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

“REDES RRBF CON APRENDIZAJE HÍBRIDO PARA LA PREDICCIÓN DE CAPTURA DE ANCHOVETAS”

PABLO ANDRÉS QUEZADA ROMÁN

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

DICIEMBRE, 2009.

Pontificia Universidad Católica de Valparaíso
Facultad de Ingeniería
Escuela de Ingeniería Informática

"REDES RRBF CON APRENDIZAJE HÍBRIDO PARA LA PREDICCIÓN DE CAPTURA DE ANCHOVETAS"

PABLO ANDRÉS QUEZADA ROMÁN

Profesor Guía: **Nibaldo Rodríguez Agurto**

Profesor Co-referente: **Claudio Cubillos Figueroa**

Carrera: **Ingeniería Civil Informática**

Diciembre, 2009.

Dedicatoria

Dedicada a mis padres por todo el cariño y esfuerzo, a mi polola por su gran amor e incondicional apoyo, y a mi familia en general, porque siempre han estado conmigo a lo largo de este proceso.

Agradecimientos

A Dios, mis padres, a mi polola y profesor guía.

Resumen

Resumen

En la presente memoria de título se presentan los objetivos, estado de arte y modelos de red desarrollados para pronosticar el nivel de captura mensual de anchovetas en la zona norte de Chile. Estos modelos consisten en la utilización de redes neuronales recurrentes con función de base radial (RRBF) cuyos parámetros internos son configurados mediante la utilización de algoritmos evolutivos como son: Algoritmos Genéticos, Optimización por Enjambre de Partículas y dos modelos híbridos en base a estos últimos. Finalmente, el modelo basado en PSO presenta el mejor desempeño con un 92,5% de la varianza explicada.

Palabras claves: Redes Neuronales Recurrentes con Función de Base Radial, Algoritmos Genéticos, Optimización por Enjambre de Partículas, Algoritmos Híbridos.

Abstract

On this thesis the objectives, state of the art and developed network model are presented to forecast the monthly capture level of anchovy in the north area of Chile. This model consists on Recurrent Neural Networks with Radial Basis Function (RRBF) whose parameters are estimated using evolutive algorithms as: Genetic Algorithms (GA), Particle Swarm Optimization (PSO) and two Hybrid model based on GA and PSO. Finally, the model based on PSO presents the best performance with a 92,5% of the explained variance.

Keywords: Recurrent Neural Networks with Radial Basis Function, Genetic Algorithms, Particle Swarm Optimization, and Hybrid Algorithms.

Índice de Contenidos

Lista de Abreviaturas o Siglas

- APE: Absolute Percentage Error
- BS: Breeding Swarm
- IP: Inicialización de Parametros
- GA : Genetic Algorithm
- MAPE: Mean Absolute Percentage Error
- PSO: Particle Swarm Optimization
- RBF: Radial Basis Function
- RMSE: Root Mean Square Error
- RNA: Red Neuronal Artificial
- RRBF: Recurrent Radial Basis Function

Índice de Figuras

Índice de Tablas

CAPÍTULO 1

Introducción

1.1 Introducción.

El sureste del Océano Pacífico, en especial, el mar del norte de Chile y sur del Perú, corresponde a un ecosistema altamente productivo en cuanto a especies pelágicas se refiere, dentro de las cuales se encuentra la especie *Engraulis Ringens* o **Anchoveta**. A partir de mediados de la década de los 50 comienza a desarrollarse una intensa actividad pesquera (Cañón, 1978) basada principalmente en la captura de la anchoveta, la cual se extiende hasta nuestros días. Tal fue la intensidad de esta actividad que desde 1983, como consecuencia de la caída de la población de anchovetas, la pesca en el área norte de Chile se encuentra sujeta a regulaciones durante los periodos de reproducción más importantes.

En este escenario, el pronóstico de las capturas es un tópico básico, debido a que juegan un papel fundamental en el papel de los stocks, previo a la toma de decisiones. En las políticas de gestión de pesca el principal objetivo es establecer el esfuerzo pesquero aplicable a un área concreta durante un periodo de tiempo conocido y manteniendo el reemplazo de las existencias. Para alcanzar esta meta, es necesario pronosticar eventos incontrolables como la posible abundancia o aumento de la biomasa.

La recopilación de información de fluctuaciones de abundancia de anchoveta en la zona norte tiene como fin ser la base desde la cual se dicten las decisiones gubernamentales respecto de la caza del recurso pelágico, es decir, tomar las medidas necesarias para asegurar la disponibilidad del recurso en el futuro. Es aquí donde se vuelve importante la habilidad de predecir, con cierto grado de exactitud, los niveles de abundancia de los recursos pelágicos de la zona, con el fin de planificar la captura y asegurar su disponibilidad para el futuro.

La información relacionada con el ecosistema en el cual abunda la anchoveta se caracteriza por su complejidad y comportamiento no lineal. La variabilidad de su nivel de abundancia se ve afectada directamente por una serie de variables ambientales, entre las cuales se encuentran: La corriente del Niño, temperatura superficial del mar y concentración de alimento (plancton).

En el presente proyecto se tomarán, como series temporales, los datos relacionados con los volúmenes de captura de anchovetas en el norte de Chile y mediante el uso de **Redes RRBF con aprendizaje Híbrido** se buscará predecir y analizar su comportamiento.

Las **Redes Neuronales Artificiales** se pueden definir como "un nuevo sistema para el tratamiento de la información, cuya unidad básica de procesamiento está inspirada en la célula fundamental del sistema nervioso: "*La Neurona*" (Matich, 2001). Tanto su estructura como estado interno proporcionan características fundamentales para su uso en el ámbito de la predicción: la capacidad de aprender, adaptarse y generalizar. Además, se pueden diferenciar distintos tipos de redes en base a su topología y estado interno. En el presente proyecto se pone énfasis en las **Redes Neuronales Artificiales Recurrentes de**

Base Radial (Redes RRBF). El hecho de que sea recurrente le permite tener conexiones hacia distintas capas y/o nodos de la red, aumentando la capacidad de representación de la esta. Por otro lado, las redes de base radial se caracterizan por que pueden aproximar cualquier función continua sobre un compacto de R^n de manera local.

Tal como se explico en el párrafo anterior, tanto el estado interno como la topología son críticas para el éxito de la red, por lo que deben ser entrenadas de tal manera que su configuración converja a la obtención de las salidas deseadas.

Existen distintas técnicas para entrenar una red neuronal, alguna implican un aprendizaje supervisado y otras uno no supervisado. En el presente caso de estudio, se realiza un aprendizaje supervisado de la red mediante la utilización de un algoritmo **Híbrido** basado en técnicas del cálculo evolutivo como son: **Algoritmos Genéticos (GA) y Optimización por Enjambre de Partículas (PSO).**

La utilización de estas 2 técnicas para definir el estado interno de la red se basará en mezclar sus cualidades de exploración y explotación, de manera que se pueda abarcar todo el espacio de búsqueda y encontradas las mejores regiones donde poder encontrar un óptimo local que se acerque lo más posible a una solución óptima de la red.

Finalmente, el proyecto propuesto comienza con el estudio del estado del arte de las Redes Neuronales, poniendo énfasis en las redes RRBF, en los Algoritmos Genéticos y Optimización por Enjambre de Partículas. Una segunda parte consiste en el diseño de una Red RRBF y la implementación de un prototipo de sistema que permita evaluar el modelo en base a un conjunto métricas con el fin de comparar los resultados con otros modelos predictivos desarrollado para el pronóstico de captura de Anchovetas.

1.2 Objetivos del Proyecto.

1.2.1 Objetivo General.

Desarrollar y evaluar un modelo neuronal de base radial recursivo utilizando algoritmos genéticos combinados con algoritmo de enjambre de partículas para pronosticar las capturas mensuales de anchovetas en área norte de Chile.

1.2.2 Objetivos Específicos.

- Comprender el estado del arte de Redes Neuronales Recursivas de Base Radial (RRBF) con aprendizaje híbrido basado en Algoritmos Genéticos (GA) y Optimización por Enjambre de Partículas (PSO).
- Diseñar la estructura y estimar los parámetros del modelo de pronóstico neuroevolutivo mediante el uso de GA y PSO.
- Implementar y evaluar el rendimiento de los modelos de pronóstico propuestos.

1.3 Organización del Texto.

El Capitulo 2 consiste en el estado del arte de las redes neuronales artificiales, comenzando con una introducción a la técnica y luego una reseña histórica. En la sección 2.3 se presenta a "la neurona biológica, para luego en la sección 2.4, se logre hacer la analogía con la red neuronal artificial. En la sección 2.5 se describen las distintas topologías de red existentes. La sección 2.6 describe una de las características importantes de las redes a estudiar: "la recurrencia", explicando el concepto de recursividad o reinyección de la data. La sección 2.7 describe otra característica importante de las redes a estudiar: la función de base radial, definiéndose así las redes RBF. La sección 2.8 presenta los métodos de aprendizaje que tiene una red. Finalmente la sección 2.9 presenta como se aplican las redes a una serie de problemas.

El Capitulo 3 corresponde a los Algoritmos Genéticos (GA). Comenzando con una introducción a la técnica y una reseña histórica. En la sección 3.3 se describen los distintos elementos que forman parte de este proceso de optimización. Luego, en la sección 3.4 se presenta paso a paso el algoritmo genético, desde la creación aleatoria de la población inicial hasta la obtención de la población final con los parámetros optimizados en base a la función fitness. En la sección 3.5 se estudian las distintas codificaciones que puede presentar un cromosoma. GA en su algoritmo tradicional se destaca la selección de la nueva generación de individuos, las modalidades de selección se describen en la sección 3.6. La sección 3.7 presenta tanto las ventajas que entrega la utilización de GA en la optimización de parámetros en la solución de problemas como las limitaciones de la técnica. Finalmente se presentan, en la sección 3.8, las aplicaciones de los GA.

El Capitulo 4 presenta al algoritmo de Optimización por Enjambre de Partículas (PSO). Luego de una introducción al método de optimización en la sección 4.1, en la sección 4.2 se presenta el fundamento de movimiento de partículas, explicando las principales componentes de la inteligencia de grupo. En la sección 4.3 se presenta el algoritmo PSO, sus ecuaciones y pasos a seguir. Parámetros importantes de este proceso son explicados en la sección 4.4. Luego las secciones 4.5 y 4.6 presentan distintos tipos de topologías y métodos de actualización respectivamente. La sección 4.7 explica los conceptos de exploración y explotación que se dan con PSO. Finalmente la sección 4.8 presenta variantes del algoritmo encontradas en la literatura y la sección 4.9 aplicaciones de PSO en distintas áreas.

El Capitulo 5 presenta a las Redes Neuroevolutivas. Comenzando con la explicación de las métricas estadísticas utilizadas en el proceso de validación de los modelos. Las secciones 5.2, 5.3, 5.4, 5.5 presentan los distintos modelos creados en esta tesis. Cada una de estas secciones incluye una descripción de modelo y el proceso de selección de topología y tipo de recurrencia.

El Capítulo 6 corresponde al Análisis de Resultados obtenidos por los modelos seleccionados en el capítulo 5. En este capítulo se busca contrastar los resultados obtenidos por los modelos en base a las métricas descritas en la sección 5.1. Se presentan graficas de estimación, correlación y MAPE. Finalmente se comparan con una técnica estadística: Regresión Funcional.

Finalmente el Capitulo 7 consiste en las Conclusiones de esta tesis, tomando en cuenta el estado del arte, las recurrencias estudiadas, los modelos generados y el resultado final del estudio.

CAPÍTULO 2

Redes Neuronales

2.1 Introducción.

El hombre se ha caracterizado siempre por su búsqueda constante de nuevas vías para mejorar sus condiciones de vida. Mediante estos esfuerzos ha logrado, por ejemplo, construir máquinas calculadoras que ayuden a resolver de manera rápida y automática determinadas operaciones que resultan tediosas de ser realizadas a mano.

Los desarrollos actuales de los científicos se dirigen al estudio de las capacidades humanas como una fuente de nuevas ideas para el diseño de nuevas máquinas. Así, la inteligencia artificial es un intento por descubrir y describir aspectos de la inteligencia humana que pueden ser simulados mediante máquinas. Esta disciplina se ha desarrollado fuertemente en los últimos años teniendo aplicación en algunos campos como visión artificial, demostración de teoremas, procesamiento de información expresada mediante lenguajes humanos, etc. (Matich, 2001).

El exacto funcionamiento del cerebro humano aún es un misterio, aunque algunos aspectos de este impresionante procesador son conocidos. En particular, el elemento más básico del cerebro es un específico tipo de célula el cual, a diferencia del resto del cuerpo, no se regenera. Porque este tipo de célula corresponde a la única parte del cuerpo que no es lentamente reemplazada, se asume que estas células son las que proveen la habilidad de recordar, pensar y aplicar experiencias previas en cada acción realizada (Anderson y McNeil, 1992). Cada una de estas, conocidas como Neuronas, puede conectarse con unas 200.000 más, aunque lo típico ronda entre 1.000 y 10.000, formando Redes Neuronales.

He aquí donde las Redes Neuronales Artificiales (RNA), son más que otra forma de emular ciertas características propias de los humanos, como la capacidad de memorizar y de asociar hechos. Estas puedes ser caracterizadas de manera más adecuada como “modelos computacionales” con propiedades particulares como la capacidad de aprender y adaptarse, generalizar, agrupar u organizar datos, y cuya operación se basa en procesamiento paralelo. Las RNA poseen distintas características, tanto de su estructura como de la forma en la que aprenden a realizar la función por la cual fueron creadas, generando así una amplia gama de posibilidades siendo algunas más adecuadas que otras frente a algún problema en particular.

En el presente capítulo, la sección 2 presenta una **Reseña Histórica** de las redes neuronales, desde sus comienzos, a través de su “receso” y hasta su regreso. Luego en la sección 3 se presenta a la **Neurona Biológica** con todas sus características, para luego llegar a la **Red Neuronal Artificial**, la cual es una analogía a nivel computacional de la red neuronal biológica. En la sección 5 se dan a conocer las distintas **Topologías de Redes Neuronales**, para luego comenzar a definir características relacionadas con el proyecto. De esta manera, en la sección 6 se presentan las **Redes Neuronales Recurrentes** destacando los principales tipos de recurrencia existentes, junto con las aplicaciones de las RRBF en la

literatura, y en la sección 7 se describen los tipos de redes utilizados en este caso de estudio: **Redes de Función de Base Radial**. Posteriormente se explican las principales formas de aprendizaje que puede tener una red y finalmente en la sección 9 de este capítulo se dan a conocer distintas **Aplicaciones** de las redes neuronales, en especial, de las redes neuronales recurrentes de base radial presentes en la literatura.

2.2 Reseña Histórica.

La historia de las redes neuronales se remonta hacia el año **1936** en donde **Alan Turing** fue el primero en estudiar el cerebro como una forma de ver el mundo de la computación. Sin embargo, los primeros teóricos que concibieron los fundamentos de la computación neuronal fueron: el neurofisiólogo **Warren McCulloch** y el matemático **Walter Pitts** quienes, en **1943**, describen una lógica de cálculo de redes neuronales que une estudios de neurofisiología y lógica matemática. Su modelo formal de una neurona sigue la ley "todo o nada". Con un suficiente número de estas simples unidades, correctas conexiones sinápticas y operando de manera síncrona, demostraron que una neurona constituida de esta manera puede, en principio, calcular cualquier función computable.

En **1949 Donald Hebb**, en (Hebb, 1949), Fue el primero en explicar los procesos del aprendizaje desde un punto de vista psicológico, desarrollando una regla de como el aprendizaje ocurría. Su idea fue que el aprendizaje ocurría cuando ciertos cambios en una neurona eran activados. También propone que la conectividad del cerebro cambia continuamente conforme un organismo aprende cosas nuevas, creándose asociaciones neuronales con estos cambios. Los trabajos de Hebb formaron las bases de la Teoría de las Redes Neuronales.

En **1950 Karl Lashley** resume su investigación de 30 años y destaca que el proceso de aprendizaje es un proceso distribuido y no local a una determinada área del cerebro.

Haibt y Duda en 1956 realizan una de las primeras simulaciones computacionales para probar un bien formulada teoría neuronal basándose en el postulado de aprendizaje de Hebb. Este mismo año, Uttley, demuestra que una red neuronal con sinapsis modificables puede aprender a clasificar un simple conjunto de patrones binarios en sus respectivas clases (Haykin 1999).

15 años luego de la publicación de McCulloch y Pitts un nuevo acercamiento al problema de reconocimiento de patrones fue introducido por **Rosenblatt** (**1958**) en su trabajo sobre el perceptrón. Éste constaba de 2 niveles y los pesos se ajustaban en proporción al error entre las salidas deseadas y obtenidas. 2 años después Widrow y Hoff introdujeron el algoritmo "least mean-square" y lo usaron para formular el ADALINE (ADAptative LINear Element). Éste último, se diferencia del perceptrón en el proceso de entrenamiento.

En **1969** se produjo la "muerte abrupta" de las redes neuronales cuando **Minsky y Papera** probaron matemáticamente que el Perceptrón no era capaz de resolver problemas tan fáciles como el aprendizaje de una función no lineal. Esto demuestra una gran debilidad, dado que las funciones no-lineales son ampliamente utilizadas en computación y en problemas del mundo real.

En **1982 Hopfield** publica un trabajo clave para el resurgimiento de las redes neuronales. En él, desarrolla la idea del uso de una función de energía para comprender la dinámica de una red neuronal recurrente con uniones sinápticas simétricas. El principal uso

de estas redes ha sido como memorias y como instrumento para resolver problemas de optimización como el problema del viajante.

En **1986 Rumelhart, Hinton y Williams**, desarrollan el algoritmo de aprendizaje de retro propagación (backpropagation) para redes neuronales multicapa, dando una serie de ejemplos en los cuales se demuestra el potencial del método desarrollado.

A partir de ese año, el número de trabajos sobre redes neuronales ha aumentado exponencialmente apareciendo un gran número de aportaciones tanto a los métodos de aprendizaje como a las arquitecturas y aplicaciones de las redes neuronales. Se podría destacar entre todas estas aportaciones lo trabajos de **Broomhead y Lowe** y el de **Poggio y Girosi** sobre el diseño de redes neuronales en capas usan RBF (Radial Basis Function).

En la actualidad, son numerosos los trabajos que se realizan y publican cada año, las aplicaciones nuevas que surgen (sobre todo en el área de control) y las empresas que lanzan al mercado productos nuevos, tanto hardware como software (sobre todo para simulación).

2.3 Red Neuronal Biológica.

El aparato de comunicación neuronal de los animales y del hombre, formado por el sistema nervioso y hormonal, tiene la misión de recibir la información, transmitirla y elaborarla, en parte también almacenarla y enviarla de nuevo en forma elaborada. El sistema de comunicación neuronal se compone de 3 partes (Isasi y Galván, 2004):

- Los **receptores**, que se encuentran en las células sensoriales, recogen la información en forma de estímulos desde el ambiento o del interior del organismo.
- El **sistema nervioso**, que recibe la información, la elabora, en parte las almacena y las envía de forma elaborada a los órganos efectores y a otras zonas del sistema nervioso.
- Órganos diana o **efectores**, que reciben la información y la interpretan en forma de acciones motoras, hormonales, etc.

El elemento estructural y funcional más esencial, en el sistema de comunicación neuronal, es la **Neurona.** La mayoría de las neuronas utilizan sus productos de secreción como señales químicas para la transmisión de la información. Esta se envía a través de distintas neuronas, formando redes, en las cuales se elabora y almacena información.

Una parte de la neurona esta en relación con los receptores, a través de los cuales llega información proveniente del exterior o del interior del organismo a las redes neuronales.

Otra parte conduce las informaciones, elaboradas en forma de órdenes, hacia los efectores. Una de las prolongaciones en la encargada de la conducción de los impulsos; se denomina **Axón.**

A partir de una distancia más o menos grande de su origen, se ramifica y forma los botones terminales, los cuales se ponen en contacto con otras neuronas o con células efectores. A esta zona de contacto se le conoce como **Sinapsis.**

Un diagrama de la neurona se muestra en la siguiente figura:

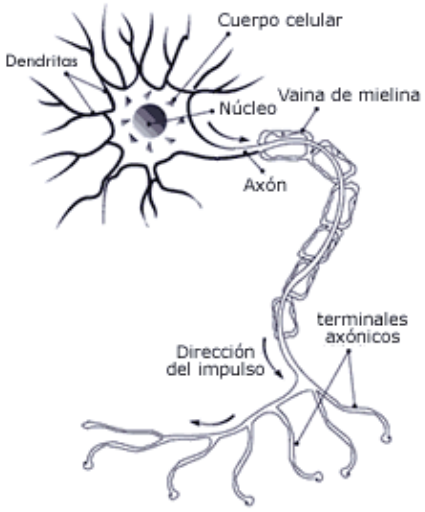

Figura 2-1 Neurona Biológica

En la figura se aprecia que la neurona se encuentra formada por un **soma** o cuerpo celular y que a su interior contiene el **núcleo**. También, cuenta con el **axón**, el cual se encuentra envuelto en la **Vaina de mielina**, que corresponde a aquel por el cual se propagan una serie de impulsos electro-químicos. Además, las ramificaciones de entrada con las que cuenta la neurona se conocen como **dendritas**, estas propagan la señal al interior de la neurona y los **terminales axónicos** que se conectan con las dendritas de la siguiente neurona para la entrega de la señal. El mecanismo es el siguiente (Isasi y Galván, 2004):

- La sinapsis recoge información electro-química de las neuronas vecinas por medio de las **dendritas**.
- La información llega al **núcleo**, en donde es procesada para generar la respuesta que viajará por el **Axón**.
- Luego, la señal propagada por el axón se ramifica en los **terminales axónicos** y llega a las **dendritas** de otras neuronas en lo que se denomina **Sinapsis.**

La capacidad de una red neuronal de aprender y de adaptarse a su ambiente puede lograrse mediante dos mecanismos: la creación de nuevas conexiones sinápticas y la modificación de las conexiones sinápticas existentes (Haykin, 1999). Esto permite a la red modificar su estructura, además de otorgarle la cualidad de ser **tolerante a fallos**, dado que se pueden realizar más de un camino para que un impulso neuronal viaje a través de la red y provoque la respuesta adecuada. Además la comunicación entre neuronas se basa en el paso de mensajes pequeños, por lo que la información crítica no se transmite directamente, mas se captura y distribuye entre las interconexiones.

2.4 Red Neuronal Artificial.

Las redes neuronales artificiales corresponden a un modelo de aprendizaje y procesamiento automático inspirado en la forma en que funciona el sistema nervioso en los animales. Consiste en un sistema de interconexión de neuronas artificiales formando una red que colabora para producir un estimulo de salida. Estas neuronas corresponden a unidades de procesamiento que se comunican unas a otras mediante el envío de señales por medio de una serie de conexiones las cuales tienen un peso asociado.

Según (Haykin, 1999), el modelo de una neurona presenta 3 elementos básicos:

- Una serie de **sinapsis**, donde cada una está caracterizada por un **peso** el cual puede tomar valores negativos como positivos.
- Un **sumador** que suma las señales de entrada pesadas por la respectiva sinapsis de la neurona.
- Una **función de activación** para limitar la amplitud de la salida de la neurona. Cabe destacar que ésta corresponde a una función no lineal que normaliza el estado interno de la neurona a un intervalo cerrado [0,1] o alternativamente [-1,1]. Ejemplos de función de activación serían: la función escalón, sigmoidea, gaussiana, polinomial, entre otras.

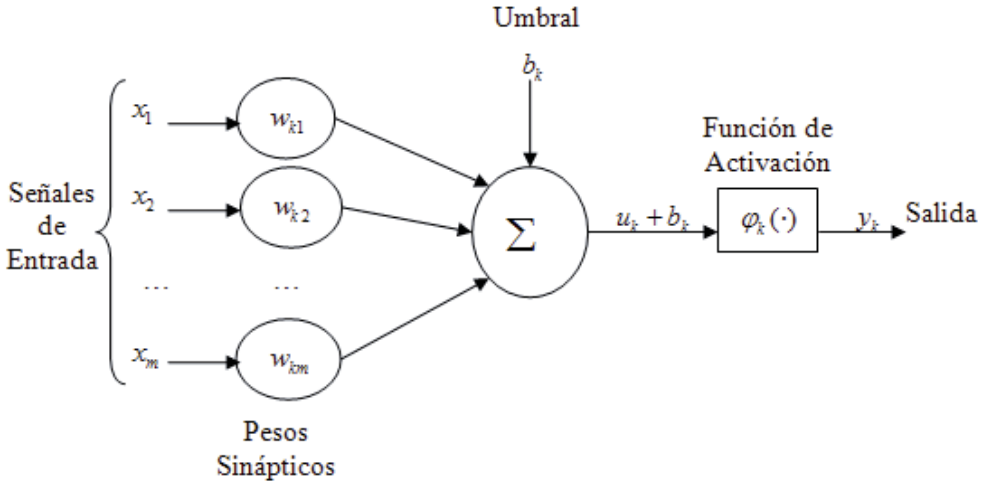

Figura 2-2 Representación de una Neurona Básica

El modelo neuronal presentado en la Fig. 2.2 incluye además un valor numérico conocido como **umbral** el cual se denota como b_k y corresponde a un valor opcional que puede aumentar o disminuir el estado interno de la neurona dependiendo si este es positivo o negativo respectivamente.

En términos matemáticos, una neurona *k* puede ser descrita por medio del siguiente par de ecuaciones:

$$u_k = \sum_{j=1}^{m} w_{kj} x_j \tag{1}$$

$$y_k = \varphi(u_k + b_k) \tag{2}$$

Donde $x_1, x_2, \ldots, x_m$ corresponden a las señales de entrada a la neurona artificial, $w_{k1}, w_{k2}, \ldots, w_{km}$ a los pesos sinápticos de la neurona k, u_k es la salida obtenida de la combinación lineal dada la señal de entrada. Luego, b_k corresponde al umbral, φ_k corresponde a la función de activación e y_k es la señal de salida de la neurona k.

A la manera mediante la cual las neuronas se conectan entre sí se le denomina **patrón de conectividad** o **arquitectura de la red** y se encuentra íntimamente relacionado con el algoritmo de aprendizaje usado para entrenar la red. Las neuronas, tal como se muestra en la Fig. 2.3, se organizan en **capas** las cuales corresponden a un conjunto de neuronas cuya entrada corresponde a la salida de otra capa o a una entrada desde el exterior, y su salida corresponde a la entrada de una capa siguiente o a la salida de la red neuronal.

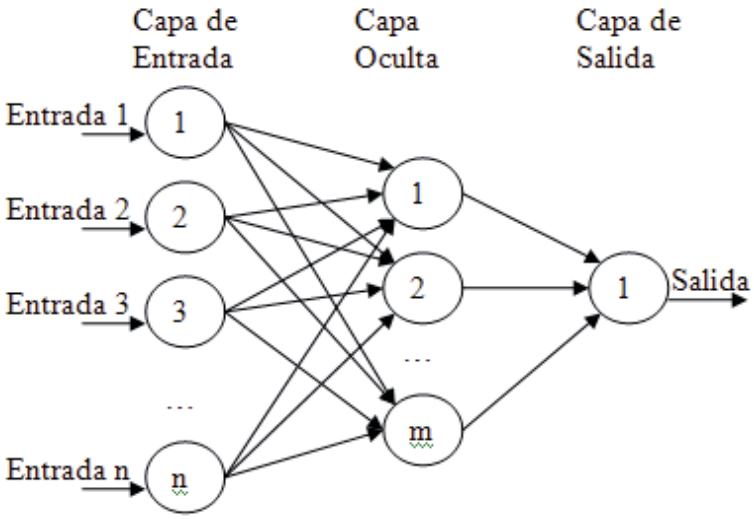

Figura 2-3 Arquitectura básica de una red neuronal

Ya definido el concepto de capa y en base a la Fig. 2.3 se pueden definir las distintas capas que forman parte de una red neuronal. La **capa de entrada** corresponde al primer nivel que recibe los valores de patrones (representados como vectores) que sirven como entrada a la red. A continuación, hay una serie de capas intermedias conocidas como **capas ocultas**, cuyas unidades corresponden a rasgos particulares que pueden aparecer a los patrones de entrada y pueden corresponder a 1 o más niveles. Finalmente, la última corresponde a la **capa de salida**, la cual sirve de salida de toda la red.

Cada interconexión entre las distintas neuronas actúa como una ruta de comunicación, a través de la cual viajan los valores numéricos de una neurona a otra. Así, la **Red Neuronal Artificial** podría definirse como un grafo cuyos nodos poseen características idénticas y cuya información se transporta a través de los arcos.

2.5 Topologías de Redes Neuronales.

La topología o arquitectura de una red neuronal consiste en la organización y disposición de las neuronas que la conforman. Estas estructuras o modelos conexionistas se encuentran fuertemente ligados al algoritmo de aprendizaje utilizado en su elaboración y se pueden clasificar de diferentes formas dependiendo del criterio de uso (Soria y Blanco, 2001):

2.5.1 Según su número de capas.

- **Redes neuronales monocapa**. Corresponden a las redes neuronales más sencillas dado que se solo se tiene una capa de entrada que proyecta la señal a una capa de salida en la cual se realizan diferentes cálculos. Dado que la capa de entrada no realiza ningún cálculo, de las 2 capas nombradas previamente, solo se cuenta una capa (la de salida). De ahí el nombre **monocapa**. Estas redes se utilizan generalmente en relacionadas con lo que se conoce como autoasociación[1].

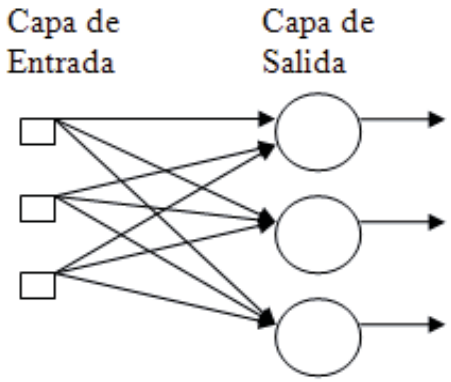

Figura 2-4 Red Neuronal Monocapa

- **Redes neuronales multicapa.** Son aquellas que disponen de un conjunto de neuronas agrupadas en varios niveles o capas. Es una generalización de la anterior existiendo un conjunto de capas intermedias, entre la capa de entrada y salida, conocidas como **capa oculta**.

[1] Consiste en la regeneración de la información de entrada a la red que se presenta de forma incompleta o distorsionada.

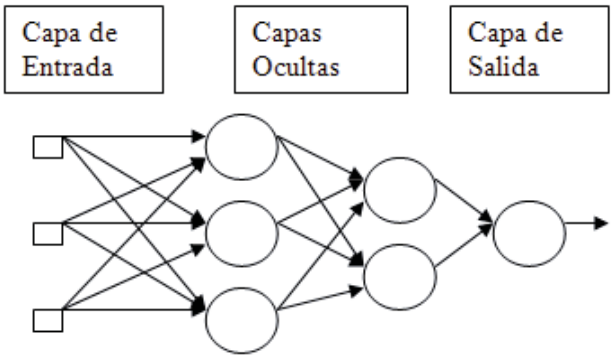

Figura 2-5 Red Neuronal Multicapa

2.5.2 Según el tipo de conexiones.

- **Redes neuronales no recurrentes.** También conocidas como redes feed-forward, son aquellas donde los nodos se conectan solo a capas posteriores, sin generar ciclos de información. La Fig. 2.5 correspondiente a la red neuronal multicapa también corresponde a una red neuronal no recurrente.
- **Redes neuronales recurrentes.** También conocida como redes de retro-propagación (feedback), son aquellas donde las neuronas de una capa se conectan con neuronas de distintas capas (anteriores y posteriores), neuronas de la misma capa, o incluso con la misma neurona. Dada la relevancia de esta topología de red en el proyecto, se explican más a fondo en la sección 2.6.

2.5.3 Según el grado de conexión.

- **Redes neuronales totalmente conectadas.** En estas redes todas las neuronas de una capa se encuentran conectadas con las neuronas de la capa siguiente y con las de la capa anterior, en caso de ser una red neurona recurrente.
- **Redes neuronales parcialmente conectadas.** En este caso no se da la conexión total entre neuronas de diferentes capas.

2.6 Redes Neuronales Recurrentes

Tal como se explico en la sección 2.5.2, las **redes neuronales recurrentes** son aquellas que se caracterizan por la generación de ciclos en red mediante el uso de las llamadas **conexiones recurrentes** pudiendo aparecer conexiones de una neurona a ella misma, conexiones entre neuronas de la misma capa o conexiones de una neurona a una capa anterior, tal como se muestra en la Fig. 2.7.

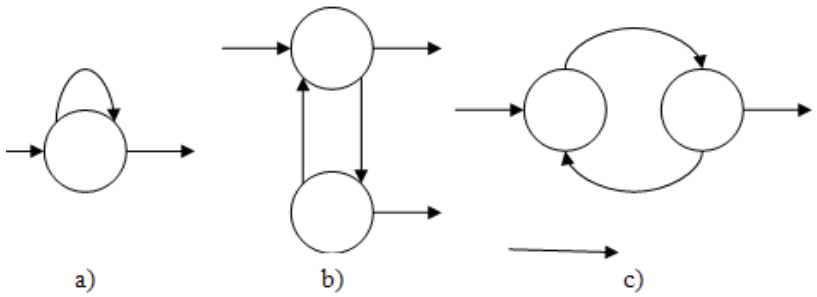

Figura 2-6 Tipos de Recurrencias

a) Conexión con la misma neurona, b) Conexión con neuronas de la misma capa y c) Conexión con neuronas posteriores y anteriores.

El uso de este tipo de redes implica un aumento en el número de pesos o parámetros ajustables de la red permitiéndose, de esta manera, aumentar la capacidad de representación de la red dado que en las redes neuronales artificiales la información se representa de manera distribuida en los pesos de las conexiones y no en las neuronas. Por otro lado, la existencia de recurrencias en la red, junto con el aumento de los parámetros, por lo general, complica el proceso de aprendizaje de ésta.

Ahora, la activación ya no depende sólo de activaciones ocurridas en la capa anterior, sino que también depende del estado o activación de cualquier otra neurona que se encuentre conectada a ella, o incluso su propia activación. Por lo tanto, es necesario incluir la **variable tiempo** en la activación de la neurona, la cual viene dada por:

$$a_i(t+1) = f_i\left(\sum_j w_{ji} a_j(t)\right) \tag{3}$$

Donde el índice *j* varía en el conjunto de todas las neuronas que se encuentran conectadas a la neurona *i*.

La presencia de $a_i(t+1)$ en las activaciones de las neuronas recurrentes hace que éstas posean un comportamiento dinámico o temporal, el cual puede entenderse de 2 formas:

- **Evolución de las activaciones de la red hasta alcanzar un punto estable.** Consiste en evolucionar la red (activaciones de las neuronas), desde un **estado inicial** dado por el patrón de entrada, hasta conseguir que todas estas activaciones de las neuronas no se modifiquen. En este momento se considera que la red ha alcanzado un **estado estable**, el cual representa el patrón de salida.

- **Evolución de las activaciones de la red en modo continuo.** En este caso, para cada instante de tiempo se tendrá una salida de la red, la cual depende de una entrada obtenida en el instante inmediatamente anterior. Este aprendizaje se puede llevar de 2 maneras diferentes:

    - **Aprendizaje por épocas:** La red evoluciona durante un intervalo de tiempo previamente definido, y cuando se alcanza este instante final se adaptan o modifican los pesos de la red. Una vez transcurrida una época, la red se reinicializa y se entra en una nueva época.
    - **Aprendizaje en tiempo real o continuo:** La ley de aprendizaje para modificar los pesos de la red se aplica en cada instante de tiempo, siempre y cuando exista la salida deseada para la red en dicho instante.

  Dentro del grupo de redes recurrentes que presentan este modo de actuación, están las redes parcialmente recurrente y totalmente recurrente. Ambos tipos utilizan algoritmos de aprendizaje supervisados para la supervisión de sus parámetros. Los algoritmos de aprendizaje supervisado se explicaran en la sección 3.8.1.

A pesar de la complejidad que poseen las redes neuronales recurrentes y la dificultad, en ocasiones, para realizar su aprendizaje, estas han sido utilizadas para abordar distintos tipos de problemas. Por ejemplo, en reconocimiento de patrones en (Zemouri, Racoceanu y Zerhouni, 2007), en la reconstrucción de procesos de escurrimiento de precipitaciones en (Pan y Wang, 2005), entre otros.

Existen distintos tipos de redes neuronales recurrentes, dentro de los cuales están las redes de Elman y Jordan. Definidas por los investigadores de mismo nombre, son redes parcialmente recurrentes, con conexión feed-forward, a las que se le han añadido algunas conexiones hacia atrás.

Generalmente cuando se habla de redes neuronales parcialmente recurrentes, existe un grupo de neuronas especiales ubicadas en la capa de entrada, conocidas como **neuronas de contexto** o **neuronas de estado.** De esta manera, se puedes distinguir 2 tipos de neurona en la entrada de la red: aquellas que actúan de entrada recibiendo las señales desde el exterior, y las neuronas de contexto (Fig.2.8) que actúan como neuronas receptoras de las conexiones recurrentes y funcionan como una memoria de la red en la que se almacenan las activaciones de las neuronas de una cierta capa de la red en el instante o iteración anterior.

Si se concatenan las activaciones de las neuronas de entrada y de las de contexto, una red parcialmente recurrente se puede ver como una red multicapa. De esta manera, en las redes parcialmente recurrentes, el cálculo de las activaciones de todas las neuronas se realiza como es una red multicapa sin recurrencias, es decir, desde la capa de entrada a la capa de salida, pasando por la capa oculta.

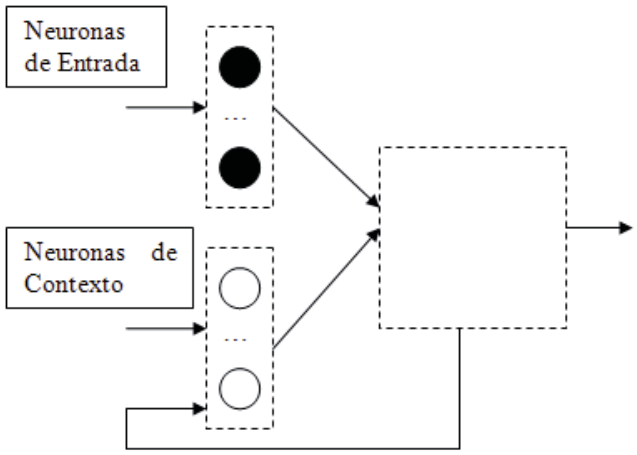

Figura 2-7 Esquema de una red parcialmente recurrente

Jordan y Elman corresponde a las redes parcialmente recurrentes más conocidas. Su diferencia fundamentalmente radica en que en las redes de Jordan las neuronas de contexto reciben copias de las neuronas de la capa de salida y de sí mismas, y en las redes de Elman las neuronas de contexto reciben copias de las neuronas ocultas.

A continuación se describe la arquitectura y características de la red de Jordan y la red de Elman (Isasi y Galván, 2004).

2.6.1 Red de Jordan.

Propuesto por Jordan en 1986, (Jordan, 1986), se caracteriza por que las neuronas de contexto reciben una copia de las neuronas de salida de la red y de ellas mismas. Además, las conexiones recurrentes de la capa de salida a las neuronas de contexto llevan un parámetro μ asociado, el cual toma un valor constante positivo y menos que 1.

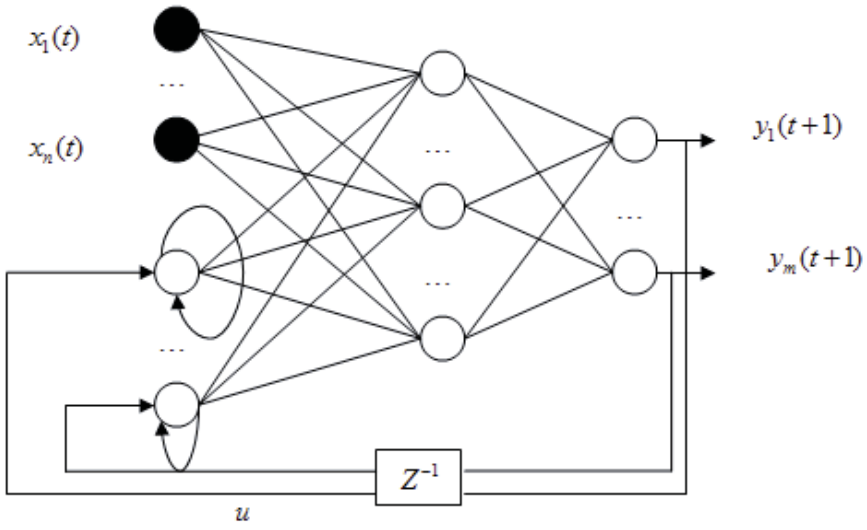

Figura 2-8 Red de Jordan

La activación de cada neurona de contexto en el instante de tiempo o iteración t, denotada por $c_i(t)$, viene dada por:

$$c_i(t) = \mu c_i(t-1) + y_i(t-1) \text{ Para } i = 1,2,...,m \quad (4)$$

Donde m corresponde al número de salidas de la red, e $y(t-1) = (y_1(t-1), y_2(t-1), ..., y_m(t-1))$ corresponde al vector de salida de la red en el instante $t-1$.

El resto de las activaciones se calculan como en una red multicapa con conexiones hacia delante, la diferencia radica en el vector de entrada en el tiempo *t*, dado que corresponderá a la concatenación de las activaciones de las neuronas de entrada junto con las de contexto. Éste vector *u* (*t*) se define como sigue:

$$u(t) = (x_1(t), ...x_n(t), c_1(t), ..., c_m(t)) \quad (5)$$

$x_i(t)$, tal como se nombra en la sección 3.4, corresponde a la señal que la red recibe desde el exterior.

Las neuronas de contexto tienen una función de activación lineal, como se visualiza en la ecuación 4. Esto hace que la activación de las neuronas de contexto se pueda desarrollar en el tiempo del siguiente modo:

$$c_i(t) = \mu^2 c_i(t-2) + \mu y_i(t-2) + y_i(t-1) = ... =$$

$$= \mu^{t-2} y_i(1) + \mu^{t-3} y_i(2) + ... + \mu y_i(t-2) + y_i(t-1)$$

Y se obtiene entonces la siguiente expresión:

$$c_i(t) = \sum_{j=1}^{t-1} \mu^{j-1} y_i(t-j) \text{ Para } i=1,2,\ldots, m \quad (6)$$

En esta expresión se observa que las neuronas de contexto acumulan las salidas de la red en todos los instantes anteriores de tiempo, y el parámetro μ determina la sensibilidad de estas neuronas de retener dicha información. Así, valores cercanos a 1 permitirán memorizar estados muy lejanos al tiempo actual, y a medida que este valor tiende a 0, dichos estados tienen una menor presencia en la activación actual de las neuronas de contexto.

2.6.2 Red de Elman.

La red de Elman (Elman, 1990) se caracteriza por que las neuronas de contexto reciben copias desde neuronas ubicadas en la capa oculta de la red, tal como se muestra en la Fig. 2.10, y no traen asociado ningún tipo de parámetro (como μ en el caso de la red de

Jordan). Por lo tanto, existirán tantas neuronas de contexto como neuronas ocultas tenga la red y su activación estará dada por:

$$c_i(t) = a_i(t-1) \text{ Para } i=1,2,\ldots r \quad (7)$$

Donde *r* corresponde al número de neuronas ocultas de la red de Elman, y $a_i(t-1)$ corresponde a las activaciones de las neuronas ocultas en el instante *t*-1.

Al igual que las redes de Jordan, el resto de las activaciones se calculan como en una red multicapa con conexiones hacia delante, formándose como entrada a la red el vector *u* (*t*), definido previamente en la ecuación 5.

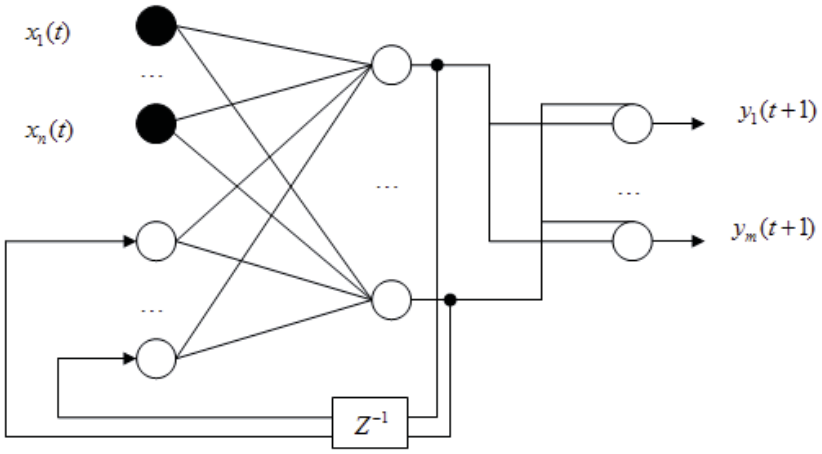

Figura 2-9 Red de Elman

Aunque las redes de Elman y Jordan son unos de los modelos de redes parcialmente recurrentes más conocidos, es posible diseñar y utilizar otros modelos diferentes, como por ejemplo, incorporar conexiones desde las neuronas de contexto a ellas mismas en el modelo de Elman, que la memoria de las neuronas de contexto estén definidas en un intervalo de tiempo, etc.

2.7 Redes con Función de Base Radial (RBF).

Las redes RBF son redes multicapa feed-forward formadas por una única capa oculta, donde cada una de las neuronas presentan un carácter local, es decir, se activa en una región diferentes del espacio de patrones de entrada. Este carácter se debe al uso de funciones de base radial como por ejemplo: la función gaussiana. Las neuronas de la capa de salida realizan una combinación lineal de las activaciones de las capas ocultas.

Estas redes nacen tras la necesidad de construir una red de neuronas que requiere un menor tiempo de aprendizaje que el Perceptrón Multicapa de manera que sea apropiada para aplicaciones en tiempo real. Esto se consiguió pues mediante las activaciones locales pocas neuronas ocultas tendrían que ser procesadas para nuevos patrones de entrada.

La función de base radial define híper esferas que dividen el espacio de entrada de tal manera que cada neurona de la capa oculta construye una aproximación local y no líneal

en una determinada región de dicho espacio. Dado que la salida es una combinación lineal de las funciones de base radial, las aproximaciones que construyen las redes RBF son combinaciones lineales de múltiples funciones locales y no lineales. De esta manera, las redes RBF aproximan relaciones completas por medio de una colección de aproximaciones locales menos complejas, dividiendo el problema en varios sub-problemas menos complejos.

Las redes RBF han sido utilizadas en distintos campos, como análisis de series de tiempo (Mordí y Darken, 1989), (Kadirk et al, 1991), procesamiento de imágenes (Saaa et al., 1991), diagnósticos médicos (Lowe y Webb, 1990), reconocimiento automático del habla (Niranjan y Fallside, 1990), etc.

2.7.1 Arquitectura de las redes RBF.

Las redes RBF se encuentran formadas por 3 capas:

- **Capa de Entrada**: reciben las señales desde el exterior y las transmiten a la siguiente capa sin realizar ningún procesamiento sobre estas señales, además no presentan ningún peso asociado.
- **Capa Oculta**: reciben las señales de la capa de entrada y realizan la transformación local y no lineal de la señal. Esta capa es la única que contiene componentes no lineales.
- **Capa de Salida**: actúa como salida de la red realizando una combinación lineal de las activaciones de las neuronas ocultas. Solo las neuronas de salida poseen umbral.

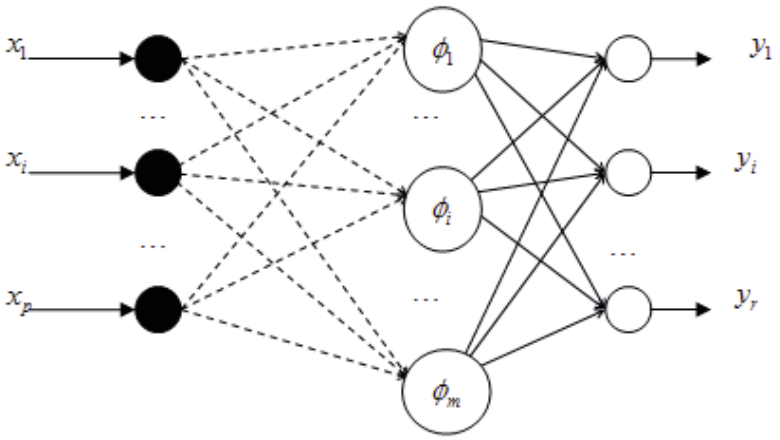

Figura 2-10 Arquitectura de una red RBF

En una red RBF, como la figura, con p neuronas en la capa de entrada, m en la capa oculta y r en la de salida, las activaciones de las neuronas de salida para el patrón de entrada n, $X(n) = (x_1(n), x_2(n), ..., x_p(n))$ denotadas como $y_k(n)$, vienen dadas por la siguiente expresión:

$$y_k(n) = \sum_{i=1}^{m} w_{ik}\phi_i(n) + b_k \text{ para } k = 1, 2, ..., r \quad (8)$$

Donde w_{ik} representa a los pesos de la conexión de la neurona oculta *i* a la neurona de salida *k*, b_k corresponde al umbral de la neurona de salida *k* y $\phi_i(n)$ son las activaciones de las neuronas ocultas para el patrón de entrada $X(n)$. ϕ_i Viene dada por la siguiente expresión:

$$\phi_i(n) = \phi\left(\frac{\|X(n) - C_i\|}{d_i}\right) \text{ Para } i=1,2,\ldots m \quad (9)$$

Donde ϕ es la función de base radial, $C_i = (C_{i1}, ..., C_{ip})$ son vectores que representan los centros, d_i son números reales que representan desviación, anchura o dilatación y $\| \ \|$ corresponde a la distancia euclídea del vector de entrada *X*(n) al centro C_i definida como:

$$\|X(n) - C_i\| = \left(\sum_{j=1}^{p} (x_j(n) - c_{ij})^2\right)^{\frac{1}{2}} \quad (10)$$

La función de base radial ϕ puede adoptar distintas formas y expresiones, entre ellas se encuentra:

- Función Gaussiana:

$$\phi(r) = e^{\left(\frac{-r^2}{2}\right)} \quad (11)$$

- Función inversa cuadrática:

$$\phi(r) = \frac{1}{1 + r^2} \quad (12)$$

- Función inversa multicuadrática:

$$\phi(r) = \frac{1}{\sqrt{1 + r^2}} \quad (13)$$

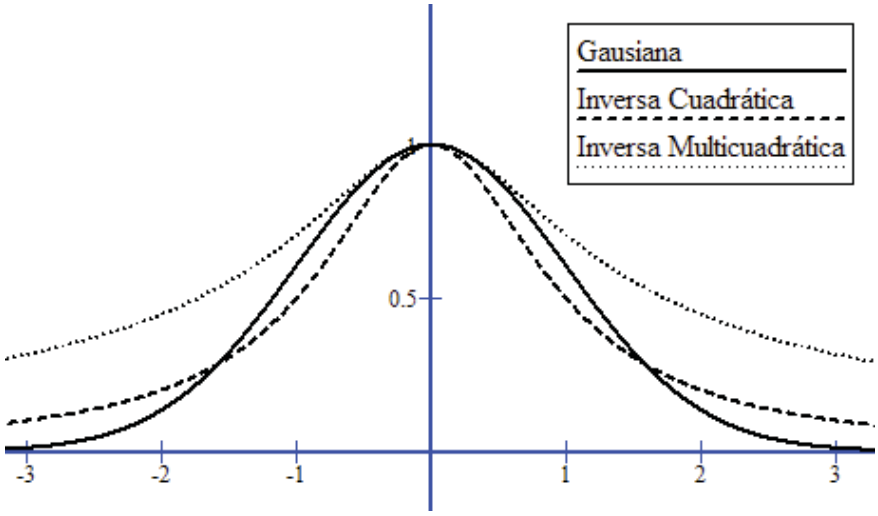

Figura 2-11 Funciones de Base Radial

2.8 Aprendizaje de una RNA.

La propiedad que es de principal importancia en una red neuronal corresponde a la habilidad de la red de aprender del ambiente y de mejorar su rendimiento a través del aprendizaje. Una red neuronal aprende del ambiente a través de un interactivo proceso de ajustes aplicados a los pesos sinápticos y umbrales. Idealmente, la red aumenta su conocimiento del entorno en cada iteración del proceso de aprendizaje (Haykin, 1999).

Según (Haykin, 1999), aprendizaje se define como un proceso en el cual los parámetros libres de una red neuronal son adaptados a través de un proceso de estimulación por el entorno en el cual se encuentra inserta la red. El tipo de aprendizaje es determinado por la manera en la cual los parámetros alterados toman lugar. Esta definición del proceso de aprendizaje implica la siguiente secuencia de eventos:

- La red neuronal es estimulada por el entorno.
- La red neuronal realiza cambios en sus parámetros libres como resultado de la estimulación.
- La red neuronal responde de una nueva forma al entorno debido a los cambios que han existido en su estructura interna.

En otras palabras, el aprendizaje corresponde al proceso mediante el cual la red neuronal modifica su estructura interna en respuesta de la información de entrada, y estos cambios que se producen se reducen a la destrucción, modificación y creación de conexiones entre las neuronas. En una neurona artificial, la creación de una conexión implica que el peso asociado pase a tener un valor distinto de 0. De la misma manera, una conexión se destruye cuando su peso pasa a ser 0.

Antes de comenzar el proceso de entrenamiento se debe determinar un estado inicial, lo cual significa, escoger un conjunto inicial de pesos para las diversas conexiones entre las neuronas de la red neuronal. Uno de los criterios a emplear consiste en otorgar valores aleatorios a las conexiones, encontrándose estos dentro de un intervalo [-n, n],

donde n es un número natural positivo. Cabe mencionar que durante el proceso de entrenamiento los pesos no se encontraran restringidos a éste intervalo.

Por otro lado, para determinar cuándo termina el proceso de aprendizaje es necesario establecer una **condición de detención.** Típicamente se distinguen 3 tipos de criterios:

- **Verificación del error de los resultados.** Se define una función que determina que tan acertados son los resultados y se entrena hasta alcanzar un rango determinado.
- **Resultado de entrenamiento irrelevante.** Condición que se da cuando los resultados de una iteración tienden a ser muy semejantes a los de la iteración anterior.
- **Alcance de un número fijo de ciclos.** Los entrenadores de la red definen un número fijo de iteraciones y una vez alcanzado éste número se detiene el entrenamiento independiente de la calidad de los resultados arrojados por la red.

Un aspecto importante respecto al aprendizaje de las redes neuronales es el conocer cómo se modifican los valores de los pesos, es decir, cuáles son los criterios que se siguen para cambiar el valor asignado a las conexiones cuando se pretende que la red aprenda una nueva información.

Existen 3 métodos de aprendizaje importantes que pueden distinguirse (Matich, 2001):

- Aprendizaje Supervisado
- Aprendizaje No Supervisado
- Aprendizaje Híbrido.

2.8.1 Aprendizaje Supervisado.

Se caracteriza porque el proceso de aprendizaje se encuentra controlado por un agente externo (supervisor, maestro) el cual determinan cuales debieran ser las respuestas entregadas por la red dada una entrada determinada. Este agente externo controla la salida y en caso de no corresponder a la deseada, se procederá a modificar los pesos de las conexiones, con el fin de que la salida obtenida se aproxime a la deseada.

Se suelen considerar 3 formas de llevar a cabo este tipo de aprendizaje, dando lugar a los siguientes aprendizajes supervisados.

- **Aprendizaje por corrección de error.** Consiste en ajustar los pesos de la red en función de las diferencias entre los valores deseados y los obtenidos de la salida de la red. Un algoritmo perteneciente a esta clasificación sería la **regla del aprendizaje Delta o regla del mínimo error cuadrado (LMS Error)**, que además de utilizar la desviación a la salida objetivo toma en consideración a todas las neuronas predecesoras que tienen la neurona de salida, cuantificando el error global y acelerando el aprendizaje (debido a que se tiene más información). También, se debe mencionar a la **regla de aprendizaje hacia atrás o de backpropagation**, la cual

consiste en una generalización de la regla Delta y fue la primera que permitió realizar cambios sobre los pesos en las conexiones de la capa oculta.

- **Aprendizaje por refuerzo.** Tipo de aprendizaje más lento que el anterior, en el cual el maestro toma un papel de crítico dado que indica mediante una señal de refuerzo si la salida de la red obtenida se ajusta a la deseada (éxito = +1 o fracaso = -1), y en función de ello se ajustan los pesos basándose en un mecanismo de probabilidades.
- **Aprendizaje estocástico.** Asocia a la red con un sólido físico que tiene cierto estado energético, de tal forma que un estado de mínima energía correspondería a una situación en que los pesos de las conexiones consiguen que su funcionamiento sea el que más se ajusta al objetivo deseado. De esta manera, el aprendizaje consistiría en asignar valores aleatorios a los pesos y determinar la energía de la red. Si el valor energético es menor después del cambio (comportamiento se acerca al deseado) se acepta el cambio, de lo contrario se aceptaría el cambio en función de una determinada y preestablecida función de probabilidades.

2.8.2 Aprendizaje No Supervisado.

Este tipo de aprendizaje se caracteriza por que la red no recibe ninguna información por parte del entorno que le indique si la salida generada en respuesta de una determinada entrada es o no es correcta.

Estas redes deben encontrar características, regularidades, correlaciones o categorías que se puedan establecer entre los datos que se presentan en su entrada. Existen varias posibilidades en cuanto a la interpretación de la salida de estas redes, que depende de su estructura y algoritmo de aprendizaje utilizado.

En cuanto a los algoritmos de este tipo de aprendizaje, en general se suelen utilizar 2 tipos que dan lugar a los siguientes aprendizajes:

- **Aprendizaje Hebbiano.** Corresponde a una regla de aprendizaje que pretende medir la familiaridad o extraer características de los datos de entrada. Consiste en que si 2 neuronas toman el mismo estado simultáneamente (ambas activas o ambas inactivas, dado que el origen de la regla es en base a la neurona biológica clásica), el peso de la conexión entre ambas se incrementa.
- **Aprendizaje Competitivo y Comparativo.** Si un patrón nuevo se determina que pertenece a una clase previamente reconocida, entonces la inclusión de este nuevo patrón a esta clase matizara la representación de la misma. En caso contrario, la estructura y los pesos de la red serán ajustados para reconocer la nueva clase.

En el caso de las redes RBF, usualmente se utilizan algoritmos no supervisados para la obtención de los centros y de las anchuras de las funciones de base radial, dentro de los cuales destaca el **K-medias,** utilizada para determinar los centros. Consiste en dividir el espacio de entrada en K clases, cada una con una coordenada (centro) que la representa para luego asignar una serie de patrones de entrada a una clase comparando las distancias euclidianas para determinar la clase en la que se quedara un determinado patrón. Finalmente se calcula el nuevo centro de la clase promediando los patrones asociados a esta. Este proceso se realiza de manera iterativa hasta que la modificación de los centros sea

menos a un mínimo establecido. Luego, las anchuras se pueden determinar utilizando las medias aritméticas de las distancias euclídeas de un centro a los centros más cercanos o bien la media geométrica de un centro a sus 2 vecinos más cercanos.

2.8.3 Aprendizaje Híbrido.

El aprendizaje híbrido corresponde a una mezcla de los tipos de aprendizaje: el aprendizaje supervisado y el aprendizaje no supervisado, descritos en las secciones 3.8.1 y 3.8.2 respectivamente.

En el caso de las redes RBF, la naturaleza dual de la red hace ideal el uso de este tipo de esquema en su aprendizaje. Esto, debido a que se requiere el ajuste de los centros y anchuras de las funciones de base radial, utilizando los patrones de entrada, y luego se deben modificar los pesos de la capa de salida, valiéndose de los resultaos esperados para la muestra ingresada.

Por lo tanto, dado que la capa oculta posee parámetros que permiten representar una zona local del espacio de entrada, se hace natural la utilización de un algoritmo de aprendizaje no supervisado que aproveche dicha característica haciendo uso de los patrones de entrenamiento. Por otro lado, los parámetros de la capa de salida pueden optimizarse haciendo uso de los resultados obtenidos a partir de los patrones de entrada frente a la salida esperada, por lo que en este caso el uso de un algoritmo supervisado es la mejor respuesta.

En conclusión, la elección del algoritmo de aprendizaje a aplicar para realizar el debido entrenamiento a una red va a depender tanto de la estructura de la red neuronal, como del tipo y la información disponible del problema que ha de resolverse.

2.9 Aplicaciones.

Las redes neuronales artificiales son una tecnología computacional que puede utilizarse en un gran número y variedad de aplicaciones. A continuación se presentan algunas áreas en las cuales se aplican redes neuronales artificiales:

- **Optimización** de soluciones que satisfacen ciertas restricciones, con el fin de maximizar o minimizar la función objetivo. En la gestión empresarial, son decisiones de optimización encontrar los niveles de tesorería, de existencias, de producción, construcción de carteras óptimas, etc.
- **Agrupamiento y Categorización** de datos que siguen un patrón común indeterminado, como es el caso de aplicaciones de compresión y minería de datos.
- **Control** de sistemas dinámicos, como por ejemplo (Kumagai y otros, 1999) componen una red neuronal recurrente para un problema de control de péndulo invertido.
- **Asociación y Clasificación.** Donde los patrones de entrada estáticos o señales temporales deben ser clasificados o reconocidos. Idealmente, un clasificador debería ser entrenado para que cuando se le presente una versión distorsionada ligeramente del patrón, pueda ser reconocida correctamente sin problemas. Un ejemplo de

reconocimiento de patrones se presenta en (Zemouri, Racoceanu y Zerhouni, 2007) donde se propone una **red RRBF** aplicada a la detección de una colisión en un brazo de robot manipulador. Esta aplicación representa una solución diagnostico mediante el **reconocimiento de patrones** donde la memoria dinámica de la red permite detectar el estado de degradación antes que se active la alarma.

- **Pronóstico y Predicción** de series de tiempo y que tienen aplicaciones en una gran variedad de áreas, por ejemplo: (Ayala y Castillo, 1999) usando RNA estudia el comportamiento del valor TRM, además del precio de cierre del dólar en el mercado regulado, (Saha y Raghava, 2006) presentan la predicción de continuos epítopos de células B en un antígeno usando redes neuronales recurrentes, (Gutierrez-Estrada y otros, 2008) busca realizar una estimación mensual de la captura de anchovetas mediante métodos heurísticos, dentro de los cuales usa redes neuronales recurrentes.

CAPÍTULO 3

Algoritmos Genéticos

3.1 Introducción.

Los Algoritmos Genéticos (GA), son métodos adaptativos que pueden y utilizarse para resolver problemas de búsqueda y optimización. Con base en los postulados de Darwin (1859), los GA se basan en el proceso genético de los seres vivos, donde las poblaciones a lo largo de las generaciones evolucionan en la naturaleza acorde con los principios de selección natural y supervivencia de los más fuertes. De esta manera, los GA son capaces de ir creando soluciones para problemas del mundo real.

Los GA usan una analogía directa con el comportamiento natural. Trabajan con una población de individuos donde cada uno presenta una solución factible a un problema dado, además, a cada individuo se le asigna un valor que se encuentra relacionado con la bondad de dicha solución (análogo al grado de efectividad de un organismo para competir por unos determinados recursos en la naturaleza). Entonces, cuanto mayor sea la adaptación de un individuo al problema, mayor será la probabilidad de que este sea seleccionado para reproducirse, cruzando su material genético con otro individuo seleccionado de igual forma. Este cruce producirá nuevos individuos (descendientes de los anteriores).

De esta manera, se produce una nueva población de posibles soluciones, que reemplazará a la anterior y se presentará con un mayor porcentaje de buenas características que la anterior. Así, a lo largo de las generaciones, si el algoritmo genético ha sido bien diseñado, se convergirá a una solución óptima del problema.

En el presente capítulo se presenta inicialmente una **reseña histórica** de los algoritmos genéticos dando a conocer como a través de la teoría de Darwin se ha llegado a lo que hoy se conocer como Algoritmos Genéticos. Luego se presentan los **elementos básicos** para así posteriormente presentar el **algoritmo básico.** En las siguientes secciones se presentan los distintos tipos de **codificación** de cromosomas más utilizados y luego un listado de algunos de los **métodos de selección** más conocidos. Finalmente se presentan las **ventajas y limitaciones de los GA** para terminar con un listado de **aplicaciones**.

3.2 Reseña Histórica.

Los primeros ejemplos de lo que hoy se puede llamar Algoritmos Genéticos aparecieron a finales de los años 50 y principios de los 60, programados por biólogos evolutivos que buscaban realizar modelos de aspectos de la evolución natural. En el año 1962, Box, Friedman, Bledsoe y Bremermann habían desarrollado, de manera independiente, algoritmos inspirados en la evolución para la optimización de funciones y el aprendizaje automático. En 1965 Rechenberg introdujo una técnica que llamo **estrategia**

evolutiva en la cual no existía población ni cruzamiento, solo un padre mutaba para crear un descendiente y se quedaba con el mejor de los 2 (Haupt y Haupt, 1998).

El siguiente desarrollo importante ocurre en 1966 Fogel, Owens y Walsh introducen la **programación evolutiva,** la cual es similar a la estrategia evolutiva con la diferencia que las soluciones candidatas para los problemas se representan como máquinas de estado finito sencillas.

En 1962 el trabajo de John Holland sobre **sistemas adaptativos** establece las bases para desarrollos posteriores, y aún más importante, Holland fue el primero en proponer de manera explícita el cruzamiento y otros operadores de recombinación. Sin embargo, el hito fundamental para los algoritmos genéticos ocurre en **1975** con la publicación del libro **"Adaptación de Sistemas Naturales y Artificiales"**, basado en investigaciones y papers de Holland y colegas de la Universidad de Michigan. Este libro fue el primero en presentar sistemática y rigurosamente el concepto de sistemas digitales adaptativos utilizando la **mutación, la selección y el cruzamiento**, simulando el proceso de la evolución biológica como estrategia para resolver problemas.

Entre principios y mediados de los 80, los algoritmos genéticos se estaban aplicando en una amplia variedad de áreas, desde problemas matemáticos abstractos y la coloración de grafos hasta asuntos tangibles de ingeniería como el control de flujo en una línea de ensamble, reconocimiento y clasificación de patrones y optimización estructural (Goldberg, 1989).

Hoy en día, la computación evolutiva es un campo floreciente, y los GA están resolviendo problemas de interés cotidiano en áreas de estudios tan diversos como la predicción en la bolsa y la planificación de la cartera de valores, ingeniería aeroespacial, diseño de microchips, bioquímicos y biología molecular, y diseño de horarios en aeropuertos y líneas de montaje. Y en el corazón de todo esto se halla nada más que la simple y poderosa idea de **Charles Darwin**: que el azar en la variación, junto con la ley de la selección, es una técnica de resolución de problemas de inmenso poder y de aplicación casi ilimitada.

3.3 Elementos de un GA.

Resulta que no existe una rigurosa definición de "Algoritmo Genético" que lo diferencie de otros métodos de computación evolutiva. Por lo general, todos los métodos llamados "Algoritmos Genéticos" tienen al menos los siguientes elementos en común (Mitchell, 1999):

- **Población de Cromosomas:** Todo algoritmo genético se basa en la existencia de cierta población de individuos, los cuales corresponden a posibles soluciones del problema y pueden representarse como un conjunto de parámetros llamados **genes** (los genes pueden tomar una forma determinada llamada **alelo**) y el conjunto de genes forma los **cromosomas.** Buena parte de la teoría en la que se fundamentan los GA utilizan el alfabeto binario para representar a los individuos.
- **Selección de acuerdo a *fitness:*** Los GA requieren de una función **fitness**, diseñada para cada problema de manera específica, la cual asigna un valor a cada cromosoma de la actual población. Este valor dependerá de que tan bien ese cromosoma resuelva

el problema en cuestión. Una vez evaluados los individuos se selecciona el que tenga un mejor desempeño.

- **Cruzamiento:** Consiste en la mezcla de cromosomas de 2 padres a partir de una posición seleccionada de manera aleatoria o en base a un patrón establecido, para la generación de nuevos descendientes. Los nuevos cromosomas presentan soluciones potencialmente mejores que las provistas por sus padres.
- **Mutación:** Consiste en el intercambio aleatorio de algunos bits del cromosoma. Esta puede ocurrir en cada posición de los bits con cierta probabilidad. Usualmente esta propiedad es muy baja (por ejemplo: 0,001).

Todos estos elementos se trabajan de manera conjunta para encontrar soluciones óptimas al problema en cuestión. El proceso consiste en una serie de iteraciones comenzando con la selección de la población inicial, la cual es generalmente aleatoria. Esta aleatoriedad permite a los GA evaluar distintos puntos del espacio en forma paralela. Luego, se evalúan los individuos mediante la función **fitness** descartando los que menos se adapten. Con los restantes, se generan los descendientes (cruzamiento) y de manera aleatoria de mutan para evitar cromosomas repetidos y además mejorar la capacidad de exploración del algoritmo. Obtenida la nueva generación, la cual provee mejores resultados que la anterior, se procede con el ciclo hasta encontrar la solución óptima.

3.4 Algoritmo Genético Básico.

Dado un problema a solucionar definido claramente, una representación en forma de bits, de las soluciones candidatas y basándose en los elementos definidos en la sección anterior. Un GA Simple funciona de la siguiente manera:

- Comenzar con una población de *n* individuos generados de manera aleatoria.
- Calcular el fitness $f(x)$ para cada cromosoma x en la población.
- Repetir los siguientes pasos hasta que *n* descendientes sean creados:
    - Seleccionar un par de cromosomas padres de la actual población, la probabilidad de selección de los individuos se calcula de acuerdo a $f(x)$. Existe la posibilidad de que un cierto individuo sea seleccionado en más de una iteración.
    - Con una probabilidad de p_c (“probabilidad de cruzamiento”) cruzar el par en un punto seleccionado de manera aleatoria para formar 2 descendientes. Si no existe cruzamiento, formar 2 descendientes que sean exactas copias de los padres.
    - Mutar los 2 descendientes con la probabilidad p_m y ubica los cromosomas resultantes en la nueva población.

      Si *n* es impar, un nuevo miembro de la población puede ser descartado de manera aleatoria
- Reemplaza la actual población por la nueva población.
- Ir al paso 2, calculo de fitness.

Cada iteración de este proceso se conoce como generación. Un GA normalmente se itera entre 50 y 500 o más veces, y al final generalmente hay 1 o más cromosomas de la población que encajan correctamente con el problema en cuestión.

El criterio de término va a depender del problema que se desee solucionar, por lo general, los criterios de término más conocidos son: el número de iteraciones, así como un criterio de convergencia, como seguir iterando hasta que la diferencia de los resultados tienda a 0. Además, cabe destacar que una de las grandes desventajas de los GA corresponde al elevado costo computacional dado por su naturaleza iterativa.

Este simple procedimiento describe la base de la mayoría de las aplicaciones de GA. Existen una serie de detalles como el tamaño de la población y las probabilidades de cruce y mutación de los cuales depende el éxito del algoritmo.

3.5 Codificación del Cromosoma

Como para cualquier método de búsqueda y aprendizaje, la manera mediante la cual las soluciones candidatas son codificadas corresponde al factor central en el éxito de un GA. La mayoría de las aplicaciones de GA usan cadenas de bits de largo y orden fijo para codificar a las soluciones candidatas. Aun así, han existido experimentos con otros tipos de codificaciones, algunas de las cuales se presentan a continuación.

3.5.1 Codificación Binaria.

Este tipo de codificación corresponde a una de las más usadas por una serie de razones. Una de ellas se debe a que gran parte de la teoría de la GA se basa en este tipo de codificación y aunque algunas se pueden extender y aplicar codificación no binaria, estas extensiones no se encuentran tan bien desarrolladas como la teoría original.

Holland dio una justificación teórica para el uso de este tipo de codificación. Comparó 2 codificaciones con aproximadamente la misma capacidad de carga de información, uno de ellos con un pequeño número de alelos y cadenas largas (por ejemplo, cadena de bits de largo 100) y el otro con un gran número de alelos y cadenas cortas (por ejemplo, cadena de decimales de largo 30). Holland sostuvo que el primero permite un mayor grado de paralelismo implícito que el segundo, dado que una instancia del binario contiene más esquemas que la del decimal (2^{100} versus 2^{30}).

A pesar de estas ventajas, la codificación binaria es antinatural y poco manejable para muchos problemas (por ejemplo, evolucionar los pesos de una RNA).

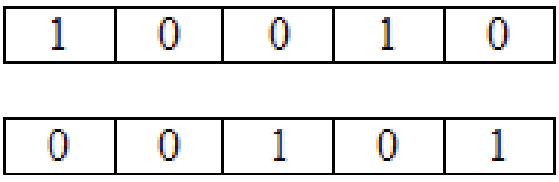

Figura 3-1 Dos cromosomas binarios

3.5.2 Codificación Real y Alfabética

Para muchas aplicaciones, es más común utilizar un alfabeto con varios caracteres o números reales para formar los cromosomas. Aunque Holland argumento que la codificación de este tipo presenta un peor resultado que la binaria, se han realizado muchas comparaciones en las cuales la codificación real y alfabética ha demostrado mejor rendimiento (Wright, 1991). Pero el problema mucho del problema y de los detalles del GA que se está utilizando.

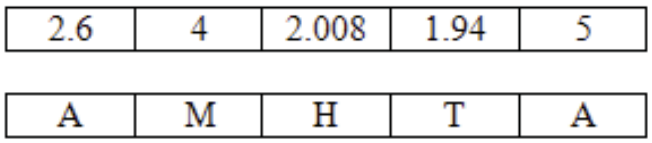

Figura 3-2 Ejemplo de Cromosoma real y alfabético

3.6 Métodos de Selección

Luego de definir cuál será el tipo de codificación que se aplicara a los cromosomas del GA, se debe decidir la manera mediante la cual se realizara la **selección**. La selección consiste en la elección de los individuos de la población para la creación de descendientes de la nueva generación con la esperanza de que éstos (descendientes) presenten un mejor desempeño (fitness).

Aunque se espera que la selección se base en alguna condición de rendimiento de los individuos, no es recomendable elegir sólo miembros de la población con altos índices de desempeño, debido a que puede mermar la capacidad exploratoria del algoritmo, al ser sensible de quedar atrapado en un desempeño sub-óptimo. Por otro lado, la selección de miembros de bajo rendimiento dará como resultado una lenta evolución. La idea es que la selección mantenga un balance entre la exploración y explotación del espacio de búsqueda.

Numerosos esquemas de selección han sido propuestos en la literatura. A continuación se presentan algunos de los métodos más comunes.

3.6.1 Selección de “Rueda de Ruleta”.

La selección de “Rueda de Ruleta”, también conocida como **selección de Ruleta** consiste en que el valor esperado de un individuo (es decir, el número de veces que un individuo será seleccionado para reproducir) estará dado por la probabilidad definida por la división del fitness del individuo por la suma de los fitness de la población.

De esta manera el tamaño asignado de la ruleta para cada individuo será proporcional a su desempeño y aunque los de mejor desempeño tendrán una mayor probabilidad de ser seleccionados, también existe la posibilidad de que uno de menor desempeño lo sea, aumentando la diversidad de la población.

3.6.2 Elitismo

Es una adición a muchos métodos de selección que obliga a los GA a conservar algunos de los mejores individuos de la población en cada generación con el fin de no perder los lugares explorados del espacio de búsqueda que son buenos candidatos. Estos individuos pueden perderse si no son seleccionados para la reproducción o si son destruidos en un cruce o mutación.

3.6.3 Selección de Boltzman

Este método se selección se basa en el uso de distintos grados de exigencia del fitness de los individuos a seleccionar a medida que avanza el numero de iteraciones, es decir, en iteraciones tempranas se busca no ser tan riguroso con el desempeño de los miembros de la población permitiendo una mayor variabilidad y maximizando las posibilidades de exploración. A medida que avanzan las iteraciones se puede ir aumentando gradualmente las exigencias de desempeño con el fin de maximizar las posibilidades de explotación del algoritmo.

3.6.4 Selección por Torneo.

La idea principal de este método consiste en realizar la selección en base a comparaciones directas entre individuos. Existen dos versiones de selección mediante torneo:

- Determinística
- Probabilística

En la versión **determinística** se selecciona al azar un número n de individuos (generalmente se escoge n=2). De entre los individuos seleccionados se selecciona el que presente un mejor fitness para pasarlo a la siguiente generación.

La versión **probabilística** se diferencia en la selección del ganador del torneo. En vez de escoger siempre el mejor se genera un número aleatorio del intervalo [0..1], si es mayor que un parámetro p (fijado para todo el proceso evolutivo) se escoge el individuo que presente mejor desempeño y en caso contrario el que presente un desempeño menor. Generalmente p toma valores en el rango $0,5 < p \leq 1$.

Si el tamaño de n es pequeño, existen mayores probabilidades de que se selecciones individuos con valores pequeños respecto de la población total, pero alto respecto del subconjunto, aumentando así la variabilidad de la nueva población y mejorando también la capacidad de exploración. Por otro lado, si el subconjunto es grande (cercano al tamaño de la población) aumenta la probabilidad de seleccionar el mejor individuo de toda la población, mejorando la explotación del espacio de búsqueda.

Elegir uno u otro método de selección determinará la estrategia de búsqueda del Algoritmo Genético. Si se opta por un método con un alto requerimiento de fitness se centra la búsqueda de las soluciones en un entorno próximo a las mejores soluciones

actuales. Por el contrario, optando por un requerimiento de desempeño menor se deja el camino abierto para la exploración de nuevas regiones del espacio de búsqueda.

3.7 Ventajas y Limitaciones de los GA.

3.7.1 Ventajas.

- Una de las ventajas más importantes de los GA es que son **intrínsecamente paralelos**. En comparación a los algoritmos en series, estos últimos exploran el espacio de búsqueda en una sola dirección al mismo tiempo. En caso de encontrarse con una solución sub-óptima debe deben comenzar todo de nuevo. Por otro lado, dado que los GA tienen descendencia múltiple pueden explorar el espacio de búsqueda en varias direcciones a la vez.

  Sin embargo, la ventaja del paralelismo va más allá de lo explicado en el párrafo anterior, por ejemplo, todas las cadenas binarias de 8 dígitos forman un espacio de búsqueda, el cual puede representarse como ******** (donde, * = 0 o 1). La cadena 01101010 es un miembro de este espacio. Sin embargo, también 01101010 es un miembro del espacio:

  - 0*******
  - 0*1*1*1*
  - 01*01**0
  - Etc.

  Por lo tanto, al evaluar el fitness de esta cadena en particular, el GA estará sondeando cada uno de los espacios a los que pertenece. De esta manera, un GA que evalúe explícitamente un número pequeño de individuos está evaluando de manera **implícita** un grupo de individuos mucho más grande.
- Debido al **paralelismo** funcionan particularmente resolviendo problemas **no lineales** los cuales presentan espacios de soluciones potenciales relativamente grandes. A diferencia de un problema lineal, donde una mejora en alguna parte dará como resultado una mejora del sistema completo, en problemas no lineales un cambio en un componente puede tener efectos en cadena en todo el sistema.
- Han demostrado efectividad escapando de los **óptimos locales** y descubrir el **óptimo global**. Tanto el paralelismo, selección, mutación y cruzamiento juegan un papel importante en esto, pero el que marca la diferencia es el **cruzamiento.** La importancia de éste ultimo radica en que permite la transferencia de información entre los candidatos de mejor rendimiento (se benefician del aprendizaje de los otros) y los esquemas pueden mezclarse y combinarse con el potencial de generar una descendencia que tenga las virtudes de los padres y ninguna de sus debilidades.
- En contraste a las estrategias de resolución de problemas que dependan de conocimiento previo, las cuales comienzan descartando caminos y perdiendo así cualquier solución novedosa pueda existir. Los GA actúan como **Relojes Ciegos**, comenzando con una población aleatoria, realizando cambios aleatorios en individuos y usando el fitness para determinar si los cambios producen mejora.

Dicho lo anterior, cualquier técnica que dependa de conocimiento previo fracasara tras la ausencia de este, en cambio, los GA no se verán afectados por tal ignorancia pues gracias a sus elementos pueden viajar por el amplio espacio de búsqueda revelando potenciales soluciones que no podrían habérseles ocurrido a diseñadores humanos.

3.7.2 Limitaciones.

- La primera limitación y la más importante consiste en la definición de la representación del problema. El lenguaje para especificar las soluciones debe tolerar cambios aleatorios que no produzcan errores fatales o resultados sin sentido. La manera más utilizada para conseguir esto corresponde a definir a los individuos como listas de números (binarios, enteros o reales) tal como se definió en la sección 4.5.
- La definición de la función fitness no es trivial, debe considerarse cuidadosamente de manera que permita encontrar el mejor desempeño y signifique una solución mejor para el problema dado. Una mala o inexacta definición puede ser incapaz de encontrar una solución o resolver un problema equivocado.
- Requiere de una cuidadosa selección del tamaño de la población, el ritmo de mutación y cruzamiento, el tipo y fuerza de la selección. Si el tamaño de la población es demasiado pequeño, puede que el GA no sea capaz de abarcar todo el espacio de posibles soluciones. Si el ritmo de cambio genético o la selección se escoge de manera inadecuada, se puede alterar el desarrollo de esquemas beneficiosos y la población puede entrar en una catástrofe de errores.

3.8 Aplicaciones.

En la literatura los GA se han utilizado para abordar una amplia variedad de problemas en un conjunto de campos sumamente diverso, demostrando claramente su capacidad y su potencial. A continuación se listan algunas aplicaciones:

- **Problemas de Optimización.** (Sato y otros, 2002) utilizaron algoritmos genéticos para diseñar una sala de conciertos que presenten propiedades acústicas óptimas, maximizando la calidad del sonido para la audiencia, el director y los músicos del escenario. Dentro de área espacial (Williams, Crossley y Lang, 2001) aplicaron GA para situar las órbitas de los satélites de manera que se minimicen los apagones de cobertura.
- **Problemas de Predicción.** (Mahfoud, y Mani, 1996) utilizaron GA para predecir el rendimiento futuro de 1600 acciones ofertadas públicamente. Además se compara con el rendimiento de una red neuronal usada a la fecha, superando a esta última notablemente.
- **Evolución de Redes Neuronales. (**Andreou, Georgopoulos y Likothanassis, 2002) utilizaron algoritmos genéticos híbridos para evolucionar redes neuronales que predijeran los tipos de cambio de monedas extranjeras hasta un mes en el futuro. Mediante el uso de GA se evolucionó la arquitectura (número de unidades de

entradas y ocultas, junto con la estructura de enlaces entre ellas). También en el mundo de los juegos (Chellapilla y Fogel, 2001) utilizaron GA para evolucionar RNAs que jugaran a las damas. Los resultados fueron tan alentadores, que llego a competir con el mejor juego de damas del mundo. Aunque perdió 4-2, la principal desventaja fue la falta de una base de datos de finales de partidas.

La evolución de las redes neuronales mediante el uso de algoritmos genéticos consiste tanto en la definición de las estructura de la red como del entrenamiento (asignación de valores a los pesos). Este proyecto busca determinar estas estructuras y pesos mediante un híbrido formado por GA y PSO. Las características de éste último se presentan en el siguiente capítulo.

CAPÍTULO 4

Optimización por Enjambre de Partículas

4.1 Introducción.

La optimización por enjambre de partícula, más conocida en la literatura científica como *Particle Swarm Optimization* (PSO), nace, al igual que otras técnicas estocásticas del cálculo evolutivo, en un intento por imitar y mimetizar el comportamiento de los procesos naturales. Siendo uno de los métodos más utilizados en la inteligencia computacional, busca imitar comportamientos sociales de un colectivo a partir de la interacción entre sus individuos y de éstos con su entorno.

Sus orígenes remontan de los estudio realizados por (Kennedy y Eberhart, 1995), quienes intentaron simular de manera gráfica el movimiento sincronizado e impredecible de bancos de peces o bandadas de aves, intrigados por la capacidad que tienen estos grupos para separarse, reagruparse o encontrar su alimento. Paralelamente con trabajos previos en el ámbito de la biología y la sociología, concluyen que el comportamiento, inteligencia y movimiento de estas agrupaciones está relacionado directamente con la capacidad para compartir información y aprovecha de la experiencia acumulada por cada uno de los congéneres.

(Kennedy y Eberhart, 1995) introducen el término **partícula,** el cual representa a cualquier tipo de individuo que muestre algún tipo de comportamiento social como grupo en forma de una colección de agentes o partículas que interactúan entre si. De acuerdo a los fundamentos teóricos del método, el movimiento de cada una de estas partículas hacia la consecución de un objetivo en común depende de 2 factores: la **memoria autobiográfica** de la partícula **o nostalgia** y la **influencia social** de todo el enjambre. Esto se puede extender, dependiendo del problema bajo análisis, a nivel computacional de D-dimensiones, siendo N el número de incógnitas. Básicamente, el proceso evolutivo se reduce a mover cada partícula dentro del espacio de soluciones con una velocidad que variará de acuerdo a su **velocidad actual**, a la **memoria de la partícula** y a la **información global** que comparte el resto del enjambre, utilizando una función de **fitness** para cuantificar la calidad de cada partícula en función de la posición que ésta ocupe.

Existen diversos tipos de esquemas para la implementación de PSO. Dependiendo de cómo se actualicen las posiciones de las partículas surgen versiones sincronías y asíncronas del método. Además, se puede distinguir entre PSO local y global dependiendo de cómo fluya la información a través del enjambre.

En el presente capítulo se comienza con la base de PSO relacionada con el **fundamento del movimiento de partículas**, para luego dar a conocer el **algoritmo básico**.

Posteriormente se estudian los **parámetros** y las **topologías** más comunes. Finalmente se presentan 2 **variantes** del método y se termina con un listado de **aplicaciones.**

4.2 Fundamentos del movimiento de partículas.

Para aplicaciones relacionadas con el ámbito de la vida artificial se deben respetar cinco principios básicos de lo que se conoce como **inteligencia de grupo**, estos se conocen como: proximidad, calidad, diversidad de respuesta, estabilidad y adaptabilidad.

- **Proximidad.** la población debiera ser capaz de realizar cálculos sencillos de espacio y tiempo, lo cual en PSO se traduce a movimientos en D-dimensiones llevados a cabo durante un intervalo de tiempo que coinciden con movimientos de la población a una determinada velocidad.
- **Calidad.** Los factores de calidad en PSO se consiguen en base a la **memoria** de la partícula junto con la historia o conocimiento social que comparten todos los congéneres.
- **Diversidad de respuesta.** Representadas por las tendencias marcadas por la memoria personal de cada partícula y por la historia de la mejor posición visitada por el conjunto.
- **Estabilidad y Adaptabilidad.** Resaltan aspectos contrapuestos, dado que por un lado, la población solo cambia su comportamiento como grupo cuando se actualiza la mejor posición históricamente visitada por alguno de los miembros que lo integran (principio de estabilidad) y, por otro lado, la población debe ser capaz debe ser capaz de modificar su comportamiento y movimiento cuando hay alguna señal que así lo recomienda. Esto último, desde el punto de vista de ahorro computacional o mejora de precisión. En PSO esto se consigue cuando alguna partícula alcanza una solución global que mejora el resultado, en ese instante la población cambia de rumbo.

4.3 Algoritmo PSO

Una partícula se define como un vector formado por la velocidad, posición y memoria de la partícula. PSO se inicializa con partículas aleatorias y optimiza sus soluciones a medida que aumentan las iteraciones. El enjambre es representado por $X = (x_{i1}, x_{i2}, ..., x_{id})$ donde i=1,2,…, n corresponde a la i-ésima partícula y d=1,2,…, D corresponde a sus dimensiones.

En cada iteración cada una de las partículas del enjambre se actualizada en base a dos valores importantes:

- Mejor posición encontrara por una **partícula,** conocida como p_{best}. El p_{best} de una i-ésima partícula se representa por $Pi = (p_{i1}, p_{i2}, ..., p_{id})$.

- Mejor posición que ha encontrado el **enjambre**, conocida como p_{gbest} .

Ambos de estos valores se actualizan en cada iteración y se utilizan ajustar la velocidad con la que se mueve una determinada partícula en cada una de las dimensiones. La influencia que tiene la mejor posición personal sobre la velocidad de una partícula se conoce como el **factor de cognición** y la influencia de la mejor partícula del enjambre se conoce como **componente social**.

En (Kennedy, Eberhart y Shi, 2001) se incorpora el **factor de inercia** *w* en la ecuación que determina la velocidad, de tal manera de que se balancee la búsqueda global y local.

$$v_{id}(t+1) = w \cdot v_{id}(t) + c_1 r_1 (p_{id}(t) - x_{id}(t)) + c_2 r_2 (p_{gd}(t) - x_{id}(t)) \tag{14}$$

$$x_{id}(t+1) = x_{id}(t) + v_{id}(t+1) \tag{15}$$

Donde r_1 **y** r_2 corresponden a valores aleatorios independientes en el intervalo [0,1], y c_1 **y** c_2 corresponden a las variables que controlan la influencia de las componentes cognitiva y social.

Para prevenir que las velocidades de las partículas incrementen infinitamente, se incorpora un parámetro V_{max} que delimita el rango de velocidad [$-V_{max}$,V_{max}] que puede tomar una partícula.

$$if\,(v_{ij} > V_{\max}) \quad or \quad (vij < -V_{\max}) \tag{16}$$

$$v_{ij} = sign(v_{ij}) \cdot V_{\max} \tag{17}$$

A continuación se presentan los seis pasos relevantes que constituyen el algoritmo de PSO Tradicional:

- **Paso 1:** Se debe definir el tamaño *n* del enjambre y se deben inicializar las posiciones de las velocidades de las partículas en un espacio D-dimensional.
- **Paso 2:** Evaluar fitness de cada partícula en base a algún criterio de optimización (función de costo, mínimos cuadrados, error cuadrático medio).
- **Paso 3:** Comparar el fitness de cada partícula obtenido en el Paso 2 con el fitness de su mejor posición personal. Si el fitness actual es mejor que el histórico, se debe actualizar esta mejor posición por la nueva obtenida.
- **Paso 4:** Comparar el fitness actual de cada partícula con el mejor fitness encontrado por el enjambre, si el actual es mejor, entonces se actualiza la mejor posición global con el valor de la posición actual.
- **Paso 5:** Ajustar la velocidad y posición de la partículas en base a las ecuaciones de velocidad y posición descritas en este capítulo. Se debe verificar que las partículas no sobrepasen la velocidad máxima definida $V_{\max}$.
- **Paso 6:** Si se cumple algún criterio de termino término, en caso contrario se debe volver al Paso 2. Los criterios de términos más conocidos pueden ser:

- Número máximo de iteraciones
- Numero de iteraciones sin mejoras
- Error mínimo de la función objetivo

4.4 Parámetros PSO.

Tal como se visualizo con los GA, el ajuste de los parámetros condicionará el rendimiento posterior del algoritmo de optimización. La selección está directamente ligada a la naturaleza del problema que se desea optimizar, siendo preciso lograr un balance óptimo entre la exploración y la convergencia. La definición de la función **fitness** cumple un papel no menor, dado que una función que no introduzca una métrica adecuada para pesar la bondad de cada partícula, hará emerger las carencias del PSO, las cuales erróneamente pueden asociarse con el algoritmo.

Para **acotar la velocidad de la partícula** se define un valor máximo $V_{\max}$, el cual restringe a la velocidad en cada dimensión del intervalo $[-V_{\max}, V_{\max}]$. Si el valor de $V_{\max}$ tiende a ser demasiado pequeño, las partículas exploraran el espacio de soluciones muy lento y podrán quedar atrapadas dentro de soluciones locales. Por otro lado, la no restricción de la velocidad llevaría a la no convergencia del enjambre en un punto, ocurriendo lo que se conoce como **explosión del PSO**, el cual corresponde a un comportamiento oscilatorio y creciente de la posición de las partículas provocando la ineficiencia del PSO como algoritmo de optimización.

El **peso de inercia *w***, regula la compensación entre la capacidad de exploración global y locales del enjambre. A mayor *w*, facilita la exploración global y en caso contrario facilita la exploración local. En un comienzo *w* se asigna como una constante, sin embargo, resultados experimentales recomiendas la asignación de mayores valores en un comienzo para que en una primera instancia se examine de manera global el espacio de búsqueda y gradualmente se disminuya para comenzar a conseguir soluciones más refinadas.

Los parámetros c_1 **y** c_2, luego de una buena sintonización, pueden acelerar la convergencia del algoritmo y aliviarlo de los mínimos locales. Típicamente a estos parámetros se les asigna 2 o 1.49. Trabajos recientes revelan que podrían ser un parámetro cognitivo c_1 más grande que el social c_2, siempre y cuando se cumpla que $c_1 + c_2 \leq 4$ (Kennedy, Eberhart y Shi, 2001).

El **tamaño de la población** debe ser seleccionado de manera rigurosa, dado que valores muy grandes pueden explorar de manera minuciosa el espacio de búsqueda pero costo computación se eleva de manera considerable debido al aumento del número de evaluaciones de la función fitness. Generalmente se utilizan poblaciones ente 10 y 50 o 100 y 200 para problemas completos.

4.5 Topologías de vecindad local y global.

En PSO los individuos mejoran sus fitness dado que en cada iteración imitan los comportamientos y tendencias que encuentran en los mejores congéneres de la población. Establecer cómo se define la vecindad de un individuo tiene una trascendencia vital en el rendimiento del algoritmo. Dependiendo de la topología que adquiera la población, la transmisión de la información entre individuos puede acelerarse o ralentizarse, lo cual está íntimamente relacionado con la velocidad de convergencia y con la capacidad del algoritmo para escapar de soluciones locales.

La topología más extendida es la **topología de vecindad global**, en la cual todos los individuos están interrelacionados y tienen acceso inmediato a los hallazgos de sus congéneres. Sin embargo, esta estructura social es vulnerable a soluciones locales, dado que dependiendo de la distribución puntual de las partículas sobre el espacio de soluciones, una de ellas apuntando a una solución local puede llegar a dominar al resto.

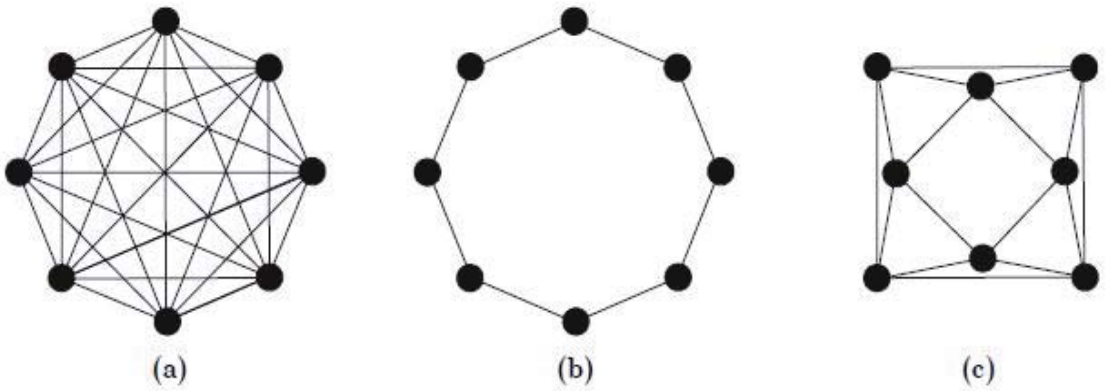

Figura 4-1Topologías de la Población

Global, (b) Local con N=2, (c) Local con N=4

El modelo del **óptimo local** permite que cada individuo sea influenciado por un reducido número de miembros adyacentes. Estas partículas vecinas no tienen relación directa con el resto de las partículas que pertenecen a otra vecindad. Por lo general, como se visualiza en la Fig. 4.1, el óptimo local tienen 2 vecinos.

Autores en (Abraham, Guo y Liu, 2006) declaran que el modelo del optimo global converge rápidamente hacia las soluciones del problema, pero tiene la debilidad de quedar atrapado en mínimos locales, mientras que el optimo local tienen la desventaja de converger lentamente hacia la solución del problema.

4.6 PSO con actualizaciones síncronas y asíncronas.

La diferencia entre una actualización asíncrona con una síncrona, radica en el instante en el cual se realiza la actualización de la memoria de cada partícula y el conocimiento social del grupo.

En el modelo síncrono, todas las partículas se mueven en paralelo. El fitness de las partículas se evalúa en cada iteración, se actualiza su memoria p_{id} y el conocimiento social p_{gd}. Por otro lado, en el PSO asíncrono al momento de desplazarse, cada partícula aprovecha la información actualizada por sus inmediatos predecesores. Es decir, en cada iteración *k*, la *i*-ésima partícula se desplaza hacia un nuevo punto Utilizando la información de los vectores p_{id} y p_{gd}, actualizados por las *i*-1 partículas previas. Posteriormente, la partícula evalúa la calidad del nuevo punto y actualiza, si procede, las variables p_{id} y p_{gd}. Esta información se transmite a las restantes partículas. Al actualizar la información partícula a partícula, el modelo asíncrono acelera la optimización, aunque la naturaleza del modelo síncrono lo hace susceptible de ser ejecutado en paralelo, sobre múltiples procesadores.

4.7 Exploración y Explotación con PSO

Uno de los problemas de los algoritmos evolutivos, es que en ocasiones quedan atrapados en óptimos locales, lo cual en la literatura se conoce como **convergencia prematura** del algoritmo y corresponde a la incapacidad de salir la zona en la cual se encontró el mínimo local, impidiendo que se exploren otras regiones del espacio de búsqueda.

Lo ideal es que las partículas representen soluciones de diferentes zonas del espacio de búsqueda. Esto se conoce como **diversificación** y en términos de los algoritmos corresponde a que las partículas sobrevuelen diferentes zonas del espacio de soluciones antes de que se comprometan con alguna en especial. Por otro lado, las partículas, una vez encontrada una zona comprometedora deben ser capaces de mejorar la solución encontrada. Esto se conoce como **convergencia de la población.**

Se define como **exploración** a la búsqueda de la solución del problema en el espacio de búsqueda de manera amplia, privilegiando la diversificación del enjambre. Por otro lado, la **explotación** consiste en la búsqueda de la solución del problema en un rango (sub-espacio) dentro del espacio de búsqueda donde el enjambre trate de tomar todos los valores posibles, privilegiando la convergencia del enjambre.

Estudios realizados en (Kennedy, Eberhart, y Shi, 2001) revelan que altos valores para el coeficiente de inercia, factores de cognición y $V_{\max}$ ($0.75<w<1$ y $2<c_1,c_2<4$), favoreces el modo de exploración del PSO, mientras que valores bajos ($0.4<w<0.75$ y $0.1<c_1,c_2<2$) favorecen el modo de explotación del PSO.

La idea es ajustar los parámetros de tal manera que el algoritmo funcione en modo de exploración en las primeras etapas y una vez transcurridas unas cuantas iteraciones cambie a modo explotación.

4.8 Variaciones del Algoritmos de PSO

En la actualidad existen una serie de variaciones del algoritmo PSO tradicional, las cuales incorporan nuevos parámetros, nuevos métodos de actualizaron, e híbridos con otros algoritmos. La mayoría de estas mejoras características como la solución encontrada y la velocidad de convergencia. Pese a esto, algunas versiones agregan un mayor nivel de complejidad a la versión tradicional. A continuación se describen algunas variaciones.

4.8.1 PSO con parámetro de Tiempo de Vida

Los autores de (Khosla y otros, 2003), proponen esta variación del algoritmo tradicional en donde introducen un nuevo parámetro llamado **tiempo de vida,** el cual se le asigna a cada una de las partículas del enjambre y decrecerá a medida que ésta no rinda en alguna iteración. Los autores dicen que la decisión de destruir la peor partícula es tomada solo después que se cumpla un número definido de oportunidades de mejora.

Al momento de la eliminación, se asume que la partícula que está teniendo el peor valor de fitness será eliminada, y para reemplazarla se generar una nueva partícula en la vecindad de las partículas restantes. Finalmente, la incorporación de los parámetros de tiempo de vida logra que el PSO rinda mejor en términos de cantidad de iteraciones para la convergencia y calidad del óptimo encontrado.

4.8.2 PSO modificado para Localizar Todos los Mínimos Globales

En (Parsopoulos y Vrahatis, 2001) los autores mencionan que los avances recientes en la ciencia, economía e ingeniería, dependen de técnicas numéricas para computar soluciones globales optimas para solucionar problemas de optimización. De acuerdo a esto, los autores proponen una estrategia para encontrar todos los mínimos globales (o algunos en caso de ser infinitos) de una función objetivo utilizando una modificación a la técnica de PSO.

Para una función objetivo f que tiene varios mínimos globales dentro del hiperespacio D. Si se usa el algoritmo tradicional de PSO solo para computar un solo minimizador global, un $\bar{x} \in D$ tal que $f(\bar{x}) \leq f(x)$, $\forall x \in D$, pueden ocurrir 2 cosas: PSO encuentre un mínimo global (pero no se sabe cuál) o que el enjambre vague por el espacio de búsqueda sin poder decidir donde parar (debido a la buena equidad de información que tiene cada minimizador global).

En varias aplicaciones, como la de entrenamiento de red neuronal (Parsopoulos y Vrahatis, 2001), el objetivo es encontrar el minimizador global de una función no negativa. El valor del mínimo global es conocido antes de examinar el asunto concreto, pero hay un

numero finito (o infinito en un caso de una red neuronal) de minimizadores globales. Para evitar este problema, los autores proponen lo siguiente: determinar un umbral no pequeño $\varepsilon > 0$ (por ejemplo, si la exactitud deseada es 10^{-5}, un valor alrededor de 0.01 ó 0.001) y cuando una partícula tiene un valor funcional que sea menor que ε, se saca la partícula de la población y se aísla. Simultáneamente, se aplicaría deflación o estiramiento 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. 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 $\bar{x}$:

$$G(x) = f(x) + \gamma_1 \frac{\| x - \bar{x} \| (sign(f(x) - f(\bar{x})) + 1)}{2} \tag{18}$$

$$H(x) = G(x) + \gamma_2 \frac{sign(f(x) - f(\bar{x})) + 1}{2 \tanh(\mu(G(x) - G(\bar{x})))} \tag{19}$$

Donde γ_1, γ_2 y μ son constantes positivas elegidas arbitrariamente, y el $sign(\cdot)$ define la función de "signo" de valores dado en (20), según:

$$sign(x) = \begin{cases} +1 & x > 0, \\ 0 & x = 0, \\ -1 & x < 0, \end{cases} \tag{20}$$

Así, después de aislar la partícula, se verifica su valor funcional. Si el valor funcional 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 minimizadores.

Si se supiera el número de minimizadores globales para *f* existentes en *D*, todos ellos serían encontrados después de algunos ciclos. En este caso no se conoce el número de minimizadores globales. Se podría preguntar por un número específico de minimizadores o dejar que el PSO corra hasta que alcance su número máximo permitido de iteraciones en un ciclo. Esto implicaría que ningún otro minimizador podría ser detectado por PSO.

4.9 Aplicaciones.

PSO ha recibido una gran atención en muchas aplicaciones y áreas de investigación durante los últimos años. Su carácter evolutivo junto con la gran cantidad de mejoras que presenta la literatura han dado pie para una gran cantidad de aplicaciones, como por ejemplo: en problemas electromagnéticos, en la síntesis de agrupaciones lineales de antenas y más aún en el aprendizaje de las Redes Neuronales. Este último es de gran importancia para motivos del presente caso de estudio.

A continuación se presentan las aplicaciones listadas en el párrafo anterior y además cabe destacar que el uso de PSO en algunos de estos casos resulta como híbrido con GA.

- **Electromagnetismo.** (Alfassio y otros, 2005) presenta un algoritmo híbrido evolutivo llamado GSO (Genetical Swarm Optimization) producto de la combinación de GA con PSO. En general, debido a la rápida búsqueda global en el espacio sin quedar atrapado en mínimos locales, GSO puede ser adoptado para resolver adecuadamente problemas de optimización en la síntesis de arrays lineales y planos, en materiales EBG o en el diseño de estructuras de protección.
- **Síntesis de Arrays.** (Correa y otros, 2005) compara PSO y Gas en la síntesis de arrays concluyendo que la aplicación de ambos metidos aplicados a la síntesis de alimentaciones en agrupaciones lineales de antenas demuestra la mutua validez pero que la sencillez de implementación, ajuste, rapidez de convergencia priman a PSO frente a los Gas.
- **Entrenamiento de RNAs.** En (Shen y otros, 2004) se ocupa un híbrido, formado por 2 variantes de PSO, para calcular los pesos de una RNA y definir la estructura, ambas entrenadas de manera simultánea demostrando una rápida convergencia. Por otro lado, (Noman y Mariyam, 2009) trabajan el entrenamiento híbrido de una red RBF mediante el uso de PSO. Los resultados se comparan con los obtenidos en estudios previos realizados con Backpropagation y RBF, estos reflejan que PSO es mucho mejor tanto en términos de convergencia y en tasas de error.

CAPÍTULO 5

Redes Neuroevolutivas

Defínase a una Red Neuroevolutiva como toda aquella Red Neuronal Artificial cuya estimación de parámetros internos se realiza mediante la utilización de Algoritmos Evolutivos.

En este capítulo se presentan 4 modelos de Redes Neuroevolutivas Recurrentes. Los Algoritmos utilizados en el proceso de estimación de parámetros son GA y PSO, y las recurrencias que presentan las redes son Elman y Jordan.

En una primera sección se presentan las principales métricas utilizadas para medir la calidad de los modelos. Luego en las siguientes secciones se presentan los modelos desarrollados, comenzando por aquellos sin aprendizaje híbrido y terminando con 2 algoritmos de híbridos correspondientes al basado en el proceso de **Inicialización de Parámetros** y **Breeding Swarm** (Settles y Otros). Finalmente se presentan los resultados del estudio comparando los modelos presentados de manera que se determine cuál de ellos presenta una mejor capacidad predictiva frente al pronóstico del nivel de captura de Anchovetas.

5.1 Métricas de Evaluación

En el presente estudio, para la selección de las topologías y, por consiguiente, de los modelos que presentan mejores resultados se utilizaron un conjunto de métricas de exactitud calculadas entre los datos observados (valores reales) y los datos pronosticados (entregados por la red). Estas métricas se nombran a continuación.

- Error Cuadrático Medio (RMSE). Consiste en la suma de las diferencias entre los datos observados y los datos proyectados por el modelo.

$$RMSE = \sqrt{\frac{\sum_{i=1}^{N} (d_i - y_i)^2}{N}} \tag{21}$$

- Coeficiente de Determinación (R^2). Mide la dependencia entre los datos reales y los pronosticados. El 0 muestra independencia y el 1 lo contrario.

$$R^2 = 1 - \frac{\sum_{i=1}^{N}(d_i - y_i)^2}{\sum_{i=1}^{N}(d_i - \bar{d})^2} \tag{22}$$

- Porcentaje de Error Medio Absoluto (MAPE). Proporciona una indicación de que tan grandes son los errores de pronóstico comparados con los valores reales de la serie. También, correspondiente a la operación interna de la sumatoria se encuentra el Porcentaje de Error Absoluto (APE).

$$MAPE = \frac{\sum_{i=1}^{N}\left|\frac{d_i - y_i}{d_i}\right|}{N} \times 100 \qquad d_i \neq 0 \tag{23}$$

Se define para las formulas presentadas a d_i al valor observado en el mes i, y_i al valor pronosticado en el mes i, $\bar{d}$ como la media de la data observada y N como el número total de meses computados.

También dentro del análisis de los resultados obtenidos se utilizan dos métricas para hacer referencia al costo computacional que conlleva el entrenamiento de los modelos. El primero de ellos corresponde al **tiempo promedio de entrenamiento** que presenta cuánto tarda en promedio un modelo en especial en realizar el entrenamiento, esto, teniendo en cuenta que para todos los modelos se utilizará la misma cantidad de número de iteraciones y solo cambiarán las topologías. Debido a esto último, se agregar otra medida de referencia correspondiente al **número de neuronas** de manera que se pueda visualizar como influye la topología en el tiempo de entrenamiento.

Finalmente, cabe destacar que el equipo en el cual se realizarán todos los entrenamientos y pruebas corresponde a un Intel Celeron 2.8 GHz y 1,50 GB de RAM.

5.2 Modelo RRBF-GA

Este modelo consiste en una red RRBF cuyos parámetros son estimados mediante la utilización de Algoritmos Genéticos. Al igual que el resto de los modelos, los métodos de recurrencia utilizados corresponden a Jordan y Elman, explicados en las secciones 2.6.1 y 2.6.2 respectivamente. El proceso de entrenamiento se presenta en la siguiente figura.

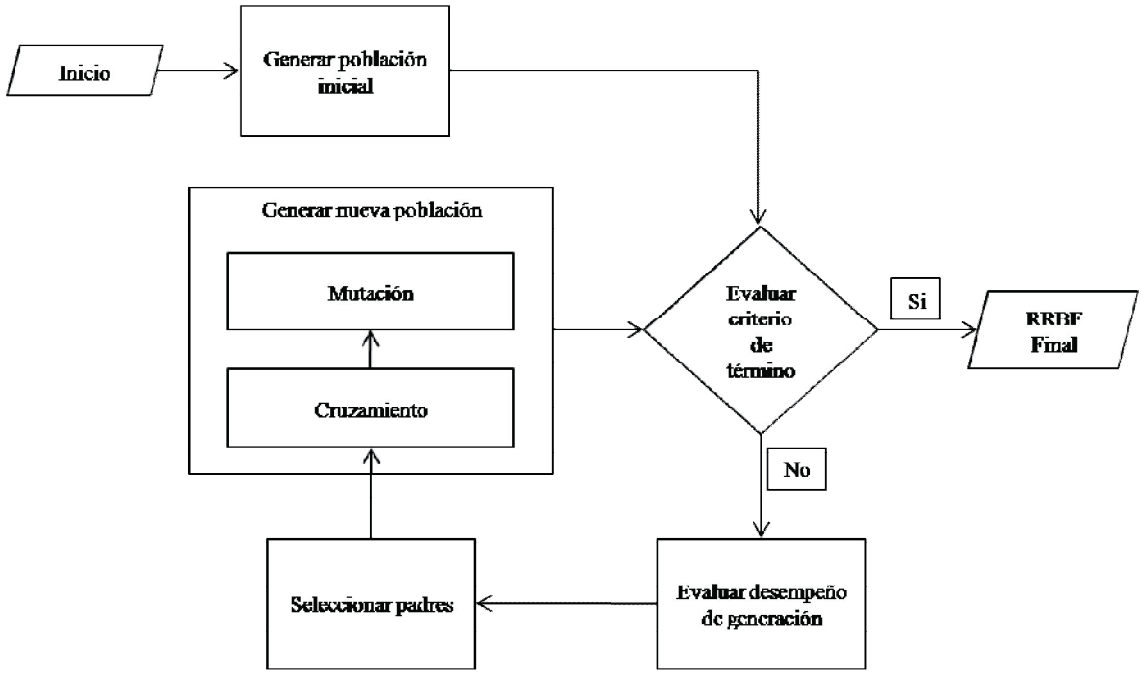

Figura 5-1 Algoritmo Modelo RRBF-GA

5.2.1 Descripción del modelo

La **población inicial** consiste en un total de N individuos, cada uno de estos representados como un **cromosoma.** La codificación de los parámetros de manera inicial fue binaria, pero dado el costo computacional exigido y los bajos resultados obtenidos se utilizo codificación real, presentando mejores resultados. El cromosoma cuenta con la siguiente estructura:

C_{11}	…	C_{1n}	σ_1	…	C_{m1}	…	C_{mn}	σ_m	W_1	…	W_m

Figura 5-2 Cromosoma Modelo RRBF – GA

Siendo m el número de neuronas ocultas y n el número de neuronas de entrada, $C_i,...,C_m$ el conjunto de centros asociados a la neurona oculta i ($0 < i < n$), $\sigma_{1...}\sigma_m$ las anchuras de la función de base radial, y $w_i,...,w_m$ son los pesos lineales ubicados entre la capa oculta y la capa de salida.

Los **criterios de término** a considerar fueron 2: exigencia mínima de error y número máximo de iteraciones. Dado que los resultados obtenidos eran completamente mejorables se omitió el primer criterio y se tomaron como límite 1500 iteraciones. Esto, debido a que la convergencia de los errores finales converge en el rango 1000 y 1500 no presentando mejores resultado a mayor número de ciclos. El desempeño de las generaciones de evaluó mediante la función RMSE.

Generada la población, tanto inicial como nueva, la **selección de los padres** se realizo utilizando el método de selección tradicional basado en la elección de los mejores, descrito

en la sección 3.6.2, tomando así la mitad de los individuos de la población y comenzando el proceso de cruzamiento.

El cruzamiento utilizado se realizo en un punto aleatorio del cromosoma, al igual que la mutación, esta última con un porcentaje de mutación de 0.5. Basándose en los pesos finales obtenidos con las distintas configuraciones probadas se opto por acotar el rango de valores, que toma un alelo tras la mutación, a [-1,1] evitando valores muy elevados que escapen de la media de valores finalmente generados. Terminada la mutación se completa la nueva generación de cromosomas y se continúa con las iteraciones.

El resumen de los parámetros utilizados, tanto en la recurrencia Elman como en la Jordan, se presentan en la siguiente tabla.

Parámetros GA	
Tamaño población	100
Tipo selección	Ranking
Probabilidad cruce	1.0
Tipo cruce	1 punto
Probabilidad mutación	0.5
Tipo de mutación	1 punto
Numero iteraciones	1500

Tabla 1 Parámetros Algoritmo Genético

5.2.2 Selección de topología y recurrencia

El método de selección de las configuraciones de prueba fue ensayo y error, ingresando a priori las configuraciones que podrían presentar un buen desempeño e intentando mejorar las que ya lo hicieron.

En términos de representación, las topologías se denotaran de la forma (E+S, O, S) en el caso de recurrencia Jordan, siendo E el número de neuronas de entrada, O el número de neuronas ocultas y S el número de neuronas de salida, E+S corresponde al total de entradas de la red considerando E entradas directas más S reinyecciones. En el caso de utilizar recurrencia Elman las topologías se presentan como (E+O, O, S) donde E+O corresponde al total de entradas donde E son directas y O son las reinyectadas desde la capa oculta de la red.

Las tablas de resultados que se presentan a continuación, al igual que para todos los modelos, corresponde a los valores obtenidos tras un total de 10 iteraciones realizadas por topología.

Topología (E+S,O,S)	RMSE Media	σ	Max	Min	R^2 Media	σ	Max	Min	TPO. TRAINING Promedio	Nº Neuronas
3+1,8,1	0,058	0,007	0,073	0,045	0,785	0,057	0,871	0,656	5min 41seg	40
3+1,12,1	0,063	0,016	0,104	0,049	0,731	0,159	0,849	0,310	7min 17seg	60
4+1,8,1	0,063	0,008	0,080	0,055	0,745	0,069	0,813	0,597	6min 14seg	48

Tabla 2 Resultados RRBF-GA Jordan

Topología (E+O,O,S)	RMSE Media	σ	Max	Min	R^2 Media	σ	Max	Min	TPO. TRAINING Promedio	Nº Neuronas
3+12,12,1	0,072	0,015	0,110	0,056	0,653	0,162	0,794	0,217	10min 43seg	192
3+8,8,1	0,070	0,002	0,073	0,066	0,702	0,019	0,731	0,680	8min 10seg	96
5+8,8,1	0,075	0,006	0,084	0,065	0,639	0,067	0,752	0,541	9min 27seg	112

Tabla 3 Resultados RRBF-GA Elman

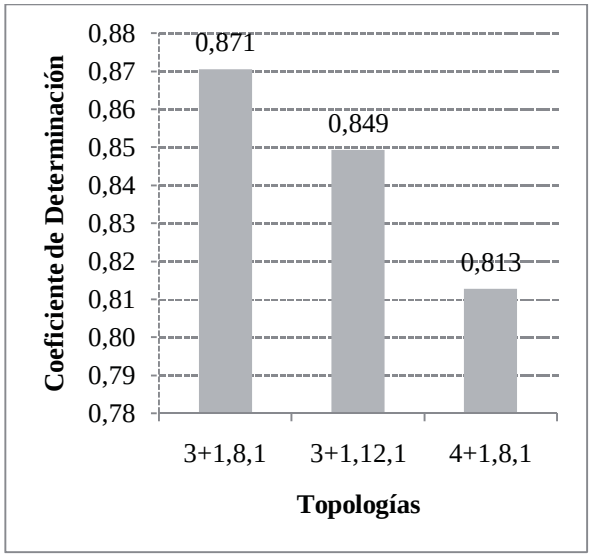

Figura 5-3 R^2 RRBF-GA Jordan

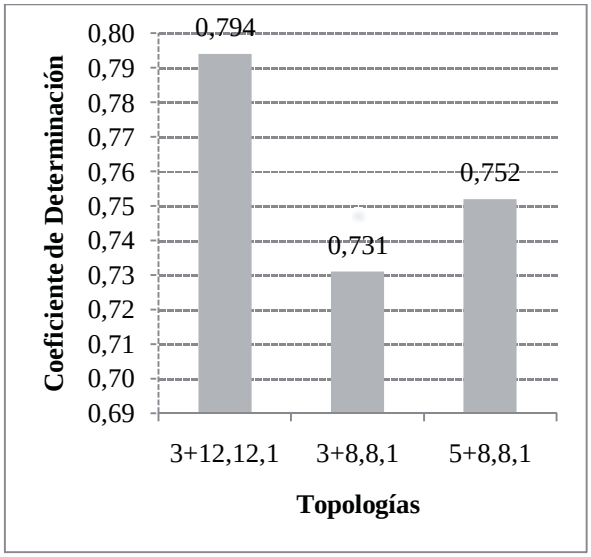

Figura 5-4 R^2 RRBF-GA Elman

Basándose en los resultados presentados en las tablas 2 y 3, se puede visualizar que en las pruebas el modelo que presenta mejor desempeño es aquel con recurrencia Jordan, presentando un menor RMSE medio, máximo y mínimo, y además, como se visualiza en las figuras 5.3 y 5.4 presenta un mayor R^2 con un 87,1% de la varianza explicada. Siguiendo en la figura 5.3, el mejor R^2 lo presenta la topología 3+1,8,1, la cual si bien presenta un leve mayor valor de R^2 que 3+1,12,1, sus valores mínimos son mayores y su desviación estándar en claramente menor, por lo que se asegura la estabilidad de esta topología por sobre las demás.

Analizando los costos, tanto la cantidad de neuronas que conforman las redes, como los tiempos de entrenamiento necesarios son menores al aplicar la recurrencia Jordan que la

Elman. Por lo tanto, en base a los casos estudiados la mejor topología encontrada para enfrentar el pronóstico de anchovetas con GA es 3+1, 8,1 Jordan.

5.3 Modelo RRBF-PSO

Este modelo consiste en una Red Neuronal Recurrente de Función de Base Radial cuyos parámetros son estimados mediante el Algoritmo de Optimización por Enjambre de Partículas. El proceso de entrenamiento de la red se presenta en la siguiente figura.

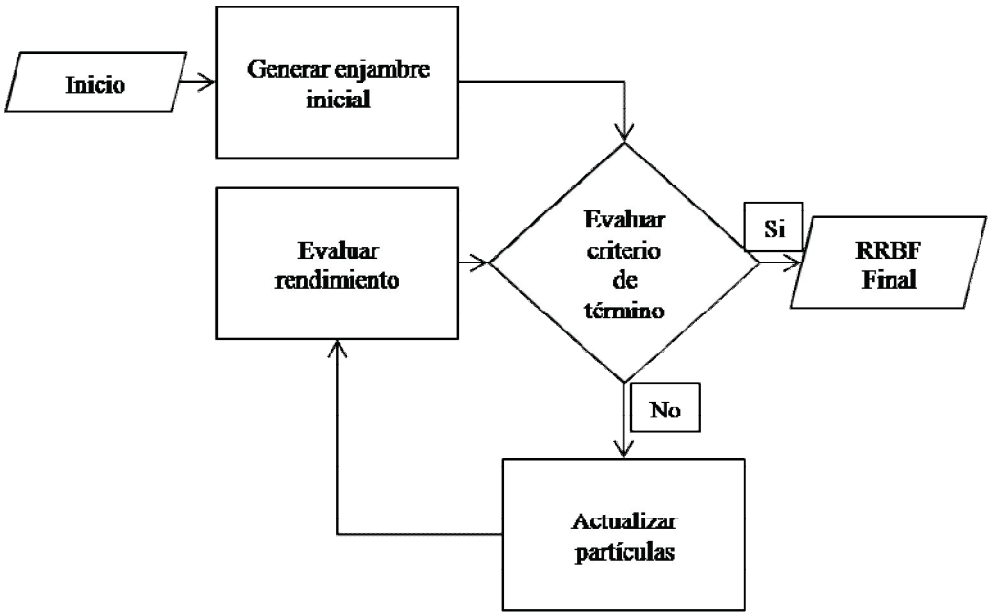

Figura 5-5 Algoritmo Modelo RRBF - PSO

5.3.1 Descripción del modelo

El proceso de entrenamiento comienza con la creación aleatoria de un **enjambre inicial** de N partículas, donde cada una de estas contendrá como posición (al igual que el cromosoma en el caso de GA) los pesos de la red (centros, anchuras y pesos), una velocidad generada de aleatoriamente y una memoria personal (P_{best}) con el mismo valor que la posición. También definida la posición, velocidad y memoria de la partícula se procede al cálculo inicial del error. Finalmente se tendrá un enjambre inicial de N partículas definiendo como mejor posición global (P_{gbest}) al menor error encontrado.

Los **criterios de término** a utilizar serán, al igual que GA corresponden a un error mínimo de 0.02 y un número de iteraciones máximo de 1500.

El proceso de **actualización de las partículas** se basa en el algoritmo descrito en la sección 4.3, específicamente aplicando las fórmulas (14) y (15) para la actualización de la velocidad y posición respectivamente. Luego se procede a la actualización del error de cada partícula, P_{best} y P_{gbest}. Esto se hace reemplazando las nuevas posiciones obtenidas por las

partículas en la red y cálculo del RMSE obtenido al procesar la data de entrenamiento y compararla con la data esperada (observada).

Los parámetros utilizados en los modelos, tanto para la recurrencia Elman como Jordan, se presentan en la siguiente tabla.

	Parámetros PSO	
	Jordan	**Elman**
Tamaño enjambre	100	100
Peso inercial	0.3	0.6
Componente cognitiva	1.5	0.4
Componente social	2	0.9
Velocidad máxima	0.04	0.05
Posición máxima	2	2
Número iteraciones	1500	1500

Tabla 4 Parámetros PSO: Jordan y Elman

5.3.2 Selección de topología y Recurrencia.

La selección de la topología se realizo siguiendo la misma línea que el modelo RRBF-GA de manera que se puedan contrastar los resultados obtenidos para cada configuración con los distintos algoritmos.

Topología	**RMSE**				**R²**				TPO. TRAINING	Nº
(E+S,O,S)	Media	σ	Max	Min	Media	σ	Max	Min	Promedio	Neuronas
3+1,5,1	0,053	0,011	0,080	0,043	0,811	0,088	0,884	0,590	6min 16seg	25
3+1,8,1	0,050	0,008	0,063	0,040	0,837	0,049	0,897	0,744	8min 31seg	40
6+1,8,1	0,039	0,004	0,048	0,035	0,901	0,023	0,925	0,856	10min 43seg	64

Tabla 5 Resultados RRBF-PSO Jordan

Topología	**RMSE**				**R²**				TPO. TRAINING	Nº
(E+O,O,S)	Media	σ	Max	Min	Media	σ	Max	Min	Promedio	Neuronas
3+5,5,1	0,054	0,005	0,060	0,044	0,803	0,032	0,872	0,765	9min 27seg	45
3+8,8,1	0,057	0,005	0,065	0,052	0,787	0,041	0,825	0,717	10min 38seg	96
5+8,8,1	0,059	0,006	0,067	0,050	0,762	0,047	0,835	0,702	20min 34seg	112

Tabla 6 Resultados RRBF-PSO Elman

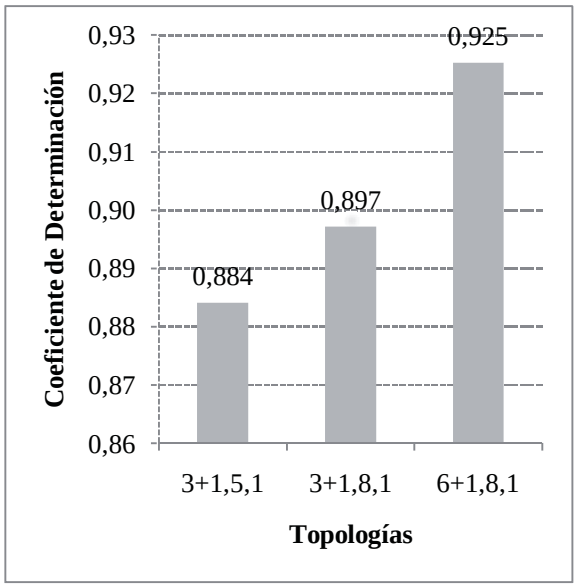

Figura 5-6 R^2 RRBF-PSO Jordan

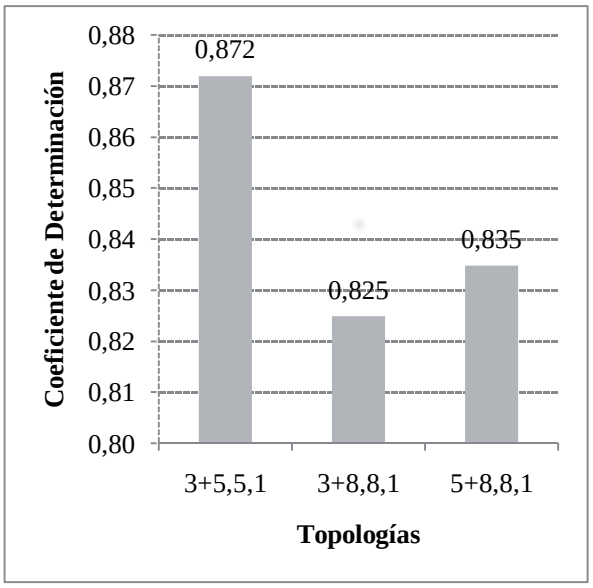

Figura 5-7 R^2 RRBF-PSO Elman

Las tablas 5 y 6 muestran en el caso de Elman que la topología 3+5, 5,1 presenta un mejor rendimiento en las pruebas realizadas frente a las otras 2 topologías. Si bien en el caso del RMSE se presenta una leve similitud entre los resultados, la diferencia se marca el R^2, donde la figura 5.7 refleja el superior porcentaje de la varianza explicada por la topología 3+5, 5,1. En el caso de Jordan, la topología 6+1, 8,1 presenta un notable mejor desempeño que el resto, presentando un bajo RMSE promedio acompañado de una desviación estándar que refleja la una baja variabilidad en la gamma de resultados. Analizando el R^2, presenta uno de los mejores resultados encontrados a la fecha con un 92,5% de la varianza explicada y manteniendo una media de un 90%.

Desde el punto de vista de los costos, se logra visualizar una cierta relación entre el número de neuronas que conforman la red y el tiempo necesario de entrenamiento, visualizando también una similitud entre los tiempos que tardarían los algoritmos con un mismo número de neuronas en entrenarse. Aun así, Jordan requiere menor cantidad y presenta mejores resultados.

Teniendo en cuenta lo anterior, junto con que los resultados medios que presenta la topología 6+1, 8,1 Jordan son superiores a los mejores encontrados por cualquier topología Elman, se determina que la topología encontrada que presenta un mejor desempeño frente a la problemática de las anchovetas descrita en este proyecto corresponde a la 6+1, 8,1 Jordan.

5.4 Modelo RRBF-IP GA+PSO

El modelo RRBF-IP GA+PSO corresponde la red RRBF utilizada en los modelos anteriores con un algoritmo de entrenamiento que consiste en una inicialización de parámetros por parte de GA, dando así un carácter inicialmente exploratorio y continuando con PSO el cual presenta un carácter local. La siguiente figura presenta la unión de los

procesos correspondientes al entrenamiento con GA y PSO descritos en los puntos 5.2 y 5.3 respectivamente.

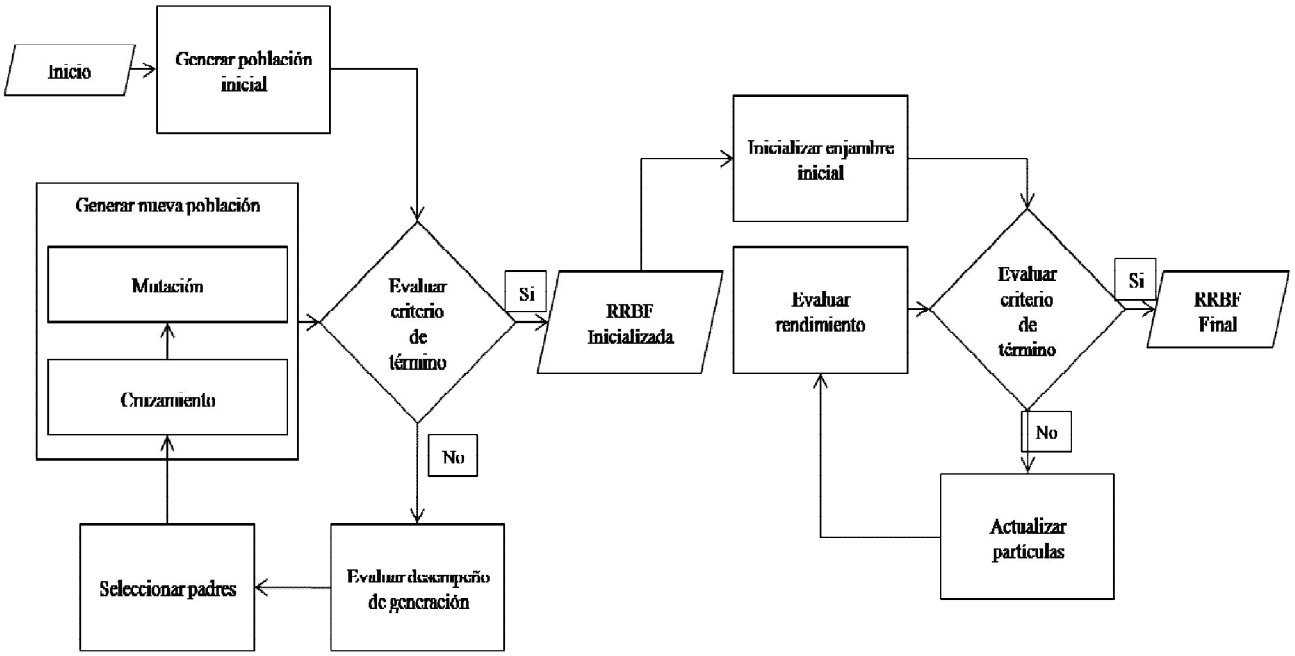

Figura 5-8 Algoritmo Modelo RRBF-IP GA + PSO

El número de iteraciones en total fueron 1500, 100 para GA y 1400 para PSO y la única diferencia, con relación a los modelos individuales, la presenta PSO con la inicialización del enjambre, dado que ahora no es de carácter aleatorio, sino que consiste en la obtención de las posiciones en base a la población de cromosoma que presento GA en la última iteración.

Los parámetros utilizados en cada una de las recurrencias y algoritmos correspondientes se muestran en las siguientes tablas.

RECURRENCIA ELMAN			
Parámetros GA		**Parámetros PSO**	
Tamaño población	100	Tamaño enjambre	100
Tipo selección	Ranking	Peso inercial	0.6
Probabilidad cruce	0.8	Componente cognitiva	0.4
Tipo cruce	1 punto	Componente social	0.9
Probabilidad mutación	0.5	Velocidad máxima	0.05
Tipo de mutación	1 punto	Posición máxima	2
Numero iteraciones	100	Número iteraciones	1400

Tabla 7 Parámetros GA-PSO Elman

RECURRENCIA JORDAN			
Parámetros GA		**Parámetros PSO**	
Tamaño población	100	Tamaño enjambre	100
Tipo selección	Ranking	Peso inercial	0.3
Probabilidad cruce	1.0	Componente cognitiva	1.5
Tipo cruce	1 punto	Componente social	1.6
Probabilidad mutación	0.5	Velocidad máxima	0.04
Tipo de mutación	1 punto	Posición máxima	2
Numero iteraciones	100	Número iteraciones	1400

Tabla 8 Parámetros GA-PSO Jordan

5.4.1 Selección de topología y recurrencia.

La selección de la topología se realizo siguiendo la misma línea del modelo RRBF-GA y RRBF-PSO de manera que se puedan contrastar los resultados obtenidos para cada configuración con los distintos algoritmos.

Topología	**RMSE**				**R^2**				TPO. TRAINING	Nº
(E+S,O,S)	Media	σ	Max	Min	Media	σ	Max	Min	Promedio	Neuronas
3+1,5,1	0,056	0,012	0,084	0,042	0,789	0,100	0,890	0,549	6min. 2seg	25
3+1,8,1	0,053	0,006	0,065	0,044	0,820	0,043	0,875	0,729	8min. 57seg	40
6+1,8,1	0,058	0,019	0,091	0,040	0,771	0,154	0,900	0,479	10min 9seg	64

Tabla 9 Resultados RRBF-IP GA+PSO Jordan

Topología	**RMSE**				**R^2**				TPO. TRAINING	Nº
(E+O,O,S)	Media	σ	Max	Min	Media	σ	Max	Min	Promedio	Neuronas
3+5,5,1	0,066	0,016	0,094	0,052	0,701	0,151	0,838	0,417	10min 9seg	45
3+8,8,1	0,055	0,006	0,065	0,047	0,795	0,048	0,856	0,722	20min 14seg	96
1+4,4,1	0,057	0,005	0,064	0,050	0,788	0,036	0,838	0,729	6min 30seg	25

Tabla 10 Resultados RRBF-IP GA+PSO Elman

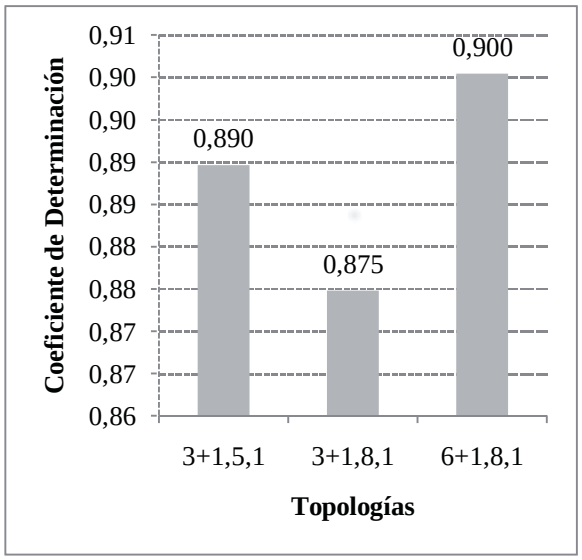

Figura 5-9 R^2 RRBF-IP GA+PSO Jordan

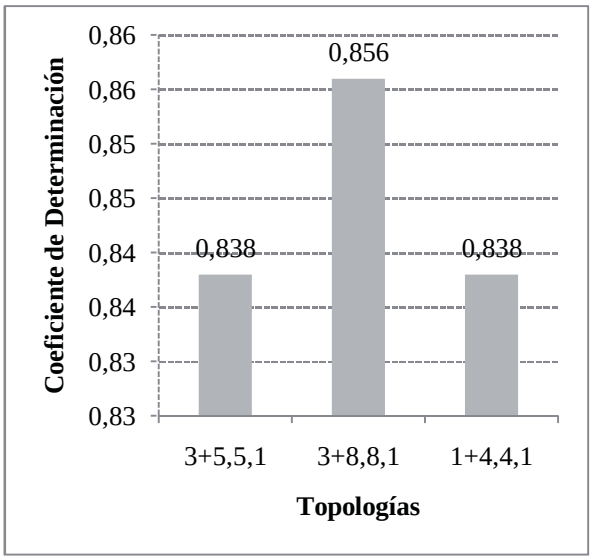

Figura 5-10 R^2 RRBF-IP GA+PSO Elman

Las tablas 9 y 10 presentan, en el caso de Jordan, que si bien la topología 6+1, 8, 1 presenta el mejor R^2 y el menor RMSE dentro de sus iteraciones, no se puede dejar pasar que también presenta los peores valores y las mayores desviaciones estándar. Esto implica que la calidad de modelo que se genera al utilizar esta topología es inferior a las demás, dadas su poca regularidad en los resultados, y teniendo en cuenta que valores mínimos como los presentados aplicados al caso de estudio representan un pronóstico fatal, se procede a analizar las 2 restantes. En este caso vuelve a suceder algo similar, debido a que la topología 3+1, 5,1 presenta un mayor porcentaje de varianza explicada con un 89% frente a un 87,5% explicado por el modelo con topología 3+1, 8,1, pero, también presenta los peores valores presentándose como una opción poco confiable. Finalmente, 3+1, 8,1 sería la que presenta un desempeño más estable en el caso de Jordan.

En el caso de Elman, si bien 3+8, 8,1 presenta un leve mayor valor de R2, la similitud en los valores obtenidos junto con 1+4, 4,1 lleva a analizar la parte de costos para seleccionar la más adecuada. En este caso, 1+4, 4,1 realiza su entrenamiento en un tiempo considerablemente menor junto con un bajo número de neuronas requeridas.

Finalmente, al comparar 3+1, 8,1 Jordan con 1+4, 4,1 Elman, la primera presenta en la mayoría de los casos un mejor valor, en especial en el caso del coeficiente de determinación, por lo que los 2 minutos aprox. de diferencia en el costo de entrenamiento pasan a segundo plano. Por esto, el modelo RRBF-IP GA+PSO Jordan con una topología de 3+1, 8,1 presenta un mejor desempeño y calidad en el pronóstico de las capturas.

5.5 Modelo RRBF-BS

Este modelo consiste en un híbrido basado en (Settles y Otros) denominado **Breeding Swarm.** Consiste en la incorporación de un nuevo operador de cruce VPAC al proceso de actualización de las partículas en PSO. El procedimiento general del entrenamiento se visualiza en la figura 5.8.

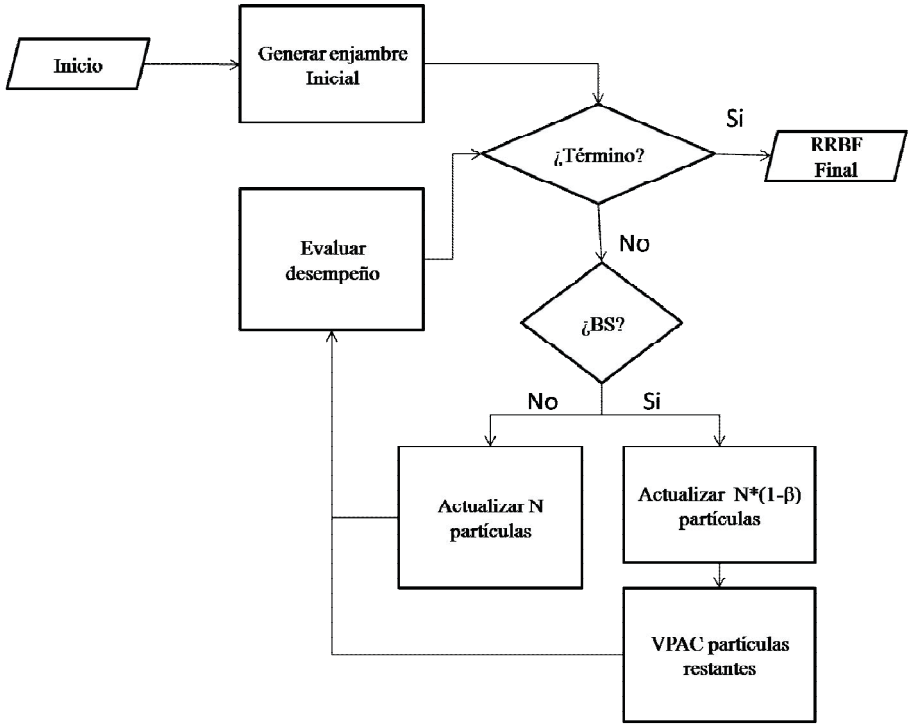

Figura 5-11 Algoritmo Modelo RRBF-BS

La generación del enjambre inicial, criterio de termino, actualización de las N partículas y evaluación del desempeño son procesos idénticos a los descritos en la sección 5.3. La diferencia, bien como se visualiza en la figura, comienza con el condicional posterior al criterio de término. **¿BS? (Breeding Swarm)** Representa el conteo del número de iteraciones mediantes las cuales no se aplicado VPAC, de manera que, si se decide que VPAC se aplicará cada 30 iteraciones cuando el numero de la iteración sea múltiplo de 30, ¿BS? Retornara **SI,** de lo contrario retornara **NO** y se prosigue con el proceso normal de PSO.

En el caso de que si se aplique BS, se deben seleccionar las mejores $N * (1 - \beta)$ partículas del enjambre, donde N corresponde al total de partículas y β se conoce como porcentaje de "respiración" (breeding) y toma valores entre [0,1]. Seleccionadas las partículas se realiza el proceso normal de actualización definido en la sección 4.3. Luego se aplica VPAC al $N\beta$ correspondiente a las peores partículas del enjambre. Cabe destacar, que las mejores y peores partículas quedaran definidas de acuerdo a su RMSE obtenido en la evaluación del desempeño.

VPAC (Velocity Propelled Averaged Crossover) corresponde a un nuevo operador de cruce que incorpora el vector de velocidad. La meta de este operador consiste en crear 2 partículas descendientes cuya posición se encuentre entre la posición de los padres, pero acelerada lejos de la actual posición en la que se encuentran, de manera que incrementen la diversidad de la población. La ecuación (24) muestra los nuevos vectores de posición calculados usando VPAC.

$$c_1(x_i) = \frac{p_1(x_i) + p_2(x_i)}{2} - \varphi_1 p_1(v_i)$$
$$c_2(x_i) = \frac{p_1(x_i) + p_2(x_i)}{2} - \varphi_2 p_2(v_i) \quad (24)$$

Donde $c_1(x_i)\ y\ c_2(x_i)$ son las posiciones de los descendientes 1 y 2 en la dimensión i. $p_1(x_i)\ y\ p_2(x_i)$ corresponden a las posiciones de los padres 1 y 2 en la dimensión i respectivamente, $p_1(v_i)\ y\ p_2(v_i)$ son las velocidades de estos padres y φ corresponde a una variable aleatoria que toma valores entre [0,1]. Las velocidades de las nuevas partículas son inicializadas con las actuales velocidades de sus padres y el P_{best}se inicializa con la posición recién generada.

5.5.1 Selección de topología y recurrencia.

Análogo a los modelos anteriores, en las siguientes tablas se muestran los resultados obtenidos tras un total de 10 iteraciones.

Topología	**RMSE**				**R²**				TPO. TRAINING	Nº
(E+S,O,S)	Media	σ	Max	Min	Media	σ	Max	Min	Promedio	Neuronas
3+1,12,1	0,048	0,005	0,056	0,042	0,852	0,028	0,887	0,803	13min 3seg	60
4+1,8,1	0,053	0,009	0,068	0,043	0,818	0,060	0,884	0,707	9min 48seg	48
6+1,8,1	0,054	0,005	0,060	0,044	0,819	0,034	0,880	0,774	11min 27seg	64

Figura 5-12 Resultados RRBF-BS Jordan

Topología	**RMSE**				**R²**				TPO. TRAINING	Nº
(E+O,O,S)	Media	σ	Max	Min	Media	σ	Max	Min	Promedio	Neuronas
3+5,5,1	0,051	0,003	0,056	0,046	0,830	0,018	0,858	0,796	8min 3seg	45
3+8,8,1	0,051	0,002	0,054	0,047	0,826	0,014	0,854	0,807	10min 37seg	96
3+12,12,1	0,064	0,010	0,084	0,049	0,727	0,088	0,842	0,538	20min 59seg	192

Figura 5-13 Resultados RRBF-BS Elman

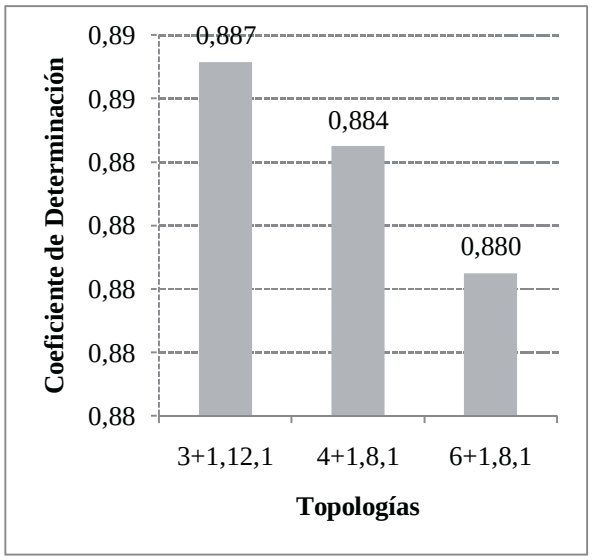

Figura 5-14 R^2 RRBF-BS Jordan

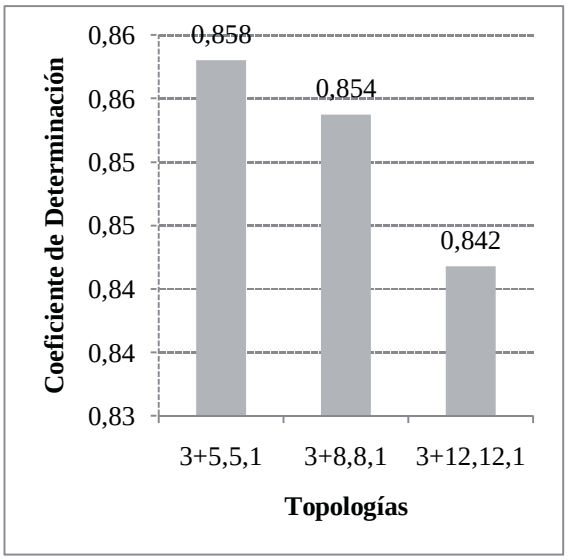

Figura 5-15 R^2 RRBF-BS Elman

En la tabla 12 se visualiza que la topología 3+1, 12,1 presenta los mejores valores y las mejores medias acompañadas de las menores desviaciones estándares. Si bien presenta buenos valores, como se visualiza en la figura 5.14, el tiempo requerido para su entrenamiento y la cantidad de nodos utilizados es superior, por lo que podría ser necesario considerar a las demás topologías dada su cercanía de R^2. Pero, aunque tarde un poco más en entrenarse la red, no pierde superioridad frente a las demás topologías debido a que estas son mucho más inestables llegando a presentar un 70,7% de la varianza explicada por solo 4 minutos de diferencia.

En el caso de Elman, existe una gran similitud en los resultados entregados por las topologías 3+5,5,1 y 3+8,8,1, al punto que la selección de la topología dependería de la importancia que se le da a la estabilidad del modelo por sobre el tiempo por sobre el tiempo que tarde en entrenarse la red. En este proyecto, dado que las diferencias de tiempo son solo de 2 minutos se valora más la estabilidad en los resultados que presenten los modelos.

Finalmente, en este modelo existe una gran competitividad entre el aporte de Jordan y Elman, debido a que en base a las topologías seleccionadas, luego del análisis previo del párrafo anterior, se visualiza una gran similitud en los resultados y si bien Jordan tarda más tiempo en entrenarse, presenta una menor cantidad de nodos creados. Debido a esto, se valora en este caso que Jordan presenta un porcentaje de la varianza explicada de un 88,7% por sobre el 85,4% explicado por Elman.

CAPÍTULO 6

Análisis de Resultados

En esta sección se busca **contrastar** los resultados obtenidos por los modelos seleccionados en el capítulo 5 frente a un total de 10 iteraciones y agregando la medida: MAPE, para la elección del modelo final. A continuación se presentan las tablas de resultados de los distintos modelos y posteriormente las graficas del mejor pronóstico, R^2 y MAPE.

	RRBF-GA 3+1,8,1			RRBF-PSO 6+1,8,1		
#Iteración	RMSE	R^2	MAPE	RMSE	R^2	MAPE
1	0,061	0,763	117	0,035	0,925	79
2	0,062	0,755	172	0,038	0,910	54
3	0,061	0,761	189	0,035	0,924	76
4	0,056	0,798	119	0,048	0,856	72
5	0,054	0,814	133	0,041	0,895	71
6	0,052	0,827	119	0,037	0,914	89
7	0,045	0,871	111	0,045	0,872	66
8	0,073	0,656	171	0,042	0,888	71
9	0,056	0,801	130	0,037	0,916	78
10	0,055	0,807	107	0,037	0,913	56

Tabla 11 Resultados modelos RRBF-GA y RRBF-PSO

	RRBF-IP GA+PSO 3+1,8,1			RRBF-BS 3+,12,1		
#Iteración	RMSE	R^2	MAPE	RMSE	R^2	MAPE
1	0,050	0,842	135	0,049	0,846	120
2	0,058	0,787	145	0,056	0,803	79
3	0,052	0,830	153	0,049	0,849	89
4	0,055	0,804	100	0,043	0,884	87
5	0,046	0,867	78	0,042	0,887	77
6	0,050	0,838	134	0,052	0,824	134
7	0,051	0,831	136	0,052	0,826	127
8	0,056	0,799	170	0,045	0,870	110
9	0,044	0,875	96	0,048	0,853	102
10	0,065	0,729	129	0,044	0,874	99

Tabla 12 Resultados modelos RRBF-IP GA+PSO y RRBF-BS

Como bien se observa en las tablas 11 y 12, en un total de 10 iteraciones los modelos GA, IP y BS presentan una gran similitud en lo que respecta al RMSE y al R^2. No obstante PSO destaca presentando los mejores valores tanto en estas métricas como el MAPE.

Como bien se ha dicho, a medida que aumente el valor del R^2 entregado por un modelo, mayor será el porcentaje de la varianza explicada por éste, es decir, la línea que grafica la data pronosticada deberá seguir la misma trayectoria que la data observada. A continuación se presentan los gráficos de estimación de los modelos.

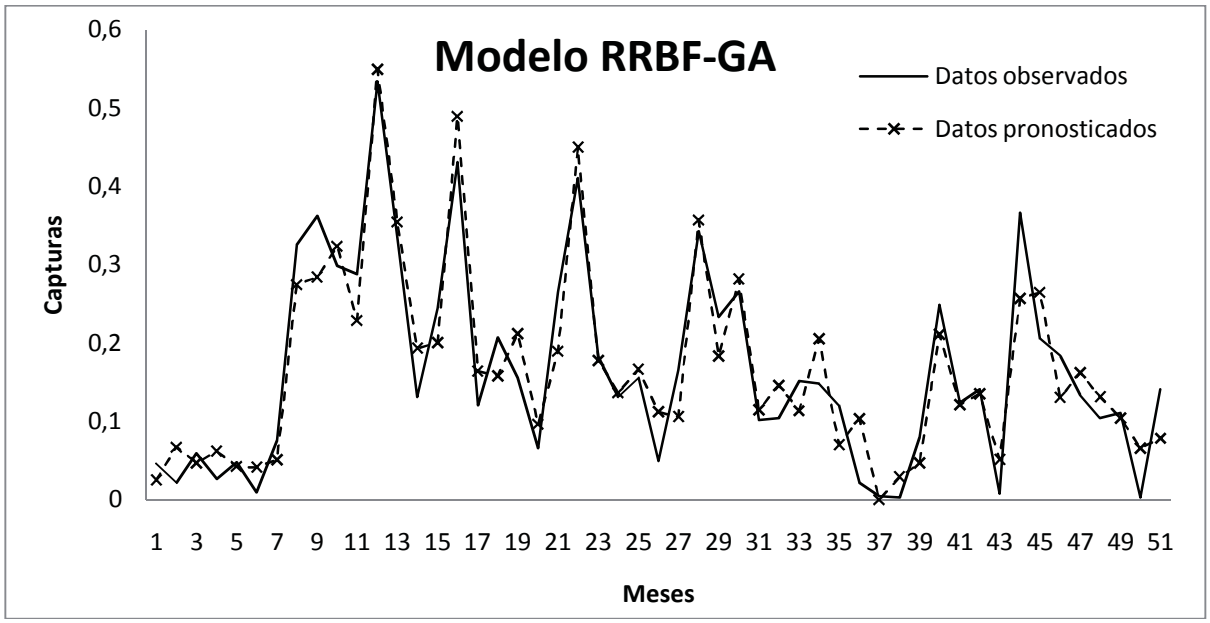

Figura 6-1 Mejor estimación modelo RRBF-GA 3+1, 8,1 Jordan

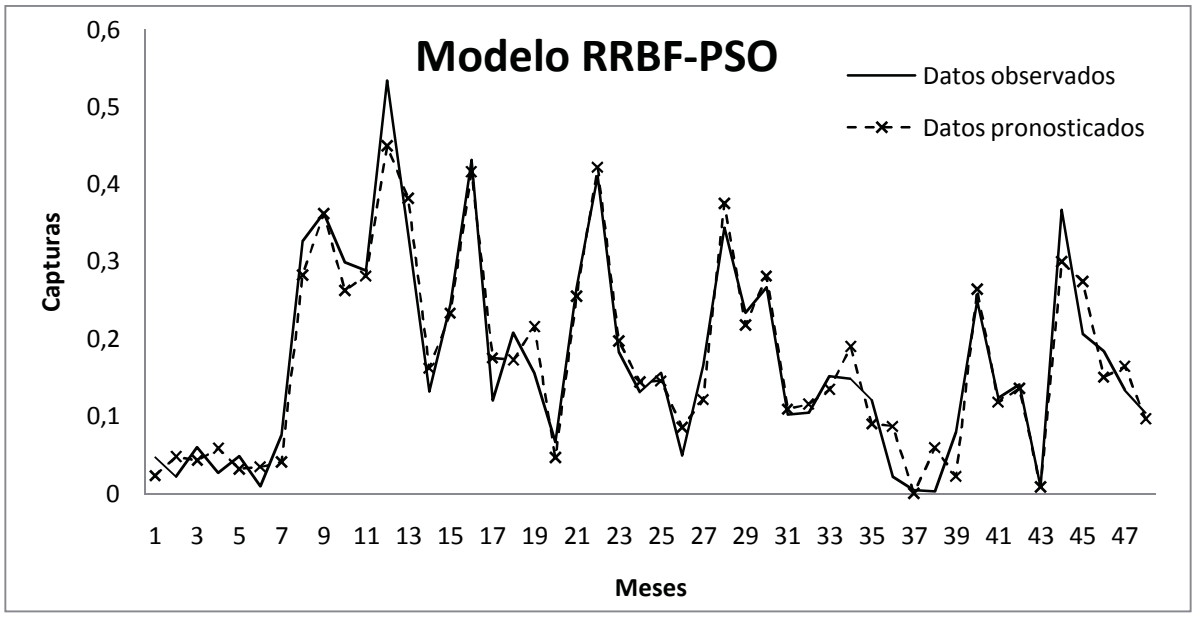

Figura 6-2 Mejor estimación modelo RRBF-PSO 6+1, 8,1 Jordan

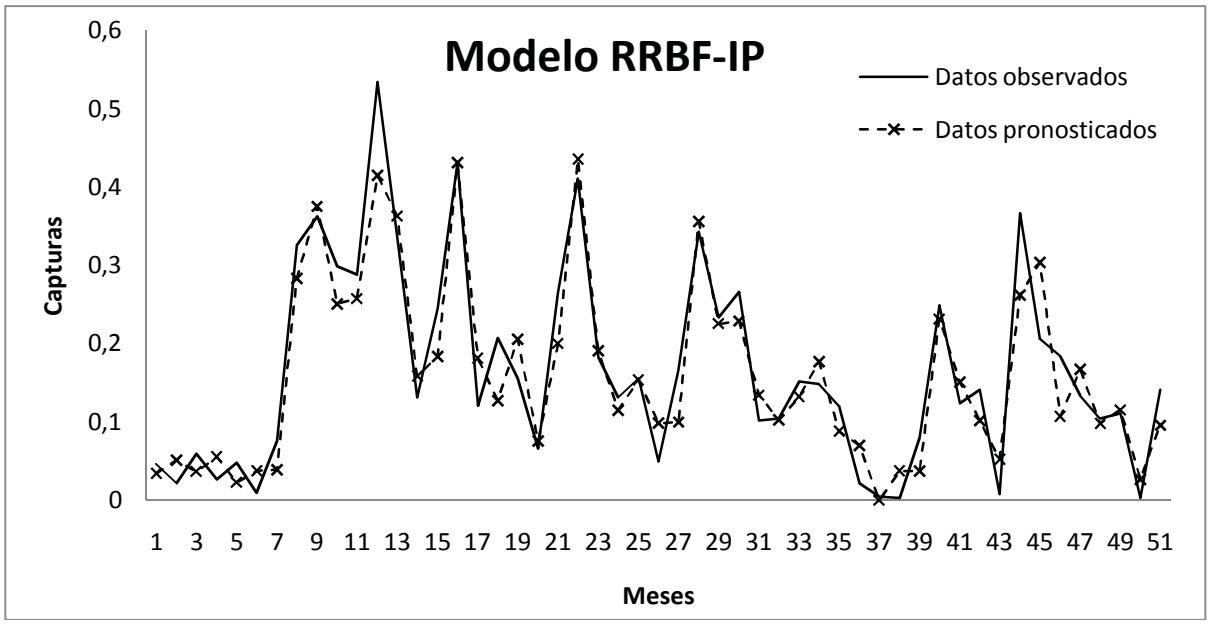

Figura 6-3 Mejor estimación modelo RRBF-IP GA+PSO 3+1, 8,1 Jordan

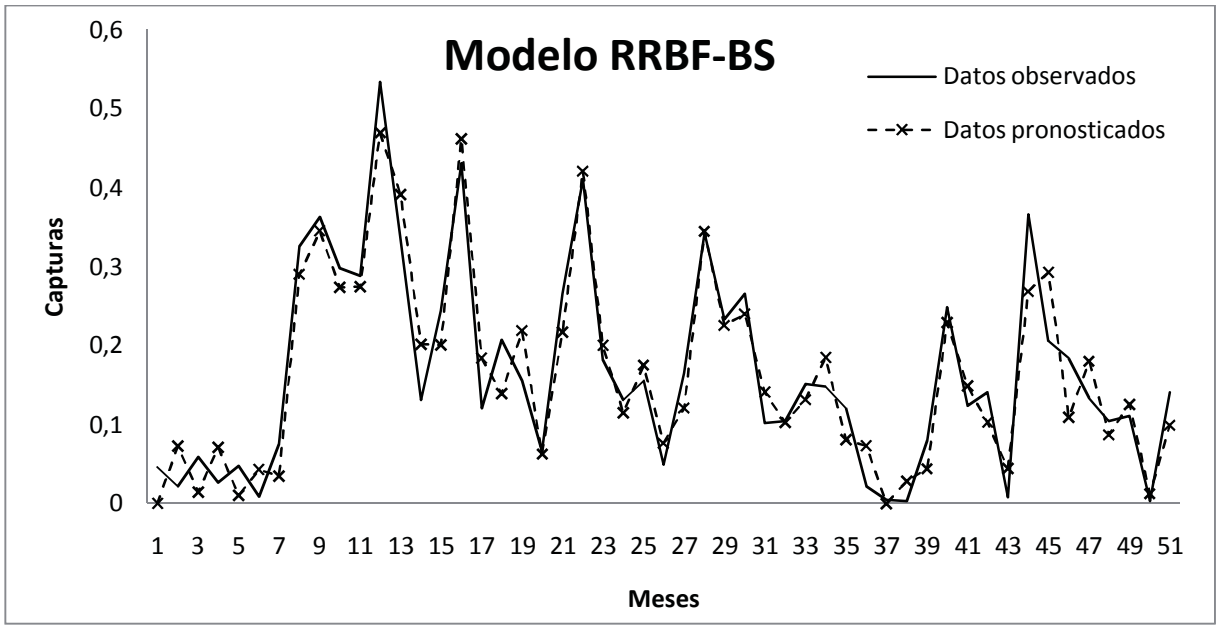

Figura 6-4 Mejor estimación modelo RRBF-BS 3+1, 12,1 Jordan

Al comparar 4 figuras anteriores, se comprueba visualmente que el modelo RRBF-PSO al presentar un menor error cuadrático medio y un mayor coeficiente de determinación, hace que la data pronosticada se superponga con la data observada en muchos más meses que los otros 3 modelos. Un ejemplo claro de la mejor predicción se visualiza entre los meses 45 y 47 donde GA, IP y BS marcan la tendencia de manera exagerada generando un nuevo pico, al contrario de PSO que atraviesa más suavemente esa zona con un grado de error menor.

Al observar las estimaciones de captura de anchoveta es importante destacar que en los meses de escases del recurso es sumamente necesario un pronóstico correcto debido a que en esos meses peligra la existencia de la especie y la regulación de captura debe ser mayor. Por ejemplo, en el mes 26 tanto PSO como BS presentan una estimación con un mínimo de error, pero GA e IP lo maximizan de cierta manera suponiendo una captura de miles de anchovetas que simplemente no existen.

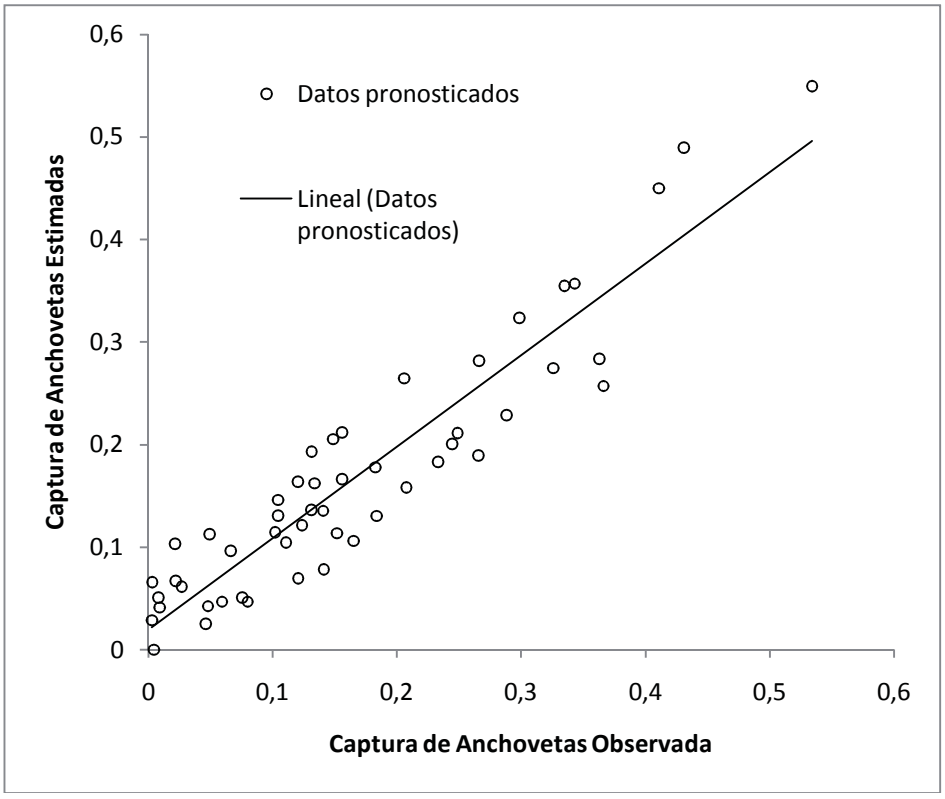

Figura 6-5 R^2 Modelo RRBF-GA 3+1, 8,1

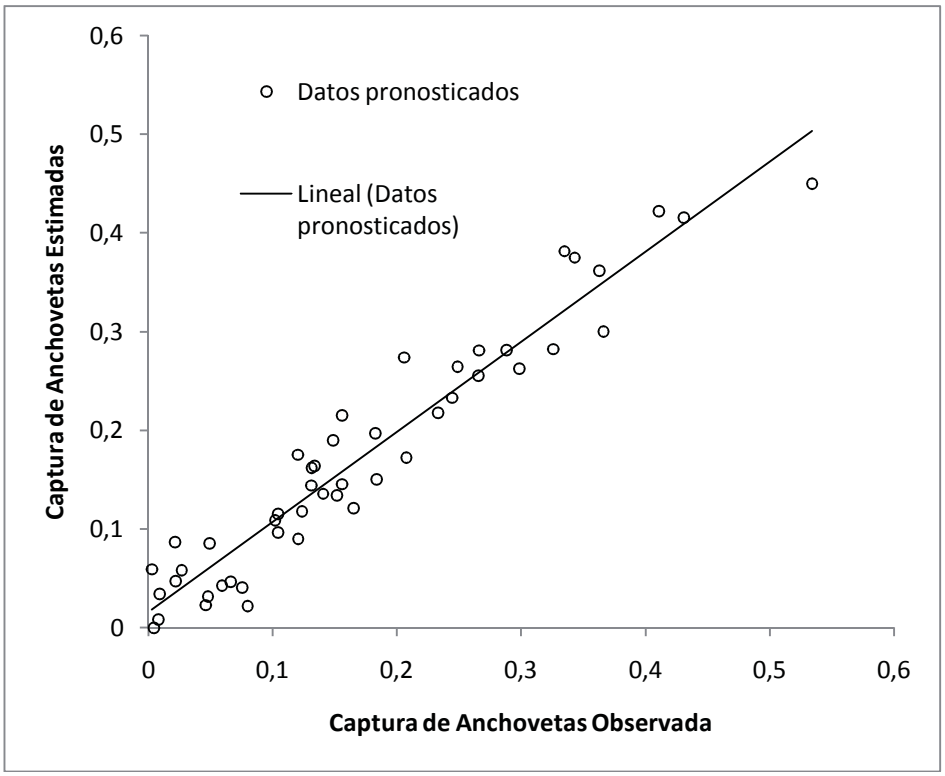

Figura 6-6 R^2 Modelo RRBF-PSO 6+1, 8,1

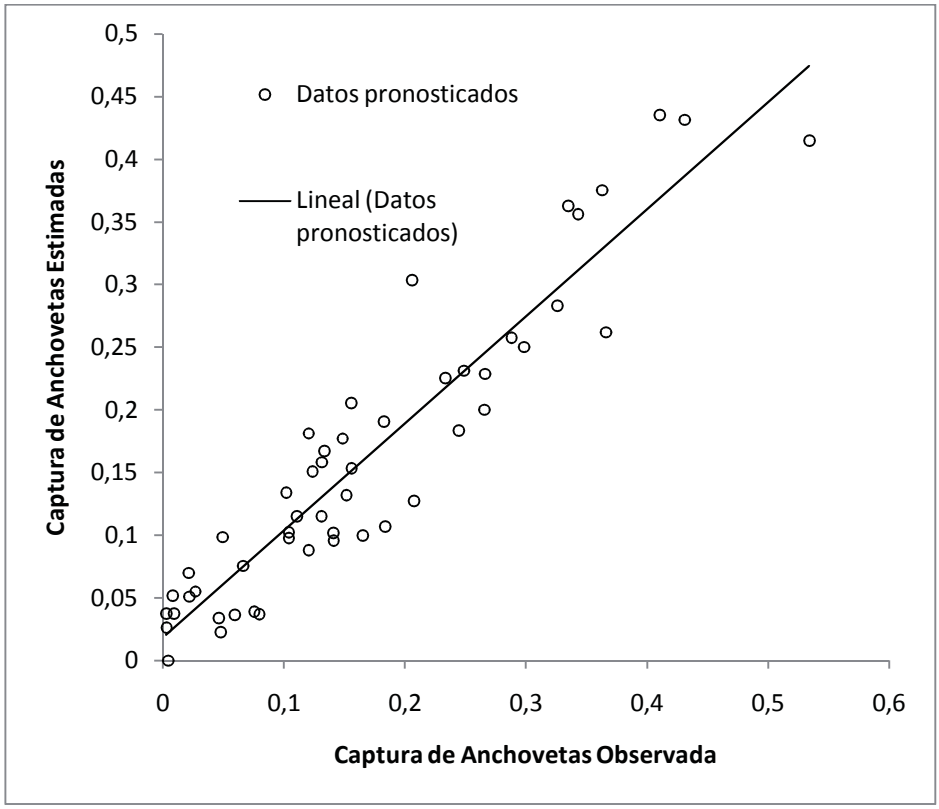

Figura 6-7 R^2 Modelo RRBF-IP GA+PSO 3+1, 8,1

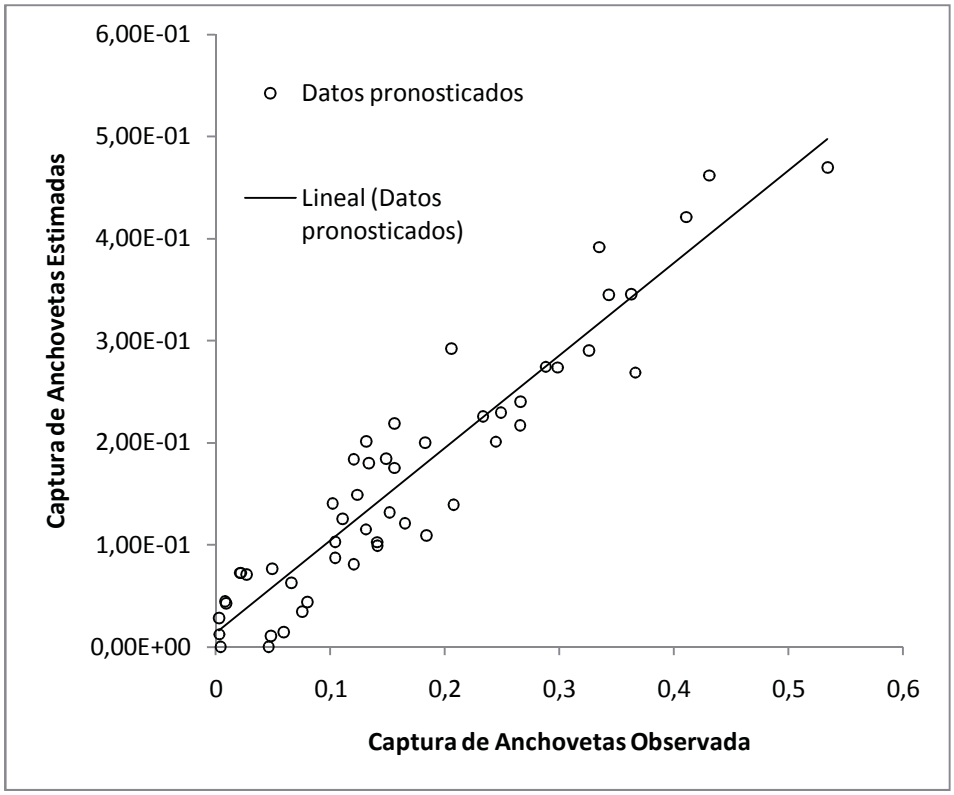

Figura 6-8 R^2 Modelo RRBF-BS 3+1, 12,1

Los gráficos correspondientes a las figuras 6.5, 6.6, 6.7 y 6.8 presentan la correlación existente entre la data pronosticada y la observada. Aquí PSO refleja su 92,5% de varianza explicada presentando los datos pronosticados más apegados a la recta que los demás. Es decir, al tener IP, GA y BS datos pronosticados más dispersos que PSO se da a entender que los datos que estos pronosticas no varían tan acorde con la data real observada como lo hace PSO.

Finalmente para terminar de visualizar los valores presentados en las tablas 11 y 12, se presentan las graficas del MAPE y del APE. Como bien se describió en la sección 5.1, un alto valor del MAPE implica un alto error de pronóstico comparado con los valores de la serie, y en la gráfica se visualiza este error de pronóstico con los altos valores del APE por sobre el MAPE.

Como bien se visualiza, todos los modelos entre el mes 10 y el 30 son capaces de mantener una baja diferencia entre el valor pronosticado y el real, volviéndose caótico entre los meses 30 y 50 donde GA, BS e IP presentan las mayores diferencias de pronóstico. A diferencia del resto, PSO fue capaz de controlar los errores cercanos al mes 30 y 50, reflejándose en la tabla 11 y 12 con un menor valor del MAPE que los demás modelos.

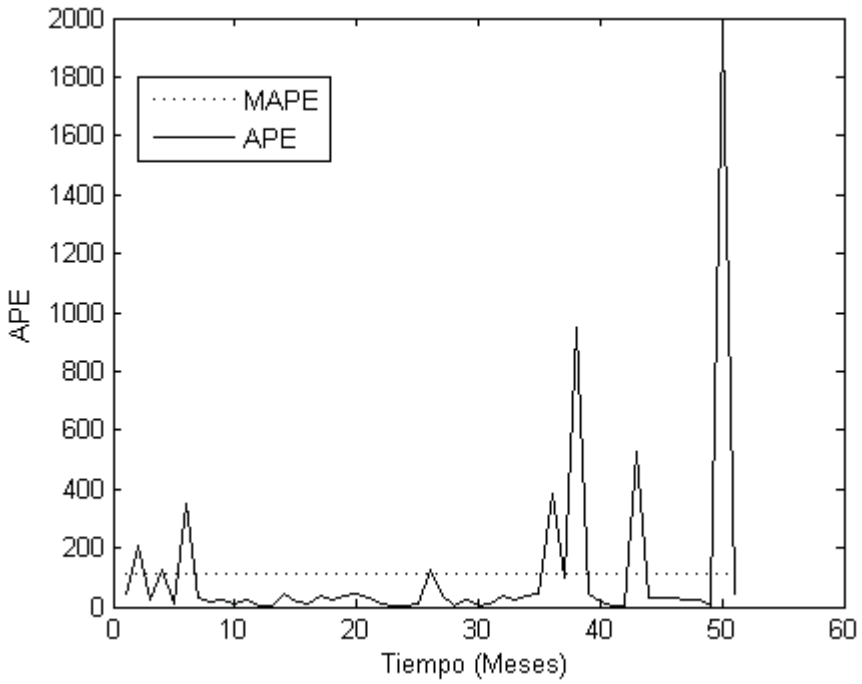

Figura 6-9 MAPE y APE RRBF-GA

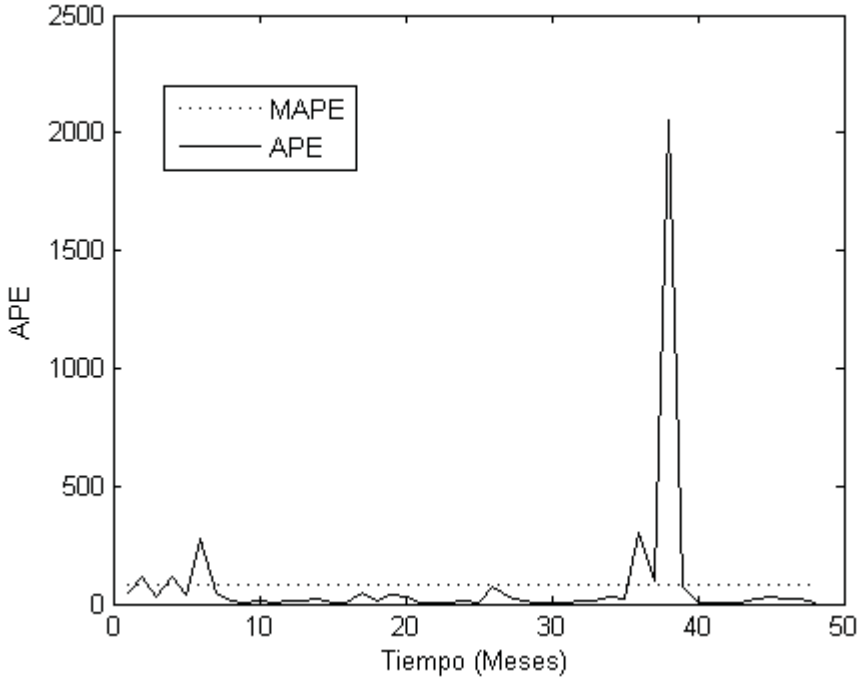

Figura 6-10 MAPE y APE RRBF-PSO

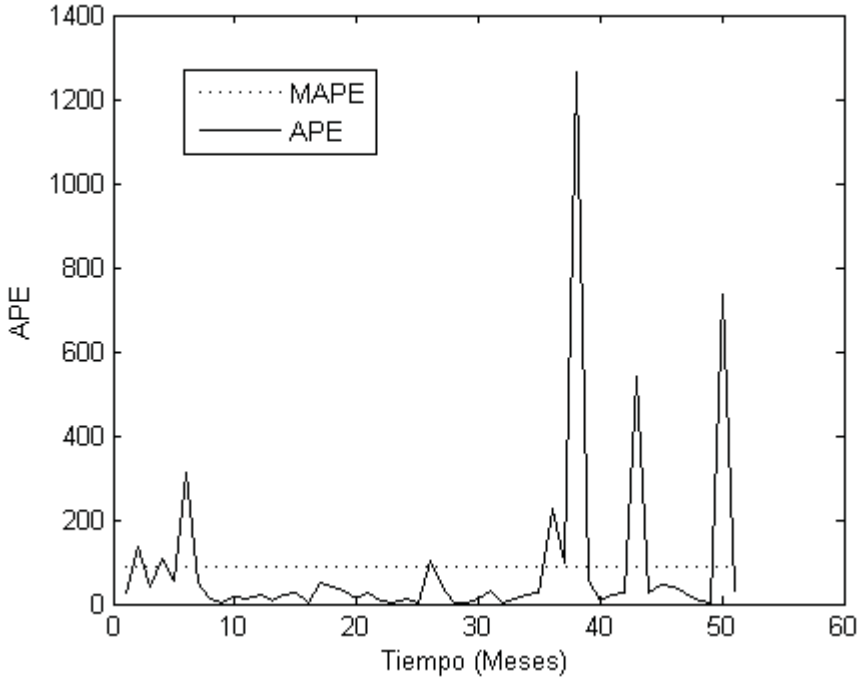

Figura 6-11 MAPE y APE RRBF-IP GA+PSO

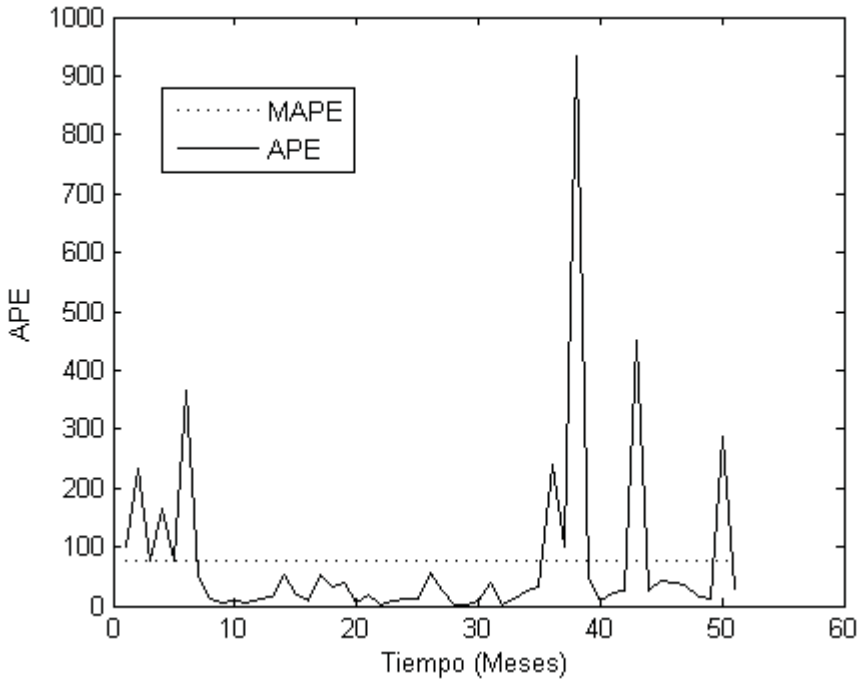

Figura 6-12 MAPE y APE RRBF-BS

Luego de contrastar los resultados de las mejores topologías encontradas por cada uno de los modelos en la sección 5, se determina que el modelo RRBF-PSO 6+1, 8,1

presenta el mejor desempeño frente al pronóstico del nivel de captura de anchovetas en el área norte de Chile.

Finalmente, se compararan los modelos de redes neuroevolutivos frente a una Regresión Funcional, la cual utiliza una técnica exacta conocida como **Mínimos Cuadrados** para encontrar sus coeficientes**.** En la siguiente tabla se presentan los resultados obtenidos utilizando un desfase desde 8 a 12 meses.

Desfase	RMSE	R2	MAPE
8	0,046	0,860	94
9	0,044	0,871	100
10	0,044	0,873	88
11	0,041	0,888	58
12	**0,039**	**0,895**	**37**

Tabla 6.13 Resultados con Mínimos Cuadrados

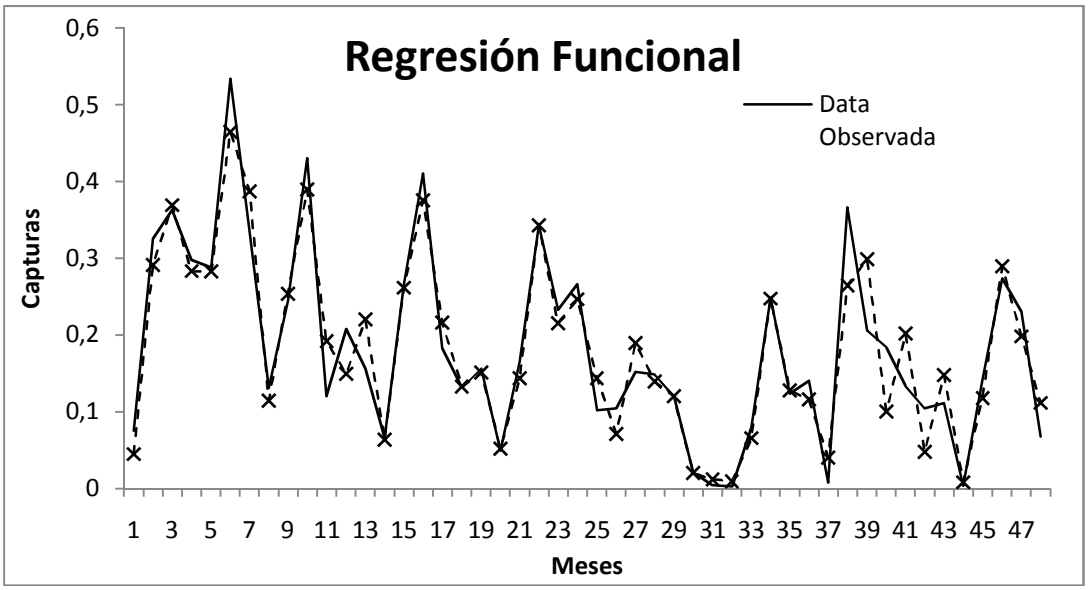

Figura 6-13 Estimación mediante regresiones funcionales

Al comparar las redes neuroevolutivas con una regresión funcional, se puede apreciar que esta última presenta valores levemente inferiores a los obtenidos mediante el modelo RRBF-PSO en lo que respecta a R2 y RMSE, pero considerablemente mejor en el valor del MAPE. Si bien al visualizar el grafico de estimación se nota la similitud en la calidad de pronóstico, la gran desventaja de la técnica de aprendizaje evolutivo recae en el costo computacional.

Aun así descrito lo anterior y en comparación al 88,7% presentado en el avance de este proyecto, la ventaja que presentan las redes neuroevolutivas frente a un modelo de Regresión Funcional es que estas pueden variar sus resultados y mediante pequeñas modificaciones en los algoritmos o en los parámetros que estos utilizan se pueden superar el desempeño obtenidos. Por lo tanto, si bien la técnica exacta puede entregar un resultado bueno y rápido las redes con una configuración adecuada pueden ser capaces de encontrar una red que supere estos resultados.

CAPÍTULO 7

Conclusiones

En el presente caso de estudio se desarrollo el estado del arte de los componentes fundamentales que se utilizarán en la confección del modelo predictivo del volumen de la captura de anchovetas mediante redes RBF con aprendizaje basado en GA y PSO. De esta manera, se logra cumplir el primer objetivo específico definido en la sección 1.2.

En base a la investigación acerca de las Redes Neuronales, el estudio de sus distintas topologías, y condiciones internas (a nivel de pesos, o centros en las redes RBF), junto con las aplicaciones encontradas en la literatura, han dejado claro que las RNAs corresponden a un modelo computacional capaz de resolver una infinidad de problemas, y que dependiendo de lo que se desee resolver se debe configurar una red que satisfaga dichos requerimientos.

Luego del análisis de los distintos modelos propuestos, se puede concluir que la recurrencia Jordan presenta un mejor desempeño en el pronóstico de las capturas de anchovetas que la recurrencia Elman. Esto, tanto los resultados obtenidos como en la velocidad de convergencia en el entrenamiento. Por otro lado, si bien las versiones de Elman para los modelos con GA y PSO no fueron suficientemente buenas, la mezcla de las capacidades explorativas de GA junto a las explotativas de PSO permitió que el modelo RRBF-IP GA+PSO y RRBF BS (mediante la incorporación de VPAC) alcanzaran mejores resultados. No obstante, la efectividad de Jordan potencio los resultados obtenidos por el modelo RRBF-PSO. Por lo tanto, se concluye con relación a esto que el componente fundamental de los modelos híbridos para poder mejorar los resultados es PSO.

Al comparar una técnica exacta frente a una evolutiva se puede dar cuenta de que si bien una Regresión Funcional puede entregar un resultado rápido y eficiente, una técnica evolutiva puede demorarse un poco más y, con una topología y parámetros adecuados, puede llegar a superar estos resultados generando un predictor de calidad.

Finalmente, a finales del proyecto se concluye que las redes RRBF si son capaces de presentar una buena calidad de pronóstico frente a la necesidad de predecir el nivel de captura de anchovetas en el norte de Chile, presentándose el modelo RRBF-PSO con un 92,5% de la varianza explicada, sin dejar de lado que los modelo híbridos desarrollados: RRBF-IP y RRBF-BS puedan mejorar su calidad de pronóstico tal como ocurrió con PSO desde el avance a la entrega final.

Referencias

[Abraham et al., 06] Abraham, A., Guo, H., y Liu. H. (2006). "Swarm Intelligence: Foundations, Perspectives and Applications", Studies in Computational Intelligence (SCI), Springer-Verlag.

[Alfassio et al., 05] Alfassio, E., Gandelli, A., Grimaccia, M., Mussetta, M., y Zich, R. (2005). A new Irbid Technique for the optimizacion of large-domain electromagnetic problems.

[Anderson and McNeil, 92] Anderson, D. y McNeill, G. (1992). Artificial Neural Networks Technology. Kaman Sciences Corporation.

[Andreou et al. 02] Andreou, Andreas, Efstratios Georgopoulos y Likothanassis. (2002). "Exchange-rates forecasting: A hybrid algorithm based on genetically optimized adaptive neural networks." Computational Economics.

[Ayala y Castillo, 04] Ayala M. y Castillo R. (2004). Un modelo de predicción para el valor TRM: un acercamiento desde las redes neuronales artificiales.

[Cañon, 78] Cañón, J. Distribución de la anchoveta (engraulis ringens) en el norte de chile en relación a determinadas condiciones oceanográficas. Serie Invest. Pesq., IFOP (Chile), 30: 1-122, 1978.

[Chellapilla et al.] Chellapilla, Kumar y Fogel. "Evolving an expert checkers playing program without using human expertise." IEEE Transactions on Evolutionary Computation.

[Correa et al., 05] Correa, A., Villanueva, J., Pérez, J. y Basterrechea, J. (2005) Aplicación de PSO y GA en la síntesis de agrupaciones lineales de antenas. Universidad de Cantabria.

[Elman, 90] Elman, J. L. (1990). Finding structure in time. Cognitive Science, 14, 179-211.

[Goldberg, 89] Goldberg, D. (1989). Genetic Algorithms in Search, Optimization, and Machine Learning. Addison-Wesley.

[Gutierrez-Estrada , 07] Gutierrez-Estrada, J.C. y otros. (2007). Monthly catch forecasting of anchovy Engraulis ringens in the north area of Chile: Non-linear univariate approach.

[Haupt et al., 98] Haupt, Randy y Sue Ellen Haupt. (1998). Practical Genetic Algorithms. John Wiley & Sons.

[Haykin, 99] Haykin, S. (1999). Neural Networks, A Comprehensive Foundation, Second Edition. Pearson Prentice Hall.

[Hebb, 49] Hebb, D. (1949). Organization of Behavior. Nueva York: John Wiley & Sons.

Isasi, V. y Galván, L. (2004). Redes Neuronales Artificiales. Un enfoque práctico. Pearson Prentice Hall.

[Kennedy and Eberhart, 95] Kennedy, J. y Eberhart, R., (1995) "Particle Swarm Optimization", Proceedings of the IEEE International Conference on Neural Networks.

[Kennedy et al., 01] Kennedy, J., Eberhart, R. y Shi, Y. (2001). "Swarm Intelligence", The Morgan Kaufmann Series in Artificial Intelligence, Morgan Kaufmann,

[Khosla et al. 03] Khosla, A., Kumar, S., Aggarwal, K., Singh, J. (2003). "Introducing Lifetime Parameter in Selection Based Particle Swarm Optimization for Improved

Performance", First Indian International Conference on Artificial Intelligence (IICAI-03), Hyderabad, India.
[Kumagi et al., 99] Kumagai T. y otros. (1999). Dynamical control by recurrent neural networks through genetic algorithms. Int. J. Adapt. Control Signal Process. 13, 261- 271.
[Mahfoud et al., 96] Mahfoud, Sam y Mani, G. (1996). "Financial forecasting using genetic algorithms." Applied Artificial Intelligence.
[Matich, 99] Matich, D. (1999). Redes Neuronales Conceptos Básicos y Aplicaciones. Rosario.
[Marczyk , 04] Marczyk, A. (2004). Genetic Algorithms and Evolutionary Computation
Mitchell, M. (1999). An Introduction to Genetic Algorithms. The MIT Press.
[Qasem y Shamsuddin, 09] Qasem, S. y Shamsuddin, S. (2009). Hybrid Learning Enhancement of RBF Network Based on Particle Swarm Optimization. Springer-Verlag Berlin Heidelberg
[Pan et al., 05] Pan, T. y Wang, R. (2005). Using recurrent neural networks to reconstruct rainfall-runoff proceddes.
[Parsopoulos y Vrahatis, 01] Parsopoulos, K y Vrahatis, M (2001) "Modification of the Particle Swarm Optimizer for Locating all the Global Minima", Proceedings of the International Conference (Artificial Neural Nets and Genetic Algorithms), Springer-Verlag.
[Saha y Raghava , 06] Saha, S. y Raghava G. (2006). Prediction of Continuous B-Cell Epitopes in an Antigen Using Recurrent Neural Network.
[Sato et al] Sato, S., Otori, K., Takizawa, A., Sakai, H., Ando, Y. y Kawamura, H. (2002). "Applying genetic algorithms to the optimum design of a concert hall." Journal of Sound and Vibration.
[Settles et al., 05] Settles, M., Nathan, P. y Soule, T. (2005). Breeding Swarm: A New Approach to Recurrent Neural Network Training. GECCCO'05.
[Shen et al., 04] Shen, Jiang, Jiao, Lin, Shen, G., Yu. (2004). Hybridized Particle Swarm Algorithm for Adaptative Structure Training of Multilayer Feed-Forward Neural Network: QSAR Studies of Bioactivity of Organic Compounds. Wiley InterScience.
[Soria y Blanco, 01] Soria, E. y Blanco, A. (2001). Redes Neuronales Artificiales. ACTA.
[William et al.] Williams, Edwin, Crossley, W. y Lang, T. (2001). "Average and maximum revisit time trade studies for satellite constellations using a multiobjective genetic algorithm." Journal of the Astronautical Sciences,
[Wright, 91] Wright, A. H. (1991). Genetic algorithms for real parameter optimization. In G. Rawlins, ed., Foundations of Genetic Algorithms. Morgan Kaufmann.
[Zemouri et al., 07] Zemouri, R., Racoceanu D. y Zerhouni N. (2007). Coupling the dynamic memory with the static memory of a neural network for the diagnosis by pattern recognition.

Anexo A – Código Fuente Red RBF

```
class RBF {

    // neuronas de la red
    private int n_entrada;
    private int n_ocultas;
    private int n_salida;
    private int n_contexto;
    private int t_entradas;
    private int t_neuronas;

    // caracteristicas red
    private int f_transferencia;
    private boolean normalizada;
    private int t_recurrencia;

    // data del proceso de entrenamiento
    private double[][] d_entrenamiento;
    private double[] d_entrenamiento_ideal;

    // informacion de la red
    private double[] d_contexto;
    private double[][] centroides;
    private double[] anchuras;
    private double[] pesos;
    private double[] salidas;

    // error que presenta la red frente al problema en cuestion
    public double error;

    public RBF(int n_entrada, int n_ocultas, int n_salida, int
f_transferencia, boolean normalizada, int t_recurrencia,
double[][] d_entrenamiento, double[] d_entrenamiento_ideal) {

        this.n_entrada = n_entrada;
        this.n_ocultas = n_ocultas;
        this.n_salida = n_salida;

        this.f_transferencia = f_transferencia;
        this.normalizada = normalizada;
        this.t_recurrencia = t_recurrencia;

        this.d_entrenamiento = d_entrenamiento;
        this.d_entrenamiento_ideal = d_entrenamiento_ideal;
```

```
        switch(t_recurrencia){
            case 1:     this.n_contexto = n_ocultas;    break;
            case 2:     this.n_contexto = n_salida;     break;
            default:    this.n_contexto = 0;            break;
        }

        t_entradas = n_entrada + n_contexto;
        t_neuronas = t_entradas + n_ocultas + n_salida;

        d_contexto = new double[n_contexto];

        centroides = new double[n_ocultas][t_entradas];
        anchuras =  new double[n_ocultas];
        pesos = new double[n_ocultas];
        salidas = new double[t_neuronas];

        this.error = 0;

    }

    // inicializacion aleatoria de parametros buscados en la red
    public void inicializar(){

        for(int i=0; i<n_ocultas; i++){

            // set de centros comparados con las entradas directas
en la red
            for(int j=0; j<n_entrada; j++){

                // los centroides se inicializan con valores
pertenecientes a la
                // data de entrada a la red
                centroides[i][j] =
d_entrenamiento[(int)Math.floor(d_entrenamiento.length*Math.random
())][(int)Math.floor(d_entrenamiento[i].length*Math.random())];
            }

            // set de centros comparados con las entradas
reinyectadas por la red
            // dependiendo del tipo de retroalimentacion que
presente
            for(int j=n_entrada; j<t_entradas; j++){

                //centroids initialized with a random entrance
data
                centroides[i][j] =
d_entrenamiento[(int)Math.floor(d_entrenamiento.length*Math.random
())][(int)Math.floor(d_entrenamiento[i].length*Math.random())];
            }

            // inicializacion aleatoria de las anchuras
            anchuras[i] = Math.random();
```

```
            // inicializacion aleatoria de los pesos
            pesos[i] = 0.5 - Math.random();

        }

        // inicializo data de contexto en 0
        for(int i=0; i<n_contexto; i++){
            d_contexto[i] = 0;
        }

    }

    // calcula la salida obtenida en cada neurona de la red,
retornando la
    // salida de la perteneciente a la capa de salida
    public double calcularSalidas(double[] d_entrada){

        // respalda nueva informacion de contexto
        double[] d_contexto_bk = new double[n_contexto];

        // salida de neuronas de entrada
        for(int i=0; i<n_entrada; i++)          salidas[i] =
d_entrada[i];
        for(int i=n_entrada; i<t_entradas; i++) salidas[i] =
d_contexto[i-n_entrada];

        // salida obtenida por las neuronas de la capa oculta de
la red
        for(int i=t_entradas; i<t_entradas+n_ocultas; i++){

            // almacena valor de distancia euclidea
            double distancia = 0;

            // calculo de distancias con centroides asociados a
las entradas
            // directas
            for(int j=0; j<n_entrada; j++)          distancia +=
Math.pow((d_entrada[j] - centroides[i-t_entradas][j]), 2);

            // calculo de distancias con centroides asociados a
las entradas
            // reinyectadas por la red
            for(int j=n_entrada; j<t_entradas; j++) distancia +=
Math.pow((d_contexto[j-n_entrada] - centroides[i-t_entradas][j]),
2);

            distancia = Math.sqrt(distancia);

            // calculo de salida dependiendo de funcion de
transferencia seleccionada
            switch (f_transferencia){
```

```
                case 0: salidas[i] = ftGaussiana(distancia,
anchuras[i-t_entradas]);  break;
                case 1: salidas[i] = ftLogistica(distancia,
anchuras[i-t_entradas]);  break;
                case 2: salidas[i] =
ftGaussianaSimplificada(distancia);    break;
                default: salidas[i] =
ftLogisticaSimplificada(distancia);    break;
            }

            // si la recurrencia es Elman, almaceno la salida de
las neuronas
            // de la capa oculta en la data de contexto
            if(t_recurrencia == 1) d_contexto_bk[i-t_entradas] =
salidas[i];

        }

        // salida de neuronas de la capa oculta
        double salida = 0;
        double s_norma = 0;

        // por cada una de las neuronas en la capa de salida de la
red
        // NOTA: dado que la red es MISO, este ciclo solo lo
realiza una vez
        for(int i=t_entradas+n_ocultas; i<t_neuronas; i++){

            // por cada una de las salidas de las neuronas ocultas
se calcula
            // una combinacion lineal entre estas multiplicadas
por los pesos
            for(int j=t_entradas; j<t_entradas+n_ocultas; j++){

                // combinacion lineal
                salida += salidas[j]*pesos[j-t_entradas];

                // valor normalizados en caso de que sea necesario
                s_norma += salidas[j]; //divisor from
normalization
            }

            // en caso de que sea normalizada, normalizo la salida
            if(normalizada) salida = salida / s_norma;

            // actualizo salidas de la red
            salidas[i] = salida;

            // si la recurrencia es Jordan, almaceno en las
neuronas de contexto
            // el valor de la salida de la red
```

```
            if(t_recurrencia == 2) d_contexto_bk[i-
(t_entradas+n_ocultas)] = salida;

        }

        // actualizo data de contexto
        d_contexto = d_contexto_bk;

        //retorno de salida de la red
        return salidas[t_neuronas-1];

    }

    // asigna nuevos valores a los parametros incognitas de la red
    public void asignarParametros(double[] parametros){

        // indice del parametros en cuestion
        int index = 0;

        // actualizacion de los centroides, anchuras y pesos
        for(int i=0; i<n_ocultas; i++){

            for(int j=0; j<t_entradas; j++){

                //actualizo centros
                centroides[i][j] = parametros[index++];
            }

            // dependiendo de la funcion actualizo las anchuras
            if(f_transferencia == 1 || f_transferencia == 0){

                // actualizo las anchuras
                anchuras[i] = parametros[index++];
            }

            // actualizo los pesos
            pesos[i] = parametros[index++];
        }

    }

    // retorna los parametros de la red
    public double[] getParams(){

        // almacenador de parametros
        double[] parametros;

        // tamaño de parametros depende del tipo de funcion de
transferencia
        if(f_transferencia == 1 | f_transferencia == 0)
            parametros = new double[n_ocultas*(t_entradas+2)];
        else
```

```
            parametros = new double[n_ocultas*(t_entradas+1)];

        // indice de parametros
        int index = 0;

        for(int i=0; i<n_ocultas; i++){
            for(int j=0; j<t_entradas; j++){

                // almaceno los centroides de la capa oculta
                parametros[index++] = centroides[i][j];
            }

            // si la f_transferencia no es simplificada almaceno
la anchura
            if(f_transferencia == 0 || f_transferencia == 1)
                parametros[index++] = anchuras[i];

        }

        for(int i=0; i<n_ocultas; i++){

            // almaceno los pesos
            parametros[index++] = pesos[i];

        }

        return parametros;

    }

    // calcula el error de la red
    public void calcularError(double dato) {
        error += Math.pow((dato - salidas[t_neuronas-1]), 2);
    }

    // retorna el error global de la red
    public double obtenerError(){

        double err =
(double)Math.sqrt(error/(d_entrenamiento.length));
        this.error = 0;
        return err;

    }

    // entrenamiento de la red: calcula sus salidas y luego el
error
    public void entrenar(){

        for(int i=0; i<d_entrenamiento.length; i++){
            calcularSalidas(d_entrenamiento[i]);
            calcularError(d_entrenamiento_ideal[i]);
```

```
        }
    }

    /*
     * Funciones de Transferencia
     */
    private double ftGaussiana(double distancia, double anchura) {
        return Math.exp((Math.pow(distancia, 2)*-
1)/(2*Math.pow(anchura, 2)));
    }

    private double ftLogistica(double distancia, double anchura) {
        return 1/(1+Math.exp(Math.pow(distancia,
2)/Math.pow(anchura, 2)));
    }

    private double ftGaussianaSimplificada(double distancia) {
        return Math.exp((Math.pow(distancia, 2)*-1));
    }

    private double ftLogisticaSimplificada(double distancia) {
        return 1/(1+Math.exp(Math.pow(distancia, 2)));
    }

    int getEntradas(){
        return n_entrada;
    }

    int getOcutlas(){
        return n_ocultas;
    }
}
```

Anexo B – Código Fuente Modelo RRBF + GA

```
public class Main_GA {

    public static void main(String[] args) {

        /*
         * Parámetros generales del proceso
         */

        // porcentaje de la data que se utilizara para la fase de
entrenamiento
        // y para la data de prueba de los modelos
        double p_dentrenamiento = 0.9;
        double p_dpruebas = 0.1;

        // numero de veces que se generaran modelos distintos y
numero de veces
        // que estos se entrenaran
        int r_modelos = 1;
        int r_entrenamiento = 1;

        /*
         * Parametros redes neuronales RBF
         */

        // numero de neuronas que conforman la RNA
        int n_entrada = 6;
        int n_ocultas = 8;
        int n_salida = 1;

        // tipo de funcion de transferencia que presentan las
neuronas ocultas
        // 0: gaussiana
        // 1: logistica
        // 2: gaussiana simplificada
        // 3: logistica simplificada
        int f_transferencia = 0;

        // define si la salida de la red es o no "normalizada"
        boolean normalizada = false;

        // tipo de feedback que presenta la RNA
        // 0: no presenta
        // 1: elman
        // 2: jordan
```

```
        int t_recurrencia = 2;

        /*
         * Parametros de Algoritmos Geneticos
         */

        // tamaño de la poblacion
        int t_poblacion = 100;

        // tipo de seleccion de los padres
        // 0: ranking
        // 1: ruleta
        // 2: torneo
        int t_seleccion = 0;

        // en caso de ser torneo, se debe definir el tamaño
        int t_torneo = 3;

        // caracteristicas del cruzamiento
        // probabilidad de cruce
        double p_cruce = 1;

        // tipo de cruzamiento
        // 0: cruzamiento en 1 solo punto
        // 1: cruzamiento en 2 puntos distintos
        int t_cruce = 0;

        // caracteristicas de mutacion
        // probabilidad de mutar
        double p_mutacion = 0.5;

        // tipo de mutacion
        // 0: mutacion en 1 solo punto
        // 1: mutacion en 2 puntos distintos
        int t_mutacion = 0;

        // factor mutacion
        // valor por el cual seran modificados los valores al
momento de
        // mutar: factor*math.random()
        double f_mutacion = 1.99;

        // division del cromosoma para una doble mutacion o
cruzamiento
        int d_cromosoma =
(n_entrada+n_salida)*n_ocultas+n_ocultas;

        // criterios de termino
        // numero maximo de iteraciones
        int n_iteraciones = 100;

        // error minimos esperado
```

```
        double e_esperado = 0.02;

        /*
         * Inicio procesamiento de data
         */

        // instancia de lector de archivos de datos xls y
almacenamiento de
        // data en respectivos lectores
        LectorXLS lector = new LectorXLS();
        double[] d_observada =
lector.leerData("C:\\dataObservada.xls");
        double[] d_suavizada =
lector.leerData("C:\\dataSuavizada.xls");

        // obtencion de cantidad de datos correspondientes al
entrenamiento y
        // a las pruebas
        int n_entrenamiento = (int)
Math.floor(d_observada.length*p_dentrenamiento);
        int n_pruebas = (int)
Math.floor(d_observada.length*p_dpruebas);

        // creacion de vectores de entrenamiento: datos de entrada
y valor esperado
        double[][]  d_entrenamiento = new double[n_entrenamiento -
n_entrada][n_entrada];
        double[]    d_entrenamiento_ideal = new
double[n_entrenamiento - n_entrada];


        // creacion de vectores de prueba: datos de entrada y
valores esperados
        double[][]  d_pruebas = new double[n_pruebas -
n_entrada][n_entrada];
        double[]    d_pruebas_ideal = new double[n_pruebas -
n_entrada];

        // llenado de los vectores de entrenamiento y pruebas con
los datos
        // obtenidos de los archivos xls: d_observada y
d_suavizada.
        // NOTA: se entrena con la data suavizada y los resultados
se comparan
        // con la data observada.
        for(int i=0; i<d_entrenamiento.length; i++){
            for(int j=0; j<n_entrada; j++){
                d_entrenamiento[i][j] = d_suavizada[i+j];
            }
            d_entrenamiento_ideal[i] = d_observada[i+n_entrada];
        }
```

```
        int cont = d_entrenamiento.length;
        for(int i=0; i<d_pruebas.length; i++){
            for(int j=0; j<n_entrada; j++){
                d_pruebas[i][j] = d_suavizada[cont+j];
            }
            d_pruebas_ideal[i] = d_observada[cont+n_entrada];
            cont++;
        }

        /*
         * Bucle que permite la generacion de r_modelos numeros de
modelos
         * distintos.
         */
        for(int a=0; a<r_modelos; a++){

            // instanciacion de una red neuronal RBF
            RBF red = new RBF(n_entrada, n_ocultas, n_salida,
f_transferencia, normalizada, t_recurrencia, d_entrenamiento,
d_entrenamiento_ideal);

            // instanciacion de una poblacion
            Poblacion P = new Poblacion(red, t_poblacion,
t_seleccion, t_torneo, p_cruce, t_cruce, p_mutacion, t_mutacion,
d_cromosoma, f_mutacion);

            /*
             * Bucle que permite la generacion de r_entrenamiento
entrenamientos
             * distintos para evaluar el comportamiento del modelo
en general
             */
            for(int b=0; b<r_entrenamiento; b++){

                // generacion de manera aleatoria de t_poblacion
nuevos cromosomas
                // pertenecientes a la poblacion P.
                P.inicializar();

                // variable de error utilizado para comparar con
el error encontrado
                // por iteracion, para ver si se cumple el
e_esperado
                double e_iteracion = 1;

                System.out.println("\nEntrenamiento #"+(b+1));

                /*
                 * Bucle correspondiente al numero de iteraciones
realizadas
                 * en el algoritmo genetico para poder encontrar
una solucion
```

```
                 * NOTA: Puede terminar antes si es que llega
primero al error
                 * minimo esperado
                 */
                //for(int i=0; i<n_iteraciones || e_iteracion <
e_esperado; i++){
                for(int i=0; i<n_iteraciones; i++){

                    // genera una nueva poblacion de cromosomas
seleccionando
                    // padres y aplicando las operaciones de
cruzamiento y de
                    // mutacion
                    P.nuevaGeneracion();

                    // calcula el error que presenta la poblacion
                    e_iteracion = P.calcularError();

                    System.out.println(e_iteracion);
                }

                // obtiene el cromosoma que presenta un mejor
desempeño en la
                // ultima poblacion generada por el algoritmo
                Cromosoma s_final = P.mejorCromosoma();

                // carga de los parametros almacenados en el
cromosoma a la red
                // para el inicio de las pruebas
                red.asignarParametros(s_final.obtenerGenes());

                // se muestran y almacenan los datos reales
                System.out.println("\nDatos observados");
                for(int i=0; i<d_pruebas_ideal.length; i++){
                    System.out.println(d_pruebas_ideal[i]);
                }

                // se muestran y almacenan los datos pronosticados
                System.out.println("\nDatos pronosticados");
                double[] d_pronosticados = new
double[d_pruebas_ideal.length];

                for(int i=0; i<d_pronosticados.length; i++){

                    // calcula las salidas de las neuronas en base
al i-esimo
                    // set de datos de entrada almacenado en
d_pruebas
                    d_pronosticados[i] =
red.calcularSalidas(d_pruebas[i]);

                    // calcula el error
```

```
                    red.calcularError(d_pruebas_ideal[i]);

                    // muestra data pronosticada
                    System.out.println(d_pronosticados[i]);
                }

                // clase que contiene las metricas necesarias para
complementar
                // el estudio del analisis de los datos.
                Metricas metricas = new Metricas();


System.out.println("\nRMSE\n"+metricas.RMSE(d_pruebas_ideal,
d_pronosticados));

System.out.println("R2\n"+metricas.R2(d_pruebas_ideal,
d_pronosticados));

System.out.println("MAPE\n"+metricas.MAPE(d_pruebas_ideal,
d_pronosticados));

            }

        }

    }

}

class Poblacion {

    private RBF red; //rbf redwork
    private int pop_size; //population size
    private int rep_size; //number of new descendents
    private int fav_size; //number of parents to procreate
    private Cromosoma[] X;

    private double cross_rate; //crossover rate
    //crossover type
    //0:one point | 1:two point
    private int cross_type;
    private int cross_section; //defines first point to crossover

    //mutation type
    //0:one point | 1:two point
    private int mut_type;
    private double mut_rate; //mutation rate

    //parents selection
    //0:ranking | 1:roulette | 2:tournament
    private int select_type;
    private int t_size; //tournament size
```

```
    double error; //population best error

    Poblacion(RBF red, int t_poblacion, int t_seleccion, int
t_torneo, double p_cruce, int t_cruce, double p_mutacion, int
t_mutacion, int d_cromosoma, double f_mutacion) {

        this.red = red;
        this.pop_size = t_poblacion;
        this.rep_size = (int) Math.floor(pop_size / 2);
        this.fav_size = (int) Math.floor(pop_size / 4);

        this.cross_rate = p_cruce;
        this.cross_type = t_cruce;
        this.cross_section = d_cromosoma;

        this.mut_rate = p_mutacion;
        this.mut_type = t_mutacion;

        this.select_type = t_seleccion;
        this.t_size = t_torneo;

        X = new Cromosoma[pop_size];

    }

    void inicializar() {

        //new pop_size random choromosomes
        for(int i=0; i<pop_size; i++){

            //new chromosome
            red.inicializar();
            X[i] = new Cromosoma();
            X[i].setGenes(red.getParams());

            //initial error calculation
            red.entrenar();
            X[i].setError(red.obtenerError());

        }

        bubble_sort(X); //chromosomes ordered from lowest to
highest error
        error = X[0].getError(); //get best error

    }

    //chromosomes ordered from lowest to highest error
    private void bubble_sort(Cromosoma[] X) {

        Cromosoma temp; //temporal chromosome for data exchange
```

```
        boolean exchange = true; //change flag
        while(exchange){
            exchange = false;
            for(int i=0 ; i<X.length-1; i++){
                if(X[i].getError()>X[i+1].getError()){
                    temp = X[i];
                    X[i] = X[i+1];
                    X[i+1] = temp;
                    exchange = true;
                }
            }
        }

    }

    void nuevaGeneracion() {
        int index = 0; //updating chromosome index
        for(int i=0; i<fav_size; i++){

            double[] p1 = new double[X[i].getNGenes()]; //parent
one
            double[] p2 = new double[X[i].getNGenes()]; //parent
two

            //ranking selection
            if(select_type == 0){
                p1 = X[i].obtenerGenes();
                p2 = X[i+1].obtenerGenes();
            }

            //roulette selection
            if(select_type == 1){
                p1 = roulete(X);
                p2 = roulete(X);
            }

            //tournament selection
            if(select_type == 2){
                p1 = tournament(X,t_size);
                p2 = tournament(X,t_size);
            }

            double[] d1 = new double[p1.length]; //get numbers of
genes
            double[] d2 = new double[p2.length]; //get numbers of
genes

            //one point crossover
            if(cross_type == 0){

                if(Math.random() < cross_rate){
```

```
                    int cut_point =
(int)Math.floor(Math.random()*(X[i].getNGenes()-1)); //random cut
point

                    for(int j=0; j<cut_point; j++){
                        d1[j] = p1[j];
                        d2[j] = p2[j];
                    }
                    for(int j=cut_point; j<p1.length; j++){
                        d1[j] = p2[j];
                        d2[j] = p1[j];
                    }
                }

            }


            //two points crossover
            if(cross_type == 1){

                //checking crossover rate
                if(Math.random() < cross_rate){

                    //first cut point
                    int pc1 =
(int)Math.floor(Math.random()*cross_section);
                    //limit of first crossover
                    int pc2 = cross_section;
                    //second cut point
                    int pc3 =
(int)Math.floor(Math.random()*(X[i].getNGenes()-
cross_section))+cross_section;
                    //limit of second crossover
                    int pc4 = X[i].getNGenes();

                    //crossover process
                    for(int j=0; j<pc1; j++){
                        d1[j] = p1[j];
                        d2[j] = p2[j];
                    }
                    for(int j=pc1; j<pc2; j++){
                        d1[j] = p2[j];
                        d2[j] = p1[j];
                    }
                    for(int j=pc2; j<pc3; j++){
                        d1[j] = p1[j];
                        d2[j] = p2[j];
                    }
                    for(int j=pc3; j<pc4; j++){
                        d1[j] = p2[j];
                        d2[j] = p1[j];
                    }
```

```
                }
            }

            //one point mutation
            if(mut_type == 0){

                if(Math.random()<mut_rate){
                    d1[(int)Math.floor((p1.length-
1)*Math.random())] = 0.99*Math.random();
                }
                if(Math.random()<mut_rate){
                    d2[(int)Math.floor((p2.length-
1)*Math.random())] = 0.99*Math.random();
                }

            }

            //two point mutation
            if(mut_type == 1){

            }

            //updating with new child chromosomes
            X[rep_size+(index++)].setGenes(d1);
            X[rep_size+(index++)].setGenes(d2);

        }
    }

    double calcularError() {

        for(int i=rep_size; i<pop_size; i++){
            red.asignarParametros(X[i].obtenerGenes());
            red.entrenar(); //calculate outputs and error
            X[i].setError(red.obtenerError());

        }

        bubble_sort(X); //chromosome ordered
        error = X[0].getError(); //best error (lowest)
        return error;

    }

    //selection of one parent with roulette selection
    private double[] roulete(Cromosoma[] X) {
        throw new UnsupportedOperationException("Not yet
implemented");
    }

    //selection of one parent with tournament selection
    private double[] tournament(Cromosoma[] X, int t_size) {
```

```
        throw new UnsupportedOperationException("Not yet
implemented");
    }

    //return the best founded configuration of values
    Cromosoma mejorCromosoma() {
        bubble_sort(X);
        return X[0];
    }

}

class Cromosoma {

    double[] genes; //unknown variables
    int num_genes; //genes size

    double error; //choromose (individual) error

    //constructor
    public Cromosoma(){
        this.error = 0;
    }

    //update chromosome genes
    public void setGenes(double[] genes){
        this.genes = genes;
        this.num_genes = genes.length;
    }

    //obtain genes
    public double[] obtenerGenes(){
        return genes;
    }

    //obtain number of genes
    public int getNGenes(){
        return num_genes;
    }

    //obtain an specific gene
    public double getGene(int pos){
        return genes[pos];
    }

    //obtain gene error
    public double getError(){
        return error;
    }

    //update chromosome error
    public void setError(double error){
```

```
            this.error = error;
        }

    }
```

Anexo C – Código Fuente Modelo RRBF + PSO

```
public class Main_PSO {

    public static void main(String args[]){

        /*
         * Parámetros generales del proceso
         */

        // porcentaje de la data que se utilizara para la fase de
entrenamiento
        // y para la data de prueba de los modelos
        double p_dentrenamiento = 0.9;
        double p_dpruebas = 0.1;

        // numero de veces que se generaran modelos distintos y
numero de veces
        // que estos se entrenaran
        int r_modelos = 1;
        int r_entrenamiento = 1;

        /*
         * Parametros redes neuronales RBF
         */

        // numero de neuronas que conforman la RNA
        int n_entrada = 6;
        int n_ocultas = 8;
        int n_salida = 1;

        // tipo de funcion de transferencia que presentan las
neuronas ocultas
        // 0: gaussiana
        // 1: logistica
        // 2: gaussiana simplificada
        // 3: logistica simplificada
        int f_transferencia = 0;

        // define si la salida de la red es o no "normalizada"
        boolean normalizada = false;

        // tipo de feedback que presenta la RNA
        // 0: no presenta
```

```
        // 1: elman
        // 2: jordan
        int t_recurrencia = 2;


        /*
         * Parametros PSO
         */
        int N = 100; //tamaño poblacion
        double w = 0.3; //peso inercial
        double c1 = 1.5; //factor cognitivo
        double c2 = 2; //factor social
        double Vmax = 0.04; //velocidad maxima
        double Xmax = 2; //posicion maxima

        // criterios de termino
        // numero maximo de iteraciones
        int n_iteraciones = 100;

        // error minimos esperado
        double e_esperado = 0.02;

        /*
         * Inicio procesamiento de data
         */

        // instancia de lector de archivos de datos xls y
almacenamiento de
        // data en respectivos lectores
        LectorXLS lector = new LectorXLS();
        double[] d_observada =
lector.leerData("C:\\dataObservada.xls");
        double[] d_suavizada =
lector.leerData("C:\\dataSuavizada.xls");

        // obtencion de cantidad de datos correspondientes al
entrenamiento y
        // a las pruebas
        int n_entrenamiento = (int)
Math.floor(d_observada.length*p_dentrenamiento);
        int n_pruebas = (int)
Math.floor(d_observada.length*p_dpruebas);

        // creacion de vectores de entrenamiento: datos de entrada
y valor esperado
        double[][]  d_entrenamiento = new double[n_entrenamiento -
n_entrada][n_entrada];
        double[]    d_entrenamiento_ideal = new
double[n_entrenamiento - n_entrada];
```

```
        // creacion de vectores de prueba: datos de entrada y
valores esperados
        double[][]  d_pruebas = new double[n_pruebas -
n_entrada][n_entrada];
        double[]    d_pruebas_ideal = new double[n_pruebas -
n_entrada];

        // llenado de los vectores de entrenamiento y pruebas con
los datos
        // obtenidos de los archivos xls: d_observada y
d_suavizada.
        // NOTA: se entrena con la data suavizada y los resultados
se comparan
        // con la data observada.
        for(int i=0; i<d_entrenamiento.length; i++){
            for(int j=0; j<n_entrada; j++){
                d_entrenamiento[i][j] = d_suavizada[i+j];
            }
            d_entrenamiento_ideal[i] = d_observada[i+n_entrada];
        }

        int cont = d_entrenamiento.length;
        for(int i=0; i<d_pruebas.length; i++){
            for(int j=0; j<n_entrada; j++){
                d_pruebas[i][j] = d_suavizada[cont+j];
            }
            d_pruebas_ideal[i] = d_observada[cont+n_entrada];
            cont++;
        }

         /*
         * Bucle que permite la generacion de r_modelos numeros de
modelos
         * distintos.
         */
        for(int a=0; a<r_modelos; a++){

            // instanciacion de una red neuronal RBF
            RBF red = new RBF(n_entrada, n_ocultas, n_salida,
f_transferencia, normalizada, t_recurrencia, d_entrenamiento,
d_entrenamiento_ideal);

            Enjambre E = new Enjambre(red,N,w,c1,c2,Vmax,Xmax);
//instancia del enjambre

             /*
             * Bucle que permite la generacion de r_entrenamiento
entrenamientos
             * distintos para evaluar el comportamiento del modelo
en general
             */
            for(int b=0; b<r_entrenamiento; b++){
```

```
                E.inicializar(); //genera enjambre inicial

                // variable de error utilizado para comparar con
el error encontrado
                // por iteracion, para ver si se cumple el
e_esperado
                double e_iteracion = 1;

                System.out.println("\nEntrenamiento #"+(b+1));

                /*
                 * Bucle correspondiente al numero de iteraciones
realizadas
                 * en el algoritmo genetico para poder encontrar
una solucion
                 * NOTA: Puede terminar antes si es que llega
primero al error
                 * minimo esperado
                 */
                //for(int i=0; i<n_iteraciones || e_iteracion <
e_esperado; i++){
                for(int i=0; i<n_iteraciones; i++){

                    E.iterar(); //actualiza X,V,Pbest,Gbest y
error
                    e_iteracion = E.getError();
                    System.out.println(e_iteracion);
                }

                red.asignarParametros(E.getGbest());

                // se muestran y almacenan los datos reales
                System.out.println("\nDatos observados");
                for(int i=0; i<d_pruebas_ideal.length; i++){
                    System.out.println(d_pruebas_ideal[i]);
                }

                // se muestran y almacenan los datos pronosticados
                System.out.println("\nDatos pronosticados");
                double[] d_pronosticados = new
double[d_pruebas_ideal.length];

                for(int i=0; i<d_pronosticados.length; i++){

                    // calcula las salidas de las neuronas en base
al i-esimo
                    // set de datos de entrada almacenado en
d_pruebas
                    d_pronosticados[i] =
red.calcularSalidas(d_pruebas[i]);
```

```
                    if(d_pronosticados[i]<0) d_pronosticados[i] =
0.00001;

                    // calcula el error
                    red.calcularError(d_pruebas_ideal[i]);

                    // muestra data pronosticada
                    System.out.println(d_pronosticados[i]);
                }

                // clase que contiene las metricas necesarias para
complementar
                // el estudio del analisis de los datos.
                Metricas metricas = new Metricas();


System.out.println("\nRMSE\n"+metricas.RMSE(d_pruebas_ideal,
d_pronosticados));

System.out.println("R2\n"+metricas.R2(d_pruebas_ideal,
d_pronosticados));

System.out.println("MAPE\n"+metricas.MAPE(d_pruebas_ideal,
d_pronosticados));

            }

        }


    }

}

public class Enjambre {

    private RBF red;
    private int N;
    private double w;
    private double c1;
    private double c2;
    private double Vmax;
    private double Xmax;
    private Particula[] P;
    private double[] Gbest;
    private double E; //error asociado al Gbest

    Enjambre(RBF red, int N, double w, double c1, double c2,
double Vmax, double Xmax){

        this.red = red;
        this.N = N; //numero de particulas
```

```
        this.w = w; //peso de inercia
        this.c1 = c1; //factor cognitivo
        this.c2 = c2; //factor social
        this.Vmax = Vmax; //velocidad maxima
        this.Xmax = Xmax; //posicion maxima
        this.E = 1; //error inicial enjambre

        P = new Particula[N]; //particulas del enjambre

    }

    public void inicializar(){

        for(int i=0; i<N; i++){
            red.inicializar();
            P[i] = new Particula();
            P[i].inicializar(red.getParams(),Vmax);

            //calculo error inicial
            red.entrenar(); //calculo salidas y genero error

            P[i].setError(red.obtenerError()); //error inicial
particula
            P[i].setBestError(P[i].getError()); //mejor error
inicial particula
        }

        Gbest = P[0].getPosicion(); //primer mejor error default
es P[0]
        E = 1;

    }

    public void iterar() {

        double Eit = 0;

        for(int i=0; i<N; i++){

            P[i].actualizar(w,c1,c2,Vmax,Xmax,Gbest); //actualiza
X,V
            Eit = calculoError(P[i]); //calcula error de particula

            if(Eit < P[i].getBestError()){
                P[i].setPbest(P[i].getPosicion());
                P[i].setBestError(Eit);
            }

            if(Eit < E){
                this.Gbest = P[i].getPosicion();
                this.E = Eit;
            }
```

```
        }

    }

    public double getError() {
        return E;
    }

    public double[] getGbest(){
        return Gbest;
    }

    public double calculoError(Particula P) {

        red.asignarParametros(P.getPosicion());
        red.entrenar();
        P.setError(red.obtenerError());

        return P.getError();

    }

    public void reinicializar() {
        for(int i=0; i<N; i++){
            red.inicializar();
            P[i] = new Particula();
            P[i].inicializar(red.getParams(),Vmax);

            //calculo error inicial
            red.entrenar(); //calculo salidas y genero error

            P[i].setError(red.obtenerError()); //error inicial
particula
            P[i].setBestError(P[i].getError()); //mejor error
inicial particula
        }

    }

    //genera enjambre con valores conocidos
    public void setEnjambre(double[][] E){

        for(int i=0; i<N; i++){
            P[i].setPosicion(E[i]);
            P[i].setPbest(E[i]);

            //calculo error inicial
            red.asignarParametros(P[i].getPosicion());
            red.entrenar(); //calculo salidas y genero error
```

```
            P[i].setError(red.obtenerError()); //error inicial
particula
            P[i].setBestError(P[i].getError()); //mejor error
inicial particula
        }

        Gbest = P[0].getPosicion(); //primer mejor error default
es P[0]
        this.E = 1;

    }

    //retorna enjambre
    public double[][] getEnjambre(){

        double[][]S = new double[N][];
        for(int i=0; i<N; i++){
            S[i] = P[i].getPosicion();
        }

        return S;

    }

    public void setError(double E) {
        this.E = E;
    }

    public void setGbest(double[] Gbest) {
        this.Gbest = Gbest;
    }

    public void setC(double d, double d0) {
        this.c1 = d;
        this.c2 = d0;
    }

}


public class Particula {

    private double[] X;
    private double[] V;
    private double[] Pbest;
    private double E;
    private double Ebest;

    public Particula(){

    }
```

```
    public void inicializar(double[] X, double Vmax){

        double rand;
        this.X = X; //posiciones iniciales
        this.Pbest = X; //mejor posicion inicial

        V = new double[X.length]; //genero vector de velocidad

        //velocidades iniciales
        for(int i=0; i<X.length; i++){
            rand = Math.random();
            V[i] = rand*Vmax + (1-rand)*(-Vmax);
        }

    }

    //actualiza posicion y velocidad de la particula
    public void actualizar(double w, double c1, double c2, double
Vmax, double Xmax, double[] Gbest){

        for(int i=0; i<X.length; i++){
            V[i] = w*V[i] + c1*Math.random()*(Pbest[i] - X[i]) +
c2*Math.random()*(Gbest[i]-X[i]);

            if(V[i] > Vmax) V[i] = Vmax;
            if(V[i] < -Vmax) V[i] = -Vmax;

            X[i] = X[i] + V[i];

            if(X[i] > Xmax) X[i] = Xmax;
            if(X[i] < -Xmax) X[i] = -Xmax;


        }

    }

    public double[] getPosicion(){
        return X;
    }

    public void setError(double E){
        this.E = E;
    }

    public void setBestError(double Ebest){
        this.Ebest = Ebest;
    }

    public void setPbest(double[] Pbest){
        this.Pbest = Pbest;
    }
```

```
    public double getError(){
        return E;
    }

    public double getBestError(){
        return Ebest;
    }

    public void setPosicion(double []X){
        this.X = X;
    }

}
```

Anexo D – Código Fuente Modelo RRBF + BS

```
import breeding_swarm.Swarm_Jordan;
import redes_neuronales.RBF_Jordan;
import utiles.*;

/*
 * Breeding Swarm
 * Basado en el paper: Breeding Swarm: A GA/PSO Hybrid
 * Algoritmo que mediante un nuevo operador VPAC se utilizada
 * para entrenar una red rbf
 */

public class BSRBF_Jordan {

    public static void main(String args[]){

        //obtengo data
        Lector_XLS r = new Lector_XLS();
        double[] DO = r.getData("C:\\dataObservada.xls");
        double[] DS = r.getData("C:\\dataSuavizada.xls");

//parametros
//red neuronal
int NE = 3; //neuronas de entrada
int NO = 5; //neuronas ocultas
int NS = 1; //neuronas de salida

//enjambre de particulas
int N = 80; //tamaño poblacion
double w = 0.6; //peso inercial
double c1 = 0.6; //factor cognitivo
double c2 = 0.9; //factor social
double Vmax = 0.06; //velocidad maxima
double Xmax = 2; //posicion maxima
double B = 0.5; //breeding ratio

//criterio de termino
int It = 300; //numero de iteraciones
double Ee = 0.01; //error esperado

        //inicializo vectores entrenamiento
        int Ne = (int) Math.floor(DO.length * 0.6); //cantidad
data entrenamiento
        int Np = DS.length - Ne; //cantidad data de pruebas
```

```
        double Xe[][] = new double[Ne][NE]; //data entrenamiento
        double Xi[] = new double[Ne]; //data entrenamiento ideal
        double Yp[][] = new double[Np - NE][NE]; //data pruebas
        double Yi[] = new double[Np - NE]; //data pruebas ideal

        //llenado vector entrenamiento
        for(int i=0; i<Ne; i++){
            for(int j=0; j<NE; j++){
                Xe[i][j] = DS[i+j]; //entreno con data suavizada
            }
            Xi[i] = DO[i+NE]; //comparo con data observada
        }

        //llenado vector de prueba
        int cont=Ne; //indica donde quede en la data asignada a
los entrenamientos
        for(int i=0; i<Np-NE; i++){
            for(int j=0; j<NE; j++){
                Yp[i][j] = DS[cont+j]; //pruebo con data suavizada
            }
            Yi[i] = DO[cont+NE]; //comparo con data observada
            cont++;
        }


        //ciclo de t repeticiones
        for(int t=0;t<20;t++){

            RBF_Jordan red = new RBF_Jordan(NE,NO,NS,Xe,Xi);
//instancia RRBF

            Swarm_Jordan2 E = new
Swarm_Jordan2(red,N,w,c1,c2,Vmax,Xmax,B); //instancia del enjambre
            E.inicializar(); //genera enjambre inicial

            double Ei = 1; //error de iteracion

            System.out.println("\nRMSE");
            for(int i=0; i<It || Ei<Ee; i++){
                E.iterar(i); //actualiza X,V,Pbest,Gbest y error
                Ei = E.getError();
                System.out.println(Ei);
            }

            red.setParametros(E.getGbest()); //obtencion de mejor
resultado

            //pruebas
            System.out.println("\nData Ideal");
            double o[] = new double[Yp.length];
            for(int i=0;i<Yi.length;i++){
```

```
                System.out.println(Yi[i]);
                o[i] = Yi[i];
            }

            System.out.println("\nData Pronosticada");
            double p[] = new double[Yp.length];
            for (int i=0;i<Yp.length;i++) {
                p[i] = red.calculoSalidas(Yp[i]);
                red.calculoError(Yi[i]);
                System.out.println(p[i]);
            }

            Info inf = new Info();
            System.out.println("\nRMSE\n"+inf.RMSE(o, p));
            System.out.println("R2\n"+inf.R2(o, p));
            System.out.println("MAPE\n"+inf.MAPE(o, p));

        }

    }

}

import redes_neuronales.RBF_Jordan;

public class Swarm_Jordan {

    private RBF_Jordan red;
    private int N;
    private double w;
    private double c1;
    private double c2;
    private double Vmax;
    private double Xmax;
    private Particle[] P;
    private double[] Gbest;
    private double E; //error asociado al Gbest
    private double B;
    private int best;
    private int worst;

    public Swarm_Jordan(RBF_Jordan red, int N, double w, double
c1, double c2, double Vmax, double Xmax, double B){

        this.red = red;
        this.N = N; //numero de Particles
        this.w = w; //peso de inercia
        this.c1 = c1; //factor cognitivo
        this.c2 = c2; //factor social
        this.Vmax = Vmax; //velocidad maxima
        this.Xmax = Xmax; //posicion maxima
        this.E = 1; //error inicial Swarm
```

```
        this.B = B; //breeding ratio

        //cantidad de mejores y peores seleccionados
        this.best    =    (int)    Math.floor(N*(1-B));    //mejores
particulas a seleccionar
        this.worst = N-this.best; //restante peor

        P = new Particle[N]; //Particles del Swarm

    }

    public void inicializar(){

        for(int i=0; i<N; i++){
            red.reset();
            P[i] = new Particle();
            P[i].inicializar(red.getParametros(),Vmax);

            //calculo error inicial
            red.entrenar(); //calculo salidas y genero error

            P[i].setError(red.getError());     //error     inicial
Particle
            P[i].setBestError(P[i].getError());    //mejor    error
inicial Particle
        }

        Gbest  =  P[0].getPosicion();  //primer  mejor  error  default
es P[0]
        E = 1;

    }

    public void iterar(int k) {

        boolean bs;
        int limit;
        if(k!=0 && k%50!=0){
            bs=true;
            limit=best;
        }else{
            bs=false;
            limit=N;
        }

        if(bs) ordenarParticulas(); //ordena particulas de menor a
mayor error

        //actualiza las "best" mejores particulas
        double Eit = 0;
        for(int i=0; i<limit; i++){
```

```
            P[i].actualizar(w,c1,c2,Vmax,Xmax,Gbest);   //actualiza
X,V
            Eit = calculoError(P[i]); //calcula error de Particle

            if(Eit < P[i].getBestError()){
                P[i].setPbest(P[i].getPosicion());
                P[i].setBestError(Eit);
            }

            if(Eit < E){
                this.Gbest = P[i].getPosicion();
                this.E = Eit;
            }

        }

        if(bs){
            //mediante   VPAC   se   generan   las   "worst"   nuevas
partículas
            for(int i=best; i<N; i=i+2){

                Particle  p1  =  seleccionTorneo();  //seleccion  de
torneo de 3
                Particle p2 = seleccionTorneo();

                double rand = Math.random();
                P[i].VPAC(p1,p2,rand); //cruzamiento BS
                rand = Math.random();
                P[i+1].VPAC(p2,p1,rand); //cruzamiento BS

                //calculo de errores
                Eit   =   calculoError(P[i]);   //calcula   error   de
Particle

                if(Eit < E){
                    this.Gbest = P[i].getPosicion();
                    this.E = Eit;
                }

                Eit  =  calculoError(P[i+1]);  //calcula  error  de
Particle

                if(Eit < E){
                    this.Gbest = P[i+1].getPosicion();
                    this.E = Eit;
                }
            }

        }

    }
```

```
    public double getError() {
        return E;
    }

    public double[] getGbest(){
        return Gbest;
    }

    public double calculoError(Particle P) {

        red.setParametros(P.getPosicion());
        red.entrenar();
        P.setError(red.getError());

        return P.getError();

    }

    public void reinicializar() {
        for(int i=0; i<N; i++){
            red.reset();
            P[i] = new Particle();
            P[i].inicializar(red.getParametros(),Vmax);

            //calculo error inicial
            red.entrenar(); //calculo salidas y genero error

            P[i].setError(red.getError());     //error     inicial
Particle
            P[i].setBestError(P[i].getError());    //mejor    error
inicial Particle
        }

    }

    //genera Swarm con valores conocidos
    public void setSwarm(double[][] E){

        for(int i=0; i<N; i++){
            P[i].setPosicion(E[i]);
            P[i].setPbest(E[i]);

            //calculo error inicial
            red.setParametros(P[i].getPosicion());
            red.entrenar(); //calculo salidas y genero error

            P[i].setError(red.getError());     //error     inicial
Particle
            P[i].setBestError(P[i].getError());    //mejor    error
inicial Particle
        }
```

```
        Gbest = P[0].getPosicion(); //primer mejor error default
es P[0]
        this.E = 1;

    }

    //retorna Swarm
    public double[][] getSwarm(){

        double[][]S = new double[N][];
        for(int i=0; i<N; i++){
            S[i] = P[i].getPosicion();
        }

        return S;

    }

    public void setError(double E) {
        this.E = E;
    }

    public void setGbest(double[] Gbest) {
        this.Gbest = Gbest;
    }

    public void setC(double d, double d0) {
        this.c1 = d;
        this.c2 = d0;
    }

    //ordena las particula de menor a mayor error
    private void ordenarParticulas() {

        Particle TMP; //particula para intercambio
        boolean intercambiado = true; //avisa cuando se realiza un
cambio
        while(intercambiado){
            intercambiado = false;
            for(int i=0 ; i<P.length-1; i++){
                if(P[i].getError()>P[i+1].getError()){
                    TMP = P[i];
                    P[i] = P[i+1];
                    P[i+1] = TMP;
                    intercambiado = true;
                }
            }
        }

    }

    private Particle seleccionTorneo() {
```

```
        Particle s1 = P[(int) Math.floor(best*Math.random())];
        Particle s2 = P[(int) Math.floor(best*Math.random())];
        Particle s3 = P[(int) Math.floor(best*Math.random())];

        if(s1.getError()  <  s2.getError()  &&  s1.getError()  <
s3.getError()){
            return s1;
        }else if(s2.getError() < s1.getError() && s2.getError() <
s3.getError()){
            return s2;
        }else{
            return s3;
        }

    }

}
```

Anexo E – Código Fuente Modelo RRBF-IP GA+PSO

```
import algoritmo_genetico.Cromosoma;
import algoritmo_genetico.Poblacion;
import enjambre_particulas.Enjambre;
import redes_neuronales.RBF_Elman;
import utiles.*;

public class InitParams_GAPSO {

    public static void main(String args[]){

        //obtengo data
        Lector_XLS r = new Lector_XLS();
        double[] DO = r.getData("C:\\dataObservada.xls");
        double[] DS = r.getData("C:\\dataSuavizada.xls");

//parametros
//red neuronal
int NE = 1; //neuronas de entrada
int NO = 4; //neuronas ocultas
int NS = 1; //neuronas de salida

//enjambre de particulas
int N = 100; //tamaño poblacion y enjambre
double w = 0.6; //peso inercial
double c1 = 0.4; //factor cognitivo
double c2 = 0.9; //factor social
double Vmax = 0.05; //velocidad maxima
double Xmax = 2; //posicion maxima

//algoritmo genetico
double Pc = 0.8; //porcentaje de cruzamiento
double Pm = 0.5; //porcentaje de mutacion

//criterio de termino
int ITg = 10; //numero de iteraciones genetico
int ITp = 140; //numero de iteraciones pso
double Ee = 0.03; //error esperado

        //inicializo vectores entrenamiento
        int Ne = (int) Math.floor(DO.length * 0.9); //cantidad
data entrenamiento
        int Np = DS.length - Ne; //cantidad data de pruebas

        double Xe[][] = new double[Ne][NE]; //data entrenamiento
```

```
        double Xi[] = new double[Ne]; //data entrenamiento ideal
        double Yp[][] = new double[Np - NE][NE]; //data pruebas
        double Yi[] = new double[Np - NE]; //data pruebas ideal

        //llenado vector entrenamiento
        for(int i=0; i<Ne; i++){
            for(int j=0; j<NE; j++){
                Xe[i][j] = DS[i+j]; //entreno con data suavizada
            }
            Xi[i] = DO[i+NE]; //comparo con data observada
        }

        //llenado vector de prueba
        int cont=Ne; //indica donde quede en la data asignada a
los entrenamientos
        for(int i=0; i<Np-NE; i++){
            for(int j=0; j<NE; j++){
                Yp[i][j] = DS[cont+j]; //pruebo con data suavizada
            }
            Yi[i] = DO[cont+NE]; //comparo con data observada
            cont++;
        }

        //ciclo de t repeticiones
        for(int t=0; t<1; t++){

            RBF_Elman red = new RBF_Elman(NE,NO,NS,Xe,Xi);
//instancia RRBF

            //algoritmo genetico
            Poblacion P = new Poblacion(red,N,Pc,Pm); //instancia
Poblacion
            P.inicializar(); //genera poblacion inicial aleatoria

            double Ei = 1; //error de iteracion

            //System.out.println("\nGA:RMSE");
            for(int i=0; i<ITg || Ei<Ee; i++){
                P.nuevaGeneracion(); //selecciona, cruza y muta
                Ei = P.calculoError();
                System.out.println(Ei);
            }

            //enjambre de particulas
            Enjambre E = new Enjambre(red,N,w,c1,c2,Vmax,Xmax);
//instancia del enjambre
            E.inicializar(); //inicializa velocidades y posiciones
            //E.setEnjambre(P.getPoblacion()); //cambia posiciones
por la poblacion del genetico

            Cromosoma solucion = P.solucionFinal(); //mejor
cromosoma (menor error)
```

```
            E.setError(solucion.getError());   //actualiza   mejor
error pso
            E.setGbest(solucion.getGenes());   //actualiza   mejor
posicion

            //System.out.println("\nPSO:RMSE");
            for(int i=0; i<ITp || Ei<Ee; i++){
                E.iterar(); //actualiza X,V,Pbest,Gbest y error
                Ei = E.getError();
                System.out.println(Ei);
            }

            red.setParametros(E.getGbest());

            System.out.println("\nData Ideal");
            double o[] = new double[Yp.length];
            for(int i=0;i<Yi.length;i++){
                System.out.println(Yi[i]);
                o[i] = Yi[i];
            }

            System.out.println("\nData Pronosticada");
            double p[] = new double[Yp.length];
            for (int i=0;i<Yp.length;i++) {
                p[i] = red.calculoSalidas(Yp[i]);
                red.calculoError(Yi[i]);
                System.out.println(p[i]);
            }

            Info inf = new Info();
            System.out.println("\nRMSE\n"+inf.RMSE(o, p));
            System.out.println("R2\n"+inf.R2(o, p));
            System.out.println("MAPE\n"+inf.MAPE(o, p));

        }

    }

}
```