, S.A., Samad, T. and Guha, A., 1989. Toward the genetic synthesis of neural networks. In: J.D. Schaffer (Editor), 3er. Int. Conf. Conf. Genetic Algorithms and Their Applications. Morgan Kaufmann, San Mateo, CA, pp. 360-369.

Hart, 1994. Adaptive global optimization with local search. Ph. D. Thesis, University of Califonia, San Diego, EEUU.

Hatsopoulos, N., Gabbiani, F. and Laurent, G., 1995. Elementary computation of object approach by a wide-field visual neuron. Science, 270: 1000-1003.

Haykin, 1994. Networks, A comprehensive Foundation. McMillan.

Herrera, F., Lozano, M. and J.L., V., 1998. Tackling real-coded genetic algorithms: operators and tools for the behavioral analysis. Artificial Inteligence Reviews, 12(4): 265-319.

Hertz, J., Krong, A. and Palmer, R., 1991. Introduction to the Theory of Neural Computation. Reading, MA. Addison-Wesley.

Hervás, C., Martínez-Estudillo, A.C., Silva, M. and Serrano, J.M., 2005. Improving the quantification of highly overlapping chromatographic peaks by using product unit neural networks modeled by an evolutionary algorithm. Journal Chemical Information and Modelling, in review.

Hervás, C., Toledo, R. and Silva, M., 2001. Use of pruned computational neural networks for processing the response of oscillating chemical reactions with a view to analyzing nonlinear multicomponent mixtures. Journal of Chemical Information and Computer Sciencies, 41(4): 1083-1092.

Hinton, G.E., McClelland, J.L. and Rumelhart, D.E., 1986. Distributed representations. In: D.E.a.M. Rumelhart, J.L. editors (Editor), Parallel Distributed Processing: Explorations in the Microstructure of Cognition. MIT Press, Cambridge, MA, pp. 77-109.

Hinton, G.E. and Nolan, S.J., 1987. How learning can guide evolution. Complex Systems, 1: 495-502.

Hofmeister, T., 1994. Depth-efficient thershold circuits for arithmethic functions. In: K.Y.S.a.A.O. V. Roychowdhury (Editor), Theoretical Advances in neural computation and learning. Norwell, MA:Kluwer, pp. 37-84.

Holland, J.H., 1975. Adaptation in Natural and Artificial Systems. Univ. Michigan Press, Ann Arbor, MI.

Holland, J.H., 1988. Using classifier systems to study adaptative nonlinear networks. Lectures in the Sciences of Complexity. Addison-Wesley, Redwood City, CA, 463-469 pp.

Horikawa, S., Furuhashi, T. and Uchikawa, Y., 1992. On fuzzy modeling using fuzzy neural networks with the back-propagation algorithm. IEEE Trans. on Neural. Networks, 3(5): 801-806.

Hornik, K., Stinchcombe, M. and White, H., 1989. Multilayer feedforward networks are universal aproximators. Neural Networks, 2: 359-366.

Hornik, K., Stinchcombe, M. and White, H., 1990. Universal aproximation of an unknown mapping and its derivates using multilayer feedforward networks. Neural Networks, 3: 551-560.

Houck, C.R., Joines, J.A. and Kay, M.G., 1996. Comparison of genetic algorithm, random start, and two-opt switching for solving large location-allocation problems. Computers Opteration Research, 23(6): 587-596.

Houck, C.R., Joines, J.A. and Kay, M.G., 1997. Empirical Investigation of the benefits of partial lamarckianism. Evolutionay Computation, 5(1): 31-60.

Ismail, A. and Engelbrecht, A.P., 1999. Training products units in feedfoeward neural networks using particle swarm optimisation. Development and practice of artificial intelligence techniques, proceeding of the international conference on artificial intelligence. In V.B. Bajic & D. Sha (Eds), Durban, South Africa, 36-40 pp.

Ito, Y., 1991a. Representation of functions by superpositions of a step or sigmoid function and their applications to neural network theory. Neural Networks, 2(6): 568-576.

Ito, Y., 1991b. Approximation of functions on a compact set by finite sums of a sigmoid function without scaling. Neural Networks, 4: 817-826.

Jankowsky, N., 1999. Flexible Transfer Functions with Ontogenic Neural Networks, Toru, Poland.

Janson, D.J. and Frenzel, J.F., 1993. Training product unit neural networks with genetic algorithms. IEEE Expert, 8(5): 26-33.

Joines, J.A. and Kay, M.G., 2002. Utilizing hybrid genetic algorithms. Evolutionary Optimization. Kluwer Academic Publishers.

Joines, J.A., Kay, M.G., King, R.E. and Culbreth, C.T., 2000. A hybrid genetic algorithm for manufacturing cell design. Journal of the Chinesse Institute of Industrial Engineers.

Kinnebrock, W., 1994. Accelerating the standard backpropagation method using a genetic approach. Neurocomputing, 6(5-6): 583-588.

Kirkpatric, S., Gellat, C.D.J. and Vecchi, M.P., 1983. Optimization by simulated annealing. Science, 220: 671-680.

Kitano, H., 1990. Designing neural networks using genetic algorithms with graph generation system. Complex Syst., 4(4): 461-476.

Knerr, S., Personnaz, L. and G., D., 1992. Handwritten digit recognition by neural networks with single-layer training. IEEE Transactions on Neural Networks, 3: 962-968.

Kollen, A. and Pesch, E., 1994. Genetic local search in combinatorial optimization. Discrete Applied Mathematics and Combinatorial Operation Research and Computer Science, 48: 273-284.

Kosinski, N. and Weigl, M., 1995. Mapping neural networks and fuzzy inference systems for approximation of mutivariate function. In: E.Kacki (Editor), System Modeling Control, Artificial Neural Networks and their Applications, Poland, pp. 60-65.

Kosko, B., 1991. Neural Networks and fuzzy systems: A dinamical systemas approach to machine inteligence. Prentice Hall.

Koza, J.R., 1989. Evolving programs using Symbolic expressions. In: e. N. S. Sridharan (Editor), Proc. of the 11th Int'l Joint Conf. on Artificial Intelligence. Morgan Kaufmann, San Mateo, CA, pp. 768-774.

Krasnogor, N., 2002. Studies on the Theory and Design space of Memetic Algorithms.Ph. D. Thesis. thesis Thesis, University of the West of England, Bristl, United Kingdom.

Kruskal, W.H. and Wallis, W.A., 1952. Use of ranks in one-criterion variance analysis. JASA, 47: 583-621.

Lang, K.J., Waibel, A.H. and E., H.G., 1990. A time-delay neural network architecture for isolated word recognition. Neural Networks, 3(1): 33-43.

Lee, E.W.M., Lim, C.P., Yuen, R.K.K. and Lo, S.M., 2004. A hybrid neural network model for noisy data regression. IEEE Trans. on Neural. Networks, 34(2): 951-960.

Lee, S.H. and Hou, C.L., 2002. An art-based construction of rbf networks. IEEE Trans. on Neural. Networks, 13(6): 1308-1321.

Leerink, L.R.G., Horne, B.G. and Jabri, M.A., 1995. Learning with products units. Advances in neural networks processing systems, 7: 537-544.

Leshno, M., Lin, V.L., Pinkus, A. and Schocken, S., 1993. Multilayer feeedforward networks with a nonpolynomical activation function can aproximate any function. Neural Networks, 6: 861-867.

Levenberg, K., 1944. A method for solution of certain non-linear problems in least squares. Quarterly Journal of Applied Mathematics II(2): 164-168.

Locatelli, M. and Schoen, F., 1999. Random Linkage A Family of acceptance/ rejection algorithms for global optimization. Mathematical Programming, 85(2): 379-396.

Lozano, M., Herrera, F., Krasnogor, N. and Molina, D., 2004. Real-Coded Memetic Algorithms with Crossover Hill-Climbing. Evolutionary Computation, 2004, in press.

Maiorov, V. and Pinkus, A., 1999. Lower bounds for approximation by MLP neural networks. Neurocomputing, 25(1-3): 81-91.

Marquardt, D.W., 1963. An algorithm for least-squares estimation of non-linear parameters. Journal of the Society of Industrial and Applied Mathematics, 11(2): 431-441.

Martí, R. and Moreno, J.M., 2003. MultiStart Methods. Revista Iberoamericana de inteligencia artificial, 19: 49-60.

Martínez-Estudillo, A.C., Hervás-Martínez, C., García-Pedrajas, N. and Martínez-Estudillo, F.J., 2002. Modelado de sistemas mediante modelos híbridos de redes neuronales y computación evolutiva, Iberamia 2002, Sevilla.

Martínez-Estudillo, A.C., Hervás-Martínez, C., Martínez-Estudillo, A.C. and García-Pedrajas, N., 2005a. Hybrid method base on clustering for evolutionary algorithms with local search. IEEE Transaction on Systems, Man and Cybernetics, Par B: Cybernetics, Submitted.

Martínez-Estudillo, A.C., Hervás-Martínez, C., Martínez-Estudillo, F.J. and Muñoz-Pérez, J., 2004. Algoritmo evolutivo para el reconocimiento de funciones de base potencial, MAEB 2004, Córdoba.

Martínez-Estudillo, A.C., Hervás-Martínez, C., Martínez-Estudillo, F.J. and Ortiz-Boyer, D., 2005b. Memetic Algorithms to Product-Unit Neural Networks for Regression, 8th International Work-Conference on Artificial Neural Networks (IWANN'2005) (Computational Intelligence and Bioinspired Systems) (accepted to be included in the LNCS-Springer proceedings).

Martínez-Estudillo, A.C., Martínez-Estudillo, F.J., García-Pedrajas, N., Hervás-Martínez, C. and García-Gimeno, R.M., 2003. Modelado de sistemas de crecimiento microbiano mediante redes de regresión evolutivas, Congreso español de metaheurísticas, algoritmos evolutivos y bioinspirados.(MAEB 2003), Gijón.

Martínez-Estudillo, A.C., Martínez-Estudillo, F.J., Hervás-Martínez, C. and García-Pedrajas, N., 2005c. Model Fitting by Evolutionary Computation Using Product Units. Neural Networks, Submitted.

McCulloch, W.S. and Pitts, W., 1943. A logical calculus od the ideas immanent in nervous activity. Bulletin of Mathematical Biophysics, 5: 115-133.

Merz, P., 2001. On the performance of memetic algorithms in combinatorial optimization, 2nd workshop on memetic algorithms. GECCO 2001, San Francisco, CA.

Michalewicz, Z., 1992. Genetic Algorithms+Data Structures=Evolution Programs. Springer-Verlag, Berlin, Germany.

Michalewicz, Z., 1996. Genetic Algorithms+Data Structures=Evolution Programs (3rd edition). Springer-Verlag, Berlin, Germany.

Michalewicz, Z. and Schoenauer, M., 1996. Evolutionary algorithms for constrained parameter optimization problems. Evolutionary Computation, 4(1): 1-32.

Miller, G.F., Todd, P.M. and Hedge, S.U., 1989. Designing neural networks using genetic algorithms. In: J.D. Schaffer (Editor), Proc. 3er Int. Conf. Genetic Algorithms and Their Applications. Morgan Kaufmann, San Mateo, CA, pp. 379-384.

Minai, A.A. and Williams, R.D., 1990. Acceleration of back-propagation through learning rate and momentum adaptation., International joint conference on neural networks, pp. 676-679.

Moscato, P., 1989. On evolution, search, optimization, genetic algorithms and martial arts: towards memetic algorithms, Inst. Technol. Pasadena, CA, California.

Moscato, P., 1992. A memetic approach for the travelling salesman problem implementation of a computational ecology for combinatorial optimization on message-passing systems. In: E.O. M. Valero, M. Jane, J.L. Larriba, B. Suarez (Editor), Parallel Computing and Transputer Applications. IOS Press, Amsterdam, The Netherlands, pp. 176-177.

Moscato, P., 1999. A gentle introduction to memetic algorithms. In: G.K. F. Glower (Editor), Handbook of Metaheuristics. Kluwer, pp. 1-56.

Myers, R.H. and Montgomery, D.C., 2002. Surface Methodology: process and product optimization using designed experiments. John Wiley and Sons.

Nadeau, C. and Bengio, Y., 2003. Inference for the generalization error. Machine Learning, 52: 239-281.

Nelder, J.A. and Mead, R., 1965. A simplex method for function minimization. Computer Journal, 7: 308-313.

Oost, E., ten Hagen, S. and Schulze, F., 2002. Extracting multivariate power functions from complex data sets, Proccedings of the 14th Dutch-Belgian Artificial Inteligence Conference, BNAIC'02, Leuven, Belgium.

Otten, R.H.J.M. and van Ginneken, L.P.P.P., 1989. The annealing algorithm. Ed. Kluwer, Boston, MA.

Pao, Y.H., 1989. Adaptative Pattern Recognition and Neural Networks. Reading, Mass. Addison-Wesley Publishing Co.

Pinkus, A., 1999. Aproximation theory of the MLP model in neural networks. Acta Numerata: 143 -195.

Rissanen, J. (Editor), 1989. Stochastic complexity in statistical inquiry. World Scientific.

Rivals, I. and Personnaz, L., 2003. Neural-Network construction and selection in nonlinear modeling. IEEE Trans. on Neural. Networks, 14, 4: 804-819.

Rudin, W., 1966. Real and Complex Análisis. McGraw-Hill, New York.

Rudin, W., 1973. Functional Analisis. McGraw-Hill, New York.

Rudolph, G., 1994. Convergence Properties of canonical genetic algorithms. IEEE Transactions on Neural Networks, 5(1): 96-101.

Rumelhart, D.E., Hinton, G.E. and McClelland, J.L., 1986. Learning internal representations by error propagation. In: D.E.R.a.J.L. McClelland (Editor), Parallel distributed processing: Explorations in the microstructure of cognition. Cambridge, MA: Mit Press, pp. 318-362.

Saito, K. and Nakano, R., 1997a. Numeric law discovery using neural networks, Proc. of the 4th International Conference on Neural Information Processing (ICONIP97), pp. 843-846.

Saito, K. and Nakano, R., 1997b. Law discovery using neural networks, Proc. of the 15th International Joint Conference on Artificial Intelligence (IJCAI97), pp. 1078-1083.

Saito, K. and Nakano, R., 1997c. Partial BFGS update and calculating optimal step-length for three-layer neural networks. Neural Computation, 9: 123-141.

Saito, K. and Nakano, R., 1997d. MDL regularizer: a new regularizer based on MDL principle, Proc. of International Conference on Neural Networks (ICNN97), pp. 1833-1838.

Saito, K. and Nakano, R., 2002. Extracting Regression Rules From Neural Networks. Neural Networks, 15: 1279-1288.

Saito, K. et al., 2000. Law discovery from financial data using neural networks, Proc of IEEE/IAFE/INFORMS Conference on Computational Intelligence for Financial Engineering (CIFEr00), pp. 209-212.

Salomon, R., 1998. Evolutionary Algorithms and Gradient Search: Similarities and Diferences. IEEE Transactions on evolutionary computations, 2(2).

Schioler, H. and Hartmann, U., 1992. Mapping neural networks derived from the the Parzen window estimator. Neural Networks, 5: 903-909.

Schmitt, M., 2001. On the Complexity of Computing and Learning with Multiplicative Neural Networks. Neural Computation, 14: 241-301.

Schwefel, H.P., 1981. Numerical Optimization of Computer Models. Wiley, Chichester, U.K.

Schwefel, H.P., 1995. Evolution and Optimum Seeking. Wiley, New York.

Setiono, R., Leow, W.K. and Zurada, J.M., 2002. Extraction of rules from artificial neural networks for nonlinear regression. IEEE Transactions on Neural Networks, 13(3): 564-577.

Siu, K.Y., Roychowdhury, T. and Kailath, V., 1995. Discrete neural computation: A theoretical foundation. Prentice Hall.

Skinner, A.J. and Brougthon, J.Q., 1995. Neural networks in computational materials science: Training algorithms. Modeling and Simulation in Materials Science and Engineering, 3(3): 371-390.

Snyman, J., 1982. A New Dynamic Method for Unconstrained Minimization. Applied Mathematical Modelling, 6: 449-462.

Snyman, J., 1983. An Improved Version of the Original LeapFrog Dyamic Method for Unconstrained Minimization:LFOP1(b). Applied Mathematical Modelling, 7: 216-218.

Souto, M.C.P., Yamazaki, A. and Ludermir, T.B., 2002. Optimization of neural network weigths and architectures for odor recognition using simulated annealing, Proceedings of the 2002 International Joint Conference on Neural Networks, pp. 547-552.

Specht, D.F., 1991. A General Regression Neural Network. IEEE Transactions on Neural Networks, 2(6): 568-576.

SPSS, I., 1989-2001. SPSS© para Windows 11.0.1. SPSS Inc., Chicago IL.

Stinchcombe, M. and White, H., 1989. Universal approximation using feedforward networks with non-sigmoid hidden layer activation functions, Proceedings of the IEEE 1989 International Joint Conference on Neural Networks. IEEE, New York, pp. 613-618.

Stinchcombe, M. and White, H., 1990. Aproximating and learning unknown mappings using multilayer feedforward networks with bounded weights, Proceedings of the IEEE 1990 International Joint Conference on Neural Networks. IEEE, New York, pp. 7-16.

Suga, N., Olsen, J.F. and Butman, J.A., 1990. Specialized subsystems for processing biologically important complex sounds: Cross-correlation analysis for ranging in the bat's brain. In: C.S.H.L. Press (Editor), Cold Spring Harbor Symposia on Quantitative Biology, Cold Spring Harbor, NY, pp. 585-597.

Sugeno, M. and Kang, G.T., 1988. Structure Identification of fuzzy model. Fuzzy Sets and Systems, 28.

Sutton, R. and Matheus, C., 1991. Learning polynomial functions by feature construction, Proceedings of the Eighth International Machine Learning Workshop, Evanston, IL, pp. 208-211.

Tamhane, A.C. and Dunlop, D.D., 2000. Statistics and Data Analysis. Prentice Hall.

Thierens, D., 1996. Non-redundant genetic coding of neural networks, IEEE Int. Conf. Evolutionary Computation, ICEC'96, pp. 571-575.

Tomandl, D. and Schober, A., 2001. A modified general regression neural network (MGRNN) with new, efficient training algorithms as a robust ´black box´-tool for data analysis. Neural Networks, 14: 1023-1034.

Turney, P.D., 1996. Lessons from the baldwin effect. Evolutionary Computation, 4(3): 271-295.

Ulder, N.L.J., Aarts, E.H.L., Bandelt, H.J., Laarhoven, P.J.M. and Pesch, E., 1991. Genetic local search algorithms for the travelling salesman problem. In: R.M. H.P. Schwefel (Editor), Parallel Problem Solving from Nature-Proceeding of 1st Workshop, PPSN I, Lecture Notes in Computer Science. Springer, Berlin, Germany, pp. 109-116.

Van den Bergh, F. and Engelbrecht, A.P., 2001. Training Product Unit Networks using Cooperative Particle Swarm Optimisers, Proceedings of IJCNN, Washington DC, USA.

White, D. and Ligomenides, P., 1993. GANNet: A genetic algorithm for optimizing topology and weights in neural networks design, Int. Workshop Artificial Neural Networks (IWANN'93), Lecture Notes in Computer Science. Springer-Verlag, Berlin, Germany, pp. 322-327.

Whitley, D., Gordon, S. and Mathias, K., 1994. Lamarckian evolution, the Baldwin effect and function optimisation. In: H.P.S. Y. Davidor, and Manner (Editor), Parallel Problem Solving from Nature-PPSN III. Springer-Verlag, pp. 6-15.

Whitley, D., Starkweather, T. and Bogart, C., 1990. Genetics algorithms and neural networks: Optimizing connections and conectivity. Parallel Computation, 14(3): 347-361.

Wilcoxon, F., Mosteller, F. and Rourke, R.E.K., 1973. Study statistics? Nonparametric and order statistics. Addison Wesley, Reading, Mass,.

Wolpert, D. and Macready, W., 1996. No Free Lunch Theorem for Search. SFI-TR-95-02-010, Santa Fe Institute.

Yao, X., 1991. Simulated annealing with extended neighbourhood. International Journal of Computer Math., 40: 169-189.

Yao, X., 1993. An empirical study of genetics operators in genetic operators in genetic algorithms. Microprocesing and Microprogramming, 38: 707-714.

Yao, X., 1993a. A review of evolutionary artificial neural networks. International Journal of Intelligent Systems, 8(4): 539-568.

Yao, X., 1997. Global Optimization by Evolutionary Algoritms. IEEE, 0: 282-291.

Yao, X., 1999. Evolving artificial neural network. Procedings of the IEEE, 9 (87): 1423-1447.

Yao, X., 2002. Evolutionary computation: A gentle introduction. Chapter 2. In: M.M. R. Sarker, and X. Yao (Editor), Evolutionary Optimization. Kluwer Academic Publishers, pp. 27-53.

Yao, X. and Liu, Y., 1997. A new evolutionary system for evolving artificial neural networks. IEEE Transactions on Neural Networks, 8 (3): 694-713.