PONTIFICIA UNIVERSIDAD CATÓLICA DE VALPARAÍSO
FACULTAD DE INGENIERÍA
ESCUELA DE INGENIERÍA INFORMÁTICA

REDES B-SPLINE CON APRENDIZAJE PSO PARA LA ESTIMACIÓN DE COSTO

LENIN ANDRÉS GONZÁLEZ SILVA

INFORME FINAL DEL PROYECTO
PARA OPTAR AL TÍTULO PROFESIONAL DE
INGENIERO CIVIL EN INFORMÁTICA

Abril 2013

PONTIFICIA UNIVERSIDAD CATÓLICA DE VALPARAÍSO
FACULTAD DE INGENIERÍA
ESCUELA DE INGENIERÍA INFORMÁTICA

REDES B-SPLINE CON APRENDIZAJE PSO PARA LA ESTIMACIÓN DE COSTO

LENIN ANDRÉS GONZÁLEZ SILVA

Profesor Guía: **Dr. Nibaldo Rodríguez Agurto**

Profesor Correferente: **Mg. José Rubio Leon**

Carrera: **Ingeniería Civil en Informática.**

Abril 2013

Dedicatoria

Dedicado a mis padres por su entrega de valores, sabiduría, amor incondicional e inspiración día a día y a Dios por su protección incondicional.

Agradecimientos

A la Pontificia Universidad Católica de Valparaíso por sus valores y enseñanzas entregadas durante mi período de formación profesional, y a mi profesor guía por su disponibilidad permanente en la entrega de conocimiento y consejos para el buen desarrollo de esta memoria de título.

Índice

Resumen

En la presente memoria de título se presenta la propuesta de un modelo neuronal evolutivo que permita realizar una temprana estimación de costos para la manufacturación de tuberías. Para ello, se presenta un estudio acabado de las Redes Neuronales Artificiales y el algoritmo evolutivo Optimización por Enjambre de Partículas, las cuales han demostrado ser herramientas altamente confiables en los diferentes campos en los que fueron ocupados. Particularmente se hace énfasis en la arquitectura de una Red Neuronal B-Spline, la cual es una herramienta que permite interpolar con polinomios locales de bajo orden evitando así el fenómeno Runge. Una Red B-Spline usualmente es entrenada con métodos basados en el gradiente descendente, el cual puede llevar a caer en mínimos locales durante la fase de aprendizaje. De acuerdo a lo anterior, se proponen algoritmos estocásticos evolutivos: PSO, QPSO, PSO con Velocidad Mínima y PSO con Velocidad Mínima Modificado, los cuales se encargaron de calibrar los parámetros de la Red Neuronal B-Spline. Los resultados numéricos presentados indican que el estimador propuesto tiene un buen comportamiento y además se logró obtener 4 modelos en total con un mape entre un 10 y 14%.

Palabras-claves: Estimación de Costos, Red Neuronal Artificial, B-Spline, Optimización por Enjambre de Partículas.

Abstract

This dissertation presents the proposal of an evolutionary neuronal model, enabling the early cost estimates for the manufacture of pipes. Here we present a complete study of Artificial Neuronal Networks and evolutionary algorithm Particle Swarm Optimization which have proven to be highly reliable tools in different fields that were occupied. It particularly emphasizes the architecture of a B-Spline Neuronal Network, which is a tool that allows interpolating to lower-order local polynomials avoiding the Runge phenomenon. A B-spline network is usually trained with methods based on gradient descent, which can lead to falling into local minima during the learning phase. For this, next stochastic evolutionary algorithms are proposed: PSO, QPSO, PSO with Minimum Speed and PSO with Modified Minimum Speed which were responsible for calibrating the parameters of the B-Spline Neuronal Network. The numerical results presented indicate that the proposed estimator has a good behavior and also managed to get 4 models in total, resulting in a mape range between 10 and 14%.

Keywords: Cost Estimation, Artificial Neuronal Networks, B-Spline, Particle Swarm Optimization

Lista de Figuras

Lista de Tablas

Lista de Abreviaturas o Siglas

IC : Ingeniería Concurrente
RNA : Red Neuronal Artificial
RNAs : Redes Neuronales Artificiales
PSO : Particle Swarm Optimization
LM : Levenberg Marquardt
DE : Operador Evolutivo Diferencial

1. Introducción

En un mercado tan competitivo como el actual resulta imperante contar con ventajas y/o habilidades que permitan utilizar efectivamente los recursos y disminuir los costos a lo mínimo posible para así maximizar las utilidades. Desde esa óptica, es decir, mejorar la competitividad empresarial, surgió el concepto de desarrollo de Ingeniería Concurrente (IC), teniendo en cuenta un período de desarrollo de productos reducido y a un bajo costo, el cual debería ser introducido a la fase de diseño del producto [1].

La IC [2] así como muchas otras propuestas [3], argumentan que los factores claves que afectan al diseño del producto durante su ciclo de vida deberían ser considerados en las fases iniciales (o conceptuales) para así reducir el ciclo de vida del costo del producto. Además, se debe considerar en esta etapa las tareas que la siguen, evitando conflictos entre el diseño y sus etapas posteriores, aumentos de costos y mal uso de los recursos.

Para aprovechar los recursos y disminuir los costos de producción se hace necesario analizar y **estimar costos**, y además controlarlos durante el período de desarrollo inicial del producto. En trabajos previos [4], [5], [6], [7], se ha expuesto que el costo del producto en las fases de diseño representa solo el 6 % del costo total, pero **determina entre un 70-80%** del costo total del producto (ver figura 1-1) y el 80% de su calidad. De aquí se puede desprender que el mayor potencial de reducción de costos está en las fases iniciales de diseño, por lo que dedicar un mayor esfuerzo a los costos de diseño es una medida razonable y necesaria para la optimización del costo del producto.

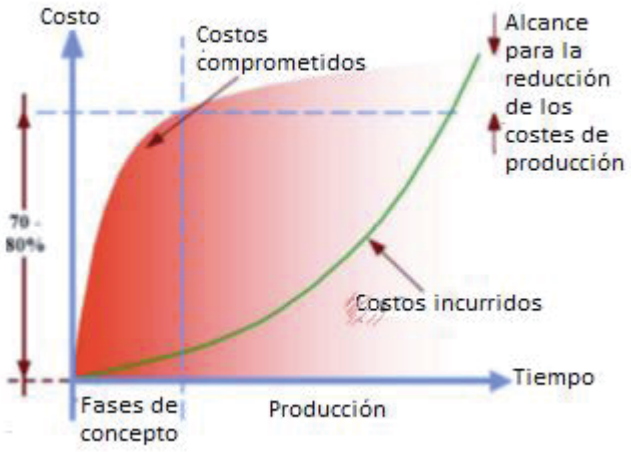

Figura 1-1 Variación del costo en las fases del proyecto [8].

Una aproximación temprana del costo en función de un conjunto de características puede ayudar a los diseñadores en decisiones tales como la selección de material, en procesos de producción y principalmente en las características morfológicas del producto [9]. Sin embargo, la información en la etapa de diseño inicial es incompleta, y los elementos de costo del interior del producto no están expresamente definidos en términos de requisitos de

calidad y tecnología factible para las funciones del producto, lo que dificulta analizar y estimar el costo del producto con precisión.

Tomar una decisión equivocada en la fase de diseño es extremadamente costoso ya que tiene un gran impacto en las etapas posteriores de desarrollo del proceso, una sobre-estimación se traduciría en el derroche de recursos, y una subestimación daría lugar a una calidad inferior. Además, dado que la estimación de costo es el punto de partida del proceso de gestión, **influye en decisiones de hacer o no hacer** (o desarrollar) un nuevo producto manufacturado. Dada esta problemática los encargados de estimar costos **necesitan una aproximación real** del verdadero costo de producir algún producto.

En la literatura se exponen una serie de técnicas y métodos para la temprana estimación de costo. Dentro de los métodos (o técnicas) analizados se puede encontrar la estimación paramétrica, **la inteligencia artificial**, técnicas basadas en analogías y técnicas de gestión de costo [8]. Pudiendo clasificarlas en técnicas cuantitativas y cualitativas [4]. No obstante, el análisis convencional de costo del producto se basa a menudo en la estimación directa de la experiencia de los diseñadores, un cierto nivel de análisis cualitativo, y la determinación subjetiva, sin un conjunto de métodos eficaces y racionales para reducir el error de estimación de costos. [9]

Existe un número incipiente de casos que utilizan Inteligencia Artificial [9]. Estas técnicas constituyen la última generación de herramientas para la estimación del costo de productos manufacturados. La Inteligencia Artificial es una rama de la ciencia de la computación que estudia los requerimientos necesarios para tareas tales como la percepción, el razonamiento, y el aprendizaje y desarrolla sistemas para realizar dichas tareas. Por ende, abarca un amplio campo de investigación donde los investigadores pueden dirigirse a un amplia gama de problemas, usar una variedad de métodos, y buscar un amplio espectro de objetivos científicos [10].

El concepto básico detrás de la aplicación de la Inteligencia Artificial en la estimación de costos es, la de imitar el comportamiento de los expertos humanos para determinar las principales variables que rigen (y en qué medida lo hacen) el costo final de un producto manufacturado [9]. En trabajos previos [11], [12], se ha demostrado que la **calidad de las predicciones** realizadas por sistemas inteligentes es comparable a la calidad garantizada por juicios expertos.

Dentro de la amplia gama de técnicas de la Inteligencia Artificial las **Redes Neuronales Artificiales (RNAs)** han sido lo más explorado en la investigación sobre la estimación de costos con resultados bastante competitivos [13], [14]. Las RNAs tratan de adaptarse a las curvas a través de datos sin proporcionar una función predeterminada con conexión de parámetros permitiendo resolver problemas no lineales. Por lo tanto, son capaces de detectar relaciones funcionales ocultas entre los atributos del producto y el costo, es decir, las relaciones desconocidas y de interés para el ingeniero de costos y además permiten capturar el conocimiento relativo a los productos producidos para su adaptación a las nuevas situaciones, es decir, los nuevos productos en fase de desarrollo.

En las últimas 2 décadas se han propuesto **técnicas híbridas** de Inteligencia Artificial para resolver problemas de estimación de Costo. La idea básica es incorporar a la Red Neuronal una Metaheurística para su aprendizaje modificando así la estructura de la misma. Dentro de las propuestas se puede encontrar Redes Neuronales con un Algoritmo Genético como también **Redes Neuronales con Optimización por Enjambre de Partículas (PSO)** obteniendo buenos resultados [2].

En este proyecto se busca desarrollar y evaluar un modelo de estimación de costo para la fabricación de productos manufacturados, **precisamente en tuberías**, estas son ocupadas como medios de transporte para diversos fluidos, gases, compuestos acuosos, polvos y masas [9]. Para ello se utilizará Redes Neuronales B-Spline con un algoritmo de aprendizaje basado en PSO [42].

.

1.1 Objetivos

1.1.1 Objetivo General

Desarrollar y Evaluar un modelo neuronal para la estimación de costo en tuberías utilizando Redes B-Spline FeedForward con un Algoritmo de Aprendizaje basado en Optimización por Enjambre de Partículas.

1.1.2 Objetivos Específicos

1. Investigar y Comprender el estado de arte de las Redes Neuronales con sus respectivas arquitecturas.
2. Investigar y Comprender el estado de arte de la Optimización por Enjambre de Partículas (PSO) y sus variaciones.
3. Diseñar y Estimar los parámetros de un modelo Neuronal Artificial B-Spline con algoritmo de aprendizaje PSO para la estimación de costo en tuberías.
4. Implementar y Evaluar el modelo Neuronal Artificial B-Spline propuesto.

1.1.3 Organización del texto

El presente informe comienza con una descripción y explicación sobre las Redes Neuronales Artificiales presentando sus características más esenciales y profundizando en la función de activación B-Spline. Luego en el capítulo 3, se presenta al algoritmo Optimización por Enjambre de Partículas el cual es un algoritmo estocástico evolutivo, este será utilizado para calibrar la red neuronal y a su vez se presentan algunas variantes del mismo. En el capítulo 4 se presenta la arquitectura general y específica del estimador de costo propuesto. El capítulo 5 contiene la metodología utilizada en el estimador de costo junto a los resultados asociados de la etapa de implementación, entregando modelos capaces de realizar tempranas estimaciones de costo en elementos de tuberías. Finalmente la conclusión y el trabajo futuro son presentados en el capítulo 6.

2. Redes Neuronales

Tradicionalmente en la ciencia de la computación la manera de confrontar los diversos problemas era generar un modelo tomando en cuenta algunos aspectos, reglas matemáticas o lógicas y proponer en base a un conjunto de sentencias iterativas con mecanismos de selección o decisión una solución. Inmediatamente surgen preguntas tales como, ¿Qué pasaría si existen tantos casos que el computador se demore siglos en responder? ¿Qué pasaría si no existe una regla o función matemática para responder al problema? ¿Qué pasa cuando el problema es del tipo no Lineal? ¿Qué pasa con el modelo si tanto los datos como el entorno cambian dinámicamente?

Para responder a esta y a otras interrogantes a lo largo de la historia se han generado diversas formas de presentar soluciones, que en cierto modo han conformado a lo que hoy se conoce como Inteligencia Artificial. Cada solución para cierto tipo de problemas tiene su propio enfoque, donde lo primordial, es ser capaz de representar el conocimiento mediante alguna abstracción precisa y no ambigua enfatizando ciertos aspectos de la resolución del problema. La clave está en presentar alguna forma de razonar. Diversas aproximaciones se han derivado como, Arboles de decisiones, Operadores de estado, Lógica Difusa, Programación con Restricciones, Metaheurísticas, Algoritmos Genéticos, Redes Neuronales, etc. Cada una con una fuente inspiradora o perspectiva para abordarlo. En particular las Redes Neuronales tienen una forma de abordar los problemas basados en el sistema nervioso biológico del Cerebro Humano. Teniendo en cuenta esta motivación se pueden generar las siguientes interrogantes: ¿Si el computador es capaz de procesar sin "errores" diversos cálculos, como es posible que para tareas a priori fáciles para el humano, tales como reconocimiento, percepción u otras, sea tan complejo de resolver para el Computador? ¿Cómo puede ser que el enfoque tradicional no dé una respuesta en tiempo record siendo que las neuronas son 6 ó 5 órdenes de magnitud más lentas que una compuerta lógica de silicio [15]? El punto de partida para estas y otras interrogantes similares está en entender cómo se estructura el Cerebro Humano y como se lleva a cabo su funcionamiento, para luego tratar de representar dicho funcionamiento en los computadores.

2.1 Fundamentos Biológicos de las Redes Neuronales Artificiales.

El Cerebro Humano es un sistema de procesamiento de información extremadamente complejo, su unidad fundamental de procesamiento es la neurona, donde el cerebro está compuesto por un elevado número de neuronas interconectadas y funcionando en paralelo. Una neurona típica posee el aspecto y las partes que se muestran en la figura 2-1.

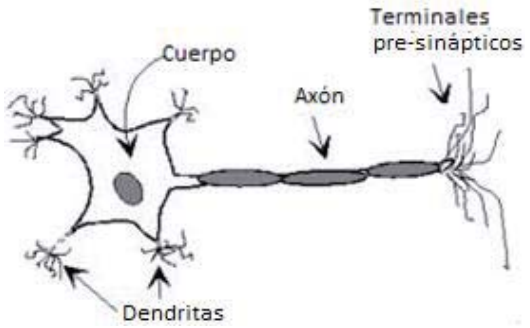

Figura 2-1 Modelo de Una Neurona

La mayoría de las neuronas codifican sus salidas como una serie de breves pulsos periódicos, llamados potenciales de acción, que se originan cercanos al Cuerpo de la célula y se propagan a través del Axón. Luego, este pulso llega a la Sinapsis y de ahí a las Dendritas de la neurona siguiente [15]. Las Dendritas son las encargadas de recibir los impulsos eléctricos provenientes de otra neurona o desde el exterior, es decir, son las zonas receptivas de la neurona.

La Sinapsis consiste en la unión del Axón de la neurona emisora con las Dendrita de la neurona receptora. Por lo que el Axón se puede ver como una línea de transmisión encargada de facilitar el proceso de la Sinapsis, sin embargo, antes deben enviar una señal a los terminales pre-sinápticos para que puedan "llamar" químicamente a las neuronas que van a recibir este mensaje [16] [17].

Una vez entendida la estructura del Cerebro Humano a grandes rasgos, es fácil ver que este es un sistema **altamente complejo, no-lineal y paralelo**. Su gran impacto está en que puede realizar **muchas operaciones simultáneamente** a diferencia de los computadores comunes que son de tipo secuencial, es decir, solo una operación a la vez. **Es tolerante a fallos** y el **conocimiento se encuentra distribuido** a lo largo de su Red Neuronal.

Dentro de los comportamientos que podrían apreciarse en las neuronas, hay 3 que son de vital importancia [15] para tomar en cuenta en una Red Neuronal Artificial (RNA):

1. El impulso que llega a una Sinapsis y el que sale de ella no son iguales en general. Puede variar durante el proceso de Aprendizaje.
2. En el Cuerpo o soma se suman las entradas de todas las Dendritas. Si estas sobrepasan un cierto umbral, entonces se transmitirá un pulso a lo largo del Axón, en caso contrario no se transmitirá. Después de transmitir un impulso, la neurona no puede transmitir durante un tiempo de entre 0,5 ms a 2 ms. A este tiempo se le llama período refractario.
3. La plasticidad inherente de las neuronas. Son capaces de cambiar dinámicamente junto con el medio. La plasticidad se percibe también en la capacidad de responder de forma correcta frente a un estímulo nunca antes recibido.

En base a esos 3 comportamientos, se construirán en general, los modelos de Redes Neuronales Artificiales (RNAs). Por lo **tanto se espera que una RNA** al inspirarse en el modelo biológico cerebral **herede la tolerancia a fallos, adquisición de conocimientos,**

plasticidad, adaptabilidad, posibilidad de modelar comportamientos no lineales y un alto grado de conectividad entre sus neuronas.

2.2 Antecedentes Históricos

En 1936 Alan Turing fue el primero en estudiar y plantear el comportamiento cerebral, sin embargo, **Warren McCulloch** y **Walter Pitts** fueron los primeros científicos que concibieron los fundamentos de la computación neuronal y propusieron un modelo matemático para las Redes Neuronales Artificiales [18]. Ellos modelaron una red neuronal simple mediante circuitos eléctricos.

En 1949 **Donal Hebb** fue el primero en explicar los procesos del aprendizaje desde un punto de vista psicológico, desarrollando una regla matemática de cómo el aprendizaje ocurría descrita en su libro *Organization of Behavior* [19].

En 1956 se organizó la primera conferencia sobre inteligencia Artificial (IA) y fue aquí donde oficialmente se acuño el término. En esta conferencia se discutió el uso potencial de los computadores para simular todos los aspectos del aprendizaje o cualquier otra característica de la inteligencia y se presentó la primera simulación de una red neuronal, aunque todavía no se sabían interpretar los datos resultantes.

En 1957 **Frank Rosemblatt** logró generalizar el modelo entregado por McCulloch y Pitts, añadiendo una técnica de aprendizaje [20], a este modelo lo llamo **Perceptron.**

En 1959, **Bernard Widrow** publica una teoría sobre la adaptación neuronal y unos modelos inspirados en esa teoría [21], el Adaline (Adaptative Linear neuron) y el Madaline (Multiple Adaline). Estos modelos fueron usados en numerosas aplicaciones y permitieron usar, por primera vez, una red neuronal en un problema importante del mundo real: filtros adaptativos para eliminar ecos en las líneas telefónicas.

En 1969, **Minsky** y **Papert** realizan una crítica al Perceptrón [22], revelando serias limitaciones, su incapacidad de resolver problemas que no eran linealmente separables como la función XOR. Esto provocó una gran caída de las investigaciones.

En los años 70, **Anderson** estudia y desarrolla modelos de memorias asociativas. Destaca el modelo brain-state-in-a-box [23] (BSB). **Kohonen** continúa el trabajo de Anderson y desarrolla modelos de aprendizaje competitivos basados en el principio de inhibición lateral. Su principal aporte consiste en un procedimiento para conseguir que unidades físicamente adyacentes aprendieran a representar patrones de entrada similares [24].

Grossberg realizó un importante trabajo teórico matemático tratando de basarse en principios fisiológicos; aportó importantes innovaciones con su modelo ART (Adaptative Resonance Theory) [25] y, junto a **Cohen**, elabora un importante teorema sobre la estabilidad de las redes recurrentes en términos de una función de energía.

En la década de los 80 se produce el renacimiento del interés por el campo gracias al trabajo del grupo PDP (Parallel Distributed Processing) creado por **Rumelhart**, **McClelland** & **Hinton** [26]. Como resultado de los trabajos de este grupo salieron los manuales con más influencia desde la crítica de Minsky y Papert.

En 1982, **Hopfield** elabora un modelo muy ilustrativo sobre los mecanismos de almacenamiento y recuperación de la memoria [27].

En 1986, **Jordan** propone un nuevo modelo considerado dentro de las redes neuronales parcialmente recurrentes, denominado Red de Jordan [28]. Luego en 1990, se publica una nueva red llamada Red de Elman [29].

Finalmente, en las últimas décadas se mantiene un continuo interés en la investigación de las Redes Neuronales Artificiales y sus posibles aplicaciones, lo que ha traído números avances no solo en informática sino también en los diversos campos de la ingeniería, medicina, economía y otras ciencias.

2.3 Redes Neuronales Artificiales

Una Red Neuronal Artificial (RNA) se puede definir como un modelo computacional inspirado en el sistema nervioso biológico, compuesto por un gran número de elementos interconectados paralelamente denominados neuronas, encargados de procesar información para producir un estímulo de salida. El comportamiento inteligente son una propiedad emergente del gran número de unidades no un resultado de reglas simbólicas o algoritmos.

Las características de una RNA como se mencionó en el punto 2.1 son heredadas de su símil biológico, es decir una RNA tiene **capacidad de aprender, clasificar, almacenar información, interpolación, adaptación, resistencia a fallas** (uno de sus elementos puede fallar sin llegar a afectar significativamente la respuesta total del sistema) y ser un **sistema distribuido no lineal**. Un posible esquema se puede apreciar en la figura 2-2.

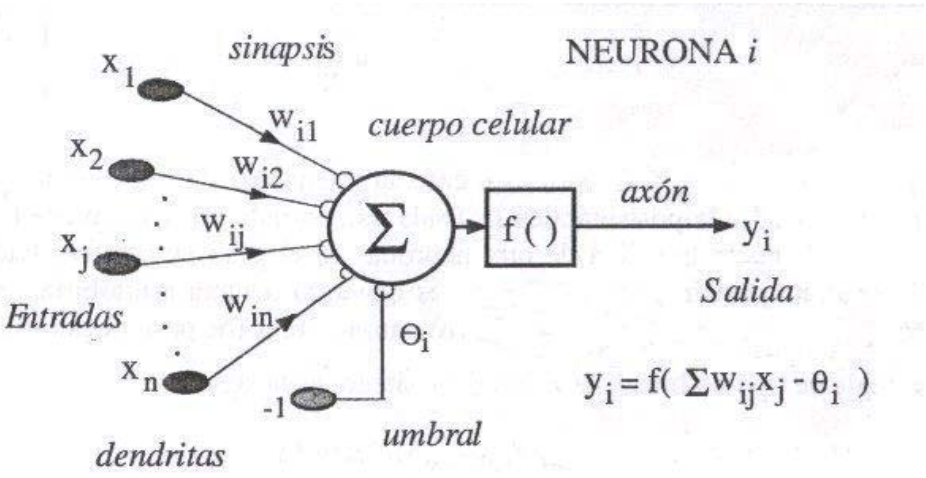

Figura 2-2 Modelo de McCulloch y Pitts.

El primer desarrollo de un modelo Neuronal propiamente tal fue en 1943, por McCulloch and Pitts [18], este modelo contenía elementos fundamentales que siguen considerándose en la actualidad, estos son:

- Un vector de Entradas $\vec{X} = (x_1, x_2, ...x_j, ...x_n)$
- Un vector de pesos $\vec{W} = (w_1, w_2, ...w_j, ...w_n)$ que define la importancia relativa de cada entrada.
- Una función de propagación $F(\vec{X}, \vec{W}) = \vec{X} \bullet \vec{W} = \sum_{i=1}^{n} x_i w_{ji}$ o $\sum_{i=1}^{n} x_i w_{ji} - \theta$ en caso de incluir un Bias para evitar salidas no deseadas.
- Una función de Activación $f = \phi(F(\vec{X}, \vec{W}))$. Que transforma el resultado de la función de propagación en una salida real de la neurona. Existen una amplia variedad de funciones de activación, McCulloch y Pitts usaron la función escalón.
- Una salida.

De lo anterior, se desprende que se generan un conjunto de conexiones, cada una de ellas caracterizadas por su propio peso. Una señal de entrada x_i tras pasar la conexión, se habrá convertido en una señal $x_i w_{ji}$. A mayor neuronas conectadas se pueden formar 1 o más capas. Por otra parte, una vez que el resultado de la función de propagación es transformado por la función de activación, este nuevo resultado puede ser comparado con algún valor umbral para determinar la salida de la neurona. Si la suma es mayor que el valor umbral, la neurona generara una señal en caso contrario no generara señal.

El modelo de Mcculloch y Pitts podía calcular funciones lógicas (AND, OR, NOT) para poder construir máquinas de estados finitos. Tomando en consideración esta arquitectura Frank Rosenblatt desarrolla un proyecto para identificar patrones ópticos, donde a la RNA se llamó Perceptron de una capa o Perceptron Simple. Este modelo recibió críticas por no poder resolver problemas que no sean linealmente separables como la función XOR. Esto implicó un nuevo desarrollo, definir nuevas arquitecturas de RNAs.

2.4 Arquitectura de Redes Neuronales Artificiales

Teniendo en cuenta el problema que tiene el Perceptron, es decir, imposibilidad de resolver problemas no separables linealmente, se puede proponer de manera intuitiva un modelo de RNA con múltiples capas, como se aprecia en la figura 2-3.

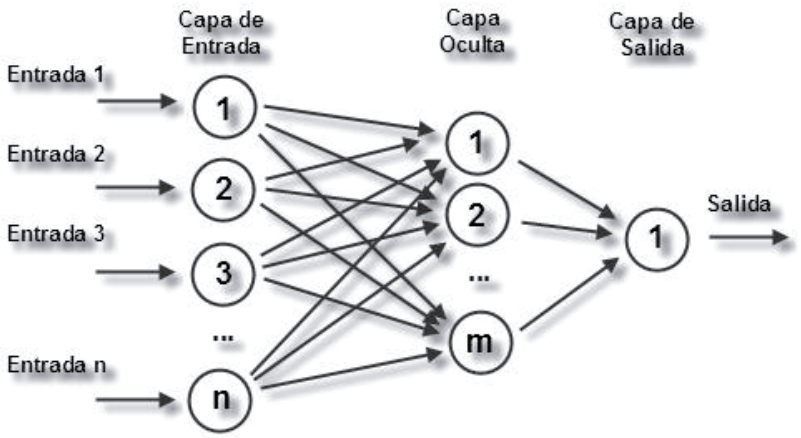

Figura 2-3 Perceptrón MultiCapa

Con este tipo de arquitecturas ya es posible resolver problemas no lineales. Sin embargo se pueden hacer **infinitas variaciones**, es decir, se puede variar el **número de capas** a utilizar, el **flujo de las señales**, **función de activación**, **el mecanismo de aprendizaje**. Una combinación de estos elementos dará la arquitectura completa de la red.

2.4.1 Arquitectura de Capas

Cuando se trabaja con grandes cantidades de neuronas, es natural ordenar aquellas que tiene comportamientos similares en capas, como en la figura 2-3. Inmediatamente se puede identificar la idea de paralelismo al observar las capas de las redes. Cada neurona de una capa no necesita de las demás en su misma capa para trabajar, son capaces por lo tanto de trabajar simultáneamente. [15].

En general, se hablará de **Redes Monocapa** cuando estas redes estén compuestas por una única capa de neuronas, en cambio una **Red Multicapa** son aquellas cuyas neuronas que se organizan en varias capas.

Tomando como referencia la figura 2-3, se puede clasificar las capas en:

- **Capa de Entrada**: Está formada por neuronas transparentes, es decir no realizan ningún proceso, su objetivo es recibir los estímulos provenientes del entorno y traspasar dicha información al resto de las neuronas que componen la red.
- **Capa Oculta**: Recibe los estímulos de la Capa de Entrada y tienen la función de proporcionar un mejor aprendizaje o separación de lo aprendido por categorías. Puede o no estar presente en una RNA, su incorporación básicamente depende de si es un Red Monocapa o una Red Multicapa. Por lo tanto puede haber 0,1 o más Capas Ocultas.
- **Capa de Salida**: Se encarga de proporcionar la respuesta de la RNA indicando, según su aprendizaje, si fue una respuesta correcta o incorrecta a lo aprendido.

Tanto la cantidad de neuronas en la Capa de Entrada como en la Capa de la Salida, están dada por la naturaleza del problema, sin embargo, ¿Cuántas neuronas debe tener la Capa Oculta y cuántas Capas Ocultas se deben usar? Para este dilema no existe una respuesta o

regla matemática formal, diversos autores proponen que con una Capa Oculta estaría bien y que con 2 sería suficiente. Para saber cuantas neuronas usar en cada capa oculta, se puede determinar sólo de manera empírica. Por lo que dependerá del problema y no se puede generalizar una estructura para otra familia de problemas. Este proyecto pretende lograr obtener un modelo de red artificial que sea capaz de estimar los costos de manufacturar tuberías.

2.4.2 Flujo de las Señales

Usualmente las capas están ordenas por el orden en que reciben las señales, esto es, Capa de Entrada, Capa Oculta (puede no estar) y Capa de Salida. Sin embargo no es obligación que las señales se propaguen sólo en una dirección, se identifican las siguientes tipos de redes según su flujo:

- **Redes Feedforward**: La información o señal circula en un único sentido desde la Capa de Entrada hacia la Capa de Salida (ver figura 2-3). Algunas de las más emblemáticas son, Perceptron Multicapa (MLP, del inglés MultilLayer Perceptron), Red de función base Radial (RBF, del inglés Radial Basis Functions).
- **Redes Feedback**: La información o señal se puede enviar a las capas anteriores, o a neuronas de su propia capa, o de una capa posterior, es decir, se pueden enviar la información en cualquier sentido. Son llamadas también recurrentes. En la figura 2-4 se puede apreciar un posible esquema.

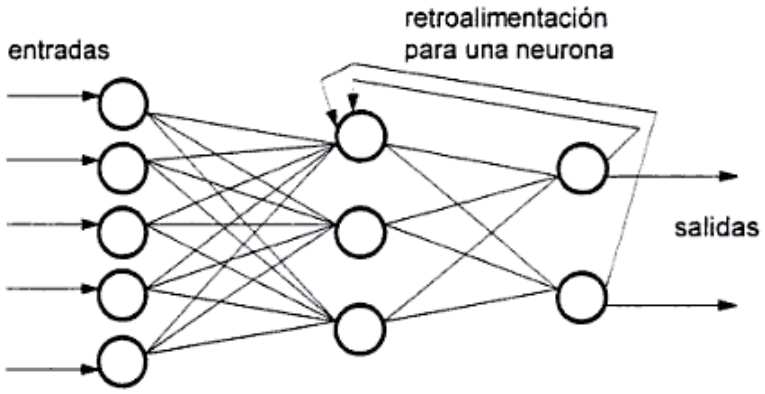

Figura 2-4 Red Recurrente o Feedback

2.4.3 Función de Activación

La función de Activación se encarga de transformar el resultado de la función de propagación en una salida real de la neurona. Existen una amplia variedad de funciones de activación, entre ellas:

- **Identidad**: $f(x) = x$, La salida calculada de la neurona será directamente lo que tenga la red como respuesta, ideal para sensores. Este tipo de función se utiliza en redes de baja complejidad, como el modelo Adaline [17].

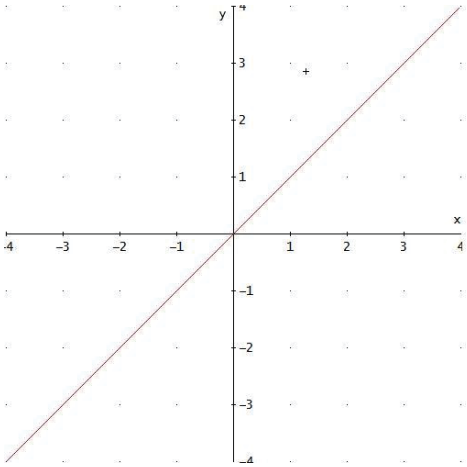

Figura 2-5 Función Identidad

- **Escalón Unitario**: $f(x) = \begin{cases} 1 & x \geq 0 \\ 0 & x < 0 \end{cases}$, Se puede comparar con un control de todo o nada, es decir, si la neurona esta inactiva significa que no contribuye con la respuesta de la red, sino implica que tiene 100% de influencia en la salida de la red.

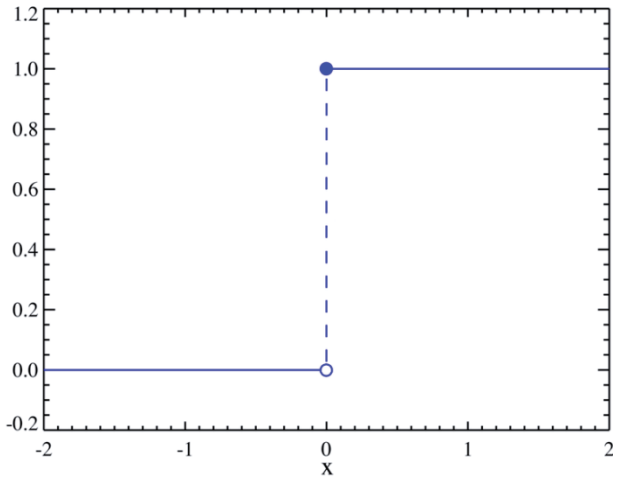

Figura 2-6 Función Escalón Unitario

- **Escalón Unitario Bipolar**: Similar que la escalón unitario con la diferencia que para valores menores que 0 toma el valor -1.
- **Sigmoidal**: $f(x) = \frac{1}{1+e^{-\theta x}}$ El valor θ indica la pendiente de la curva. Esta es una de las funciones de activación más utilizada. Básicamente permite describir procesos naturales donde hay una progresión temporal desde unos niveles bajo de inicio hasta acercarse a un clímax transcurrido un cierto tiempo; la transición se produce en una

región caracterizada por una fuerte aceleración intermedia. Dentro de las ventajas más notables es ser una función no lineal acotable, diferenciable y su derivada es simple, es decir, $f'(x) = f(x)(1 - f(x))$

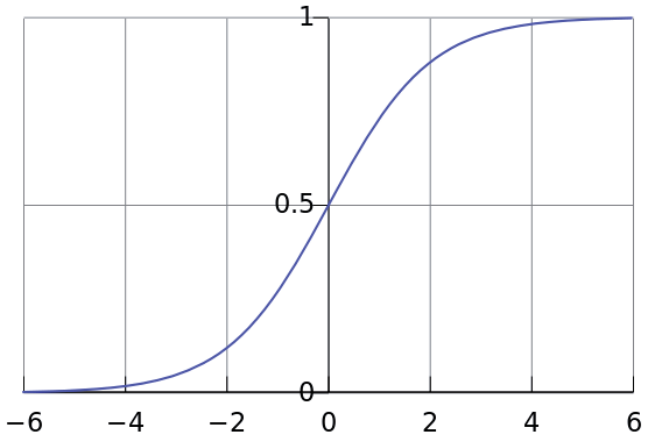

Figura 2-7 Función Sigmoidal

- **Tangencial**: $f(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$, El comportamiento de esta función es similar al de la Sigmoidal, sin embargo no cuenta con un θ para controlar la pendiente de la función.

La función de **activación** a utilizar es **responsabilidad del creador** de la Red Neuronal, una de las características determinantes en su elección es si la salida tomara valores binarios o valores continuos comprendidos entre un rango. En este trabajo se utilizará la función de activación llamada **B-Spline** descrita a continuación.

2.4.3.1 B-Spline

Las funciones polinomiales se caracterizan por tener todas las derivadas en cualquier punto de soporte; no obstante, cuando ciertas funciones no poseen un **alto grado de suavidad** en determinados puntos, el ajuste debido a estas funciones de polinomios no siempre será satisfactorio en estos tramos. Para hacer frente a esta situación, se propone el **ajuste de polinomios de bajo orden localmente**, con discontinuidades en ciertos puntos (knots), método conocido como Spline [30]. Un Spline es una curva diferenciable definida en porciones (tramos o trozos) mediante polinomios. Los Splines se utilizan para aproximar formas complicadas [31].

Una de las desventajas del método es su sensibilidad al número y ubicación de los nodos, motivo por el cual han sido propuestos distintos métodos para su selección. El proceso de suavizamiento a través de este método está dado por [30]:

$$\sum_{i=1}^{n} (y_i - m(x_i))^2 + \lambda \int m''(x)^2 dx$$

Donde λ es una constante especificada de suavizamiento.

Sea C una curva parametrizada $\vec{\phi}(t)$ definida en un intervalo real [*a, b*]. Se dice que C es suave si se cumplen simultáneamente las siguientes condiciones [35]:

1. $\vec{\phi}(t)$ es una función vectorial de clase C^1 en [*a, b*].
2. $\vec{\phi}'(t) \neq \vec{0} \ \forall t \in [a,b]$.

Si existe un número finito de puntos $t_1, t_2, ..., t_n$ en el intervalo [*a, b*] en los que se verifica que la derivada no existe, es discontinua o $\vec{\phi}(t) = \vec{0}$ se dice que la curva C es suave a trozos. Definiendo $t_0 = a$, $t_n = b$ y C_i como el arco de la curva caracterizado por:

$C_i : \vec{\varphi} : [t_{i-1}, t_i] \subset R \rightarrow R^q$, con $i = 1, 2, ..., n$

La curva suave a trozos se puede expresar como una unión de los arcos C_i, es decir:

$$C = \bigcup_{i=1}^{n} C_i$$

En la figura 2-8 se puede ver una curva unida por 3 trozos *C1, C2* y *C3* respectivamente.

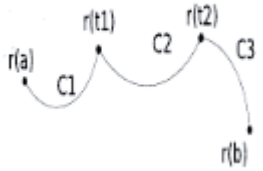

Figura 2-8 Curva Suave a trozos

Un teorema fundamental establece que cada función Spline de un determinado **grado, suavidad y partición del dominio**, se puede representar como una **combinación lineal de B-Splines del mismo grado y suavidad, y sobre la misma partición**. Por lo tanto una B-Spline no es más que una función Spline [32].

Cabe resaltar que B-Spline es una generalización de una curva de Bézier, que puede evitar el fenómeno Runge (oscilaciones en los extremos) sin **necesidad de aumentar** el grado de la B-Spline. Suponer que se quiere construir una curva que se ajuste a la función $f(x) = \frac{1}{1+25x^2}$ de la figura 2-9 (curva roja). Se selecciona una serie de puntos y los interpolamos mediante un polinomio de grado 5 (curva azul) y un polinomio de grado 9 (curva verde). Como se observa en la figura 2-9, cuando el orden del polinomio es grande la interpolación no es adecuada (inestabilidad en los extremos). Una alternativa para subsanar este inconveniente es utilizar funciones polinómicas a trozos B-Splines, bastaría con incrementar el número de trozos del polinomio que se usan para construir el Spline en lugar de incrementar el grado.

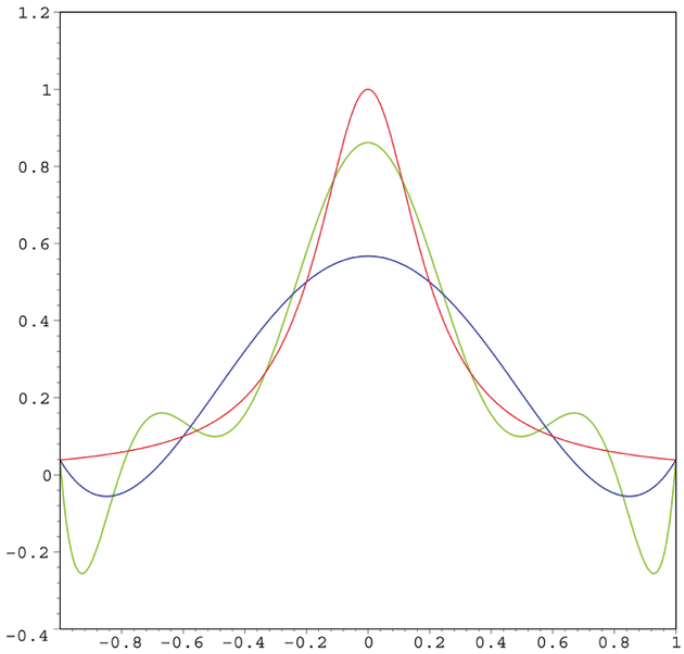

Figura 2-9 Fenómeno de Runge.

Las B-Splines tienen una naturaleza no global, es decir, son de carácter local. Cada vértice del polígono generador de la curva está asociado a una única función base. Lo cual implica, que la superficie cambia de manera predecible conforme lo indican éstos; a esto se le conoce como la propiedad de soporte local y permite que el movimiento de un punto de control sólo afecte la superficie localmente [33].

Dado el orden *k* de la curva, los vértices de control $B_0, B_1, ..., B_N$, y el vector de nodos $[x_o, x_1, ..., x_{n+k+1}]$, definimos la curva B-Spline asociada a estos datos como:

$$P(t) = \sum_{i=0}^{n} N_i^k(t) B_i$$

Donde N_i^k corresponde a las funciones base y $t \in [t_k, t_{n+1}]$. En la figura 2-10 se puede apreciar un ejemplo.

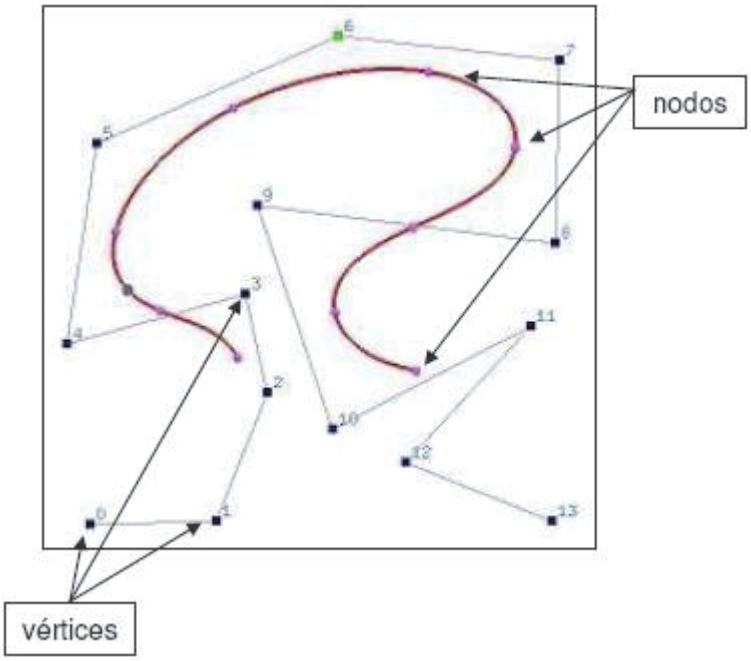

Figura 2-10 Curva B-Spline

Las funciones Base se pueden obtener a través de la formula recursiva de Cox-de Boor descrita como:

$$N_i^0(t) = \begin{cases} 1 \text{ si } t \in [x_i, x_{i+1}] \\ 0 \text{ en otro caso} \end{cases}$$

$$N_i^r(t) = \frac{(t - x_i) N_i^{r-1}(t)}{x_{i+r} - x_i} + \frac{(x_{i+r+1} - t) N_{i+1}^{r-1}(t)}{x_{i+r+1} - x_{i+1}} \quad r \geq 1$$

Si algún denominador es 0, se entiende que ese sumando es 0. En cuanto al vector de nodos, si cada punto está equidistante, se dice que es un vector de nodos uniforme. Por ejemplo el vector [*0,1,2,3,4*] seria uniforme, ya que $x_i = x_{i-1} + 1$.

Las B-Spline son empleadas para diversas tareas como la reconstrucción de imágenes 3D y el Diseño Asistido por Computador (CAD/CAM), entre otras [34].

2.4.4 Mecanismo de Aprendizaje

Una de las propiedades más importantes de las RNAs es su capacidad de aprender a partir de un conjunto de patrones de entrenamientos, es la clave de la **plasticidad**. Por lo tanto el proceso de aprendizaje es aquel por el cual la RNA se adapta al estímulo modificando sus pesos y eventualmente produce un resultado esperado.

Retomando el concepto biológico de las neuronas, en la mayoría de los seres vivos con sistema biológico nervioso, la inteligencia está dada por el **número, organización y modo de cambio** de las conexiones sinápticas. Es aquí donde radica la importancia del mecanismo de aprendizaje, ya que este se encargara de emular este comportamiento.

La cantidad de patrones de entrenamientos de la red debe ser una muestra representativa y un número significativo (de ejemplos) para que la red ajuste sus pesos de manera eficaz. Se puede distinguir 3 tipos de aprendizajes: Aprendizaje Supervisado, Aprendizaje No Supervisado y Aprendizaje Híbrido.

2.4.4.1 Aprendizaje Supervisado

Incluye a un supervisor que le indica a la red cuan cerca está de aprender y va modificando los pesos neuronales. Suponer:

$$N = \left\{ \{x_1, resp(x_1)\}, \{x_2, resp(x_2)\}, ..., \{x_i, resp(x_i)\}, ..., \{x_n, resp(x_n)\} \right\}$$

Un conjunto de *n* pares de ejemplos de entrenamiento, donde para la entrada *i-ésima* x_i , $resp(x_i)$ es su respuesta correcta. Además suponer,

$$M = \left\{ est(x_1), est(x_2), ..., est(x_i), ..., , est(x_n) \right\}$$

Donde M se corresponde unívocamente con N, es decir, para cada entrada *i-ésima* x_i , $est(x_i)$ es su respuesta estimada. Cuando se introduzca un ejemplo *i-esimo* y sea procesado por la RNA, su salida estimada $est(x_i)$ será compara con la salida que debería haber producido $resp(x_i)$. Su diferencia denominada tradicionalmente cómo $Error = resp(x_i) - est(x_i)$ será la que influirá en cómo se modificaran los pesos. Usando el Error (o modificaciones de este, como el error cuadrático medio) un algoritmo dado por el supervisor modificara los pesos uno por uno tratando de minimizar el error. La figura 2-11 muestra el proceso descrito:

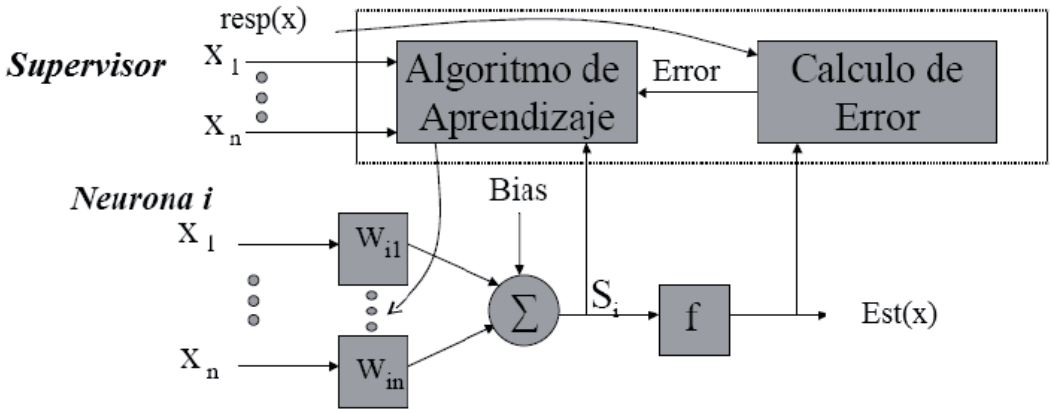

Figura 2-11 Aprendizaje Supervisado

Algunos algoritmos que se encuentran son:

- **Backpropagation**: Es un algoritmo de gradiente descendente, en el cual los pesos de la red son modificados mediante el gradiente de la función de error. Una vez generada una(s) salida(s) de un patrón, comienza una etapa hacia atrás donde se modifican los pesos de la red, uno por uno, de manera que coincida la salida deseada por el usuario con la salida obtenida por la red ante la presentación de un determinado patrón de entrada o con un error aceptable [36]. La actualización de los pesos está dada por :
  - $w_{ij}(k+1) = w_{ij}(k) - \mu \frac{\partial Error}{\partial w_{ij}}$
    - $w_{ij} = peso$
    - μ = Factor de Aprendizaje
- **Levenberg Marquardt** (LM): Algunos algoritmos calculan matrices Hessianas que agregan un segundo orden al algoritmo de propagación hacia atrás, pero esto implica cálculos demasiados complejos. LM también busca hallar la dirección de cambio en base a derivadas de segundo orden, pero sin requerir el cálculo del Hessiano, sino una aproximación de este. Se dice que puede tomar la dirección del algoritmo Gauss-Newton o la del gradiente descendiente, es una mezcla de ambos. Tiene un buen desempeño cuando el rendimiento de la red este determinado por el error cuadrático medio [36]. La actualización de los pesos está dada por:
  - $w_{ij}(k+1) = w_{ij}(k) - \mu[J \cdot J^T + \mu \cdot I]J^T \cdot Error$
    - J = Matriz Jacobiana que contiene las primeras derivadas de los errores de la red con respecto a los pesos.
    - $Error$ = Vector de errores de la red.

En este proyecto se desarrollará un aprendizaje Supervisado para el ajuste de los pesos y la arquitectura de la red. Se emplearán Algoritmos de Optimización por Enjambre de Partículas (PSO) para llevar a cabo el entrenamiento. El detalle de su funcionamiento esta descrito en el capítulo 3.

- **OTROS**: Adeline, Delta Learning, Madeline, Reforzado, Competitivo.

2.4.4.2 Aprendizaje No Supervisado

Se presenta solo un conjunto de patrones a la RNA, y el objetivo del algoritmo de aprendizaje es ajustar los pesos de la red de manera tal que la red encuentre alguna estructura o configuración presente en los datos. No dependen de un supervisor. Por lo que ajusta los parámetros internamente, adaptándose al entorno. Se distingue del supervisado por que no hay un conocimiento a priori, es decir, no se le presenta una salida correcta o deseada.

La red deba descubrir por si sola características, patrones, correlaciones o categorías en los datos de entrada, y se obtengan de forma codificada a la salida [37].

Algunos algoritmos no supervisados son Hebbian Learning, Self Organizing Map (SOM) [24], Redes de Resonancia Adaptativa (ART), Redes de Kohonen.

2.4.4.3 Aprendizaje Mixto

Corresponde a una mezcla del esquema Supervizado y No Supervizado, definiendo dos fases para la etapa del entrenamiento, donde en una se utiliza aprendizaje supervisado y en otra un aprendizaje no supervisado. Normalmente estos se emplean en distintas capas. Un ejemplo de aprendizaje híbrido corresponde al entrenamiento de las redes de base radial [38].

2.5 Aplicaciones de las Redes Neuronales Artificiales

Las redes neuronales pueden utilizarse en diversas actividades tales como [39]:

- Clustering (Agrupación): Un algoritmo de clustering explora la similitud entre sus patrones y otros patrones similares en un clúster.
- Clasificación o reconocimiento de patrones: La tarea de reconocer patrones consiste en asignar un patrón de entrada a una de las muchas clases. Esta categoría incluye implementaciones tales como memorias asociativas. Un ejemplo es el reconocimiento de patrones en imágenes, tales como escritura manual, visión artificial.
- Aproximación de Funciones: La idea es encontrar y estimar una función desconocida sujeta a ruido. Variadas ingenierías y disciplinas científicas requieren aproximaciones de funciones.
- Procesamiento del lenguaje natural: Conversión de texto escrito a lenguaje hablado, La idea es transformar el código escrito a fonemas. Aprendizaje de gramáticas, trata de generalizar y conjugar verbos desconocidos.
- Predicción: La idea es pronosticar algunos valores futuros de una serie temporal de datos. La predicción tiene un gran impacto en la toma de decisiones. La predicción difiere de aproximaciones de funciones ya que considera el factor de tiempo. Un ejemplo de aplicación es en el campo de indicadores económicos.
- Problemas de Combinatoria: Acercamientos a resolver problemas complejos del tipo NP-completos, por ejemplo el problema del vendedor viajero [40].

3. Optimización por Enjambre de Partículas

Como se mencionó en el capítulo 2, a lo largo de la historia se han generado diversas formas de presentar soluciones, cada una con su propia perspectiva o fuente inspiradora. Intentar abordar problemas complejos de optimización uniobjetivo y multiobjetivo con los métodos clásicos se torna en muchos casos impracticable por su complejidad o por desconocimiento de las funciones mismas. Una alternativa a estos problemas son los algoritmos de optimización metaheurísticos que permiten encontrar soluciones de buena calidad en tiempos de ejecución computacionalmente aceptables. Esta área es uno de los campos más activos en optimización [41].

Desde un punto de vista matemático los algoritmos heurísticos no aseguran encontrar el mínimo o máximo absoluto de una función, sin embargo, no siempre es necesario obtener dicho valor máximo o mínimo sino uno que satisfaga las necesidades de diseño o esté dentro de un margen de error aceptable.

Especial interés ha adquirido la Optimización por Enjambre de Partículas o PSO (de las palabras en inglés Particle Swarm Optimization), método perteneciente a las técnicas estocásticas de computo evolutivo, siendo considerada una de las más emblemáticas de la rama de la inteligencia de enjambre o SI (de la palabra en inglés Swarm Intelligence). Este proceso trata de imitar los comportamientos sociales de una población a partir de la interacción de los individuos entre sí y con el entorno [41]

El origen de PSO se puede remontar a la siguiente interrogante: ¿Cómo pueden las aves o peces desarrollar un comportamiento colectivo y coordinado? Kennedy y Eberhart [42] intentaron emular gráficamente el movimiento sincronizado de grupos de seres vivos, motivamos por la capacidad que poseen estos grupos para separarse, reagruparse o encontrar alimento, para ello observaron el comportamiento de bancos de peces y bandadas de aves. Las conclusiones a las cuales llegaron se relaciona directamente con la capacidad que tienen estos organismos para compartir información y la experiencia acumulada que aporta cada uno de los integrantes. Esto resulta ser un proceso evolutivo que crea sinergia.

En [42], Kenndy y Eberhart proponen el concepto denominado partícula o agente, para representar a un individuo, tal como un pez, abeja, ave, o cualquier otro del que se pretenda emular su comportamiento social. El fundamento teórico de este método dice que el movimiento de una partícula cualquiera dirigida hacia un objetivo en común de D dimensiones estaría limitado por la memoria autobiográfica de la partícula y la influencia social de todo el enjambre. Cada partícula se dice que sobrevuela el espacio de decisión en busca de soluciones óptimas. Por lo que la posición instantánea de cada una de estas en el espacio vectorial de D dimensiones representa una solución potencial. Cuando se habla de un espacio D-dimensional, se cuentan con d incógnitas a solucionar.

El proceso evolutivo se basa en mover a cada una de las partículas dentro del espacio de soluciones, con una velocidad que varía de acuerdo a la memoria de la partícula, información global y su velocidad actual, utilizando una función de aptitud (fitness) que se encargará de cuantificar la calidad de cada partícula en función de la posición que esta ocupe [43].

La ventaja de utilizar PSO es que no recurre el gradiente para optimizar el problema, por lo que el método puede usarse en una amplia gama de problemas, ya que no requiere que el problema de optimización sea diferenciable como si lo es requerido en métodos clásicos tales como el gradiente descendente o método de cuasi-Newton. Sin embargo esta versatilidad tiene el inconveniente que PSO no siempre funcione bien lo que conlleva a ajustar sus parámetros de modo tal que se adapte al problema específico en estudio, dando inicio a una serie de variantes de PSO. Estas variantes PSO pueden aumentar en gran medida la complejidad del método original pero han demostrado tener rendimientos satisfactorios para clases de problemas específicos.

Para poder hacer modificaciones a PSO se debe tener claro cómo funciona el PSO tradicional y que parámetros se pueden modificar, por lo tanto, en lo que sigue de este capítulo, se presentara de manera formal el Modelo Tradicional de PSO, luego se presentara los parámetros que se pueden modificar, las topologías, algunas variantes que se han encontrado en la literatura de PSO y finalmente sus áreas de aplicaciones.

3.1 Algoritmo PSO Modelo Tradicional

Definición: PSO es un algoritmo evolutivo, iterativo y estocástico, que se basa en buscar soluciones en un espacio vectorial *N* dimensional desplazando a un conjunto de partículas sobre el mismo, siendo cada partícula una posible solución.

Este algoritmo parte de una población(o enjambre) para iniciar un proceso de búsqueda, donde cada partícula esta agrupada dentro una población con posiciones y velocidades aleatorias. Una partícula es considerada como un punto en el espacio de búsqueda *D*-dimensional, donde cada dimensión representa una de las incógnitas del problema. La nomenclatura para representar el enjambre está dada por $X = (x_{i1}.x_{i2}, x_{i3}, ..., x_{id})$, donde *i=1,2,…, n* corresponde a la i-ésima partícula y *n=1,2,…..D*, corresponde a sus dimensiones.

Ya teniendo inicializadas las partículas comienza el proceso iterativo. Por cada iteración, cada partícula actualizara el valor de su mejor posición encontrada, denominada mejor personal p_{best}. Para representar al p_{best} de una i-ésima partícula se tiene la siguiente expresión $P_i = (p_{i1}, p_{i2}, p_{i3}, ..., p_{id})$. Además de actualizar él p_{best} se debe preocupar de actualizar el valor de la mejor posición encontrada en el enjambre, denominado mejor global p_{gbest}. Estos valores se utilizan para ajustar la velocidad con que se mueven una partícula i-ésima en cada una de las dimensiones. Además, con el valor de la velocidad, la partícula podrá calcular su nueva posición para la siguiente iteración. La influencia que tiene la mejor posición personal sobre la velocidad de una partícula se conoce como el factor o componente de cognición, y la influencia de la mejor posición global o del enjambre se conoce como componente social [44].

Las ecuaciones que determinan la posición y la velocidad de cada partícula dentro del enjambre está dada por:

$$v_{id}(t+1) = w \cdot v_{id}(t) + c_1 r_1 (p_{id}(t) - x_{id}(t)) + c_2 r_2 (p_{gd}(t) - x_{id}(t)) \quad (3.1)$$

$$x_{id}(t+1) = x_{id}(t) + v_{id}(t+1) \tag{3.2}$$

Los parámetros c_1 y c_2 corresponde a las variables que controlan la influencia de las componentes cognitiva y social. Los valores de r_1 y r_2 corresponden a números aleatorios independientes pertenecientes al intervalo [0,1]. Sin embargo hasta ahora no se ha hablado del el parámetro *w*. En un trabajo previo [43], se realizó una modificación al algoritmo agregando un componente denominado factor de inercia *w*, el cual se utiliza para balancear la velocidad de la partícula respecto a la búsqueda global. Para prevenir que las velocidades de las partículas incrementen infinitamente, estas se acotan a un rango determinado por el valor V_{max}, el cual delimita el rango de velocidad al intervalo [$-V_{max}$, V_{max}].

A continuación se mostrara los pasos a seguir en el algoritmo Optimización por Enjambre de Partículas (PSO) tradicional:

1. Se inicia con una población de partículas de tamaño *n*, con posiciones y velocidades aleatorias en un espacio *D*-dimensional.
2. Para cada partícula, evaluar su función fitness de acuerdo a algún criterio de optimización (función de costo, mínimos cuadrados, error cuadrático medio).
3. Se compara el valor de la función fitness de la partícula con el fitness del mejor personal (p_{best}). Si el valor del fitness actual resulta ser mejor, entonces se actualiza p_{best} al valor de la posición actual x_{id}.
4. Se compara el *fitness* actual de cada partícula con el mejor *fitness* encontrado por el enjambre. Si el *fitness* actual resulta ser mejor, entonces se actualiza la mejor posición global (p_{gbest}) al valor de la posición actual x_{id}.
5. Modificar las posiciones y velocidades de las partículas de acuerdo a las ecuaciones 3.1 y 3.2, verificando que las partículas no sobrepasen la velocidad máxima definida el intervalo generado por $V_{max.}$
6. Si el criterio de parada no se ha cumplido, volver al paso 2.

Las condiciones de términos más recurridas en la literatura para finalizar el proceso de optimización son:

- Número de iteraciones sin mejora: El proceso termina después de alcanzar un número fijo de iteraciones sin presentar mejoras en la solución.
- Número máximo de iteraciones sin mejora: El proceso termina después de realizar un número fijo de iteraciones prestablecidas.
- Fitness logrado: El proceso termina cuando llega a un valor menor o igual al fitness establecido.

El éxito del algoritmo depende en la medida que se ajusten las posiciones de las partículas en el espacio de búsqueda, estos ajustes dependen de un número reducidos de parámetros lo que se traduce en la gran potencia de PSO. Sin embargo muchísimas variaciones de los parámetros se podrían hacer. Es por ello que se a continuación se presenta un apartado para revisar las posibles modificaciones que se aconsejan en la literatura para ajustar los parámetros y obtener soluciones prometedoras.

3.2 Ajuste de los Parámetros de PSO

El ajuste de los parámetros condiciona el rendimiento posterior del algoritmo de optimización. Existen múltiples estudios [45] que intentan generalizar la selección de estos parámetros, sin embargo dicha selección está directamente relacionada a la naturaleza del problema [46].

Retomando la ecuación (3.1), se podrían variar sus parámetros de manera tal que afecte la búsqueda en el espacio de soluciones. A continuación se examinaran los parámetros que afectan al algoritmo.

$$v_{in}(t+1) = \underbrace{w \cdot v_{in}(t)}_{Inercia} + \underbrace{c_1 r_1 (p_{in}(t) - x_{in}(t))}_{Influencia\ Personal} + \underbrace{c_2 r_2 (p_{gn}(t) - x_{in}(t))}_{Influencia\ Social} \tag{3.3}$$

- **Velocidad**: Para acotar la velocidad de la partícula se especifica un valor máximo, V_{max}, que restringe la velocidad en cada dimensión al intervalo [-V_{max} , V_{max}]. En caso de que no pertenezca a dicho rango se hace el siguiente ajuste:

$$v_{ij} = sign(v_{ij}) \cdot V_{\max} \tag{3.4}$$

$$sign(x) = \begin{cases} 1 & x>0 \\ 0 & x=0 \\ -1 & x<0 \end{cases} \tag{3.5}$$

  Si el valor de V_{max} es extremadamente pequeño las partículas exploraran el espacio de soluciones muy lentamente y podrán quedar atrapadas en un óptimo local [47], este fenómeno se conoce como convergencia prematura del algoritmo [46]. No incluir el parámetro de control V_{max}, implicaría que el enjambre no tienda a converger hacia un punto, produciéndose el fenómeno conocido como explosión del PSO, consistente en un comportamiento oscilatorio y creciente de la posición de las partículas, provocando la ineficiencia del PSO como algoritmo de optimización [43]. El valor de V_{max} a tomar, dependerá única y exclusivamente de la naturaleza del problema a optimizar. Con el objetivo de reducir el efecto de V_{max} y perfeccionar el control del alcance de la búsqueda sobre el espacio de soluciones se incorporó el concepto de peso inercial [44].
- **Peso de inercia *w***: Es considerado un parámetro crítico para la convergencia de PSO. Se utiliza para controlar el impacto de la historia previa de las velocidades actuales. En consecuencia, el parámetro *w* regula la compensación entre las capacidades de exploración global y locales del enjambre. Un peso inercial grande facilita la exploración global (buscando nuevas áreas). Si es un valor pequeño facilita la exploración local. Algunos resultados experimentales han indicado que es mejor asignar un valor *w* grande al principio, para promover la exploración global del espacio de búsqueda, y gradualmente disminuirlo para conseguir soluciones más refinadas. En [44] se menciona que una buena opción para el

parámetro *w* seria asignarle un valor inicial alrededor de 1.0 y luego, ir disminuyéndolo hasta llegar cercano a 0. Otra alternativa, seria usar alguna aproximación adaptable (un controlador difuso por ejemplo), en el cual este parámetro puede ser sintonizado según las características del problema a considerar [41].

- **Parámetros c_1 y c_2**: No son críticos para la convergencia, sin embargo, si se calibran adecuadamente podrían acelerar la convergencia del algoritmo y aliviarlo de posibles mínimos locales. El valor comúnmente usados para estos parámetros es igual a 2 ($c_1 = c_2 = 2$) [44], pero algunos resultados experimentales indican que se pueden asignar a ambos parámetros el valor de 1.49. Trabajos recientes han agregado la restricción que la suma de los parámetros no supere el valor 4, es decir, $c_1 + c_2 \leq 4$ [43].
- **Tamaño de la población** (o enjambre): Debe ser seleccionado con rigurosidad ya que poblaciones excesivamente grandes exploran minuciosamente el espacio de búsqueda, pero con un elevado coste computacional. Los estudios paramétricos revelan que una población de alrededor de 30 partículas es suficiente para múltiples problemas, y típicamente se utilizan poblaciones que oscilan entre 10 y 50 partículas, o entre 100 y 200 partículas para abordar problemas más complejos [44].

Lo ideal es que las partículas representen soluciones de diferentes zonas del espacio de búsqueda. Este concepto se conoce diversificación de la población, y se refiere a que las partículas sobrevuelen diferentes zonas del espacio de solución antes de que se comprometan con alguna zona. Una vez que convergen a una zona prometedora, se espera que sean capaces de mejorar la solución encontrada es decir, que logren encontrar y converger a un óptimo cada vez mejor [48], [49].

Estos 2 modos se conocen en la literatura exploración y explotación [50]. Se define como **exploración**, a la búsqueda de la solución del problema en un espacio de búsqueda amplio, es decir, que las partículas vuelen por diferentes zonas del espacio de búsqueda privilegiando la diversificación. **Explotación** se entenderá como a la búsqueda de la solución del problema encontrando sub-espacio dentro del espacio de búsqueda, en donde el enjambre trate de tomar todos los valores posibles que comprendan a sub-espacio. [48].

Lo ideal es ajustar los parámetros para que el algoritmo funcione en modo exploración inicialmente, y que luego cambie al modo explotación. Debido a esto, en la actualidad se han propuesto una seria de técnicas [48], [51] en donde los parámetros son autoajustados durante la ejecución del algoritmo, cambiando de modo exploración a explotación y de explotación a exploración, según como sea el rendimiento del algoritmo [52].

En un trabajo previo [43], se da cuenta que valores altos para el coeficiente de inercia, factores de cognición y V_{max} favorecen el modo exploración del PSO, mientras que valores bajos para estos parámetros favorecen el modo explotación, es decir:

$$Exploración \Rightarrow 0.75 < w < 1 \text{ y } 2 < c_1, c_2 < 4 \tag{3.6}$$

$$Explotación \Rightarrow 0.4 < w < 0.75 \text{ y } 0.1 < c_1, c_2 < 2 \tag{3.7}$$

Finalmente una de las decisiones claves es la elección de la función objetivo para evaluar la aptitud o fitness de una partícula. La elección de esta función será clave para alcanzar el éxito del algoritmo, debido a que es ella quien se encarga de guiar la búsqueda durante el proceso de optimización.

3.3 Modelo Óptimo Local y Óptimo Global

En el modelo óptimo local las partículas tienen solo la información de sus vecinos más cercanos del enjambre, permitiendo que cada individuo sea influenciado por un reducido número de miembros adyacentes de la población. Las partículas seleccionadas para ser un subconjunto del enjambre no tienen relación directa con el resto de las partículas que pertenecen a otra vecindad [41].

En el modelo del óptimo global las partículas consideran la información de todo el enjambre. En el modelo del óptimo global, la trayectoria de búsqueda de cada partícula es influenciada por la mejor posición encontrada por cualquier miembro de la población. La mejor partícula atrae a todas las partículas hacia ella.

Se concluye que el modelo de optimo global converge rápidamente hacia las soluciones del problema, pero tiene aumenta la probabilidad de quedar atrapado en óptimos locales, mientras que el modelo de optimo local tiene la desventaja de converger lentamente hacia la solución del problema. Por ello, los autores de [44] recomiendan usar el modelo de óptimo global fuerte para funciones unimodales y un modelo local variable para funciones multimodales.

3.4 Variaciones del Algoritmo de PSO

Retomando al algoritmo PSO tradicional, la nueva posición de una partícula está dada por la ecuación:

$$x_{in}(t+1) = x_{in}(t) + v_{in}(t+1)$$

Suponiendo un caso hipotético que la partícula está recorriendo un sub-espacio donde el punto de interés está en el centro de dicho espacio, podría resultar más atractivo que se vaya acercando a dicho punto tanto por la izquierda como la derecha de este, por lo que una modificación a la ecuación clásica podría ser:

$$x_{in}(t+1) = \left[x_{in}(t) + v_{in}(t+1) \right] \cdot \underbrace{(-1)^{i}}_{\substack{Alternar \\ espacio}} \tag{3.8}$$

Sin embargo al hacer esto se agrega un mayor nivel de complejidad al algoritmo tradicional. Actualmente existen una serie de versiones alternativas al algoritmo PSO tradicional, las cuales incorporan nuevos parámetros, nuevos métodos de actualización, e híbridos con otros algoritmos. Estas versiones tienen por objeto mejorar algunas de las características del PSO tales como mejora de la solución encontrada y mejora de velocidad de convergencia. Las variaciones que se pueden hacer son infinitas, sin embargo no todas tienen

porque tener un buen comportamiento. A continuación se describirán algunas variaciones del algoritmo PSO tradicional que han demostrado tener resultados competitivos.

3.4.1 PSO con un parámetro de Tiempo de Vida

Esta variación está inspirada en que tan productivo o eficiente es un empleado realizando su trabajo dentro de la organización, centrándose particularmente en aquellos que son ineficientes. En una organización la producción de los empleados se podría clasificar en 3 grupos: los que rinden dentro del promedio esperado, los que son más eficientes que el resto, y grupos que simplemente producen bajo la media. En ocasiones, a los trabajadores ineficientes tienen la oportunidad de cambiar y mejorar su rendimiento antes de ser despedidos. Si no logran mejorar su rendimiento dentro de un periodo asignado, son despedidos y se asignan nuevos trabajadores a los puestos vacantes con la idea de que estos últimos trabajen mejor que los anteriores.

Considerando una analogía en relación al periodo de tiempo que se les asigna a los trabajadores ineficientes, la variación consiste en introducir un nuevo parámetro llamado tiempo de vida, el cual se les asignará a cada partícula y decrecerá de acuerdo a su falta rendimiento en cada iteración [53]. La decisión de destruir la peor partícula es tomada solo después que se cumpla un número definido de oportunidades de mejora, dado por el parámetro "tiempo de vida". La condición está definida por:

$$rem\left(\frac{Iteración - Número}{TiempoVida}\right) = 0 \tag{3.9}$$

Una vez que las partículas cumplen la condición dada por 3.9 implica que están listas para ser relevadas. Se elige aquella partícula que tenga el peor valor de fitness y pasa a ser automáticamente candidata para ser destruida. Para remplazar a la partícula destruida se genera una nueva partícula en la vecindad de una de las partículas restantes, elegida a través de algún método de selección.

Esta mejora fue analizada en tres funciones de pruebas de patrón concluyendo que la incorporación del parámetro de tiempo de vida logra que el PSO rinda mejor en términos de cantidad de iteraciones para la convergencia y calidad del óptimo [54].

3.4.2 PSO modificado para Localizar Todos los Mínimos Globales

En aplicaciones prácticas, existen diversos métodos que permiten encontrar solo soluciones sub-óptimas de una determinada función objetivo. En muchos casos estas soluciones sub-óptimas son aceptables, pero hay aplicaciones donde una solución óptima es indispensable. Además muchos problemas de optimización poseen mínimos globales los cuales tienen que ser computados rápidamente. Debido a lo anterior, se propone una estrategia para encontrar todos los mínimos globales (o alguno de ellos) de una función objetivo de manera eficiente [55].

Si se utiliza el PSO tradicional para encontrar un solo minimizador global, es decir, un $\bar{x} \in D \; tal \; que \; f(\bar{x}) \leq f(x), \forall x \in D,$ dada una función objetivo f que posee varios mínimos globales dentro del híper-espacio *D*, podría pasar que encuentre un mínimo global (pero no sabe cuál) o que el enjambre avance por el espacio de búsqueda sin poder decidir saber dónde parar [55].

En varias aplicaciones el valor del mínimo global es conocido antes pero puede haber un número finito (o infinito en algunos casos) de minimizadores globales. Para evitar este problema, se determina un umbral no pequeño $\varepsilon > 0$ y cuando una partícula adquiera un valor funcional que sea menor que ε, se saca la partícula de la población y se aísla. Simultáneamente, se aplicaría deflación o estiramiento [55] a la función objetivo original f en ese punto, para repeler al resto del enjambre y lograr que las demás partículas no se muevan hacia dicho punto, sino que se adhieran a una nueva partícula (seleccionada al azar) del enjambre.

El "estiramiento" es una técnica que provee una vía de escape de un mínimo local cuando el PSO es incapaz de converger [55]. Esto consiste en una transformación de dos estados en cuanto a la forma de la función f original, la cual puede ser aplicada antes de que la función f haya detectado un mínimo global $\bar{x}$:

$$G(x) = f(x) + \gamma_1 \frac{\|x - \bar{x}\|(sign(f(x) - f(\bar{x})) + 1)}{2} \tag{3.10}$$

$$H(x) = G(x) + \gamma_2 \frac{sign(f(x) - f(\bar{x})) + 1}{2\tanh(\mu(G(\bar{x}) - G(\bar{x})))} \tag{3.11}$$

Donde γ_1, γ_2, y μ son constantes positivas arbitrarias, y el *sign(.)* es la ecuación definida en 3.5 .

Una vez aislada la partícula se verifica su valor funcional. Si está lejos de la exactitud deseada, se podrá generar una pequeña población de partículas alrededor de ella y obligar a este pequeño enjambre (en la vecindad aislada de f) para que ejecute una búsqueda mejor, mientras que el enjambre grande continúe explorando el resto del espacio de búsqueda para encontrar los otros mininizadores. Esto permite la versatilidad de ejecutar en paralelo al algoritmo, por un lado se ejecuta el enjambre pequeño y por otro el enjambre grande.

Este algoritmo fue probado en 2 funciones, una bidimensional y Levy No 5. Concluyendo que presenta mejoras al encontrar todos los minimizadores globales con la exactitud dada y no se necesitó de una búsqueda local adicional [55].

3.4.3 PSO con Operador Diferencial Evolutivo

Varios de los investigadores han notado que durante todo el proceso de optimización, las partículas oscilan en diferentes ondas sinusoidales, convergiendo rápidamente incluso antes de tiempo. Esto se debe a que la diferencia entre la mejor posición personal de una partícula y

la mejor posición global del enjambre son muy cerradas, es decir, $p_{id} - p_{gd} \approx 0$. Esto provocaría que algunas partículas quedaran "inactivas" en una cierta etapa [56].

La variación consiste en un algoritmo híbrido entre PSO y un Operador Evolutivo Diferencial (DE), el cual provee mutaciones acampanadas (gaussianas) de acuerdo a la diversidad de la población, mientras se mantiene el dinamismo del enjambre de partículas.

La idea consiste en que una vez encontradas las mejores posiciones por las partículas se realicen mutaciones proporcionadas por el operador DE sobre estas, es decir, sobre el vector p_{id}, con un punto de rastro $T_i = p_i$. La dimensión está dada por *D*, y *k* adquiere un valor entero aleatorio entre el intervalo [*1, D*]. El parámetro $\vec{\delta}_N$ representa la diferencia de vector general dada por [56]:

$$\vec{\delta}_N = \frac{\sum_{1}^{N} \vec{\Delta}}{N} \tag{3.12}$$

El vector Δ es el vector de diferencia esencial, es decir, significa la diferencia entre 2 elementos que son escogidos aleatoriamente de un conjunto de puntos comunes, el cual incluye las mejores posiciones personales de las partículas en el caso actual [41]. El parámetro *N* es el número de Δ implicadas. Entonces para una *d*-dimensión, el vector diferencia esencial estar dado por:

$$\vec{\Delta}_N = \vec{p}_{A,d} - \vec{p}_{B,d} \tag{3.13}$$

Donde los vectores $\vec{p}_{A,d}$ y $\vec{p}_{B,d}$ son escogidos del conjunto de las mejores posiciones personales al azar. Por lo tanto, para un problema que considera una *d*-dimensiones se tiene [56]:

$$Si(r_i < CR \text{ o } d == k) \text{ actualizar } T_{id} = p_{gbest} + \delta_{2,d} \tag{3.14}$$

Donde *CR* es una constante de cruce, r_i corresponde a los valores aleatorios r_1 y r_2 de la ecuación de la velocidad. La mutación se realiza sobre la mejor posición personal p_{id} en vez de la posición actual x_{id}, para evitar conductas inesperadas del enjambre, ya que el remplazo de p_{id} seguirá la estrategia, es decir, T_i remplazará a p_i si y solo si su valor es mejor que el de p_{id}. Este tipo de mutación proporciona mayor capacidad de aprendizaje social, posibilitando una mayor rapidez de convergencia del algoritmo [56].

3.4.4 PSO con Restricción de Velocidad Mínima

Se propone la incorporación de un parámetro V_{min} a modo de velocidad mínima, para impedir que las partículas desciendan rápidamente su velocidad, y evitar que el algoritmo PSO converja prematuramente [57]. Teniendo en cuenta la misma estructura diseñada para el V_{max},

después de actualizada las velocidades de las partículas, se debe verificar que si estas bajan su velocidad por debajo de la velocidad mínima:

$$si(v_{id} < V_{\min}) \text{ y } (v_{id} > -V_{\min}) \text{ entonces,}$$
$$v_{id} = sign(v_{id}) \cdot \bar{v} \tag{3.15}$$

Donde $\bar{v}$ representa el promedio de las velocidades de las partículas dado por:

$$v_{id} = \frac{1}{N \cdot D} \sum_{i=1}^{N} \sum_{d=1}^{D} \left(\left| \mathrm{v_{id}} \right| \right) \tag{3.16}$$

N representa las dimensiones de las partículas y *D* el tamaño del enjambre. La propuesta se probó con 4 funciones de prueba de patrón, en las cuales se obtuvo una buena mejora en términos de calidad del óptimo encontrado. Sin embargo, se podría hacer menos costoso el cálculo del promedio de la velocidad del enjambre, al considerar solo el promedio de las dimensiones en la cual la velocidad decreció por debajo de la velocidad mínima [58].

3.4.5 PSO Quantum

Esta propuesta consiste en eliminar el concepto de velocidad e incorporar una función de movimiento. Esta función consiste en comparar la posición actual de una partícula con el promedio de las mejores posiciones del enjambre, a diferencia del PSO tradicional que compara la posición de una partícula con la mejor posición del enjambre [59]. La expresión de moviente está dada por:

$$x(t+1) = p \pm \beta \left| m_{best} - x(t) \right| \cdot \ln(1/u) \tag{3.17}$$

Donde *u* es un valor aleatorio entre [*0,1*], β es un coeficiente de expansión-contracción, perteneciente al rango [0,1.7], m_{best} representa el vector con las medidas de las mejores posiciones encontradas por las partículas a nivel de dimensión y está dado por:

$$m_{best} = \frac{1}{M} \sum_{i=1}^{M} p_i = \left(\frac{1}{M} \sum_{i=1}^{M} P_{i1}, \frac{1}{M} \sum_{i=1}^{M} P_{i2}, \ldots, \frac{1}{M} \sum_{i=1}^{M} P_{id} \right) \tag{3.18}$$

Sin embargo calcular la media de las mejores posiciones en cada iteración del algoritmo tiene un alto coste computacional, por lo que se debe considerar un número reducido de partículas.

El parámetro *p* en 3.18 depende de la mejor posición de la partícula p_{id} y de la mejor posición global, es decir:

$$p = \phi \cdot p_{id} + (1 - \phi) \cdot p_{gd} \tag{3.19}$$

Donde ϕ corresponde a un valor aleatorio entre [0,1]. Esta propuesta fue probada en 3 funcione obteniendo resultados superiores en términos de velocidad de convergencia al PSO tradicional [59].

3.5 Aplicaciones de PSO

La flexibilidad que otorga PSO permite ser usado en una amplia gama de problemas. De manera general se dice que PSO se puede llevar a problemas multimodales y/o a problemas donde no se conozca algún método especial que arroje resultados satisfactorios. Por otro lado no se puede ser más específico debido a que PSO ha reportado numerosas aplicaciones. En el trabajo [60], se expone un extenso análisis sobre los campos reportados que se ha utilizado PSO, donde solo en la base de datos *IEEE Xplore* se encuentran alrededor de 1,100 artículos. Algunos campos de aplicación son:

- Antenas.
- Biomedicina.
- Clasificación.
- Optimización Combinatorial.
- Control Automático.
- Diseño
- Sistemas Distribuidos.
- Fenómenos electromagnéticos.
- Electrónica.
- Entretenimiento (juegos)
- Finanzas
- Lógica difusa
- Imagen y Video
- Redes Neuronales.
- Predicción y Estimación.
- Etc.

El área de aplicación que se centrara este proyecto es para ayudar a obtener la estructura de una red neuronal en la fase de aprendizaje ajustando los pesos de la red.

4. Arquitectura del Modelo Neuronal Evolutivo FeedFodward Propuesto

En esta sección se presenta el estimador de costo para tuberías. **Es una Red B-Spline de 3 capas feedforward, 4 neuronas de entradas normalizadas (diámetro, dificultad, peso, soldadura) y H neuronas en la capa oculta**. Cada neurona de la capa oculta es sometida a una **función de activación B-Spline de orden q**, la figura 4-2 muestra un esquema genérico. **La salida de la red corresponderá al costo que tiene producir un elemento de tubería manufacturado**. El aprendizaje de la red está dado por los algoritmos PSO, QPSO, PSO con Velocidad Mínima y PSO con Velocidad Mínima modificado, cada uno probado por separado. Estos se encargaran de encontrar los siguientes parámetros:

- El Vector Knot $\Omega = [\lambda_{(j-q)}, ..., \lambda_{(j)}]$, para cada entrada.

- El vector de pesos $W = [w_1, ...w_i, w_p]$, donde el vector de pesos corresponde a los puntos de control de una función B-Spline.

La dimensión de las partículas estará dada por: $(P+q)\times E + P$. La figura 4-1 representa una partícula i-ésima.

$$\underbrace{\lambda_1, \lambda_2, \lambda_3, \ldots, \lambda_i, \ldots, \lambda_{(j-q)}, \ldots, \lambda_j, \ldots, \lambda_{(p+q)\times E}}_{\text{Vector Knot Para Todas Las Entradas}}, \underbrace{w_1, w_2, \ldots, w_p}_{\text{Vector Puntos De Control B-Spline}}$$

Figura 4-1 Partícula que contiene lambdas para cada entrada y los puntos de control w.

La salida del estimador es obtenida como sigue:

$$y = \sum_{j=1}^{p} w_j N_q^j(x_k) \tag{4.1}$$

Dónde:

p=1, 2, 3…., H. Donde H será la cantidad total de nodos ocultos.
q=1, 2, 3…., N. Donde N será el orden (grado) de la función B-Spline.
k=1, 2, 3…, E. Donde E es el número de entradas.

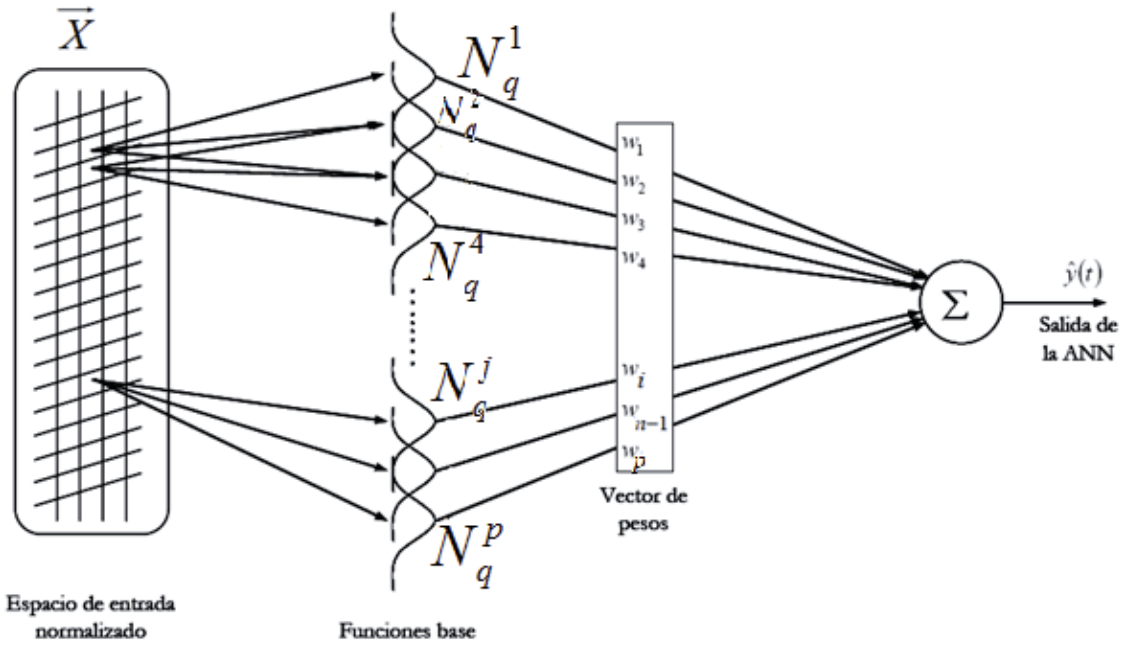

Figura 4-2 Arquitectura genérica del Estimador Propuesto

Sin embargo, dado que hay más de una entrada para una función de base monovariable, **se propone** realizar el **producto tensorial de *L* funciones monovariables**, es decir, generar una función de base multivariable en base a las ***L*** funciones monovariables para cada entrada. La figura 4-3 muestra el esquema específico. El estimador de costo neuronal queda determinado por:

$$y = \sum_{j=1}^{p} w_j N_{L,q}^j(x) \tag{4.2}$$

Donde ***L*** es el número de entradas, luego la función multivariable está dada por:

$$N_{L,q}^j(x) = N_q^j(x_1) \otimes N_q^j(x_2) \ldots \otimes N_q^j(x_L) = \prod_{i=1}^{L=E} N_q^j(x_i) \tag{4.3}$$

Una alternativa podría ser realizar una combinación de lineal de funciones de base monovariable en contraposición al producto tensorial.

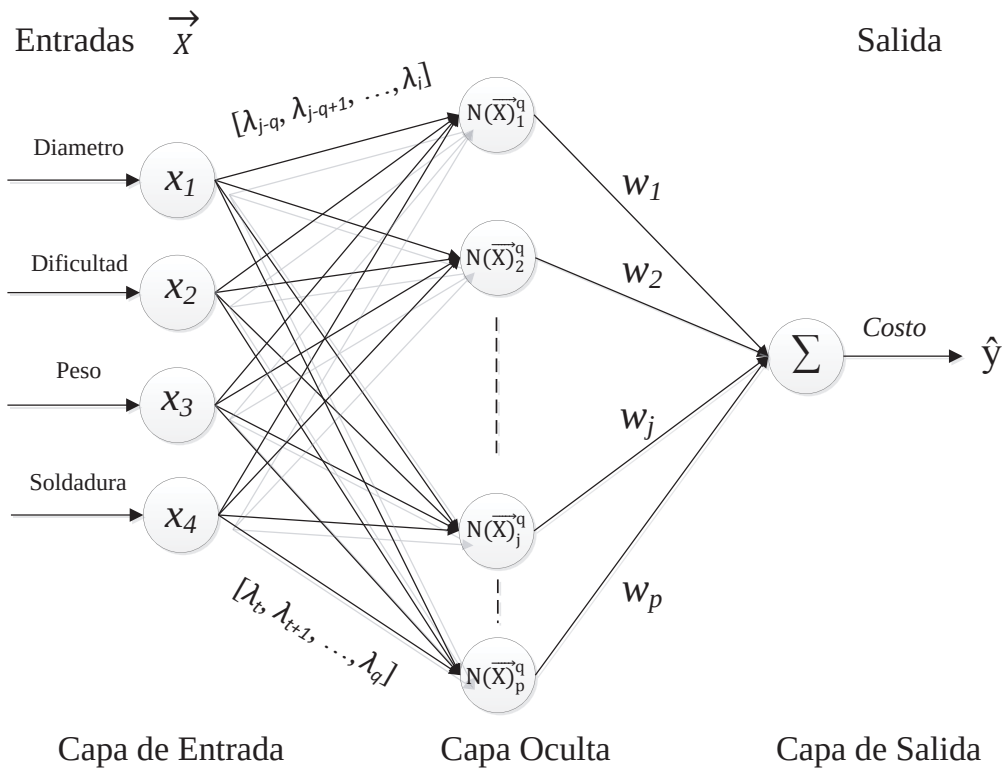

Figura 4-3 Estimador de costo propuesto para fabricar tuberías.

Por lo que las ecuaciones 4.2 y 4.3 se transforman en:

$$y = \sum_{j=1}^{p} w_j N_{L,q}^{j}(\vec{X}) \tag{4.4}$$

$$N_{L,q}^{j}(\vec{X}) = N_q^j(x_1) \otimes N_q^j(x_2) \otimes N_q^j(x_3) N_q^j(x_4) = \prod_{i=1}^{L=E=4} N_q^j(x_i) \tag{4.5}$$

Donde el j-ésimo nodo oculto es sometido a una función base multivariable de orden q, originada por el producto tensorial de 4 funciones de base monovariable. Cabe destacar que el orden de la función base es de responsabilidad del diseñador de la red neuronal y no es necesario que las funciones de base monovariable sean del mismo orden polinomial, es decir, para el j-ésimo nodo oculto su función multivariable se puede crear a partir de 2 funciones monovariables de orden 2, 1 función monovariable de orden 3 y una función de orden 4.

5. Estimador Neuronal Evolutivo FeedFodward Propuesto

5.1 Introducción

Para probar el estimador propuesto se configuraron casos de pruebas aplicables a cada algoritmo de aprendizaje (PSO, QPSO, PSO con Velocidad Mínima, PSO con Velocidad Mínima Modificado). Los pasos realizados fueron:

1. Obtención y pre-procesamiento de los datos: Los datos para probar el estimador son de la misma naturaleza que los utilizados en la investigación [9]. Las variables de entradas seleccionadas luego de un análisis estadístico (coeficiente de Pearson, covarianza y multicolinearidad) son: **Diámetro, dificultad, peso, soldadura.**
2. División de los datos en 2 partes: De un total de 2253 datos que representa los costos de manufacturar tuberías, un 80% ha sido destinado para entrenamiento de la red neuronal y el resto del conjunto de datos, para realizar pruebas(o validaciones). Cabe resaltar que tanto los datos de entrenamiento como los de prueba fueron normalizados generando valores entre 0 y 1.
3. Entrenamiento o calibración de la red: Se calibró la Red B-Spline **manteniendo fijo el número de entradas en 4 y el orden de la función B-Spline en 2 y 3.**
4. **Dado que no existe trabajos previos disponibles sobre los parámetros a configurar se recurrió a variar la cantidad de iteraciones conforme aumentan las partículas de manera tal de generar directrices. Es decir, para 20, 40, 60 y 80 partículas se usó 3000, 2000, 1500, 1400 iteraciones respectivamente.**
5. La función Fitness escogida para elegir una partícula sobre otra (tanto para pso como para sus variaciones) fue el MAPE.

5.2 Métricas de Rendimiento

5.2.1 Error porcentual Absoluto Medio (Mape)

Está dado por:

$$MAPE = \frac{\sum_{i=1}^{N} \left| \frac{Y_i - \hat{Y}_i}{Y_i} \right|}{N} \times 100 \tag{5.1}$$

Donde Y es la salida deseada, $\hat{Y}$ es la salida estimada y N el tamaño de la muestra. El Error Porcentual Absoluto Medio es una medida de precisión comúnmente usado en series temporales. Representa la precisión en forma de porcentaje.

5.2.2 Coeficiente de Correlación *(R)*

Está dado por:

$$R_{yy'} = \frac{\sum y_i y_i' - n\overline{yy'}}{n s_y s_{y'}} = \frac{n\sum y_i y_i' - \sum y_i \sum y_i'}{\sqrt{n\sum y_i^2} - \sqrt{n\sum y'^2_i - \left(\sum y_i'\right)^2}} \tag{5.2}$$

Esta métrica mide qué tanto se relacionan linealmente la salida deseada ***y*** y la salida estimada ***y'.***

5.2.3 Coeficiente de Determinación *(R^2)*

El Coeficiente de Determinación mide el porcentaje de variabilidad en ***y'*** que se puede explicar con el conocimiento de la variabilidad en la variable independiente ***y***.

5.3 Resultados Trainning/Testing

Nota: En las tablas de resultados H y N significan: Número de Nodos Ocultos, Tamaño del Enjambre.

5.3.1 Red B-Spline y PSO Clásico.

La configuración elegida para PSO se basó en iniciar con una estrategia que favoreciera la exploración y conforme se avanzaba en las iteraciones aumentar la explotación y disminuir la exploración. El decaimiento fue aplicado al parámetro W de manera lineal, además a los parámetros c_1 y c_2 se les otorgó la misma importancia. Finalmente esta configuración el PSO clásico tomo la forma de modelo Optimo Global donde las partículas son influenciadas por la mejor posición del enjambre.

$$W = W_{\max} - \left(\frac{W_{\max} - W_{\min}}{N°iteraciones} \right) \times k$$

Donde $W_{\max} = 0.9$, $W_{\min} = 0.4$, $c_1 = c_2 = 1.4$ y k es la k-esima iteración.

La tabla 5-1 contiene los resultados de las diversas configuraciones a las cuales fue sometida de la Red Neuronal con PSO en la fase de entrenamiento.

Tabla 5-1 Resultados Fase Entrenamiento PSO Clásico

H	Mape N=20	Mape N=40	Mape N=60	Mape N=80
1	12.813282	12.813295	12.8150	13.1772
2	13.1504	12.8384	12.8132794881	12.813274
3	12.813279	26.2435	12.8227	12.0227
4	12.9683	13.1805	12.8287	13.1647
5	12.8436	12.8134	12.813288	13.1647
6	14.7681	14.9743	14.6855	26.2437
7	15.3049	12.8198	12.8305	12.7912
8	12.8641	15.3135	12.8174	13.1722
9	13.1925	26.2435	13.3761	13.1787
10	15.2954	14.7913	12.7795	13.1585
11	14.9743	15.1173	14.9889	12.8697

La tabla 5-2 contiene el tiempo en segundos de inicializar el algoritmo (partículas) junto con la realización de una iteración por cada configuración.

Tabla 5-2 Tiempo por cada Iteración en Fase Entrenamiento

H	Tiempo (s) N=20	Tiempo (s) N=40	Tiempo (s) N=60	Tiempo (s) N=80
1	1.830576	3.501679	5.287361	6.933661
2	1.862161	3.661643	5.470420	7.231025
3	1.942711	3.781631	5.702516	7.549111
4	1.990024	3.955115	5.860476	7.828306
5	2.038801	3.953710	6.160193	8.067172
6	2.176161	4.227435	6.374313	8.501302
7	2.325275	4.485129	6.634270	8.743777
8	2.353658	4.600747	6.906077	9.063937
9	2.428893	4.841917	7.061691	9.455524
10	2.463697	4.868083	7.372231	9.694990
11	2.551267	5.079777	7.560395	10.001286

Se puede apreciar que el menor valor del Mape es cuando se utilizó **80 partículas y 3 nodos ocultos.** A continuación se adjunta las gráficas de cada enjambre (N=20, 40, 60, 80) y como se fue comportando el Mape con las diversas cantidades de nodos ocultos con la finalidad de tener una idea más clara sobre cuantas iteraciones son necesarias.

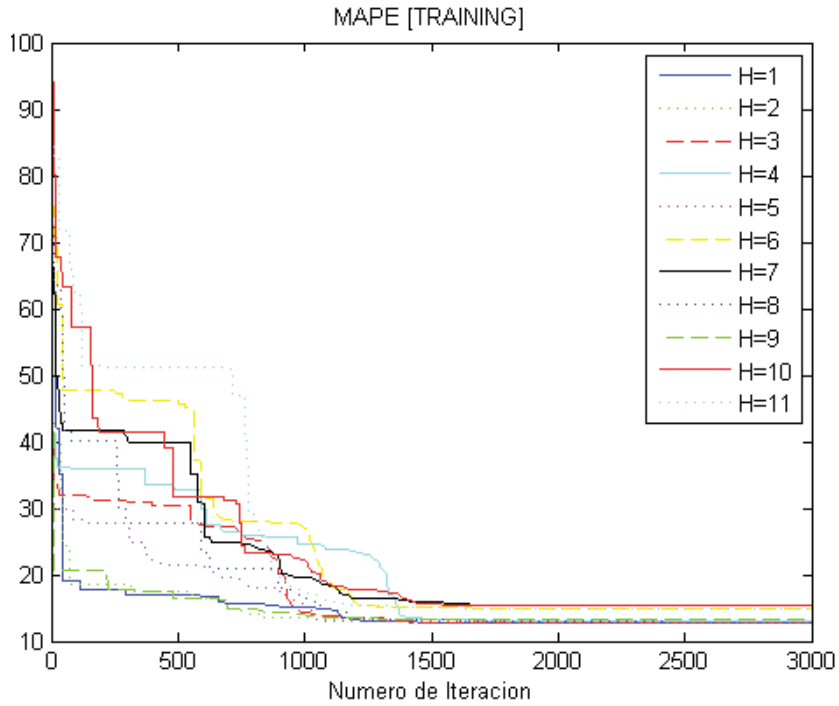

Figura 5-1 Mape de los Nodos Ocultos para un Enjambre 20 partículas.

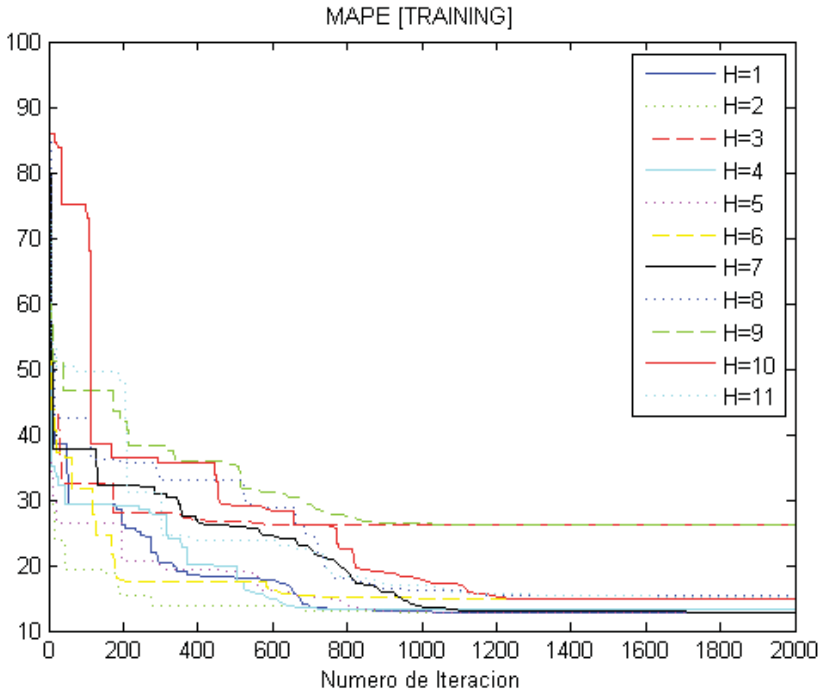

Figura 5-2 Mape de los Nodos Ocultos para un Enjambre 40 partículas.

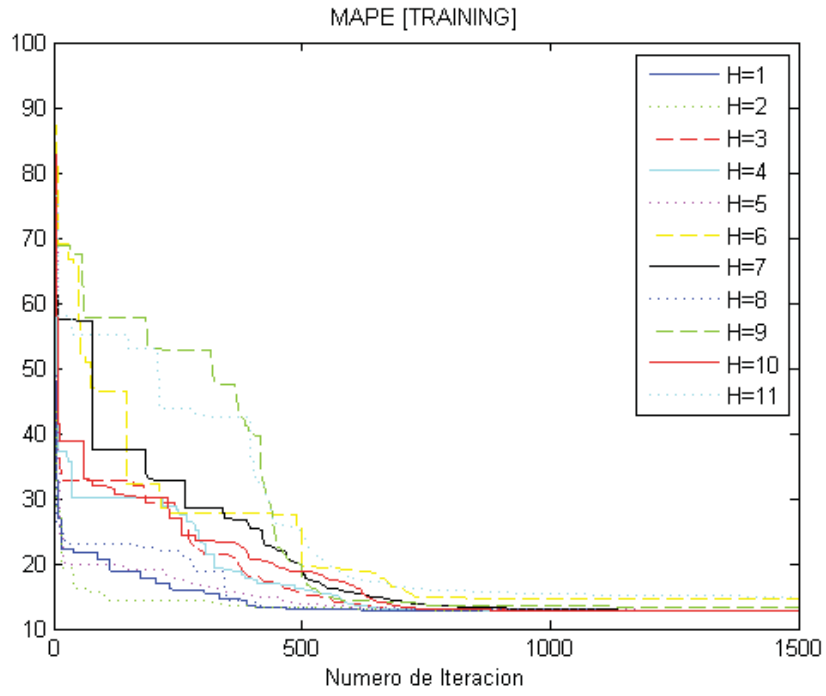

Figura 5-3 Mape de los Nodos Ocultos para un Enjambre 60 partículas.

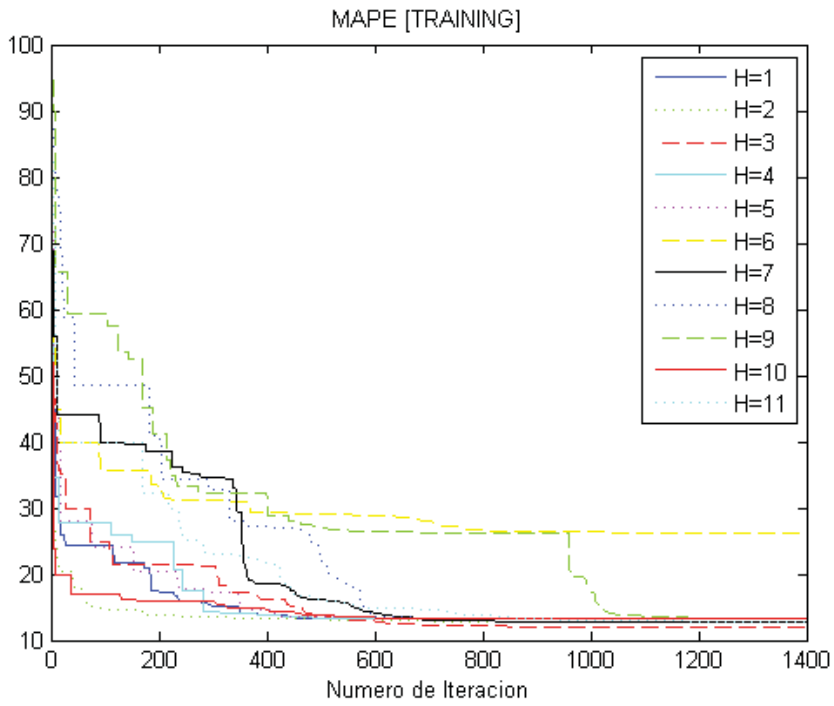

Figura 5-4 Mape de los Nodos Ocultos para un Enjambre 80 partículas.

De las gráficas se puede concluir que con máximo 1700 iteraciones se puede trabajar como punto de partida, para los diversos tamaños de enjambre, aunque puede ser menos (800), se prefiere dejar un margen hacia arriba ya que el número de iteraciones tiene incidencia directa sobre W. Luego un intervalo de trabajo puede ser [800 – 1700] iteraciones.

A continuación se muestra los gráficos asociados a la fase Testing de la mejor configuración obtenida, esto es, **80 partículas y 3 nodos ocultos:**

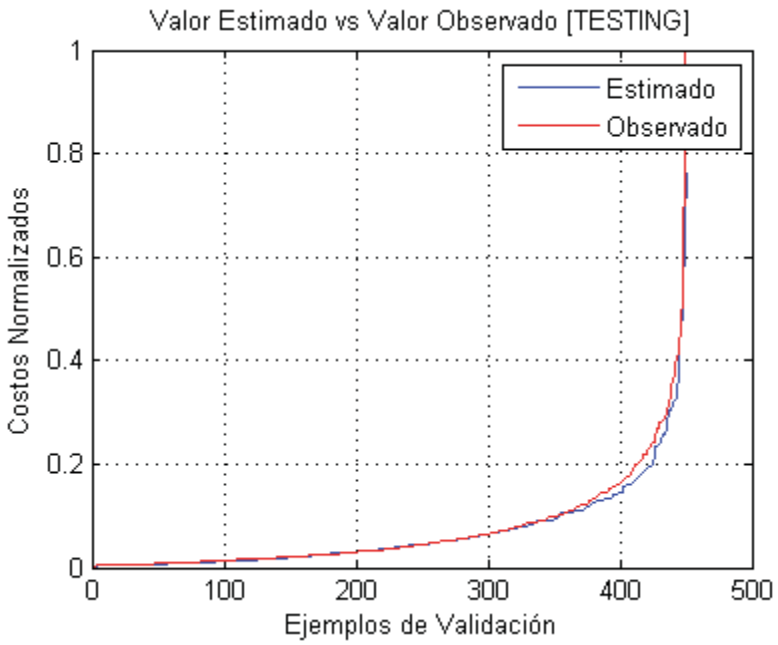

Figura 5-5 Test: 80 Partículas, 3 Nodos Ocultos, PSO Clásico.

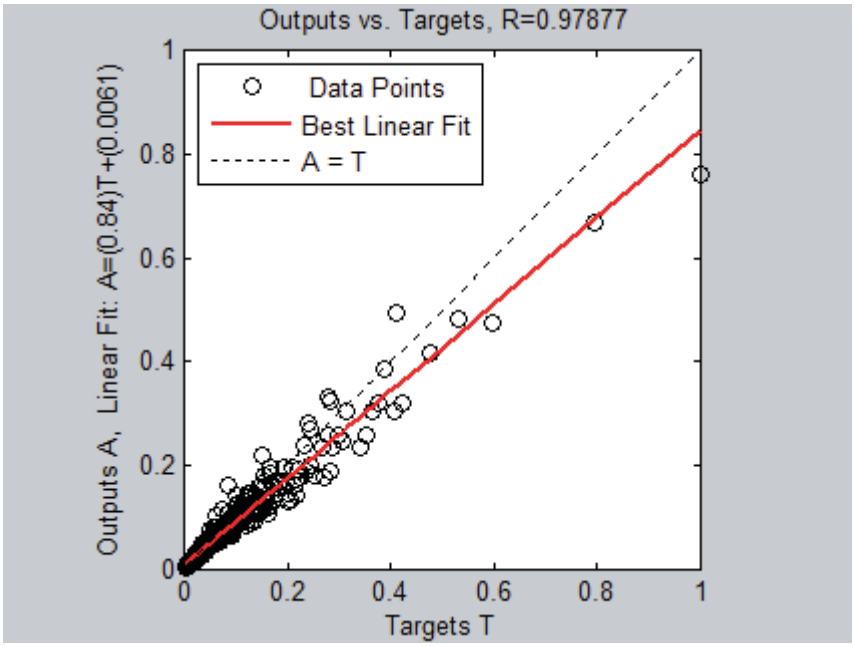

Figura 5-6 Correlación: 80 Partículas, 3 Nodos Ocultos, PSO Clásico.

La tabla 5-3 resume los valores de las métricas para la fase Testing, se puede apreciar buenos resultados.

Tabla 5-3 Testing PSO Clásico: 80 Partículas, 3 Nodos Ocultos

Métrica	Valor
Error Cuadrático Medio (MSE)	0.0006
Raíz Error Cuadrático Medio (RSME)	0.0249
Coeficiente de Correlación(R)	0.9788
Coeficiente de Determinación(R^2)	0.9580
Mape	12.9045

5.3.2 Resultados de PSO con Velocidad Mínima

Manteniendo la base de la configuración descrita en PSO clásico (sección 5.3.1), se agregó el parámetro $V_{\min} = 0.0001$

Tabla 5-4 Resultados Fase Entrenamiento PSO Velocidad Mínima

H	Mape N=20	Mape N=40	Mape N=60	Mape N=80
1	12.8447	12.9919	13.1559	13.2139
2	13.0579	13.0122	13.0310	13.1656
3	13.1854	13.1949	14.8168	13.0357
4	12.9319	13.3211	13.2092	14.9570
5	13.2502	12.9720	13.0105	13.1627
6	13.1196	13.2008	15.3267	13.2106
7	15.6266	13.3629	13.1590	12.4751
8	15.8691	13.3240	13.1675	15.4155
9	15.4931	26.3536	15.9084	13.3727
10	16.1813	15.5327	16.1516	13.0805
11	13.1830	26.7446	15.7926	15.7015

La tabla 5-5 contiene el tiempo en segundos de inicializar el algoritmo (partículas) junto con la realización de una iteración por cada configuración.

Tabla 5-5 Tiempo por cada Iteración en Fase Entrenamiento

H	Tiempo (s) N=20	Tiempo (s) N=40	Tiempo (s) N=60	Tiempo (s) N=80
1	1.754811	3.540959	5.166235	6.888881
2	1.890006	3.728751	5.417722	7.086275
3	1.908640	3.741451	5.715276	7.610023
4	1.977961	3.985979	5.849928	7.840641
5	2.117662	4.036377	6.191703	8.250064
6	2.162600	4.260575	6.412710	8.546648
7	2.296792	4.410781	6.681890	8.947678
8	2.310081	4.579286	6.773811	9.074027
9	2.385157	4.750140	7.191051	9.479938
10	2.514000	4.828585	7.320779	9.752881
11	2.541730	5.012417	7.623933	10.193273

Se puede apreciar que el menor valor del Mape es cuando se utilizó **80 partículas y 7 nodos ocultos.** A continuación se adjunta las gráficas de cada enjambre (N=20, 40, 60, 80) y como se fue comportando el Mape con las diversas cantidades de nodos ocultos con la finalidad de tener una idea más clara sobre cuantas iteraciones son necesarias.

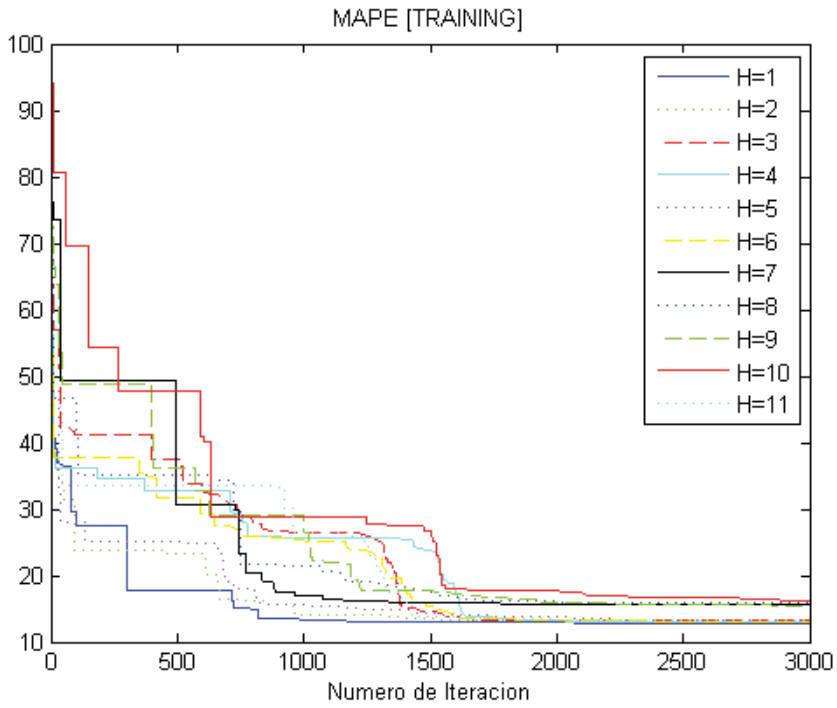

Figura 5-7 Mape de los Nodos Ocultos para un Enjambre 20 partículas.

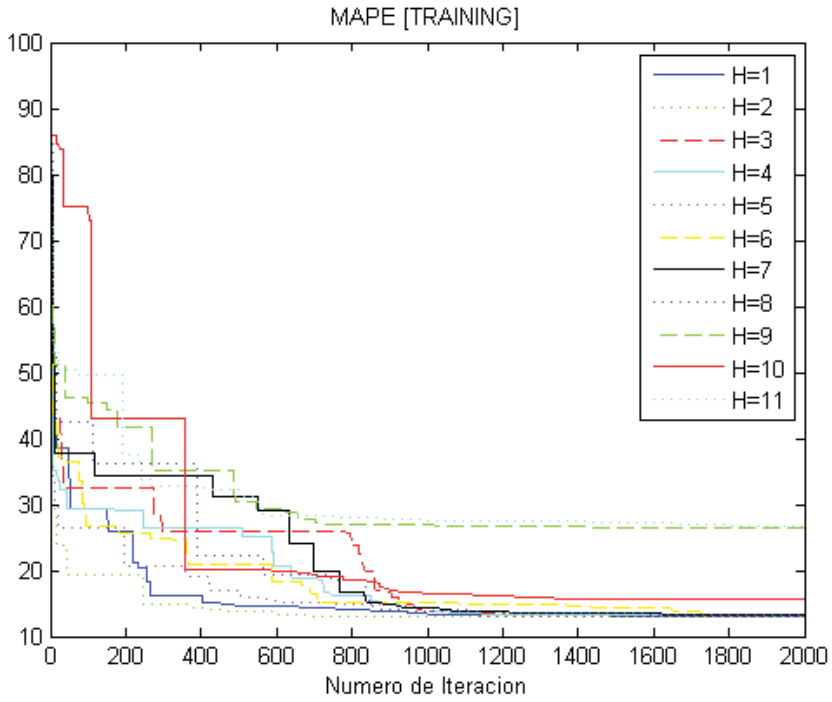

Figura 5-8 Mape de los Nodos Ocultos para un Enjambre 40 partículas.

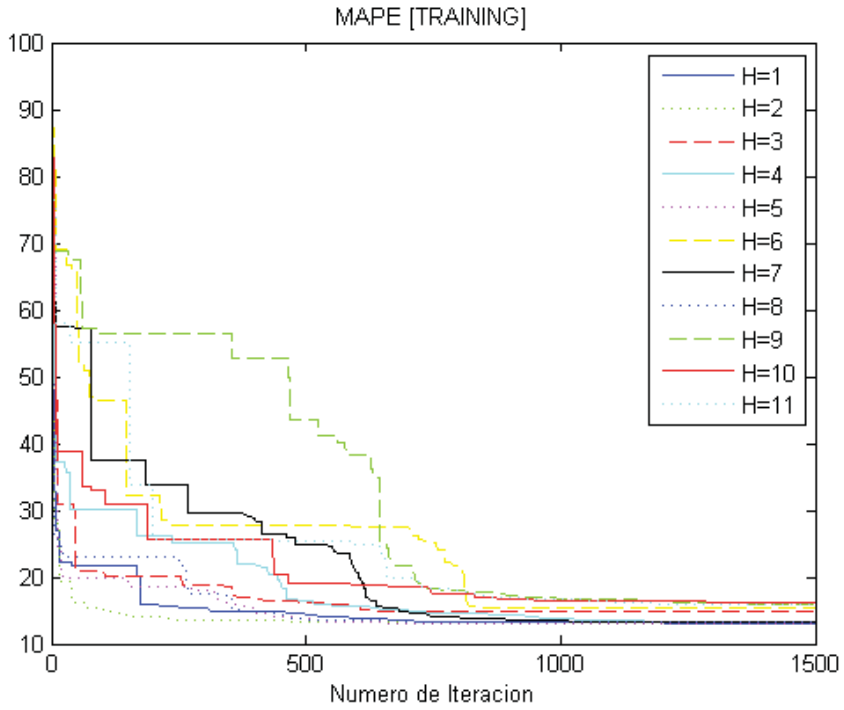

Figura 5-9 Mape de los Nodos Ocultos para un Enjambre 60 partículas.

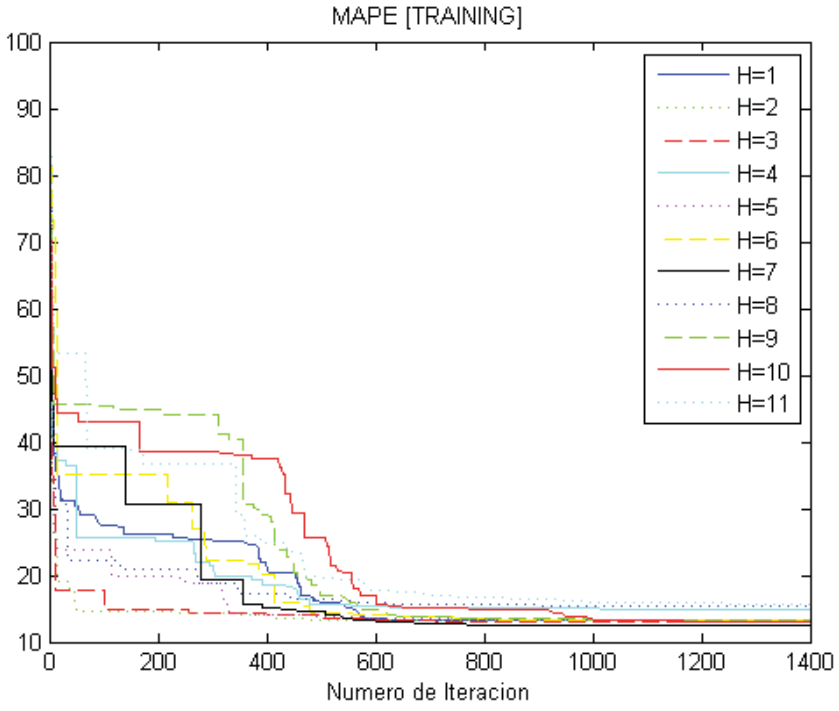

Figura 5-10 Mape de los Nodos Ocultos para un Enjambre 80 partículas.

De las gráficas se puede concluir que con máximo 1700 iteraciones se puede trabajar como punto de partida, para los diversos tamaños de enjambre, aunque puede ser menos (800), se prefiere dejar un margen hacia arriba ya que el número de iteraciones tiene incidencia directa sobre W. Luego un intervalo de trabajo puede ser [800 – 1700] iteraciones.

A continuación se muestra los gráficos asociados a la fase Testing de la mejor configuración obtenida, esto es, **80 partículas y 7 nodos ocultos:**

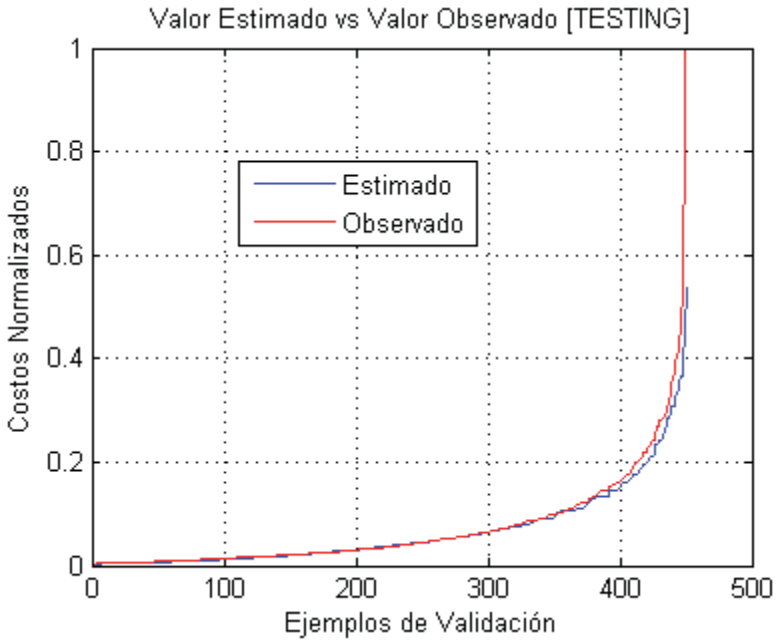

Figura 5-11 Test: 80 Partículas, 7 Nodos Ocultos, PSO Velocidad Mínima.

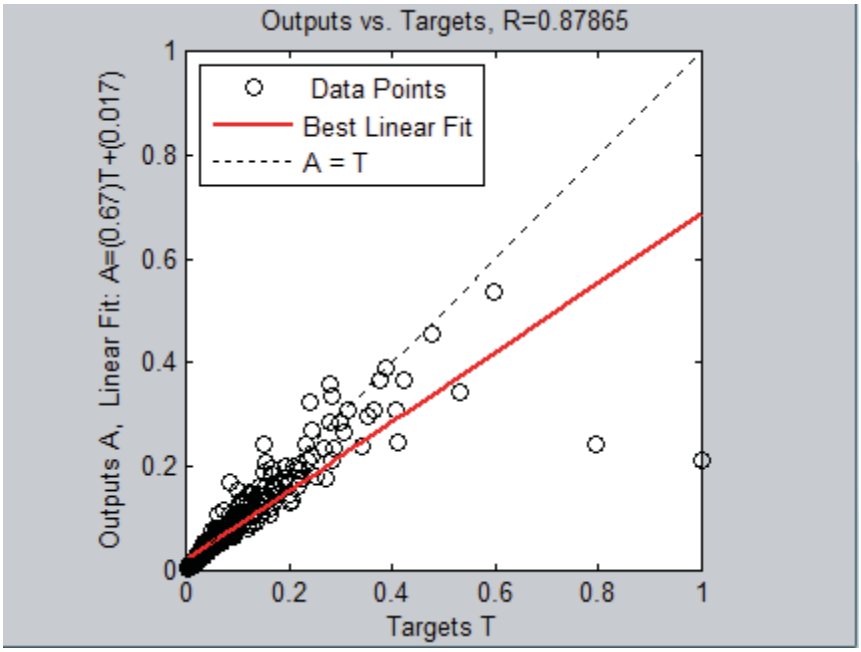

Figura 5-12 Correlación: 80 Partículas, 7 Nodos Ocultos, PSO Velocidad Mínima.

La tabla 5-6 resume los valores de las métricas para la fase Testing, se puede apreciar buenos resultados.

Tabla 5-6 Testing PSO Velocidad Mínima: 80 Partículas, 7 Nodos Ocultos

Métrica	Valor
Error Cuadrático Medio (MSE)	0.0026
Raíz Error Cuadrático Medio (RSME)	0.0507
Coeficiente de Correlación(R)	0.8787
Coeficiente de Determinación(R^2)	0.7730
Mape	13.5839

5.3.3 Resultados de PSO Velocidad Mínima Modificado

Manteniendo la base de la configuración descrita en PSO clásico (sección 5.3.1), se agregó el parámetro $V_{min} = 0.0001$

Tabla 5-7 Resultados Fase Entrenamiento PSO Velocidad Mínima Modificado

H	Mape N=20	Mape N=40	Mape N=60	Mape N=80
1	13.3315	13.4469	13.1780	12.8385
2	12.9298	13.1769	13.1538	13.0634
3	13.0846	12.9058	15.2135	12.8428
4	14.8403	26.2780	13.0573	15.2337
5	13.2149	12.9974	13.0360	13.0567
6	15.3529	13.6278	17.8073	12.9257
7	14.0389	15.8283	26.5117	12.2809
8	13.0366	13.4395	13.0631	15.3768
9	26.5938	26.7397	26.5908	12.5220
10	26.4550	13.3044	13.1874	13.2376
11	14.4924	13.5416	15.7394	13.327

La tabla 5-8 contiene el tiempo en segundos de inicializar el algoritmo (partículas) junto con la realización de una iteración por cada configuración.

Tabla 5-8 Tiempo por cada Iteración en Fase Entrenamiento

H	Tiempo (s) N=20	Tiempo (s) N=40	Tiempo (s) N=60	Tiempo (s) N=80
1	1.740373	3.490304	5.194185	6.901405
2	1.869456	3.621409	5.426374	7.191483
3	1.979713	3.782676	5.712096	7.432518
4	2.018947	3.924404	5.891404	7.789662
5	2.096561	4.116841	6.082605	8.205055
6	2.219784	4.268852	6.310704	8.506720
7	2.268364	4.500003	6.635373	8.721667
8	2.359259	4.580820	6.796136	8.985976
9	2.415452	4.745188	7.152586	9.483251
10	2.521876	4.881715	7.362636	9.732157
11	2.566638	5.054857	7.493197	9.917823

Se puede apreciar que el menor valor del Mape es cuando se utilizó **80 partículas y 7 nodos ocultos.** A continuación se adjunta las gráficas de cada enjambre (N=20, 40, 60, 80) y como se fue comportando el Mape con las diversas cantidades de nodos ocultos con la finalidad de tener una idea más clara sobre cuantas iteraciones son necesarias.

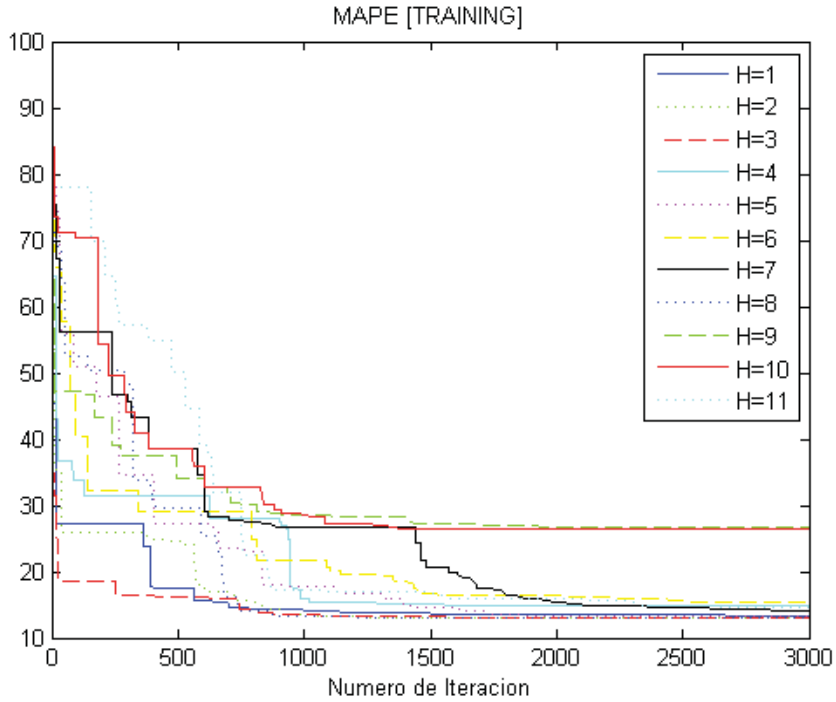

Figura 5-13 Mape de los Nodos Ocultos para un Enjambre 20 partículas.

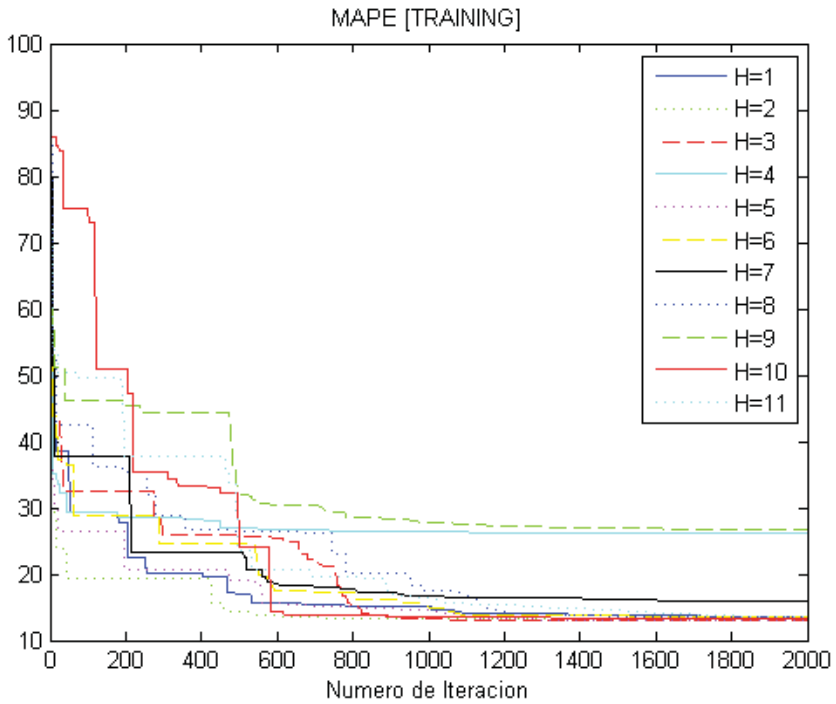

Figura 5-14 Mape de los Nodos Ocultos para un Enjambre 40 partículas.

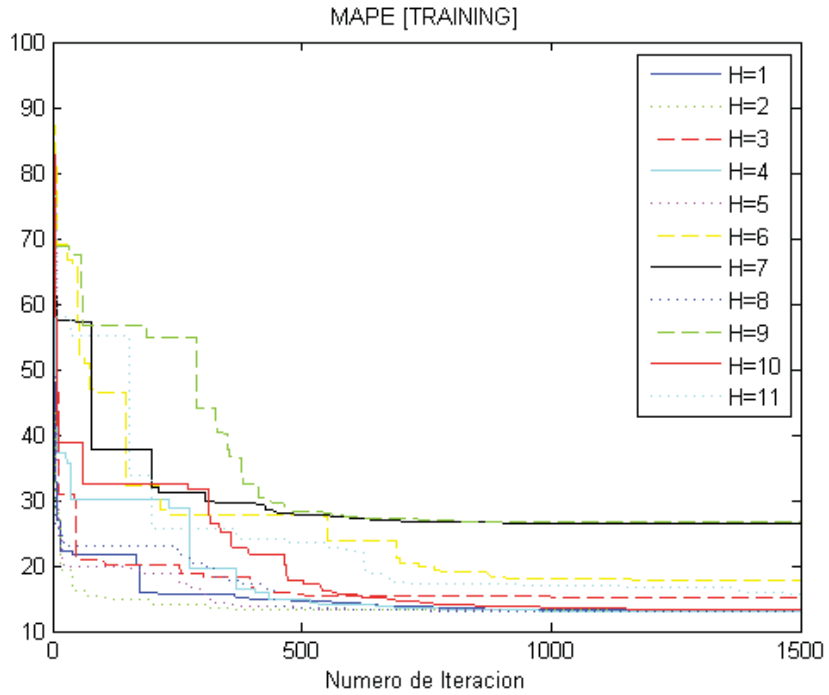

Figura 5-15 Mape de los Nodos Ocultos para un Enjambre 60 partículas.

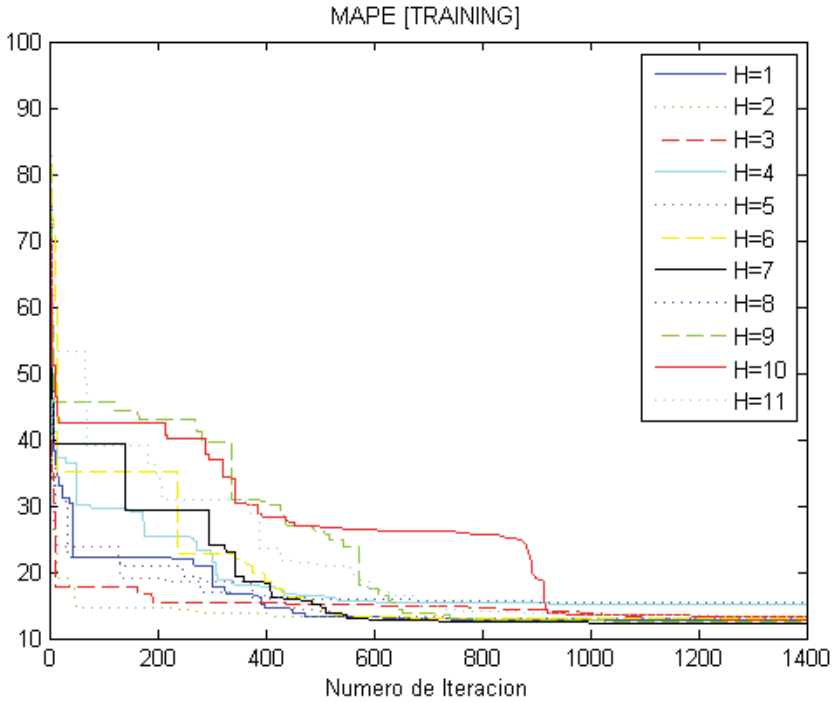

Figura 5-16 Mape de los Nodos Ocultos para un Enjambre 80 partículas.

Se puede concluir que con 1700 iteraciones se puede trabajar como punto de partida, para los diversos tamaños de enjambre, aunque puede ser entre 8000 y 2000, sin embargo se opta por mantener 1700 iteraciones para aplicar a cada algoritmo de aprendizaje la misma cantidad de iteraciones, es decir, intentar en el rango de [800-1700] iteraciones.

A continuación se muestra los gráficos asociados a la fase Testing de la mejor configuración obtenida, esto es, **80 partículas y 7 nodos ocultos:**

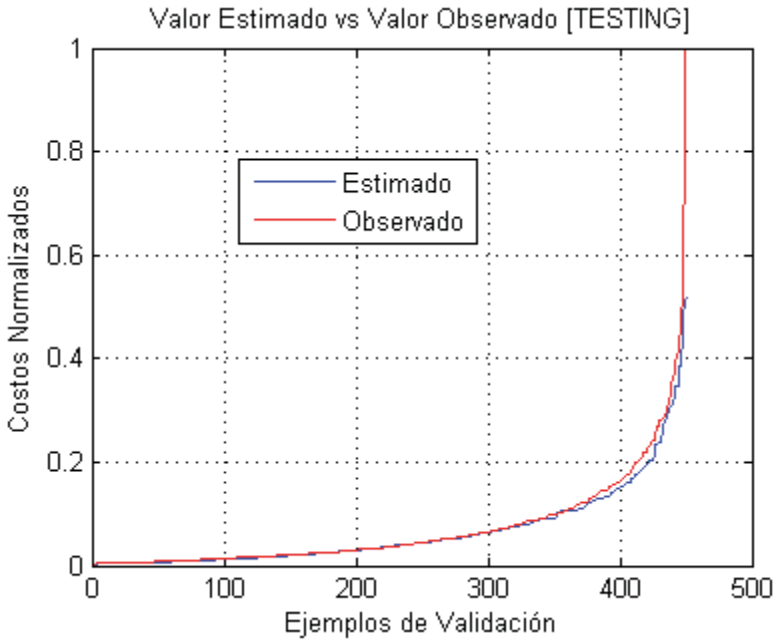

Figura 5-17 Test: 80 Partículas, 7 Nodos Ocultos, PSO Velocidad Mínima Modificado.

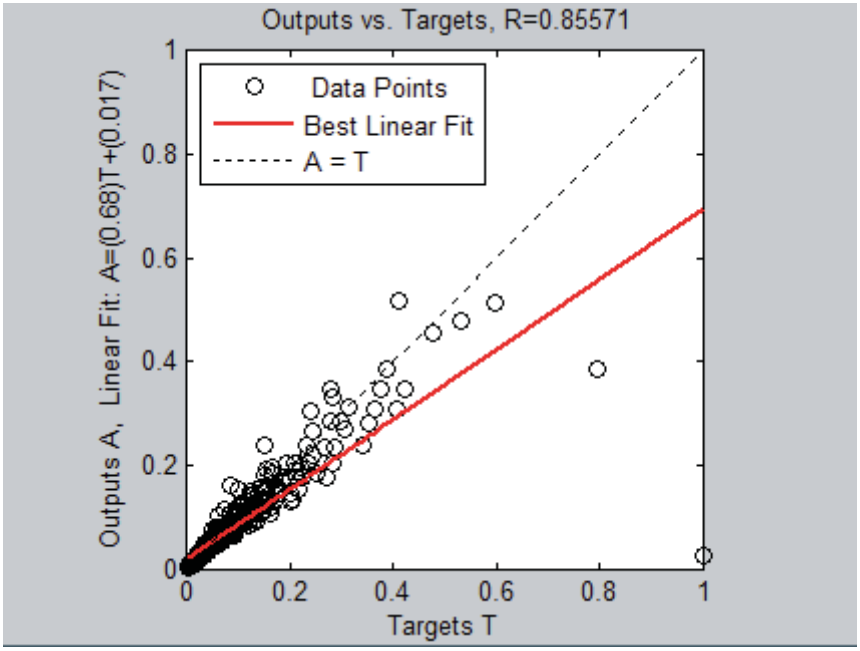

Figura 5-18 Correlación: 80 Partículas, 7 Nodos Ocultos, PSO Velocidad Mínima Modificado.

La tabla 5-9 resume los valores de las métricas para la fase Testing, se puede apreciar buenos resultados.

Tabla 5-9 PSO Velocidad Mínima Modificado: 80 Partículas, 7 Nodos Ocultos

Métrica	Valor
Error Cuadrático Medio (MSE)	0.0029
Raíz Error Cuadrático Medio (RSME)	0.0537
Coeficiente de Correlación(R)	0.8557
Coeficiente de Determinación(R^2)	0.7322
Mape	13.2554

5.3.4 Resultados de QPSO

El parámetro que configura a QPSO está dado por el coeficiente contracción/expansión obtenido mediante la siguiente relación lineal:

$$\beta = (1 - A) \times (MaxIter - iter) / MaxIter + A$$

Donde $A = 0.6$, *MaxIter* es el número máximo de iteraciones e *iter* es la i-esima iteración.

Tabla 5-10 Resultados Fase Entrenamiento QPSO

H	Mape N=20	Mape N=40	Mape N=60	Mape N=80
1	12.8083	18.1405	100	100
2	14.2392	15.6310	17.1472	100
3	13.2150	27.1610	14.2257	16.9182
4	26.9737	18.1492	16.2655	16.3981
5	24.7846	16.3904	24.5973	100
6	15.6259	13.6680	14.8886	17.4498
7	15.3912	16.5105	13.8609	14.2050
8	14.8079	18.4628	21.0528	23.8474
9	16.3845	16.7285	29.9344	14.4846
10	14.4878	20.3962	14.5750	20.4117
11	18.9674	18.8168	18.1517	17.0008

La tabla 5-11 contiene el tiempo en segundos de inicializar el algoritmo (partículas) junto con la realización de una iteración por cada configuración.

Tabla 5-11 Tiempo por cada Iteración en Fase Entrenamiento

H	Tiempo (s) N=20	Tiempo (s) N=40	Tiempo (s) N=60	Tiempo (s) N=80
1	1.745420	3.512437	5.174686	6.971513
2	1.954140	3.639957	5.437239	7.229216
3	1.966684	3.820039	5.775180	7.515574
4	2.141095	4.008745	5.950345	7.977715
5	2.159367	4.209824	6.174169	8.224301
6	2.196670	4.338955	6.428081	8.740460

7	2.355575	4.494690	6.714014	8.950027
8	2.403095	4.697096	6.972627	9.283862
9	2.501225	4.823031	7.328913	9.666669
10	2.574475	4.976379	7.548272	10.043482
11	2.714905	5.218463	7.728368	10.256564

Se puede apreciar que el menor valor del Mape es cuando se utilizó **20 partículas y 1 nodo oculto.** A continuación se adjunta las gráficas de cada enjambre (N=20, 40, 60, 80) y como se fue comportando el Mape con las diversas cantidades de nodos ocultos con la finalidad de tener una idea más clara sobre cuantas iteraciones son necesarias.

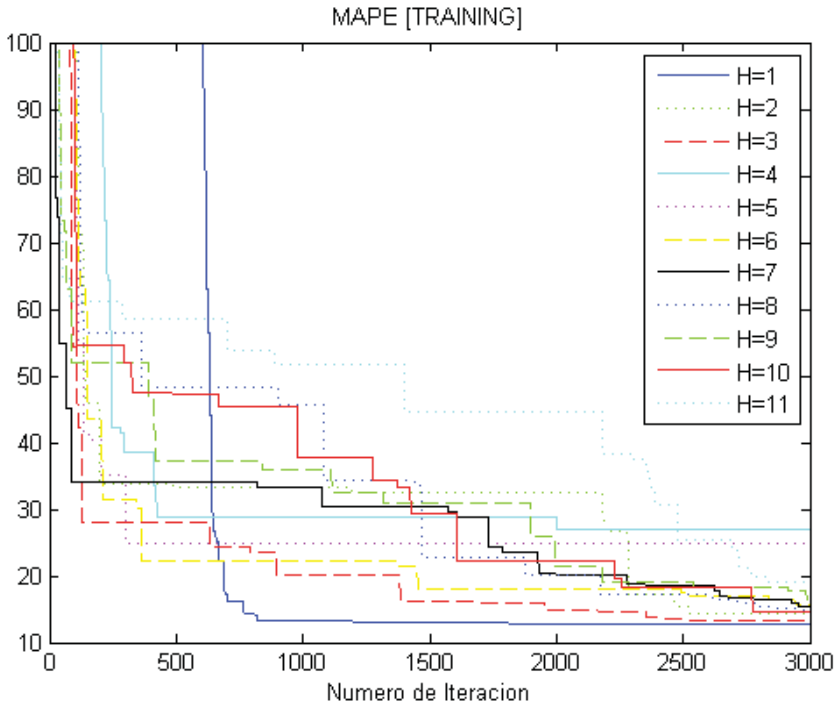

Figura 5-19 Mape de los Nodos Ocultos para un Enjambre 20 partículas.

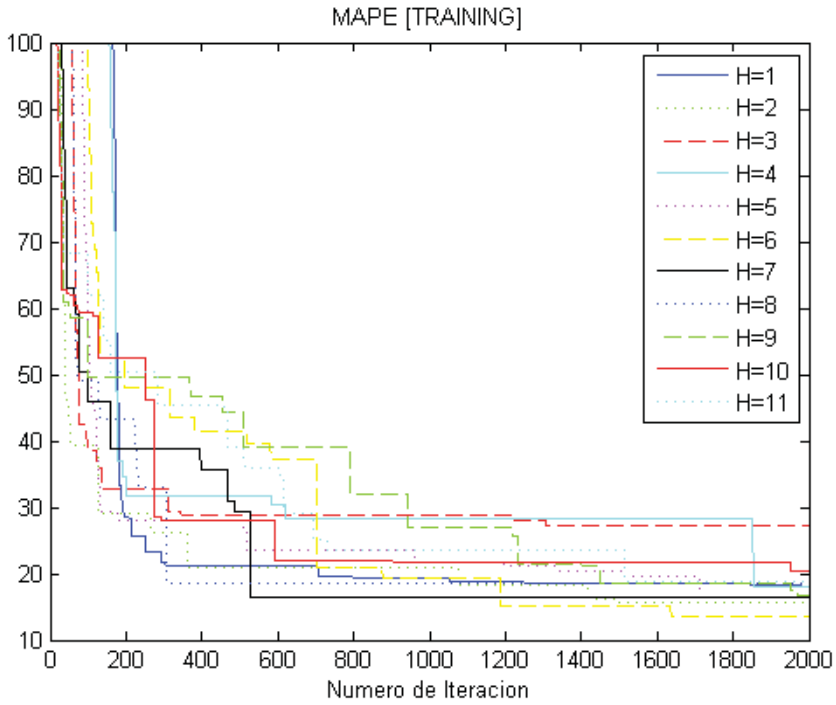

Figura 5-20 Mape de los Nodos Ocultos para un Enjambre 40 partículas

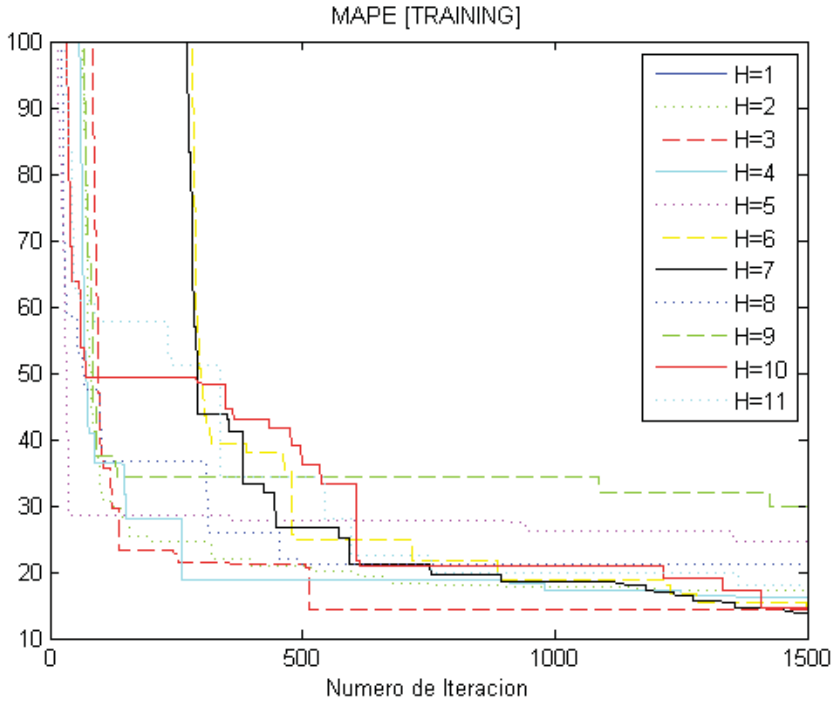

Figura 5-21 Mape de los Nodos Ocultos para un Enjambre 60 partículas.

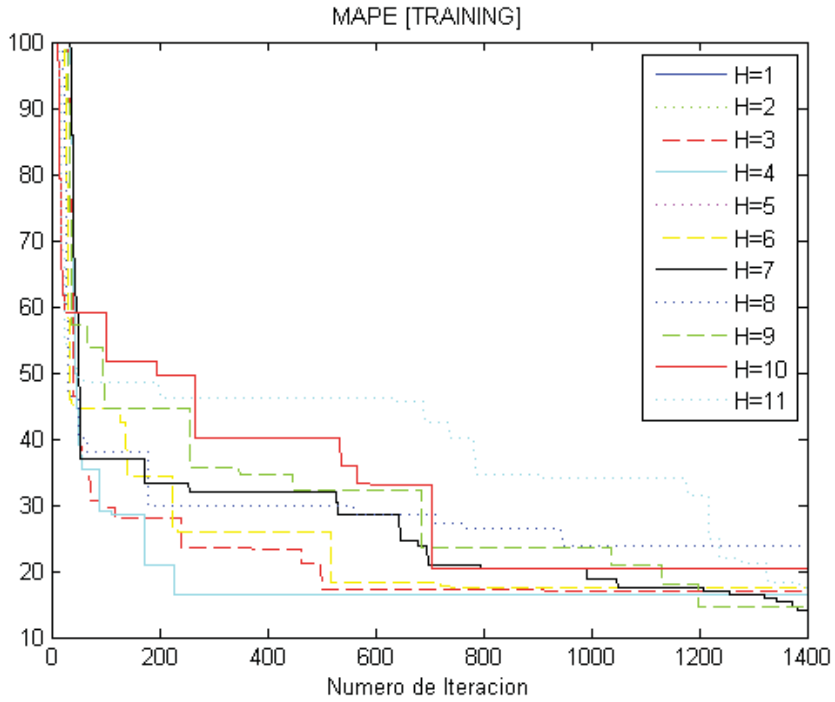

Figura 5-22 Mape de los Nodos Ocultos para un Enjambre 80 partículas.

Se puede concluir que con 2000 iteraciones se puede trabajar como punto de partida, para los diversos tamaños de enjambre, aunque puede ser entre 1700 y 2500.

A continuación se muestra los gráficos asociados a la fase Testing de la mejor configuración obtenida, esto es, **20 partículas y 1 nodo oculto:**

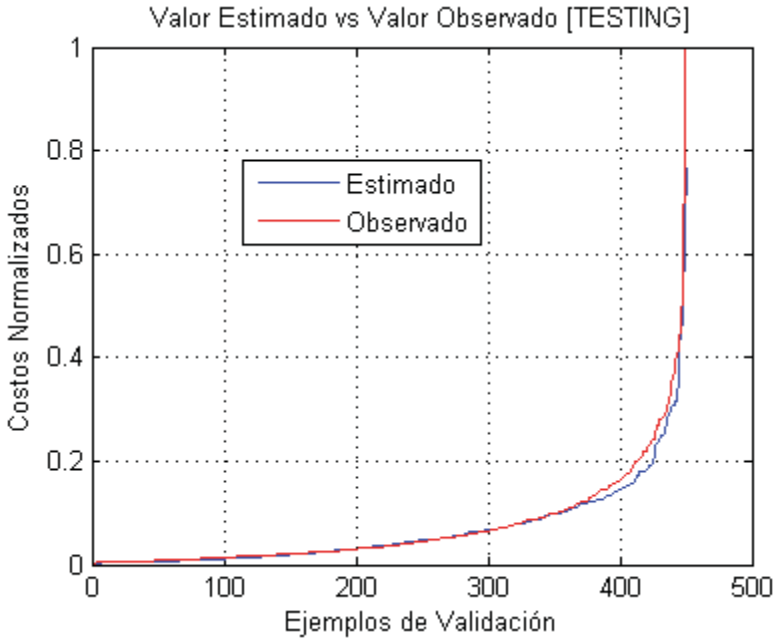

Figura 5-23 Test: 20 Partículas, 1 Nodo Oculto, QPSO

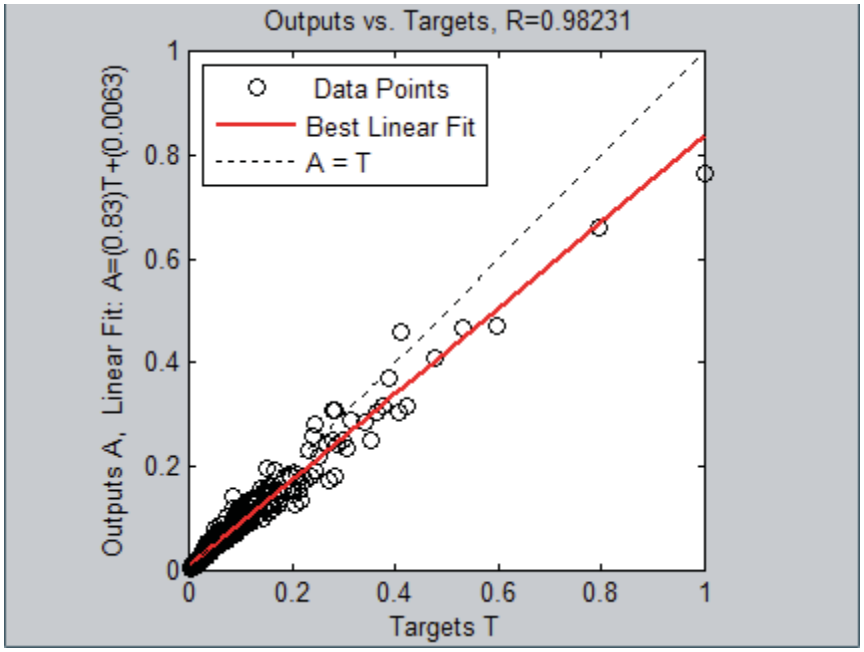

Figura 5-24 Correlación: 20 Partículas, 1 Nodos Ocultos, QPSO.

La tabla 5-12 resume los valores de las métricas para la fase Testing, se puede apreciar buenos resultados.

Tabla 5-12 Testing QPSO: 20 Partículas, 1 Nodo Oculto

Métrica	Valor
Error Cuadrático Medio (MSE)	0.0006
Raíz Error Cuadrático Medio (RSME)	0.0244
Coeficiente de Correlación(R)	0.9823
Coeficiente de Determinación(R^2)	0.9649
Mape	14.6332

5.3.5 Comparación de los Modelos

La tabla 5-13 muestra los resultados para la mejor configuración obtenida por cada algoritmo de aprendizaje en la fase de entrenamiento.

Tabla 5-13 Mejores Resultados Fase Entrenamiento

Indicador	PSO	PSO VelMin	PSO VelMinMod	QPSO
N	80	80	80	20
Iteraciones	1400	1400	1400	3000
H	3	7	7	1
Mape	12.0227	12.4751	12.2809	12.8083
Tiempo (s) Inicialización+1 iteración (Entrenamiento)	7.549111 =3.906835 + 3.642276	8.947678 =4.585089 + 4.362589	8.721667 =4.527420 + 4.194247	1.745420 =0.925914 + 0.819506

La tabla 5-14 muestra los resultados para la mejor configuración obtenida por cada algoritmo de aprendizaje en la fase de prueba o testing.

Tabla 5-14 Resultados de la Fase Testing

Métrica	PSO	PSOVelMin	PSOVelMinMod	QPSO
MSE	0.0006	0.0026	0.0029	0.0006
RSME	0.0249	0.0507	0.0537	0.0244
R	0.9788	0.8787	0.8557	0.9823
R^2	0.9580	0.7730	0.7322	0.9649
Mape	12.9045	13.5839	13.2554	14.6332

Se puede desprender que PSO clásico obtiene el mejor Mape y un mayor equilibrio. La figura 5-5 muestra una buena aproximación, sin embargo, se puede apreciar segmentos donde la curva estimada esta más alejada de la original. Matemáticamente cuando se desea suavizar una curva una alternativa es subir el orden del polinomio, por lo tanto, con la finalidad de lograr suavizar **aun más la curva y por ende reducir el mape** se procedio a aumentar el orden las funciones polinomiales base a el orden 3. Ademas considerando las directrices generadas en el orden 2 se tomo en cuenta el número máximo de iteraciones en 1700 y hasta 7 nodos ocultos por cada enjambre dado que los mejores resultados se encontraron entre 1 a 7 nodos ocultos. Los resultados se aprecian en la tabla 5-15 donde *iter* significa el número de iteraciones:

Tabla 5-15 Resultados Fase Entrenamiento Red -Bspline de orden 3 con PSO

H	Mape N=20 Iter=1700	Mape N=40 Iter=1700	Mape N=60 Iter=1500	Mape N=80 Iter=1400
1	11.4445	10.4867	13.5727	12.2813
2	12.3059	12.8322	11.3999	11.0152
3	10.5073	12.4820	10.5086	12.2828
4	12.3213	10.6607	10.5414	11.1334
5	26.2177	11.3842	11.6906	12.2908
6	17.4718	11.9304	10.4927	10.5889
7	12.9319	10.7183	10.71	12.2795

La tabla 5-16 contiene el tiempo en segundos de inicializar el algoritmo (partículas) junto con la realización de una iteración por cada configuración.

Tabla 5-16 Tiempo por cada Iteración en Fase Entrenamiento

H	Tiempo (s) N=20 Iter=1700	Tiempo (s) N=40 Iter=1700	Tiempo (s) N=60 Iter=1500	Tiempo (s) N=80 Iter=1400
1	3.228045	6.381810	9.473066	12.796082
2	4.766022	9.460228	14.101986	18.824760
3	6.316430	12.686116	18.940332	25.088784
4	7.782468	15.479484	23.368536	31.366767
5	9.341673	18.452259	28.081573	36.792307
6	10.869369	21.514240	33.416632	43.141083
7	12.840670	24.481167	37.762806	49.418464

Los resultados de la tabla 5-15 muestran que al aumentar el grado de 2 a 3 en las funciones bases B-Spline se logra una disminución del mape tanto a nivel general (resultado natural de comparar la tabla 5-15 con la tabla 5-1) como a nivel de mínimo, es decir, se logró reducir el mape a un nuevo valor mínimo. Por lo tanto el nuevo modelo es capaz de suavizar la curva y por ende lograr una **mayor exactitud** en la estimación de costo. Este modelo utilizo 40 partículas, 1700 iteraciones, y 1 nodo oculto, sin embargo el tiempo requerido por iteración **a nivel general es mayor que los algoritmos de orden 2** (resultado natural de comparar la tabla 5-16 con la tabla 5-2, 5-5, ,5-8 y 5-11), sin embargo al comparar solo los tiempos de las mejores configuraciones (tiempo para que convergiera) podemos apreciar los siguientes resultados:

Tabla 5-17 Tiempo que demora en converger el algoritmo

Algoritmo	Tiempo (en minutos)
PSO Orden 2	84,99084931666670
PSOVelMin	101,79745166666700
PSOVelMinMod	97,87131621666670
QPSO	40,97707346666670
PSO Orden 3	87,96235510000000

Se puede desprender que hay poca diferencia entre PSO de orden 2 y orden 3, sin embargo también hay que enfatizar que si PSO de orden 3 hubiese tenido su mejor configuración con 80 partículas y 7 nodos ocultos la diferencia sería abismal, esto hay que tenerlo en cuenta ya que la cantidad de nodos ocultos se determina de manera empírica. Los resultados de la fase de testing (validación) se adjuntan en la tabla 5-18, figura 5-25 y 5-26.

Tabla 5-18 Testing Red Bspline orden 3 con PSO: 40 Partículas, 1 Nodo Oculto

Métrica	Valor
Error Cuadrático Medio (MSE)	0.0007
Raíz Error Cuadrático Medio (RSME)	0.0258
Coeficiente de Correlación(R)	0.9788
Coeficiente de Determinación(R^2)	0.9581
Mape	12.3153

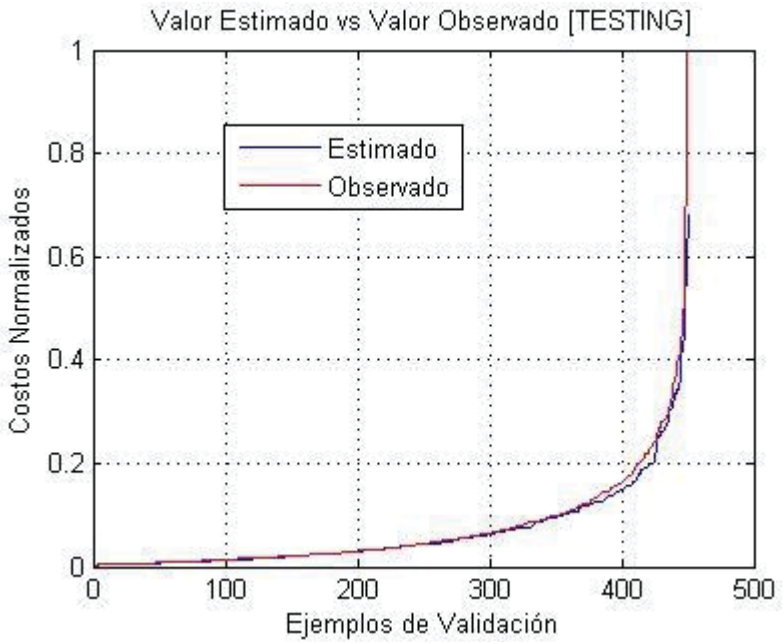

Figura 5-25 Test: 40 Partículas, 1 Nodo Oculto, B-spline Orden 3,PSO

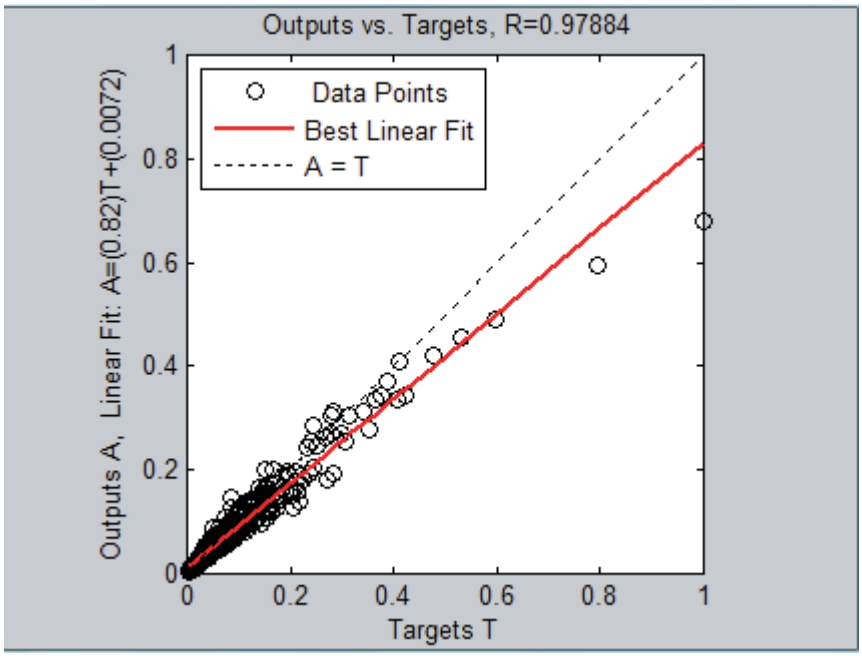

Figura 5-26 Correlación: 40 Partículas, 1 Nodo Oculto, B-spline Orden 3,PSO

6. Conclusión

La importancia de estimar tempranamente los costos de los productos manufacturados radica en que determina hasta un 80% del costo total del producto y también un 80% de su calidad. El análisis convencional se basa a menudo en la estimación directa de la experiencia de los diseñadores, es decir, una determinación subjetiva, sin un conjunto de métodos eficaces y racionales para reducir el error de estimación de costos. Además cabe destacar que en las fases iniciales de diseño hay solo un pequeño conjunto de características que se conocen de los productos a fabricar por lo que la pregunta implícita en esta memoria fue si las redes neuronales serían capaces de dar buenas estimaciones en base a poca información disponible.

Las Redes Neuronales han demostrado ser una buena alternativa para resolver problemas no lineales, particularmente el punto de interés es que permiten resolver problemas donde se tiene poco conocimiento ya que son capaces de aprender de los datos. En este proyecto se desarrollaron modelos neuronales basados en las Redes Neuronales B-Spline ajustando sus parámetros con PSO y 3 variaciones de PSO como mecanismo de aprendizaje en detrimento de optar por mecanismos que utilicen el gradiente descendente, donde la finalidad de estos, fue ver si eran capaces de realizar estimaciones de costos tempranas para la fabricación de elementos de tuberías.

Realizar la etapa de entrenamiento en una red neuronal con técnicas estocásticas es una tarea que requiere muchos ensayos y en definitiva tiempo, dado que no existe una metodología exacta que permita encontrar los parámetros ideales para las condiciones del problema, siendo este hecho la principal desventaja al utilizar este tipo de técnicas evolutivas. Si bien se puede llegar a buenos resultados eso no significa que sea la mejor alternativa al momento de calibrar una red neuronal. Existen técnicas que no tienen necesidad de realizar iteraciones para poder calibrar la red, razón que las transforman en una alternativa muy viable y también para establecer comparaciones. Por otra parte, la principal ventaja de implementar una Red B-Spline es que se puede recurrir a polinomios locales de bajo orden, pero al aumentar el orden de los polinomios implica mayor complejidad y tiempo para converger a la solución.

Los resultados indican que se obtuvieron 4 buenos modelos siendo la Red B-Spline de orden 3 con PSO clásico el que obtuvo una **mayor exactitud**. Su configuración está dada por 40 partículas y 1 nodo oculto, la cual obtuvo el menor mape (10.4876 entrenamiento y 12.3153 en training), su coeficiente de correlación (97.88%) y determinación es bastante alto (95.81%), esto significa que hay una fuerte relación entre la salida estimada y la salida deseada. Un 100% indicaría que están totalmente correlacionadas y se podría esperar un comportamiento funcional idéntico, por lo que se generó un modelo de estimación confiable. Por otra parte los modelos que usan PSO y variaciones de PSO, tienen mayor rapidez de convergencia a nivel general al usar funciones bases de orden 2, dado que el orden de complejidad de las funciones bases es menor, aunque puntualmente en el caso de comparar los mejores resultados o configuraciones dicho tiempo resulte marginal, resulta crucial tener en cuenta la cantidad de nodos ocultos ya que esto determina fuertemente los tiempos de convergencia y puede pasar de un tiempo con diferencia marginal a un tiempo con diferencia bastante considerable.

Con respecto a los objetivos específicos que se buscan del proyecto se han cumplido el 1, 2, 3 y 4 satisfactoriamente, logrando proponer 4 modelos neuronales que permitan realizar una temprana estimación de costos de fabricar tuberías superando la expectativa del objetivo general de proponer un modelo.

El trabajo futuro apunta a mejorar o disminuir el mape de los modelos desarrollados, tomando en cuenta que se puede hacer a los algoritmos de aprendizaje más competitivos, basado en la evidencia recopilada en esta memoria, como disminuir enormemente la cantidad de iteraciones, probar con orden de funciones bases superior, entre otras posibilidades y, analizar y comparar sus resultados con métodos tradicionales de entrenamiento tales como Levenberg-Marquardt, gradiente descendente y filtro de Kalman entre otros.

7. Referencias

[1] Wei, H. Q, *Concurrent design process analysis and optimization for aluminum profile extrusion product development*, International Journal of Advanced Manufacturing Technology, 33(7–8), 652–661, 2007.

[2] Che, Z. H, *PSO-based back-propagation artificial neural network for product and mold cost estimation of plastic injection molding*, Computers and. Industrial Engineering, 58(4), 625–637, 2010.

[3] Piedras, H., Yacout, S., & Savard, G., *Concurrent optimization of customer requirements and the design of a new product*, International Journal of Production Research, 44(20), 2201-2216, 2006.

[4] Niazi, A., Dai, J. S., Balabani, S., & Seneviratne, L., *Product cost estimation: Technique classification and methodology review*, Journal of Manufacturing Science and Engineering, Transactions of the ASME, 128(2), 563-575, 2006.

[5] Yang, C. O., & Lin, T. S., *Developing an integrated framework for feature based early manufacturing cost estimation*, International Journal of Advanced Manufacturing Technology, 13(9), 618-629, 1997.

[6] Rehman, S., & Guenov, M. D, *A methodology for modeling manufacturing costs at conceptual design*, Computers & Industrial Engineering, 35(3–4),623–626, 1998.

[7] Tu, Y. L., Xie, S. Q., & Fung, R. Y. K., *Product development cost estimation in mass customization*. IEEE Transactions on Engineering Management, 54(1), 29–40, 2007

[8] Rush, C., & Roy, R., *Analysis of cost estimating processes used within a concurrent engineering environment throughout a product life cycle*, In 7th ISPE international conference on concurrent engineering: Research and applications, Lyon, France, (pp. 58–67), July 17–20th, PA, 2000.

[9] Duran, o., et al, *Comparisons between two types of neural networks for manufacturing cost estimation of piping elements*, Expert System with Applications, 2012.

[10] Leake, D., *Artificial Intelligence* ,Van Nostrand Scientific Encyclopedia, ninth Edition, 2002

[11] Graham, C., & Smith, S. D., *Estimating the productivity of cyclic construction operations using case-based reasoning*, Advanced Engineering Informatics, 18, 17–28, 2004.

[12] Ficko, M., Drstvensek, I., Brezocnik, M., Balic, J., & Vaupotic, B., *Prediction of total manufacturing costs for stamping tool on the basis of CAD-model of finished product*, Journal of Materials Processing Technology, 164–165, 1327–1335, 2005.

[13] Bode, J., *Neural networks for cost estimation: Simulations and pilot application.* International Journal of Production Research, 38(6), 1231–1254. 2000.

[14] Cavalieri, S., Maccarrone, P., & Pinto, R., *Parametric vs. neural network models for the estimation of production costs: A case study in the automotive industry*. International Journal of Production Economics, 91, 165–177, 2004.

[15] Izaurieta, F., Saavedra, C., *Redes Neuronales Artificiales*, Revista Charlas de Física No. 16, Fac. Ciencias - Universidad de Talca, 0-15, 1999.

[16] Carlson, N.; Fundamentos de psicología fisiológica, Pearson Education, tercera edición, Mexico, 1996.

[17] Floréz, R., Fernandez, J.; *Redes Neuronales Artificiales: Fundamentos teóricos y prácticas, Netbiblio*, Primera edición, España, 2008.
[18] W. McCulloch y W. Pitts, *A logical calculus of the ideas immanent in nervous activity*, Bulletin of Mathematical Biophysics Vol. 7, 1943.
[19] Hebb, D., *Organization of Behavior*, John Wiley & Sons, 1949.
[20] Rosenblatt, F., *The perceptron: A perceiving and recognicing automation*. Technical Report, Cornell Aeronautical Laboratory, 1957.
[21] Widrow, B. *Adaptative sampled-data systems, a statistical theory of adaptation*. IRE WESCON Convention Record, parte 4. New York, Institute of Radio Engineers, 1959.
[22] Minsky. M.L., Paper, S. *Perceptrons*. Cambridge MA, Editorial MIT Press, 1969.
[23] Anderson J.A, Silverstein J.W., Ritz S.A., Jommes R.S. *Distinctive features categorical perception and probability learning: some applications of a neural model.* Psychological Review, Nro 84, 1977.
[24] Kohonen, T. *Self-organization and Associative Memory, Series in Information Sciences*, Editorial Springer-Verlag, Vol. 8, Berlin, 1984
[25] Grossberg, S. *Competitive learning: from interactive activation to adaptative resonance*. Cognitive Science, Nro 11, pag. 23-63, 1987.
[26] Rumelhart D.E., McClelland, J.L & Group, *Parallel Distributed Processing. Explorations in the Microstructure of Cognition*. Cambridge, MA:MIT Press, 1986.
[27] Hoplfield, J.L. *Neural networks and physical systems with emergent collective computational abilities.* Proceedings of the National Academy of Science USA, Nro 79, pag. 2554-2558, 1982.
[28] Jordan, M., *Serial order: A parallel distributed processing approach* Technical report, Institute for Cognitive Science, 1986.
[29] Elman, J., *Finding structure in time*, Vol. 14, Cognitive Science, 1990.
[30] J, Santana., *La Curva de Rendimientos: Una revisión metodológica y nuevas aproximaciones de estimación*, Cuadernos de Economía, v. XXVII, n. 48, Bogotá, pag. 71-113, 2008.
[31] http://es.wikipedia.org/wiki/Spline. Revisada por última vez el 17 de abril del 2012.
[32] Boor, C., *A Practical Guide to Spline*, Springer-Verlag. pp. 113–114, 197.
[33] Mateus, S., *Una técnica de Inteligencia Artificial para el Ajuste de uno de los Elementos que definen una B-Spline Racional No Uniforme (NURBS)* , Politécnico Colombiano JIC, Facultad de Ingenierías, Cra. 48 N° 7-151 P19-146, Medellín-Colombia, 2010.
[34] http://es.wikipedia.org/wiki/B-spline. Revisada por última vez el 17 de abril del 2012.
[35] http://es.scribd.com/doc/51749931/71/Curvas-suaves. Revisada por última vez el 17 de abril del 2012.
[36] Pérez R., *Predicción de Captura de Anchovetas utilizando Redes Neuronales*, Memoria de título, Pontificia Universidad Católica de Valparaíso, diciembre 2007.
[37] Vergara, J., *Red Sigmoidal Recurrente con Aprendizaje Híbrido para el Pronóstico del Volumen de Captura de Anchovetas*, Memoria de título, Pontificia Universidad Católica de Valparaíso, abril 2010.
[38] Pedro Isasi Viñuela e Inés Galván León, *Redes Neuronales Artificiales: Un Enfoque Práctico*, Pearson Prentice Hall, 2004.

[39] http://www.myreaders.info/08_Neural_Networks.pdf. Revisada por última vez el 12 abril 2012.
[40] Aravena, R., *Comparación de MLP-r y MLP-c para aproximación de funciones no lineales*, Memoria de título, Pontificia Universidad Católica de Valparaíso, 2006.
[41] Durán, J., *Optimización por Enjambre de Partículas para Reducción de Distorsión no lineal en Sistemas OFDM*, Memoria de título, Pontificia Universidad Católica de Valparaíso, Diciembre 2007.
[42] J.Kennedy y R. Eberhart, *Particle Swarm Optimization*, Proceedings of the IEEE International Conference on Neural Networks, vol. 4, pages1942-1948, Diciembre 1995.
[43] J. Kennedy, R. Eberhart, Y. Shi, *Swarm Intelligence*, The Morgan Kaufmann Series in Artificial Intelligence, Morgan Kaufmann, San Francisco – California, pages 3-80, April 2001.
[44] A. Abraham, H. Guo y H. Liu, *Swarm Intelligence: Foundations, Perspectives and Applications*, Studies in Computational Intelligence (SCI), Springer-Verlag, vol. 26, pages 3-25, Noviembre 2006.
[45] Magnus Erik Hvass Pedersen, *Good Parameters for Particle Swarm Optimization*, Hvass Laboratories, Technical Report no. HL1001, 2010.
[46] J.Gimmler, T. Stutzle, T.E. Exner *Hybrid Particle Swarm Optimization: An Examination of the Influence of Iterative Improvement Algorithms on Performance*, Lecture Notes in Computer Science (LNCS), Springer-Verlag, vol. 4150, pages 72-83, August 2006.
[47] J.R. Pérez, *Contribución a los Métodos de Optimización Basados en Procesos Naturales y su Aplicación a la Medida de Antenas de Campo Próximo*, a Thesis presented to the University of Cantabria, October 2005.
[48] A-E. Eiben, Z.Michalewicz, M. Schoenauer, J.E Smith, *Parameter Control in Evolutionary Algorithms*, Studies in Computational Intelligence (SCI), Springer-Verlag vol. 54, pages 19-46, April 2007.
[49] A.E Eiben, E. Marchiori, V.A Valkó, *Evolutionary Algorithms with On-the-Fly Population Size Adjustment*, Lecture Notes in Computer Science (LNCS), Springer-Verlag, vol. 3242 pages 41-50, December 2004.
[50] J. Maturana, F. Saubion, *Automated Parameter Control for Evolutionary Algorithms*, 1st Workshop on Autonomous Search, September 2007.
[51] V. Nannen, A.E Eiben, *Relevance Estimation and Value Calibration using MetaAlgorithms*, Proceedings of International Joint Conference on Artificial Intelligence (IJCAI 2007), pages 975-980, January 2007.
[52] A. Khosla, S. Kumar, K.K Aggarwal, J. Singh, *Introducing Lifetime Parameter in Selection Based Particle Swarm Optimization for Improved Performance*, First Indian International Conference on Artificial Intelligence (IICAI-03), Hyderabad, India, Pages 1175-1181, December 2003.
[53] A. Khosla, S.Kumar, K.K Aggarwal, J. Singh, *Particle Swarm for Fuzzy Models Identification*, Studies in Computational Intelligence (SCI), Springer-Verlag, vol. 26, pages 149-173, November 2006.
[54] K.E. Parsopoulos M.N Vrahtis, *Modification of the Particle Swarm Optimizer for Locating all the Global Minima*, Proceedings of the International Conference (Artificial Neuronal Nets and Genetic Algorithms), Springer-Verlag, pages 324-327, April 2007.

[55] X. Xie y W. Zhang, *DEPSO: Hybrid Particle Swarm with Differential Evolution Operator,* IEEE Internat. Conf. on System, vol. 4, pages 3816- 3821, Octubre 2003.

[56] P.L Gilabert, G. Montoro, *Predistorsión Digital y Adaptativa de Amplificadores de RF con Modelos de Hammerstein*, 7th European Conference on Wireless od the Scientific International Union on Radio (URSI' 05 Valencia-Spain), September 2005.

[57] X. Pu, Z. Fang y Y. Liu, *Multilayer Perceptron Networks Training Using Particle Swarm Optimization with Minimum Velocity Constraints*, Lecture Notes in Computer Sciencie (LNCS), Springer-Verlag, vol. 4493, pages 237-245, Julio 2007.

[58] L. Fang, P. Chen, S. Lui, *Particle Swarm Optimization with simulated Annealing for TSP*, Proceedings of the 6th Conference on 6th WSEAS, vol. 6, pages 206-210, February 2007.

[59] J. Liu, J. Sun y W. Xu, *Quantum-Behaved Particle Swarm Optimization with Adaptive Mutation Operator*, Lecture Notes in Computer Science (LNCS), Springer-Verlag, vol. 4221, pages 959-967, Septiembre 2006.

[60] R, Poli., *An Analysis of Publications on Particule Swarm Optimization Applications*, Department of Computer Science ,University of Essex, Technical Report CSM-469 ISSN: 1744-8050, May 2007

Anexo A: Red B-Spline + PSO

```
function mejor_posicion_enjambre =
PSO(iteraciones,particulas,nodos_entradas,nodos_ocultos,entradas_training,d
eseada_training)

dimension=(nodos_ocultos+3)*nodos_entradas+nodos_ocultos; %Donde 3 es el
orden de la función bspline
Wmin = 0.4; % Valor minimo W ==>>> valor de inercia minimo ===>> PESO
valores bajos busqueda localizada
Wmax = 0.9; % Valor máximo W ==>>> valor de inercia maximo ===>> PESO
valores altos busqueda exhaustiva
c1 = 1.4; % Coeficiente de aprendizaje cognitivo, este coeficiente asegura
convergencia
c2 = 1.4; % Coeficiente de aprendizaje social, este coeficiente asegura
convergencia
mape=zeros(iteraciones, 1);
posiciones=rand(particulas,dimension)*0.01;
velocidad=rand(particulas,dimension)*0.01;
mejor_posicion_particula=posiciones;
fitness_mejor_posicion_particula=zeros(particulas,1);
salida_estimada=zeros(1803,1);
L=1803;
% ======================================================
% ============== INICIALIZACION CUMULO ===============
% ======================================================



for i = 1 : particulas
    for j = 1 : dimension
        posiciones(i,j) = rand(1); %almacena posicion de la particula
        mejor_posicion_particula(i,j) = posiciones(i,j); %posicion de la
mejor solucion encontrada
        velocidad(i,j) = rand(1); %direccion en la que se moveran las
particulas, entre el intervalo [-Vmax, Vmax]
    end
    for l=1:L
            %salida_estimada(l)= bs(entradas_training(l,:), posiciones(i,:),
nodos_ocultos , 3);
            salida_estimada(l)= bsOrden3(entradas_training(l,:),
posiciones(i,:), nodos_ocultos);
    end

    %error_cuadratico = (sum((deseada_training - salida_estimada).^2) /
length(deseada_training));
    errorMape = mean((abs(deseada_training -
salida_estimada)./abs(deseada_training))*100);
    fitness_mejor_posicion_particula(i) = errorMape; % Se guarda el error
cuadratico medio para cada particula
end

fitness_mejor_posicion_enjambre = min(fitness_mejor_posicion_particula); %
Se obtiene la particula que tenga menor fitness dentro del enjambre
mape(1)=fitness_mejor_posicion_enjambre ;
```

```
for i = 1 : particulas
    if fitness_mejor_posicion_particula(i) ==
fitness_mejor_posicion_enjambre
        mejor_posicion_enjambre = posiciones(i,:);
        break;
%        i = particulas;
    end
end

% ========================================
% ============== ITERACIONES ==============
% ========================================
for k = 2 : iteraciones

    for i = 1 : particulas
        W = Wmax - ((Wmax - Wmin)/iteraciones)*k;

        for j = 1 : dimension
            % ===== ACTUALIZACION VELOCIDAD  ====
            velocidad(i,j) = W * velocidad(i,j) +
c1*rand(1)*(mejor_posicion_particula(i,j) - posiciones(i,j)) +
c2*rand(1)*(mejor_posicion_enjambre(1,j) - posiciones(i,j)); %
Actualizacion de la velocidad

            % Se comprueba si la velocidad de la particula i se encuentra
            % dentro del rango [-Vmax,Vmax]
            %if(velocidad(i,j) > Vmax)
             %   velocidad(i,j) = Vmax;
            %end
            %if(velocidad(i,j) < -Vmax)
            %    velocidad(i,j) = -Vmax;
            %end

            posiciones(i,j) = posiciones(i,j) + velocidad(i,j);

        end
        for l=1:L
           %salida_estimada(l)= bs(entradas_training(l,:), posiciones(i,:),
nodos_ocultos , 3);
           salida_estimada(l)= bsOrden3(entradas_training(l,:),
posiciones(i,:), nodos_ocultos);
        end
%        error_cuadratico = (sum((deseada_training - salida_estimada).^2) /
length(deseada_training));
        errorMape = mean((abs(deseada_training -
salida_estimada)./abs(deseada_training))*100);

        % Comprobacion de error cuadratico medio para la particula
        if(fitness_mejor_posicion_particula(i) > errorMape)
            fitness_mejor_posicion_particula(i) = errorMape;
            mejor_posicion_particula(i,:) = posiciones(i,:);
        end
```

```
        % Comprobacion de error cuadratico medio para el enjambre
        if(fitness_mejor_posicion_enjambre > errorMape)
            fitness_mejor_posicion_enjambre = errorMape;
            for d=1:dimension
                mejor_posicion_enjambre(1,d)= posiciones(i,d);
            end
        end

    end
    mape(k)=fitness_mejor_posicion_enjambre;
    fprintf('(%d)\n', k);
end
fprintf('Mape: %g',mape(k-1));
fOut =
sprintf('Orden3/Config_Training_PSO/Part%dIter%d/Part%d_NodOcu%d_Iter%d',pa
rticulas,iteraciones,particulas,nodos_ocultos,iteraciones);
save (fOut , 'mape' , 'mejor_posicion_enjambre');

scrsz = get(0,'ScreenSize');
figure('Name','MAPE [TRAINING]','Position',[1 1 scrsz(3)/3 scrsz(4)/2.5])
plot(1:iteraciones, mape,'b'), xlabel('Numero de Iteracion'), title('MAPE
[TRAINING]');
legend('MAPE');
```

Anexo B: Red B-Spline + PSO vel Min

```
function
PSOVELMin(iteraciones,particulas,nodos_entradas,nodos_ocultos,entradas_trai
ning,deseada_training)

dimension=(nodos_ocultos+2)*nodos_entradas+nodos_ocultos;
Wmin = 0.4; % Valor minimo W ==>>> valor de inercia minimo ===>> PESO
valores bajos busqueda localizada
Wmax = 0.9; % Valor máximo W ==>>> valor de inercia maximo ===>> PESO
valores altos busqueda exhaustiva
c1 = 1.4; % Coeficiente de aprendizaje cognitivo, este coeficiente asegura
convergencia
c2 = 1.4; % Coeficiente de aprendizaje social, este coeficiente asegura
convergencia
mape=zeros(iteraciones, 1);
posiciones=rand(particulas,dimension)*0.01;
velocidad=rand(particulas,dimension)*0.01;
mejor_posicion_particula=posiciones;
fitness_mejor_posicion_particula=zeros(particulas,1);
salida_estimada=zeros(1803,1);
Vmin=0.0001;
L=1803;
% ======================================================
% ============== INICIALIZACION CUMULO ===============
% ======================================================



for i = 1 : particulas
    for j = 1 : dimension
        posiciones(i,j) = rand(1); %almacena posicion de la particula
        mejor_posicion_particula(i,j) = posiciones(i,j); %posicion de la
mejor solucion encontrada
        velocidad(i,j) = rand(1); %direccion en la que se moveran las
particulas, entre el intervalo [-Vmax, Vmax]
    end
    for l=1:L
           salida_estimada(l)= bs(entradas_training(l,:), posiciones(i,:),
nodos_ocultos , 3);
    end

    %error_cuadratico = (sum((deseada_training - salida_estimada).^2) /
length(deseada_training));
    errorMape = mean((abs(deseada_training -
salida_estimada)./abs(deseada_training))*100);
    fitness_mejor_posicion_particula(i) = errorMape; % Se guarda el error
cuadratico medio para cada particula
end

fitness_mejor_posicion_enjambre = min(fitness_mejor_posicion_particula); %
Se obtiene la particula que tenga menor fitness dentro del enjambre
mape(1)=fitness_mejor_posicion_enjambre ;
```

```
for i = 1 : particulas
    if fitness_mejor_posicion_particula(i) ==
fitness_mejor_posicion_enjambre
        mejor_posicion_enjambre = posiciones(i,:);
        break;
%        i = particulas;
    end
end

% =========================================
% ============== ITERACIONES ==============
% =========================================
for k = 2 : iteraciones

    for i = 1 : particulas
        W = Wmax - ((Wmax - Wmin)/iteraciones)*k;

        for j = 1 : dimension
            % ===== ACTUALIZACION VELOCIDAD  ====
            velocidad(i,j) = W * velocidad(i,j) +
c1*rand(1)*(mejor_posicion_particula(i,j) - posiciones(i,j)) +
c2*rand(1)*(mejor_posicion_enjambre(1,j) - posiciones(i,j)); %
Actualizacion de la velocidad
            if(velocidad(i,j) < Vmin && velocidad(i,j)> -Vmin)
                %Caso Promedio todo el enjambre
                sumaParticulas=sum(velocidad);
                sumaEnjambre=sum(sumaParticulas);
                promedioEnjambre=sumaEnjambre/(dimension*particulas);
                velocidad(i,j) = promedioEnjambre*sign(velocidad(i,j));
            end
            posiciones(i,j) = posiciones(i,j) + velocidad(i,j);

        end
        for l=1:L
           salida_estimada(l)= bs(entradas_training(l,:), posiciones(i,:),
nodos_ocultos , 3);
        end
%         error_cuadratico = (sum((deseada_training - salida_estimada).^2) /
length(deseada_training));
        errorMape = mean((abs(deseada_training -
salida_estimada)./abs(deseada_training))*100);

        % Comprobacion de error cuadratico medio para la particula
        if(fitness_mejor_posicion_particula(i) > errorMape)
            fitness_mejor_posicion_particula(i) = errorMape;
            mejor_posicion_particula(i,:) = posiciones(i,:);
        end

        % Comprobacion de error cuadratico medio para el enjambre
        if(fitness_mejor_posicion_enjambre > errorMape)
            fitness_mejor_posicion_enjambre = errorMape;
            for d=1:dimension
                mejor_posicion_enjambre(1,d)= posiciones(i,d);
            end
```

```
        end

    end
    mape(k)=fitness_mejor_posicion_enjambre;
end

fOut =
sprintf('Config_Training_PSOVELMin/Part%dIter%d/Part%d_NodOcu%d_Iter%d',par
ticulas,iteraciones,particulas,nodos_ocultos,iteraciones);
save (fOut , 'mape' , 'mejor_posicion_enjambre');

scrsz = get(0,'ScreenSize');
figure('Name','MAPE [TRAINING]','Position',[1 1 scrsz(3)/3 scrsz(4)/2.5])
plot(1:iteraciones, mape,'b'), xlabel('Numero de Iteracion'), title('MAPE
[TRAINING]');
legend('MAPE');
```

Anexo C: Red B-Spline + PSO Vel Mínima Modificado

```
function
PSOVelMinModificado(iteraciones,particulas,nodos_entradas,nodos_ocultos,ent
radas_training,deseada_training)

dimension=(nodos_ocultos+3)*nodos_entradas+nodos_ocultos; %Donde 3 es el
orden de la bspline
Wmin = 0.4; % Valor minimo W ==>>> valor de inercia minimo ===>> PESO
valores bajos busqueda localizada
Wmax = 0.9; % Valor máximo W ==>>> valor de inercia maximo ===>> PESO
valores altos busqueda exhaustiva
c1 = 1.4; % Coeficiente de aprendizaje cognitivo, este coeficiente asegura
convergencia
c2 = 1.4; % Coeficiente de aprendizaje social, este coeficiente asegura
convergencia
mape=zeros(iteraciones, 1);
posiciones=rand(particulas,dimension)*0.01;
velocidad=rand(particulas,dimension)*0.01;
mejor_posicion_particula=posiciones;
fitness_mejor_posicion_particula=zeros(particulas,1);
salida_estimada=zeros(1803,1);
Vmin=0.07;
L=1803;
% ====================================================
% ============== INICIALIZACION CUMULO ===============
% ====================================================



for i = 1 : particulas
    for j = 1 : dimension
        posiciones(i,j) = rand(1); %almacena posicion de la particula
        mejor_posicion_particula(i,j) = posiciones(i,j); %posicion de la
mejor solucion encontrada
        velocidad(i,j) = rand(1); %direccion en la que se moveran las
particulas, entre el intervalo [-Vmax, Vmax]
    end
    for l=1:L
           %salida_estimada(l)= bs(entradas_training(l,:), posiciones(i,:),
nodos_ocultos , 3);
           salida_estimada(l)= bsOrden3(entradas_training(l,:),
posiciones(i,:), nodos_ocultos);
    end

    %error_cuadratico = (sum((deseada_training - salida_estimada).^2) /
length(deseada_training));
    errorMape = mean((abs(deseada_training -
salida_estimada)./abs(deseada_training))*100);
    fitness_mejor_posicion_particula(i) = errorMape; % Se guarda el error
cuadratico medio para cada particula
end

fitness_mejor_posicion_enjambre = min(fitness_mejor_posicion_particula); %
Se obtiene la particula que tenga menor fitness dentro del enjambre
```

```
mape(1)=fitness_mejor_posicion_enjambre ;

for i = 1 : particulas
    if fitness_mejor_posicion_particula(i) ==
fitness_mejor_posicion_enjambre
        mejor_posicion_enjambre = posiciones(i,:);
        break;
%        i = particulas;
    end
end

% ========================================
% ============== ITERACIONES ==============
% ========================================
for k = 2 : iteraciones

    for i = 1 : particulas
        W = Wmax - ((Wmax - Wmin)/iteraciones)*k;

        for j = 1 : dimension
            % ===== ACTUALIZACION VELOCIDAD  ====
            velocidad(i,j) = W * velocidad(i,j) +
c1*rand(1)*(mejor_posicion_particula(i,j) - posiciones(i,j)) +
c2*rand(1)*(mejor_posicion_enjambre(1,j) - posiciones(i,j)); %
Actualizacion de la velocidad
            if(velocidad(i,j) < Vmin && velocidad(i,j)> -Vmin)
                %Caso Promedio Dimensión.
                promedioDimensionEnjambre=mean(velocidad(:,j));
                velocidad(i,j) =
promedioDimensionEnjambre*sign(velocidad(i,j));
            end
            posiciones(i,j) = posiciones(i,j) + velocidad(i,j);

        end
        for l=1:L
           %salida_estimada(l)= bs(entradas_training(l,:), posiciones(i,:),
nodos_ocultos , 3);
           salida_estimada(l)= bsOrden3(entradas_training(l,:),
posiciones(i,:), nodos_ocultos);
        end
%        error_cuadratico = (sum((deseada_training - salida_estimada).^2) /
length(deseada_training));
        errorMape = mean((abs(deseada_training -
salida_estimada)./abs(deseada_training))*100);

        % Comprobacion de error cuadratico medio para la particula
        if(fitness_mejor_posicion_particula(i) > errorMape)
            fitness_mejor_posicion_particula(i) = errorMape;
            mejor_posicion_particula(i,:) = posiciones(i,:);
        end

        % Comprobacion de error cuadratico medio para el enjambre
        if(fitness_mejor_posicion_enjambre > errorMape)
            fitness_mejor_posicion_enjambre = errorMape;
            for d=1:dimension
```

```
                mejor_posicion_enjambre(1,d)= posiciones(i,d);
            end
        end

    end
    mape(k)=fitness_mejor_posicion_enjambre;
    fprintf('Mape: %g\n ', mape(k));
end
fOut =
sprintf('Orden3/Config_Training_PSOVelMinModificado/Part%dIter%d/Part%d_Nod
Ocu%d_Iter%d',particulas,
iteraciones,particulas,nodos_ocultos,iteraciones);
save (fOut , 'mape' , 'mejor_posicion_enjambre');

scrsz = get(0,'ScreenSize');
figure('Name','MAPE [TRAINING]','Position',[1 1 scrsz(3)/3 scrsz(4)/2.5])
plot(1:iteraciones, mape,'b'), xlabel('Numero de Iteracion'), title('MAPE
[TRAINING]');
legend('MAPE');
```

Anexo D: Red B-Spline + QPSO

```
function
QPSO(iteraciones,particulas,nodos_entradas,nodos_ocultos,entradas_training,
deseada_training)

L=1803;
salida_estimada=zeros(L,1);
SSE=zeros(particulas,1);
NewSSE=zeros(particulas,1);

MaxIter    = iteraciones;
A          = 0.6;
MSE        = zeros(MaxIter,1);
Dim        = (nodos_ocultos+3)*nodos_entradas+nodos_ocultos;% Dimensión del
problema (número de parámetros del modelo) donde 3 es el orden de la
funcion
X          = rand(particulas,Dim)*0.01;     % Enjambre de partículas N*Dim
pbest      = X;                    % Mejor Partícula anterior inicial


for i = 1 : particulas
    for l=1:L
          %salida_estimada(l)= bs(entradas_training(l,:), X(i,:),
nodos_ocultos , 3);
          salida_estimada(l)= bsOrden3(entradas_training(l,:), X(i,:),
nodos_ocultos);
    end
    SSE(i) =mean((abs(deseada_training -
salida_estimada)./abs(deseada_training))*100);
end

% Valores iniciales del Global Best y el mejor anterior
[MinSSE,g] = min(SSE);
gbest=pbest(g,:);
MSE(1)     = MinSSE;


for (iter=2:MaxIter)

    Alfa = (1.0-A)*(MaxIter-iter)/MaxIter + A;
    for (i=1:particulas)

        % Media de la mejor población
        mBest = mean(pbest);

        for(j=1:Dim)

            r1 = rand;
            r2 = rand;

            % pbest es la mejor particula previa y gbest es la mejor
particula global
```

```
            p = r1*pbest(i,j) + (1-r1)*gbest(j);

            % Asignando posicion de la partícula
            if (rand>=0.5)
                X(i,j) = p - Alfa * abs(mBest(j)-X(i,j))*log(1/r2);
            else
                X(i,j) = p + Alfa * abs(mBest(j)-X(i,j))*log(1/r2);
            end

        end
    end

    for i = 1 : particulas
        for l=1:L
            %salida_estimada(l)= bs(entradas_training(l,:), X(i,:),
nodos_ocultos , 3);
            salida_estimada(l)= bsOrden3(entradas_training(l,:), X(i,:),
nodos_ocultos);
        end
        NewSSE(i) =mean((abs(deseada_training -
salida_estimada)./abs(deseada_training))*100);
    end


    % Se obtienen los índices con los errores menores de la iteración
anterior.
    Index = find(NewSSE<SSE);

    if(isempty(Index)~=1)
        pbest(Index,:) = X(Index,:);
        SSE(Index) = NewSSE(Index);
    end

    [NewMinSSE,g] = min(SSE);
    MSE(iter) = NewMinSSE;

    % Actualizando mejor Global Best encontrado
    if (NewMinSSE < MinSSE)
        MinSSE = NewMinSSE;
        gbest = pbest(g,:);
    end
    fprintf('Mape: %g\n', MSE(iter));
end
mape=MSE;
fOut =
sprintf('Orden3/Config_Training_Qpso/Part%dIter%d/Part%d_NodOcu%d_Iter%d',p
articulas,iteraciones,particulas,nodos_ocultos,iteraciones);
save (fOut , 'mape' , 'gbest');
scrsz = get(0,'ScreenSize');
figure('Name','MAPE [TRAINING]','Position',[1 1 scrsz(3)/3 scrsz(4)/2.5])
plot(1:iteraciones, mape,'b'), xlabel('Numero de Iteracion'), title('MAPE
[TRAINING]');
legend('MAPE');
```

Anexo E: Función de Activación B-Spline Orden 3

```
function y= bsOrden3(X,R,H)
%BSPLINe(l)= bs(training(l,:), Pesos(:,p),H,VectorKnot);
%Autor:Lenin Gonzalez
%Argumentos de la Funcion B-Spline
%- X = Vector de datos de entrada.
%- R = Partículas para una configuración i-esima
%- H = Numero de Nodos Ocultos.

%Separacion del Vector P
%           - W = Vector de Pesos de Entrada.
%           - V = Vector de Pesos de Salida.


E=length(X); %Numero de entradas
P=R;
lambda=zeros(E,H+3);%POR CADA ENTRADA GENERE UN "VECTOR KNOT" . y 3 es el
orden de la bspline
%POR EJEMPLO para la entrada X1, lambda(1,:)=[L1,L2,L3,L4,L5,L6] CASO DE 4
NODOS
%DONDE EL PRIMER NODO TOMARA L1,L2,L3... EL SEGUNDO NODO TOMARA L2,L3,L4 Y
%ASI SUCESIVAMENTE
inicio=1;
fin=H+3; %donde 3 es el orden de la función
for j=1:E;
          lambda(j,:)=sort(P(inicio:fin)); %Saco el pedazo que quiero y
ademas ordena de forma creciente los lambda
          inicio=fin+1; %Voy generando los pedazos subsiguientes...
          fin=fin+(H+3); %El ultimo elemento o lambda...
end
V=P(inicio:length(P)); %Registro los pesos.
y=0;
for j=1:H
    NTj=1;
    for i=1:E
        denominador1=lambda(i,j+2)-lambda(i,j);%bien
        denominador2=lambda(i,j+3)-lambda(i,j+1);%bien
        termino1=0;
        termino2=0;
        if(denominador1 ~= 0) %IMPLICA QUE ALGUNA FUNCION MONOVARIABLE EL
DENOMIDADOR DE LOS LAMBDAS DIO != 0  Y POR ende no es infinito
            termino1=(( X(i)-lambda(i,j) ) /denominador1);
        end
        if(denominador2 ~= 0)
            termino2=(( lambda(i,j+3)- X(i) ) /denominador2);
        end
        bs1=bsOrden2(X(i),lambda(i,(j:j+2))); %estos terminos son los de
bspline orden 2
        bs2=bsOrden2(X(i),lambda(i,(j+1:j+3))); %estos terminos son los de
bspline orden 2
        Nj= termino1*bs1+ termino2*bs2;%aca multiplicar por la de orden 2
        NTj=Nj*NTj; %Producto tensorial de cada funcion monovariable;
    end
```

```
    y=y+NTj*V(j); %LA COMBINACION DE LAS FUNCIONES LINEALES, COMPUESTAS DE
H FUNCIONES MULTIVARIABLES Y ESTAS, A SU VEZ, DESCOMPUESTAS EN VARIAS
MONOVARIABLES
end

function Nj= bsOrden2(X,lambda)

constante1=0;
constante2=0;
if(X>=lambda(1) && (X< lambda(2) ) )
    constante1=1;
end
if(X>=lambda(2) && (X< lambda(3) ) )
    constante2=1;
end
denominador1=lambda(2)-lambda(1);
denominador2=lambda(3)-lambda(2);
termino1=0;
termino2=0;
if(denominador1 ~= 0) %IMPLICA QUE ALGUNA FUNCION MONOVARIABLE EL
DENOMIDADOR DE LOS LAMBDAS DIO != 0  Y POR ende no es infinito
    termino1=(( X-lambda(1) ) /denominador1)*constante1;
end
if(denominador2 ~= 0)
    termino2=(( lambda(3)- X ) /denominador2)* constante2;
end
Nj= termino1+ termino2;
```

Anexo F: Testing

```
clear all
clc
%--------------------------------------------------------------------------
%Sección que carga los datos necesarios para el proceso de training.
load DataLista.mat;
x=1:450; % Coordenas para el eje X.
%--------------------------------------------------------------------------

%Tomar la mejor configuración y copiar los datos a las variables:
nodos_ocultos=1;
%Se copia el enjambre con mejor configuracion
mejor_posicion_enjambre=[1230.6025,-0.0199,1.0932,0.2413,958.4498,-
0.2433,2.2014,4294688.2361,-0.3458,5576.4841,2.153,-5.646,1.8545];

%Salida estimada TESTING
salida_estimada=zeros(450,1);
for l=1:450
          salida_estimada(l)= bs(testing (l,:), mejor_posicion_enjambre,
nodos_ocultos , 3);
end

% Grafico de valor estimado vs valor observado
scrsz = get(0,'ScreenSize');
figure('Name','Valor Estimado vs Valor Observado [TESTING]','Position',[1 1
scrsz(3)/3 scrsz(4)/2.5])
plot(x, sort(salida_estimada),'b-', x , sort(y_test), 'r-') ,
xlabel('Ejemplos de Validación'), ylabel('Costos Normalizados')
title('Valor Estimado vs Valor Observado [TESTING]') ,
legend('Estimado','Observado') , grid on

% Calculo error cuadratico medio
error_cuadratico = (sum((y_test - salida_estimada).^2) / length(y_test));
fprintf('\n | Error Cuadratico Medio                                    -->
MSE   = %.4f',error_cuadratico);

% Calculo raiz error cuadratico medio
RMSE=sqrt(error_cuadratico);
fprintf('\n | Raiz Error Cuadratico Medio                               -->
MSE   = %.4f',RMSE);

% Calculo de R^2
figure('Name','Correlación Pronostico vs Deseado
[TESTING]','Position',[scrsz(3)/3 1 scrsz(3)/3 scrsz(4)/2.5])
[M,B,R]=postreg(salida_estimada', y_test'); % Se obtiene el coeficiente de
correlacion de Pearson
R2_testing = R*R;
fprintf('\n | Coeficiente de Correlacion   --> R    = %.4f',R);
fprintf('\n | Coeficiente de Determinación --> R^2  = %.4f',R2_testing);

% Calculo MAPE
MAPE = mean((abs(y_test - salida_estimada)./abs(y_test))*100);
```

```
fprintf('\n | Error Porcentual Absoluto Medio                  --> MAPE =
%.4f',MAPE);

fprintf('\n ------------------------------------------------------------------
------- \n \n \n');
```