SEP
SECRETARÍA DE EDUCACIÓN PÚBLICA

TECNOLÓGICO NACIONAL DE MÉXICO

INSTITUTO TECNOLÓGICO DE CIUDAD MADERO

DIVISIÓN DE ESTUDIOS DE POSGRADO E INVESTIGACIÓN

MAESTRÍA EN CIENCIAS DE LA COMPUTACIÓN

"POR MI PATRIA Y POR MI BIEN"

TESIS

ANÁLISIS DE DESEMPEÑO DE REDES NEURONALES EVOLUTIVAS PARA EL PRONÓSTICO DE SERIES DE TIEMPO

Que para obtener el Grado de
Maestro en Ciencias de la Computación

Presenta
Ing. Moisés Israel Herrera Ramos
G05070523

Director de Tesis
Dr. Juan Frausto Solís

Co-Director de Tesis
Dra. Guadalupe Castilla Valdez

Cd. Madero, Tamaulipas Junio 2019

SEP
SECRETARÍA DE EDUCACIÓN PÚBLICA

TECNOLÓGICO NACIONAL DE MÉXICO

Instituto Tecnológico de Ciudad Madero

"2019, Año del Caudillo del Sur, Emiliano Zapata"

Cd. Madero, Tams., a 29 de Mayo de 2019

OFICIO No.: U5.082/19
ÁREA: DIVISIÓN DE ESTUDIOS DE POSGRADO E INVESTIGACIÓN
ASUNTO: AUTORIZACIÓN DE IMPRESIÓN DE TESIS.

ING. MOISÉS ISRAEL HERRERA RAMOS
No. DE CONTROL G05070523
P R E S E N T E

Me es grato comunicarle que después de la revisión realizada por el Jurado designado para su Examen de Grado de Maestro en Ciencias de la Computación, se acordó autorizar la impresión de su tesis titulada:

"ANÁLISIS DE DESEMPEÑO DE REDES NEURONALES EVOLUTIVAS PARA EL PRONÓSTICO DE SERIES DE TIEMPO"

El Jurado está integrado por los siguientes catedráticos:

PRESIDENTE:	DR. JOSÉ ANTONIO MARTÍNEZ FLORES
SECRETARIO:	DRA. LAURA CRUZ REYES
VOCAL:	DR. JUAN FRAUSTO SOLIS
SUPLENTE:	DRA. GUADALUPE CASTILLA VALDEZ
DIRECTORA DE TESIS :	DR. JUAN FRAUSTO SOLIS
CO-DIRECTORA:	DRA. GUADALUPE CASTILLA VALDEZ

Es muy satisfactorio para la División de Estudios de Posgrado e Investigación compartir con Usted el logro de esta meta. Espero que continúe con éxito su desarrollo profesional y dedique su experiencia e inteligencia en beneficio de México.

A T E N T A M E N T E
Excelencia en Educación Tecnológica®
"Por mi patria y por mi bien"®

DR. JOSÉ AARÓN MELO BANDA
JEFE DE LA DIVISIÓN DE ESTUDIOS DE POSGRADO E INVESTIGACIÓN

SECRETARÍA DE EDUCACIÓN PÚBLICA
TECNOLÓGICO NACIONAL DE MÉXICO
INSTITUTO TECNOLÓGICO DE CIUDAD MADERO
DIVISIÓN DE ESTUDIOS DE POSGRADO E INVESTIGACIÓN

c.c.p.- Archivo
Minuta

JAMB 'JAMF 'jar

Av. 1° de Mayo y Sor Juana I. de la Cruz Col. Los Mangos, C.P. 89440, Cd. Madero, Tam.
Tel. 01 (833) 357 48 20, e-mail: dir01_cdmadero@tecnm.mx
www.tecnm.mx | www.cdmadero.tecnm.mx

Declaración de Originalidad

Declaro y prometo que este documento de tesis es producto de mi trabajo original y que no infringe los derechos de terceros, tales como derechos de publicación, derechos de autor, patentes y similares.

Además, declaro que en las citas textuales que he incluido (las cuales aparecen entre comillas) y en los resúmenes que he realizado de publicaciones ajenas, indico explícitamente los datos de los autores y las publicaciones.

Además, en caso de infracción de los derechos de terceros derivados de este documento de tesis, acepto la responsabilidad de la infracción y relevo de ésta a mi director y codirectores de tesis, así como al Instituto Tecnológico de Cd. Madero y sus autoridades.

Mayo de 2019, Cd. Madero, Tamps.

Ing. Moises Israel Herrera Ramos

Tabla de contenido

Capítulo 1

Introducción

El Problema del Pronóstico (Forecasting Problem) surge en diferentes ámbitos tales como logística, transporte, producción y finanzas. En estas áreas, es de gran importancia para los tomadores de decisión contar con herramientas de pronóstico con el fin de hacer frente a la estacionalidad, los cambios repentinos en los niveles de demanda, las maniobras de reducción de precios de la competencia y la oscilación en la economía. En los mercados financieros las corporaciones y los inversionistas requieren de un marco referencial sobre cómo se comportará el mercado en el futuro y consecuentemente tomar buenas decisiones para conformar carteras de inversión. El pronóstico se utiliza para analizar los activos de los mercados financieros con el fin de obtener información anticipada sobre la valorización de las cotizaciones de los valores de forma que se puedan realizar operaciones de compra venta con beneficios. Una serie de tiempo es una sucesión de observaciones tomadas secuencialmente en el tiempo [Chatfield, 2000].; algunos ejemplos de series de tiempo son: la cotización del dólar en el mes de noviembre de 2012, la temperatura ambiental de Londres en el mes de enero del año actual y la presión atmosférica en la ciudad de Tampico en el periodo de verano de 2016.

Un pronóstico es un proceso de predicción de valores futuros, el cual toma datos pasados y presentes de una serie de tiempo. El pronóstico del precio de las acciones de los mercados financieros es una de las tareas más difíciles de realizar debido a naturaleza compleja de los mercados de valores [Pai,2005][Wang,2012][Wei,2013]. En estos mercados donde se realizan operaciones de compra y venta de valores de millones de dólares, los inversionistas necesitan contar con buenos métodos de pronóstico con el fin de realizar sus operaciones financieras con la mayor garantía de un máximo beneficio y un mínimo riesgo. Esta situación ha motivado a los investigadores a desarrollar nuevos modelos predictivos [Atsalakis,2011]. En los últimos años, varios modelos y técnicas han sido utilizados para la predicción del precio de las acciones. Entre los más populares están las Redes Neuronales Artificiales (RNAs) debido a su capacidad de aprender patrones a partir de datos e inferir datos desconocidos. En estos modelos el elemento básico es la neurona artificial, que busca emular una neurona del cerebro humano debido a que cuenta con uno o varios elementos de entrada que emulan el comportamiento de las neuronas biológicas, es decir se excitan, generan una respuesta y se comunican con otras neuronas. Al conjunto de todas las neuronas se le denomina red neuronal.

Los métodos clásicos de pronóstico buscan que los datos se ajusten a un modelo, la diferencia con el uso de redes neuronales es que estas se ajustan a los datos generando un modelo que permite hacer predicciones.

Este trabajo busca crear un nuevo modelo híbrido de redes neuronales evolutivas con algoritmos heurísticos para el pronóstico de series de tiempo financieras.

1.1 Antecedentes

Dentro del grupo de trabajo de trabajo del Instituto Tecnológico de Ciudad Madero de Inteligencia Artificial y de optimización inteligente se desarrollan actualmente para este problema la optimización de portafolios con activos de la BMV, el pronóstico de series de tiempo utilizando máquinas de soporte vectorial, métodos caóticos y métodos clásicos. Así mismo, existe otro trabajo que se está realizando y tiene por nombre diseño de portafolios de inversión utilizando pronóstico de series de tiempo con máquinas de soporte vectorial en el cual se crean portafolios de inversión utilizando el modelo de Markowitz.

El trabajo de tesis propuesto en este documento, es el primer trabajo que se realiza dentro del grupo en el cual se plantea utilizar redes neuronales evolutivas para el pronóstico de series de tiempo de la BMV.

1.2 Objetivos del proyecto

En esta sección se plantean los principales objetivos que se tienen en este trabajo de tesis.

1.2.1 Objetivos General

Desarrollar un algoritmo híbrido basado en modelos de redes neuronales evolutivas y algoritmos heurísticos para el pronóstico de series de tiempo financieras.

1.2.2 Objetivos Específicos

Explorar diferentes modelos de neuroevolución para el pronóstico de series de tiempo financieras.

Diseñar e implementar las estrategias neuroevolutivas seleccionadas.

Diseñar e implementar un modelo híbrido de basado en redes neuronales y algoritmos evolutivos.

Evaluar el modelo híbrido neuroevolutivo sobre diferentes series de tiempo financieras comparándolo con el método clásico ARIMA.

1.3 Justificación

El problema de pronóstico de series de tiempo tiene importantes aplicaciones tales como en finanzas para el pronóstico del valor de las acciones en los mercados financieros y en

el área de la toma de decisiones dentro de las organizaciones. Para este problema se han utilizado ampliamente los métodos clásicos, sin embargo, actualmente los métodos de Inteligencia artificial tales como las máquinas de soporte vectorial (SVR) [Santamaría et al,2015] [Law, Shawe, 2017], las redes neuronales[Zhang,2003] y las redes neuronales evolutivas se han aplicado con éxito [Ariyo,2014], [Srinivasan,1998], [Chen, Chang,2009], [González et al., 2012] y se espera tener un desempeño competitivo para el pronóstico de series de tiempo financieras.

1.4 Alcances y Limitaciones

Se utilizan únicamente las series de tiempo de la bolsa mexicana de valores como instancias de prueba para la validación del modelo de red neuronal propuesto.

Se comparan los resultados del algoritmo desarrollado con los métodos clásicos de pronóstico.

Se diseña un algoritmo híbrido de redes neuronales evolutivas con algoritmo heurístico para el pronóstico de series de tiempo.

Se utilizará software estadístico para la comparación de resultados entre el algoritmo híbrido y los métodos clásicos.

Capítulo 2

Marco Teórico

2.1 Serie de tiempo

Es un conjunto de observaciones medidas secuencialmente a través del tiempo [Chatfield, 2000].

El análisis de las series de tiempo financiera se refiere a la teoría y la práctica de la valoración de activos a lo largo del tiempo [Tsay,2010].

Cualquier variable que consta de datos reunidos, registrados u observados sobre incrementos sucesivos de tiempo [Hanke,2010].

2.1.1 Componentes de una serie de tiempo

Las series de tiempo tienen 4 componentes que son: tendencia, un componente cíclico, un componente estacional y un componente irregular. A continuación, se describen estos componentes.

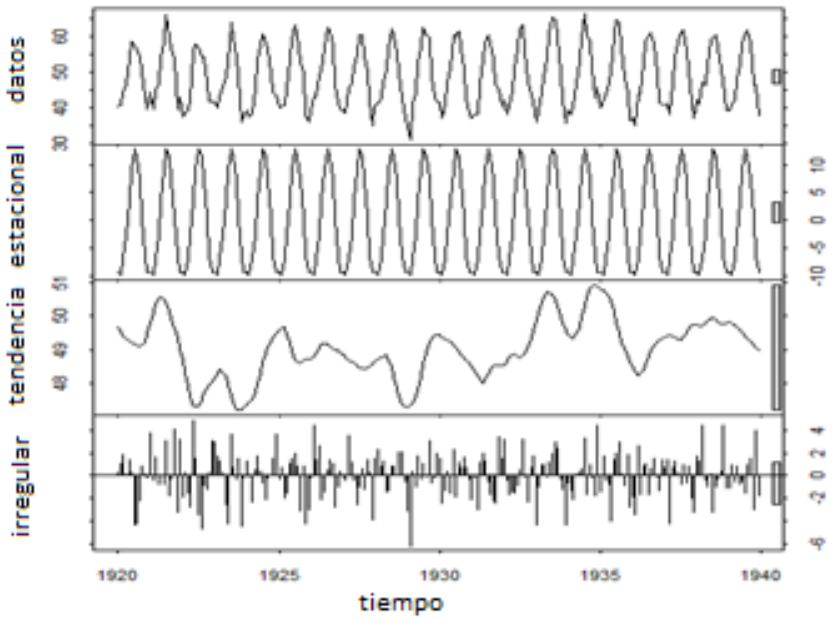


Figura 1 Ejemplo de descomposición de la serie de tiempo de temperaturas del aire promedio en el castillo de Nottingham en grados Fahrenheit por 20 años

Tendencia. La tendencia es el componente que representa el crecimiento (o la declinación) subyacente en una serie de tiempo sobre varios periodos de tiempo sucesivos. La tendencia se produce, por ejemplo, como resultado de la inflación, de cambios demográficos persistentes o de cambios tecnológicos e incrementos en la productividad. [Hanke,2010][Chatfield,2000].

Componente cíclico. Es una serie de fluctuaciones en forma de onda o ciclos de más de un año de duración [Hanke,2010]. Las condiciones cambiantes de la economía generalmente producen ciclos.

Componente estacional. Los componentes estacionales se encuentran comúnmente en datos trimestrales, mensuales o semanales. La variación estacional se refiere a un patrón de cambio más o menos estable que aparece anualmente y se repite un año tras otro [Hanke,2010]. Los patrones estacionales ocurren por la influencia del tiempo climatológico o por acontecimientos relacionados con el calendario, como las vacaciones escolares y los días feriados nacionales. Si una serie solo es medida anualmente no es posible obtener el componente estacional.

Componente irregular. El componente irregular consiste en fluctuaciones impredecibles o aleatorias [Hanke,2010].

2.2 Normalización

Para trabajar con los datos de una serie de tiempo se le realiza un pre-procesamiento, mediante el cual se realiza un escalado dentro de un rango numérico, a este proceso se le denomina normalización y se realiza solo al inicio del proceso, al finalizar el proceso se hace la operación inversa para tener la serie original. [S. Panigrahi et. al 2013] [González, B. P et al 2012]

La normalización de datos utilizada es la siguiente

$$N_i = Min_N + \frac{T_i - Min_T}{Max_T - Min_T} * (Max_N - Min_N) \quad (1)$$

Donde:

N_i es el elemento Normalizado en el momento i
Min_N es el valor mínimo que existirá en nuestra serie al normalizarla
Max_N es el valor máximo que existirá en nuestra serie al normalizarla
Max_T es el valor que existe actualmente en la serie de tiempo T
Min_T es el valor mínimo que existe actualmente en la serie de tiempo T
T_i es el valor actual de la serie T en el momento i que será normalizado

$$T_i = \left(\frac{(N_i - Min_N)}{Max_N - Min_N}\right) * (Max_T - Min_T) + Min_T \quad (2)$$

Esta fórmula produce un valor en el rango de [0,1] para cada elemento de la serie de tiempo

2.3 Métodos de pronóstico

Pare el pronóstico de series de tiempo, existen los métodos clásicos y los métodos de computo inteligente, dentro de los métodos clásicos existen ARIMA, suavizamiento exponencial, medias móviles, además de estos existen los métodos de computo inteligente, estos incluyen redes neuronales, máquinas de soporte vectorial, algoritmos heurísticos, además existen hibridaciones la cuales pueden incluir por ejemplo el uso de redes neuronales con métodos clásicos de pronóstico ARIMA ANN, o SVR con ARIMA, SVR con algoritmos genéticos.

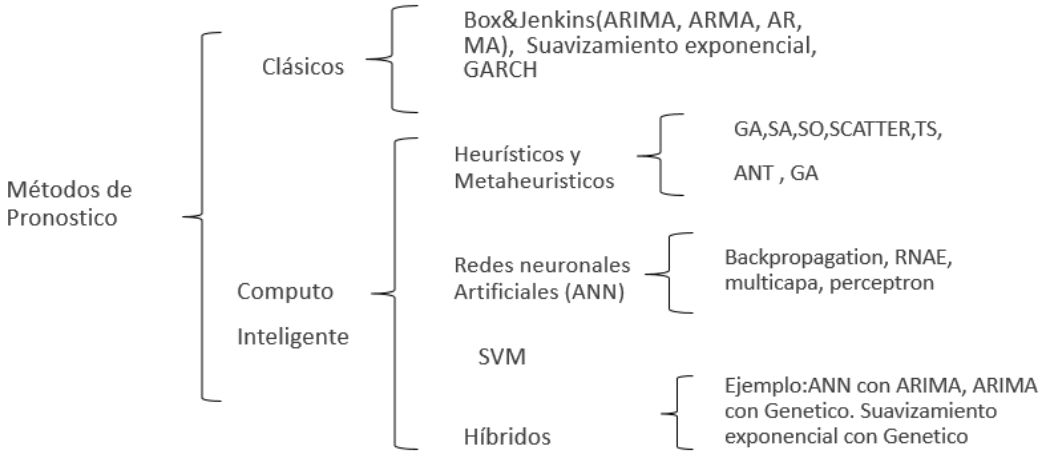


Figura 2 Diagrama de métodos de pronóstico

2.3.1 Medias móviles

El promedio móvil para el periodo de tiempo t es la media aritmética de las k observaciones más recientes. En un promedio móvil, se asignan pesos iguales a cada observación. Conforme está disponible, cada nuevo punto de datos se incluye en el promedio y el punto de datos más antiguo se descarta. El porcentaje de respuesta a los cambios en el patrón subyacente de datos depende del número de periodos, k, incluidos en el promedio móvil. [Hanke,2010].

$$\hat{Y}_{t+1} = \frac{Y_t + Y_{t-1} + Y_{t-2} + \cdots + Y_{t-k+1}}{k} \quad (3)$$

Donde:

$\hat{Y}_{t+1}$ = nuevo valor pronosticado en el periodo t

k = número de periodos
Y_t = el valor real en el periodo *t*

2.3.2 Suavizamiento exponencial

El suavizamiento exponencial revisa continuamente un estimado a la luz de las experiencias más recientes. Este método se basa en promediar (suavizar) valores pasados de una serie de manera exponencialmente decreciente. La observación más reciente recibe el peso más grande, *a* (donde 0 < *a* < 1); la siguiente observación más reciente recibe menos peso, a(l - *a*); la observación de dos periodos en el pasado recibe incluso menos peso, *a*(l - *a*)2; y así sucesivamente[Hanke,2010].

Nuevo pronóstico = [α X (nueva observación)] + [(1 - α) X (último pronóstico)]

$$\hat{Y}_t = \alpha Y_{t-1} + (1-\alpha)\hat{Y}_{t-1} \quad (4)$$

Donde:

$\hat{Y}_t$ = nuevo valor suavizado o el valor del pronóstico para el periodo *t*
α = constante de suavizamiento (0< α <1)
Y_{t-1} = nueva observación o el valor real de la serie de tiempo en el periodo *t -1*
$\hat{Y}_{t-1}$ = último valor suavizado o el pronóstico del periodo *t -1*

2.3.3 ARIMA

ARIMA[Box&Jenkins,2008] significa modelo auto regresivo integrado de medias móviles, este método de pronóstico fue propuesto por George E. P. Box y Gwilym M. Jenkins realiza un proceso iterativo que consta de cuatro pasos los cuales son: postular una clase general de modelos(AR, MA, ARMA, ARIMA), identificar el modelo que se utilizara de los mencionados anteriormente, se estiman los parámetros del modelo, se diagnostica el modelo si es adecuado o no y si es adecuado se utiliza este modelo para realizar los pronósticos[Box &Jenkins, 2008].

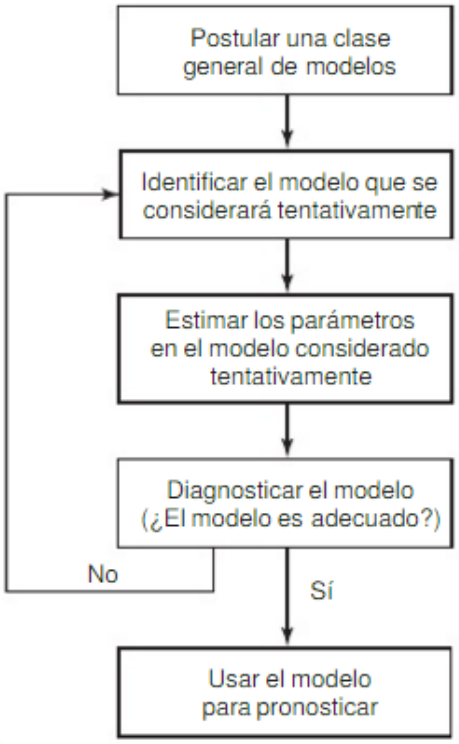


Figura 3 Metodología Box-Jenkins (Box &Jenkins 2008)

Un modelo ARIMA(p,0,0) se reduce a un modelo AR y su fórmula para calcular es

$$Y_t = \varphi_0 + \varphi_1 Y_{t-1} + \varphi_2 Y_{t-2} + \cdots + \varphi_p Y_{t-p} + \varepsilon_t \quad (5)$$

Donde:Y_t Es la variable dependiente en el tiempo, $\varphi_0, \varphi_1, \cdots, \varphi_n$ son los coeficientes a calcular,$Y_{t-1}, Y_{t-2}, \cdots, Y_{t-p}$, son las variables de retraso, ε_t es el término de error en el tiempo t.

Un modelo ARIMA(0,0,q) se reduce a un modelo MA y su fórmula es

$$Y_t = \mu + \varepsilon_t - \omega_1 \varepsilon_{t-1} + \omega_2 \varepsilon_{t-2} + \cdots + \omega_q \varepsilon_{t-q} \quad (6)$$

Donde: Y_t Es la variable dependiente en el tiempo, $\omega_1, \omega_2, \cdots, \omega_n$ son los coeficientes a calcular, μ es el promedio constante en el proceso $\varepsilon_{t-1}, \varepsilon_{t-2}, \cdots, \varepsilon_{t-p}$, , son los errores en periodos anteriores, ε_t es el término de error en el tiempo *t*.

Un modelo ARIMA(p,0,q) se reduce a un modelo ARMA y su fórmula es

$$Y_t = \varphi_0 + \varphi_1 Y_{t-1} + \varphi_2 Y_{t-2} + \cdots + \varphi_p Y_{t-p} + \varepsilon_t - \omega_1 \varepsilon_{t-1} + \omega_2 \varepsilon_{t-2} + \cdots + \omega_q \varepsilon_{t-q} \quad (7)$$

Donde: Y_t Es la variable dependiente en el tiempo, $\varphi_0, \varphi_1, \cdots, \varphi_n, \omega_1, \omega_2, \cdots, \omega_n$ son los coeficientes a calcular, $\varepsilon_{t-1}, \varepsilon_{t-2}, \cdots, \varepsilon_{t-p}$, , son los errores en periodos anteriores, ε_t es el término de error en el tiempo t

2.4 Medidas de error de pronóstico

Las medidas de error nos sirven para identificar que tan bueno es nuestro pronóstico, existen diferentes formas de medir estos errores las cuales se muestran a continuación.

Las diferentes formas de medir un error de pronóstico utilizan la diferencia entre el valor real y el valor pronosticado, la cual tiene por nombre residuo.

$$e_t = Y_t - \hat{Y}_t \tag{8}$$

Donde:

e_t es el error de pronosticar el periodo *t*.
Y_t es el valor real del periodo *t*.
$\hat{Y}_t$ es el valor pronosticado del periodo *t*.

2.4.1 Desviación media absoluta

Esta medida de error(MAD, por sus siglas en inglés) esta en las mismas unidades que la serie original, nos indica que tan alejado está el pronóstico con respecto a la serie original sin importar su dirección. La expresión MAD se presenta en la siguiente ecuación.

$$MAD = \frac{1}{n} \sum_{t=1}^{n} |Y_t - \hat{Y}_t| \tag{9}$$

Donde:

Y_t valor real del periodo t.
$\hat{Y}_t$ valor pronosticado del periodo t.
n número de elementos pronosticados.

2.4.2 Error cuadrático medio

Esta medida de error (MSE, por sus siglas en inglés) eleva al cuadrado la diferencia del valor real menos el pronosticado y al final lo promedia, esta medida castiga aquellas diferencias mayores, lo que significa que es preferible tener un pronostico con errores moderados a un pronóstico con errores mínimos que de vez en cuando produce errores extremadamente grandes. La expresión MSE se presenta en la siguiente ecuación.

$$MSE = \frac{1}{n} \sum_{t=1}^{n} (Y_t - \hat{Y}_t)^2 \tag{10}$$

Donde:

Y_t valor real del periodo t.
$\hat{Y}_t$ valor pronosticado del periodo t.
n número de elementos pronosticados.

2.4.3 Raíz cuadrada del error cuadrático medio

Esta medida de error(RMSE, por sus siglas en inglés) es similar al MSE, con la diferencia de que realiza la raiz cuadrada al finalizar el cálculo,esta medida tambien castiga las diferencias mayores, esta en las mismas unidades de la serie original por lo que es mas facil interpretar su valor. La expresión RMSE se presenta en la siguiente ecuación.

$$RMSE = \sqrt{\frac{1}{n}\sum_{t=1}^{n}(Y_t - \hat{Y}_t)^2} \tag{11}$$

Donde:

Y_t valor real del periodo *t*.
$\hat{Y}_t$ valor pronosticado del periodo *t*.
n número de elementos pronosticados.

2.4.4 Error porcentual absoluto medio

Esta medida (MAPE, por sus siglas en inglés) divide la diferencia absoluta entre el valor real menos el pronosticado entre el valor absoluto del valor real, lo promedia en los n periodos y después lo multiplica por 100. Esta medida calcula el porcentaje de error del pronóstico sin importar la dirección; esta medida es útil cuando los valores reales son muy grandes. Este valor no puede calcularse cuando el valor real de cualquier periodo es cero. La expresión RMSE se presenta en la siguiente ecuación. La expresión MAPE se presenta en la siguiente ecuación.

$$MAPE = \frac{100}{n}\sum_{t=1}^{n}\frac{|Y_t - \hat{Y}_t|}{|Y_t|} \tag{12}$$

Donde:

Y_t valor real del periodo t.
$\hat{Y}_t$ valor pronosticado del periodo t.
n número de elementos pronosticados.

2.4.5 Error porcentual medio

Esta medida(MPE, por sus siglas en inglés) obtiene un promedio de *n* periodos de la diferencia del valor real menos el valor pronosticado entre el valor real y al final se obtiene el porcentaje. Un valor resultante muy grande ya sea positivo o negativo indica que se esta sobreestimando o subestimando consistentemente los valores ya que no utiliza el valor asoluto para calcularse. De la misma manera que el MAPE el MPE no puede calcularse si el valor real de cualquier periodo es cero. La expresión MPE se presenta en la siguiente ecuación.

$$MPE = \frac{100}{n}\sum_{t=1}^{n}\frac{(Y_t - \hat{Y}_t)}{Y_t} \tag{13}$$

Donde:

Y_t valor real del periodo *t*.

$\hat{Y}_t$ valor pronosticado del periodo *t*.
n número de elementos pronosticados.

Con estas medidas de error es posible comparar dos o más métodos de pronóstico distintos con el fin de elegir el mejor. Por otro lado, estas medidas permiten medir la confiabilidad de alguna técnica utilizada.

2.5 Redes Neuronales

2.5.1 Neurona Artificial

Las neuronas son células que forman parte del sistema nervioso, las cuales en conjunto forman un sistema llamado red neuronal. Estas unidades están constituidas por dendritas, núcleo y axón.

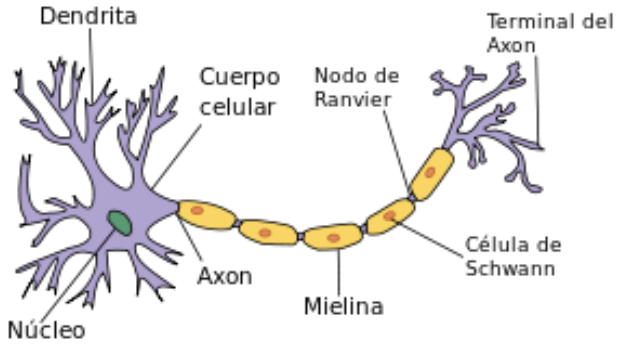


Figura 4 Partes de una neurona biológica

Las conexiones neurona a neurona se forma en las dendritas. A esta conexión se le denomina sinapsis. Las dendritas son las entradas que recibe la neurona, la función de las dendritas es recibir los impulsos de otras neuronas y enviarlos hasta el cuerpo celular o soma para posteriormente emitir una respuesta por medio de las ramificaciones del axón. Las señales recibidas por las dendritas pueden ser de excitación o de inhibición. Las primeras provocan que la neurona genere un impulso eléctrico, y las segundas detienen un impulso eléctrico. Entonces las funciones básicas de una neurona son: recibir información, procesarla y posteriormente comunicar señales.

2.5.2 Redes neuronales artificiales

La red neuronal artificial es un grupo interconectado de neuronas artificiales que utiliza un modelo matemático o modelo computacional para el procesamiento de la información basado en un enfoque conexionista a la computación. Las redes neuronales artificiales están hechas de neuronas artificiales de interconexión que pueden compartir algunas propiedades de las redes neuronales biológicas. [Chakraborty, 2010].

Las redes neuronales artificiales son redes masivamente paralelas e interconectadas de elementos simples (generalmente adaptativos) y sus organizaciones jerárquicas que tienen la intención de interactuar con los objetos del mundo real de la misma manera que lo hacen los sistemas nerviosos biológicos. [Kohonen,1988].

Una red neuronal es un triple ordenado (N, V, w) con dos conjuntos N, V y una función w, donde N es el conjunto de neuronas y V un conjunto $\{(i, j) \mid i, j \, N \in \mathbb{N}\}$ cuyos elementos son llamadas conexiones entre la neurona i y la neurona j. La función $w: V \rightarrow \mathbb{R}$ define los pesos, donde $w\,((i, j))$, el peso de la conexión entre la neurona i y la neurona j, se acorta a w_{ij}. Dependiendo del punto de vista, es indefinido o 0 para conexiones que no existen en la red[Kriesel, 2007].

Las Redes Neuronales Artificiales surgieron originalmente como una simulación abstracta de los sistemas nerviosos biológicos, constituidos por un conjunto de unidades llamadas neuronas conectadas unas con otras.

El primer modelo de red neuronal fue propuesto en 1943 por McCulloch y Pitts en términos de un modelo computacional de actividad nerviosa. Este modelo era un modelo binario donde cada neurona tenía un escalón o umbral prefijado, y sirvió de base para los modelos posteriores. Debido a las limitantes que había en su época, éstas comenzaron primero como redes de una sola capa y con el descubrimiento de los métodos de entrenamiento a partir de 1980 fueron evolucionando hasta las actuales redes [Zilouchian,Jamshidi, 2000].

Las redes neuronales artificiales están compuestas de muchos elementos simples que emulan diversas actividades cerebrales. Una de las principales motivaciones para introducir la RNA entre muchos investigadores ha sido la exploración y reproducción de tareas de procesamiento de información humana como el habla, la visión, el procesamiento del conocimiento y el control motor [Zilouchian,Jamshidi, 2000].

Considerando a la neurona artificial como la unidad básica de las RNA, se puede observar que existe una gran similitud entre la neurona biológica y la neurona artificial. Ambas tienen entradas, utilizan pesos y generan salidas.

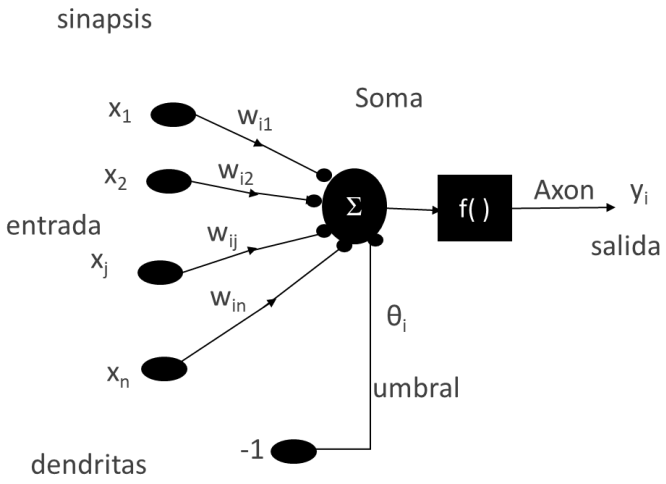


Figura 5 Ejemplo de una neurona artificial (Larranaga et al., 1997)

Una neurona consta de tres componentes básicos: pesos, umbrales y una sola función de activación [Chakraborty,2010]

La entrada de una neurona pasa por una regla de propagación, definida por el conjunto de entradas y los pesos sinápticos, los valores de los pesos sinápticos determinan la fuerza que tiene el vector de entrada [Larranaga et al., 1997], [Kriesel, 2007], [Chakraborty,2010]

La regla más común de propagación es la suma ponderada [Kriesel, 2007].

Suma ponderada: es la suma de todos los valores de entrada a la neurona, multiplicados por sus correspondientes pesos

$$\text{VE}=\sum_i (in_{ij} w_{ij}) \tag{14}$$

2.5.3 Funciones de activación

Las neuronas artificiales al igual que las biológicas pueden estar excitada (activada) o no excitada (no activada)

Sea *j* una neurona. El valor de umbral se asigna únicamente a *j* y marca la posición del valor de gradiente máximo de la función de activación [Kriesel,2007]

Las funciones de activación más comunes son:

función lineal

$$f(x) = x \tag{15}$$

función sigmoidea

$$f(x) = \frac{1}{1+ e^x} \tag{16}$$

función tangente hiperbólica

$$f(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} \quad (17)$$

función seno o coseno

$$f(x) = \sin x \quad or \quad f(x) = \cos x \quad (18)$$

De estas funciones la más popular es la sigmoidea[Kriesel,2007]

Funciones de salida: El último componente que una neurona necesita es la función de salida. El valor resultante de esta función es la salida de la neurona *i* (out_i); por ende, la función de salida determina qué valor se transfiere a las neuronas vinculadas. Si la función de activación está por debajo de un umbral determinado, ninguna salida se pasa a la neurona subsiguiente.

Dos de las funciones de salida más comunes son:

Ninguna: este es el tipo de función más sencillo, tal que la salida es la misma que la entrada. Es también llamada función identidad.

Binaria: donde vale 1 si el valor actual sobrepasa al umbral o de lo contrario vale 0.

2.5.4 Topología de una red neuronal

La topología o arquitectura de una red neuronal consiste en la organización y disposición de las neuronas en la misma, formando capas o agrupaciones de neuronas más o menos alejadas de la entrada y salida de dicha red. En este sentido, los parámetros fundamentales de la red son: el número de capas, el número de neuronas por capa, el grado de conectividad y el tipo de conexiones entre neuronas. [Matich,2001]

Existen redes monocapa y redes multicapa, en las redes monocapa significa que las neuronas están interconectadas en la única capa de la red

Las redes multicapa son redes que tienen una capa de entrada, una capa oculta y una capa de salida, la capa oculta puede conformarse por 1, 2, 3, …,ó hasta *n* capas

Estas dos posibilidades permiten distinguir entre dos tipos de redes con múltiples capas: las redes con conexiones hacia adelante o redes *feedforward,* y las redes que disponen de conexiones tanto hacia adelante como hacia atrás o redes feedforward/feedback.

El nombre de backpropagation resulta de la forma en que el error es propagado hacia atrás a través de la red neuronal, en otras palabras, el error se propaga hacia atrás desde la capa de salida. Esto permite que los pesos sobre las conexiones de las neuronas ubicadas en las capas ocultas cambien durante el entrenamiento.

2.6 Redes neuronales artificiales evolutivas

La computación evolutiva simula un proceso evolutivo en una computadora para generar buenas soluciones a un problema. El proceso atrae la inspiración de alto nivel de la evolución biológica. Inicialmente se genera una población de soluciones potenciales, y éstas son mejoradas iterativamente a lo largo de muchas generaciones simuladas. En iteraciones sucesivas del algoritmo, la selección basada en la aptitud tiene lugar dentro de la población de soluciones. Preferentemente se seleccionan mejores soluciones para sobrevivir en la siguiente generación de soluciones, introduciendo la diversidad en las soluciones seleccionadas en un intento de descubrir soluciones aún mejores a través de múltiples generaciones. Los algoritmos que emplean un enfoque evolutivo incluyen algoritmos genéticos (AGs), estrategias evolutivas (ES), programación evolutiva (EP) y programación genética (GP). La evolución diferencial (DE) también extrae la inspiración de los procesos evolutivos [Brabazon, O'Neill, McGarraghy, 2015]

Muchos investigadores entre 1950 y 1960 desarrollaron algoritmos inspirados en la evolución para resolver problemas de optimización y Aprendizaje de Máquina. Estos algoritmos fueron haciéndose más complejos y eficaces a medida que los recursos informáticos avanzaban. Aun así y a pesar de los importantes avances en el área, no fue sino hasta (1975) que Holland definió los principios básicos de los algoritmos genéticos. Estos se encuentran bien descritos por (Mitchell, 1999). De los desarrollos de esta época se pueden subrayar los avances en algoritmos genéticos, las estrategias evolutivas y la programación evolutiva[Henoa,2013]

Una red neuronal evolutiva es una red neuronal artificial que utiliza algoritmos evolutivos para hacer evolucionar la red, los tipos de evolución que se pueden presentar son evolución de pesos de entrada, evolución de topología de la red.

Un algoritmo evolutivo se puede utilizar para seleccionar de las opciones para cualquiera o todos de los siguientes: entradas de modelo, número de capas ocultas, número de nodos en cada capa oculta, naturaleza de las funciones de transferencia en cada nodo, estructura de conexión entre cada nodo, algoritmo de aprendizaje y parámetros asociados, o pesos de conexión entre cada nodo.

2.7 Algoritmos evolutivos

El algoritmo genético enfatiza la importancia de la cruza sexual sobre la mutación y usa selección probabilística. [Coello, 2004]

Un algoritmo genético consta de generar una población inicial, una función de evaluación la cual nos permite identificar la calidad de los individuos de la población, posteriormente realiza operadores genéticos los cuales pueden ser de cruza o mutación, y dependiendo el método de selección se eligen los hijos que conforman la siguiente generación de soluciones [Holland, 1992].

Los procesos básicos son los siguientes [Coello,2004]:

- Generar (aleatoriamente) una población inicial.
- Calcular aptitud de cada individuo.
- Seleccionar probabilísticamente en base a aptitud.
- Aplicar operadores genéticos (cruza y/o mutación) para generar la siguiente población.
- Ciclar hasta que cierta condición se satisfaga.

Para aplicar un algoritmo genético se requiere de una forma de representación de la solución del problema, una forma de crear una población inicial, una función de evaluación, operadores genéticos que alteren la composición de los hijos que se producirán y los valores para los diferentes parámetros que utiliza el algoritmo (tamaño de la población, probabilidad de cruza, probabilidad de mutación, número máximo de generaciones).

Capítulo 3

Estado del arte

3.1 Trabajos relacionados

Se realizó una búsqueda de trabajos de predicción de series de tiempo utilizando redes neuronales, encontrándose los siguientes trabajos que tienen relevancia para esta investigación.

3.1.1 Análisis y predicción de la serie de tiempo del precio externo del café colombiano utilizando redes neuronales artificiales

Este trabajo realiza un pronóstico utilizando la serie de tiempo del precio promedio mensual del café colombiano en la Bolsa de Valores de Nueva York, desde enero de 1913 hasta julio de 2001 utiliza una red neuronal Feed-forward con una capa oculta y con el algoritmo retropropagación, este trabajo demuestra que las redes neuronales son superiores a los métodos clásicos de pronóstico [Martín,2003]

3.1.2 Redes Neuronales Artificiales en predicción de series de tiempo. Una aplicación en la industria.

Se utilizan redes neuronales artificiales con el fin de realizar una predicción en dos series de tiempo, una de ellas es la generación mensual de electricidad total y la otra serie de tiempo utilizada es el consumo de gas natural ambas series con un periodo desde enero 2001 hasta junio 2007 y fueron tomadas de Estados Unidos, utiliza una red multicapa con algoritmos de aprendizaje *backpropagation y resilient propagation,* la medida de error utilizada es el MAPE [Escobar,2009].

3.1.3 Pronósticos mediante redes neuronales artificiales y modelos ARIMA:el caso de los CETES en México.

Morales y Torres realizan un pronóstico utilizando un modelo autoregresivo con una red neuronal, la topología de red es una red multicapa, para medir el error de pronóstico utilizan un algoritmo de retropropagación. La serie de tiempo utilizada para realizar este pronóstico fueron la serie de cotizaciones de cierre del futuro de los certificados de tesorería (cetes) a 91 días en el mercado mexicano de derivados (Mexder). Concluye que fue mejor la utilización de una red neuronal para el pronóstico comparándose contra el método ARIMA [Morales, 2008].

3.1.4 A Time series forecasting using a hybrid ARIMA and neural network model

Este trabajo realiza la hibridación de una red neuronal con el método ARIMA. El algoritmo híbrido consta de dos pasos; en el primer paso se utiliza un modelo ARIMA para analizar la parte lineal del problema, en el segundo paso se desarrolla un modelo de red neural para modelar los residuos del modelo ARIMA. Las series de tiempo utilizadas en este trabajo son los datos de las manchas solares que contienen el número anual de manchas solares de 1700 a 1987, dando un total de 288 observaciones, la serie de lince contiene el número de lince atrapado por año en el río Mackenzie Distrito del norte de Canadá con 114 observaciones y el tipo de cambio entre la libra esterlina y el dólar estadounidense. Las medidas de error utilizadas en este trabajo son MSE y MAD [Zhang,2003].

3.1.5 Parallelization of an Evolving Artificial Neural Networks System to Forecast Time Series using OPENMP and MPI

En este trabajo se presenta un enfoque evolutivo (EANN) para el pronóstico de serie de tiempo basado en el algoritmo de distribución de estimación (EDA) que evoluciona totalmente conectado., Utiliza 5 series temporales y paraleliza utilizando open MP y MPI. La métrica utilizada es el SMAPE y las series de tiempo utilizadas son: el número de pasajeros de una línea aérea internacional en millares; medidos mensualmente a partir de enero de 1949 hasta diciembre de 1960. La temperatura sobre la media mensual de la temperatura del aire medida en el castillo de Nottingham a partir de 1920 hasta 1939. Los cierres mensuales del índice industrial Dow-Jones desde agosto de 1968 hasta agosto de 1981. Las ventas de papel mensuales en Francia desde 1963 hasta 1972 y la demanda de gasolina en Ontario, en millones de galones, de 1960 a 1975. [González et al., 2012].

3.1.6 Evolutionary artificial neural networks for hydrological systems forecasting

Este artículo propone una red neuronal evolutiva para la predicción hidrológica de series de tiempo, se utilizó la serie de tiempo caótica Mackey-Glass muestra que el uso de una red neuronal evolutiva es eficiente, eficaz y robusta eficiencia, la medida de error utilizada es el RMSE la red evoluciona utilizando operadores genéticos [Chen, Chang,2009]

3.1.7 A hybrid neural network and ARIMA model for water quality time series prediction

En este estudio se propone un modelo híbrido de ARIMA y red neuronal que es capaz de explotar las fortalezas de los enfoques tradicionales de series temporales y redes neuronales artificiales. El enfoque propuesto consiste en una metodología ARIMA y una estructura de red de backpropagation feed-forward. El método híbrido para la predicción de series de tiempo se prueba usando observaciones de 108 meses de datos de calidad del agua,

incluyendo la temperatura del agua, boro y oxígeno disuelto, durante 1996-2004 en el río Büyük Menderes, Turquía. La medida de error utilizada son el MAPE, RMSE y el coeficiente Nash–Sutcliffe (Faruk,2010)

3.1.8 Stock index forecasting based on a hybrid model

Aquí se realiza la hibridación de una red neuronal retropropagación con el método ARIMA y suavizamiento exponencial, las medidas de error que utilizaron fueron MAE, RMSE, MAPE, ME y DA, los datos utilizados fueron el índice de cierre mensual Shenzhen Integrated Index (SZII) de China y el índice de apertura mensual Dow Jones Industrial Average Index (DJIAI) de los Estados Unidos. El conjunto de datos del índice de cierre de SZII abarca el período comprendido entre enero de 1993 y diciembre de 2010. En el conjunto de datos hay un total de 216 valores. El conjunto de datos de índice de apertura de DJIAI contiene precios de apertura de existencias de enero de 1991 a diciembre de 2010 y hay un total de 240 valores en el conjunto de datos (Wang, Wang y Zhang, 2012)

3.1.9 Evolving artificial neural networks for short term load forecasting

Este artículo presenta redes neuronales artificiales desarrolladas por un algoritmo genético para pronóstico de carga a corto plazo. Utilizando valores de carga real y datos meteorológicos pronosticados. Se observó que eran consistentemente superiores en comparación con un método estadístico comúnmente utilizado. La medida de error utilizada para la comparación es el MAPE (Srinivasan, 1998).

3.1.10 ARIMA forecasting of ambient air pollutants (O3, NO, NO2 and CO)

Este artículo utiliza el método ARIMA para pronosticar la concentración media diaria de contaminantes del aire ambiente en un sitio de tráfico urbano (ITO) de Delhi, India. Para estimar el modelo utilizan los criterios de evaluación AIC (Criterio de Información de Akaike), HIC (Criterio de Información de Hannon-Quinn), BIC (Criterio Bayesiano de Información) y FPE (Error Final de Predicción) además de la inspección ACF (función de autocorrelación) y PACF (función de autocorrelación parcial). Para evaluar el desempeño del modelo ARIMA(p,q) ARMA(p,d,q) seleccionados utilizaron las medidas de error se utilizaron el MAPE, el MAE y el RMSE, el objetivo fue adaptar un procedimiento de modelado que produjera resultados satisfactorios y viables para el pronóstico de contaminantes atmosféricos ambientales. (Kumar,Jain, 2010).

3.1.11 ARIMA forecasting of primary energy demand by fuel in Turkey

En este estudio se utilizaron los métodos ARIMA y ARIMA estacional para estimar la demanda futura de energía primaria de Turquía de 2005 a 2020, en este trabajo utiliza datos de series temporales de energía primaria total, incluyendo hulla, lignito, asfaltita, petrococo, madera, restos de animales y plantas, petróleo, gas natural, energía hidroeléctrica, calor geotérmico y electricidad y energía solar, para verificar la precisión del pronóstico realizado

utiliza el MSE. Este trabajo compara sus resultados con el Modelo para el Análisis de la Demanda de Energía (MAED). (Ediger & Akar,2007)

3.1.12 Utilización de modelos ARIMA para la vigilancia de enfermedades transmisibles

Este trabajo utiliza el modelo ARIMA para las series temporales de atenciones médicas por enfermedades diarreicas agudas (EDA) y atenciones médicas por enfermedades respiratorias agudas (IRA) en Cuba durante los años 1998-2004 con frecuencia semanal, la medida de error utilizada en esta serie fue el MAPE(Coutin,2007)

Tabla 1 Resumen de trabajos relacionados

Autor	Algoritmo	Evolución	Hibridación	Comp. Métodos Clásicos	Series de tiempo usadas	Medición de error
(Srinivasan,1998)	RNA	si	si	no	Temperatura mensual por hora para pronosticar las cargas eléctricas	MAPE
(Ediger,Akar,2007)	ARIMA	no	no	no	Demanda de energía en Turquía	MSE
(Kumar,Jain, 2010)	ARIMA	no	no	si	Nivel de contaminación del aire ((O3, NO, NO2 and CO)	MAPE, MAE, RMSE
(Coutin,2007)	ARIMA	no	no	si	Enfermedades diarreicas agudas, enfermedades respiratorias agudas	MAPE
(Wang, Wang y Zhang, 2012)	RNA	no	si	si	(SZII) de China, (DJIAI) de los Estados Unidos	MAE, RMSE, MAPE, ME y DA
(Faruk,2010)	RNA	no	si	si	calidad del agua midiendo temperatura, boro y oxígeno disuelto	RMSE, MAPE
(Chen, Chang,2009)	RNA	si	si	si	Datos generados para la serie de tiempo caótica Mackey Glass	RMSE
(Gonzalez, Sánchez, Donate, Cortez, de Miguel, 2012)	RNA	si	no	si	Pasajeros de una aerolínea, temperatura media en Nottingham , cierres mensuales Dow-Jones, venta de papel , demanda de gasolina en Ontario	SMAPE
(Zhang,2003)	RNA	no	si	si	Manchas solares, linces capturados en Canadá, tasa de cambio libra dólar	MSE, MAD
(Morales,2008)	RNA	no	no	si	CETES	RMSE, MAPE
(Escobar,2009)	RNA	no	no	no	Electricidad, gas	MAPE
(Martín,2003)	RNA	no	no	si	café colombiano en la Bolsa de Valores de Nueva York	MSE
Este proyecto	RNA	si	si	si	Makridakis y BMV	MAPE

Capítulo 4

Métodos de pronóstico

En este capítulo se describen métodos de pronóstico utilizados para realizar un pronóstico de series de tiempo utilizando redes neuronales

4.1 Red neuronal feedforward

Como primer acercamiento al pronóstico de redes neuronales se creó una red neuronal simple de 3 capas con estructura feedforward configurada de la siguiente manera:

- numero de capas:3
- capa de entrada: 4 Neuronas
- numero de capas ocultas:1
- numero de neuronas en la capa oculta:10 Neuronas
- numero de neuronas en capa de salida: 1 Neurona
- función de activación: sigmoidea

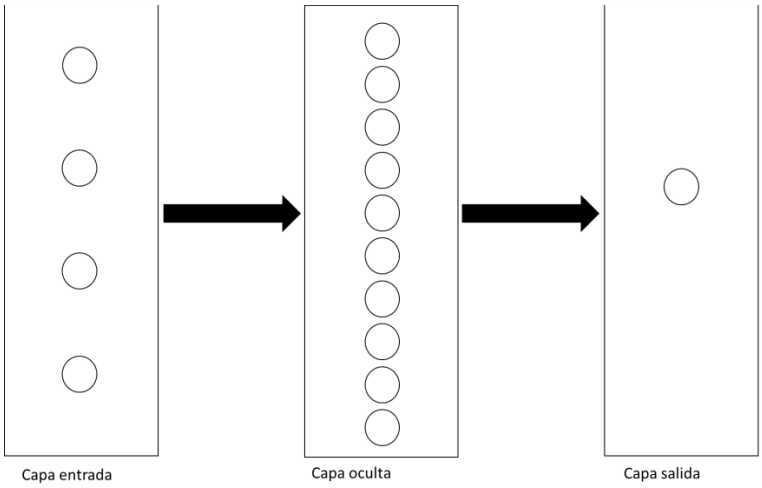


Figura 6 Red neuronal usada para pronóstico compuesta por 3 capas (entrada, oculta, salida)

La figura 6 muestra la red neuronal con pesos iniciales aleatorios para realizar pronósticos de series de tiempo. Este primer acercamiento no tiene entrenamiento y se utilizó para ejemplificar el uso de redes neuronales en pronóstico, con fines de simplificar el dibujo la flecha significa que cada neurona de la capa anterior está conectada con todas las neuronas de la capa siguiente, este tipo de red es una red feedforward completamente conectada.

4.2 Redes Neuronales Evolutivas

En esta sección se describen los métodos neuroevolutivos que se probaron para el pronóstico de series de tiempo

4.2.1 Evolución por backpropagation

En esta red desarrollada se utilizó la red neuronal de configuración inicial similar a la de la red neuronal que se creó como acercamiento inicial de redes neuronales para pronóstico

- numero de capas:3
- capa de entrada: 4 Neuronas
- numero de capas ocultas:1
- numero de neuronas en la capa oculta:10 Neuronas
- numero de neuronas en capa de salida: 1 Neurona
- función de activación: sigmoidea

Se ejecutó el entrenamiento de pesos por medio del algoritmo de backpropagation, en este tipo de evolución solo se evolucionaron los pesos para una red de topología fija con la configuración descrita anteriormente.

4.2.2 Evolución por fuerza bruta

En esta implementación se optó por crear una red neuronal de 3 capas (entrada, oculta, salida) como base para este tipo de evolución, comienza con 2 neuronas en la capa de entrada, 1 neurona en la capa oculta y 1 neurona en la capa de salida.

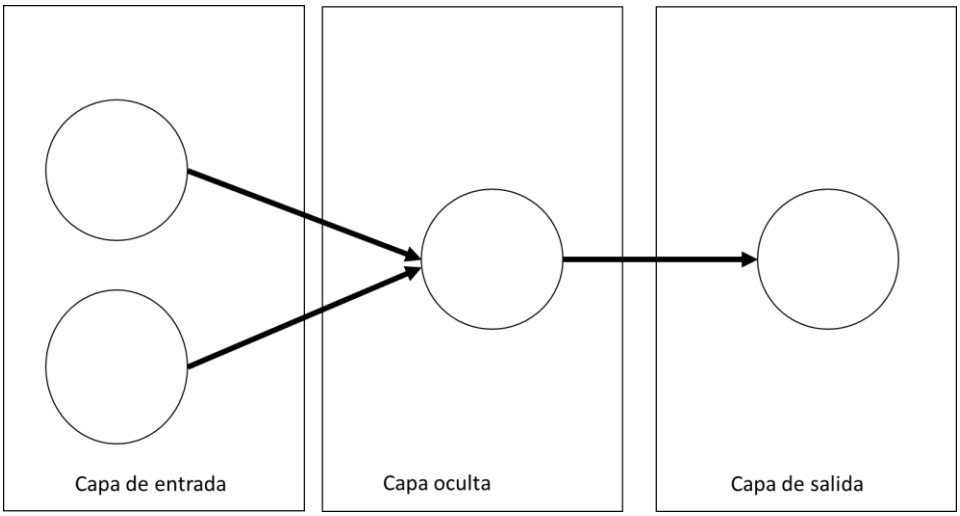


Figura 7 Topología inicial del algoritmo de evolución por fuerza bruta

En este tipo de evolución se utilizó el algoritmo backpropagation para entrenar los pesos de las redes, se utilización 2 ciclos para realizar la evolución topológica, en el primer ciclo se incrementa las neuronas de la capa de entrada, el segundo ciclo incrementa el número de neuronas de la capa oculta, dentro de este ciclo una nueva red es creada con el número de neuronas del valor actual del ciclo 1 para la capa de entrada y el número de neuronas ocultas del valor actual del ciclo2, esta red es posteriormente es entrenada por medio de backpropagation.

Se compara el MAPE de la nueva red contra el MAPE de la mejor red hasta el momento, si el valor MAPE de la nueva red es menor que el MAPE de la mejor red, la nueva red se almacena como la mejor red hasta el momento y el proceso continúa incrementando neuronas en la capa de entrada y neuronas en la capa oculta, para este proceso se definió un número no mayor a 20 neuronas en la capa de entrada y 40 neuronas en la capa oculta.

Se implementó una medida de convergencia en la cual, en caso de que la diferencia absoluta entre la mejor red y la mejor red actual sea menor que 0.00001, este valor se guarda en una variable que cuenta el número de no mejoras, si este valor es igual a 3 significa que aunque sigamos aumentando el número de neuronas en la red no mejorará y el proceso finaliza obteniendo la mejor red del proceso.

4.3 Redes Neuronales Evolutivas propuestas

En esta sección se describen los métodos neuroevolutivos propuestos, estos métodos usan algoritmo genético para modificar la arquitectura de la red (número de capas, número de neuronas por capa, conexiones entre neuronas), el último método utiliza el algoritmo recocido simulado para ajustar los pesos de las conexiones entre neuronas.

El objetivo del algoritmo genético aplicado en este trabajo es encontrar la mejor arquitectura para realizar pronósticos de series de tiempo, la función de aptitud es medida por medio del MAPE, y por medio de evolución el principal objetivo es la búsqueda de la mejor configuración de red neuronal que arroje el menor error de pronóstico medido del MAPE, medida recomendada para pronóstico de series de tiempo (Tsay,2005) (Hanke,2006).

Para representar el cromosoma de la red neuronal se utiliza un vector de conexiones con origen, destino, peso y estatus dentro de la conexión (habilitada o deshabilitada), este trabajo comienza con una población de redes conformada por redes de una misma topología, esto es, redes que se componen únicamente de 3 neuronas en la capa de entrada y solo una neurona en la capa de salida, esto es debido a que se realiza la predicción one step ahead.

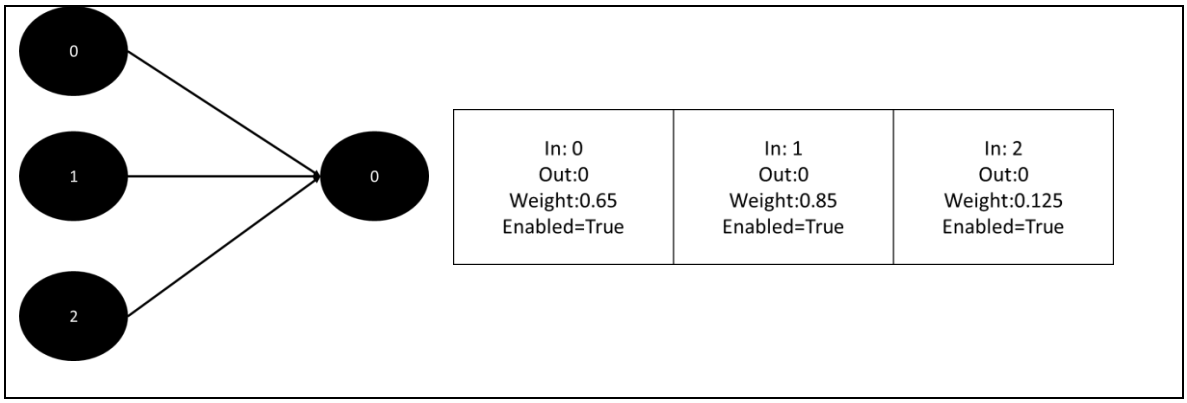


Figura 8 Ejemplo de representación de cromosoma para algoritmo genético

Otra restricción dentro de este trabajo es que todas las redes que se crean por medio de evolución deben conservar la estructura feedforward, esto quiere decir que no se crean redes recurrentes.

Los métodos descritos a continuación siguen la estructura básica de un algoritmo genético (Holland,1992).

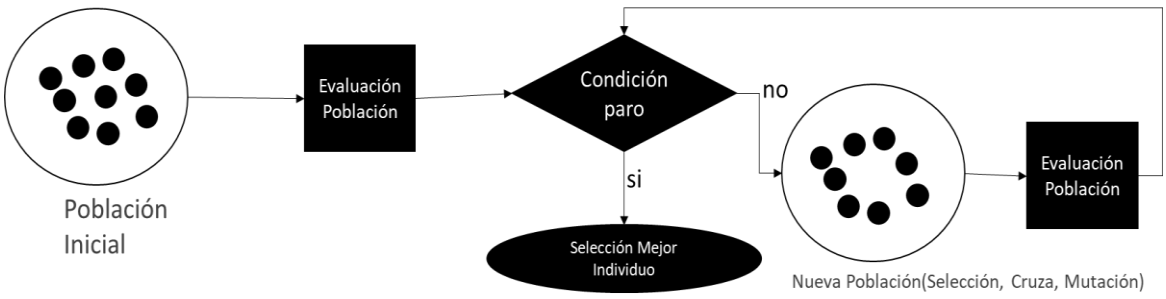


Figura 9 Diagrama básico de algoritmo genético (Holland,1992)

Antes de usar los algoritmos evolutivos en este trabajo los datos fueron pre procesados usando la normalización Min-Max

Debido a que la función de activación de estas redes es la sigmoide, la normalización usada para transformar la serie en un rango de 0 a 1 los valores para Max_N y Min_N fueron asignados como 0.1 y 0.9 respectivamente.

4.3.1 Algoritmo de mutación feedforward (FFM)

Este algoritmo se implementó con el fin de evolucionar las redes neuronales solo por mutación, esto quiere decir que no existe cruza, por lo cual se usa programación evolutiva (Fogel,1966).

La estructura del algoritmo es la siguiente:

	Algoritmo FFM
1:	Inicializar población *p*
2:	Evaluar población *p*
3:	Repetir
4:	Ordenar población
5:	Seleccionar el mejor individuo *b*
6:	Repetir
7:	Reemplazar elemento de la población mutando mejor individuo *b*
8:	Hasta número de población
9:	Evaluar nueva población
10:	Hasta número de generaciones
11:	Ordenar población
12:	Seleccionar mejor individuo

En el paso 1 se inicializa la población de redes de topología básica de 100 elementos, esto quiere decir que se crearon 100 redes neuronales con 3 neuronas en la capa de entrada 1 neurona en la capa de salida y sin capa oculta.

En el paso 2 se evalúa la población inicial calculando el MAPE

En cada iteración del ciclo del paso 3 se realiza un ordenamiento de la población de redes de menor a mayor en el paso 4, esto quiere decir que la red con un valor cercano a cero tiene mejor desempeño pronosticando que una red con un valor mucho mayor, posteriormente en el paso 5 se selecciona ese individuo que está en la posición número 1 de la población.

En cada iteración del paso 6 se reemplaza un elemento de la población por el individuo que se obtiene aplicando una mutación al mejor individuo de la generación anterior en el paso 7; esta se selecciona aleatoriamente de las 4 posibles mutaciones. Los tipos de mutaciones posibles son las siguientes:

- agregar neurona
- agregar o borrar una conexión
- agregar una capa
- perturbar peso

El proceso *Agregar neurona* consiste en seleccionar aleatoriamente una de las capas existentes de la red, ya sea de entrada u ocultas y se agrega una neurona con conexiones hacia las capas adyacentes, con lo cual se busca preservar una estructura de tipo *feedforward*.

En el proceso *Agregar o borrar una conexión* se selecciona aleatoriamente una neurona de entrada y una de salida, si esta conexión existe se deshabilita, en caso de no existir se agrega.

En el proceso Agregar una capa se aplica una mutación que agrega una capa a la red neuronal con una neurona completamente conectada con su capa adyacente. La capa nueva es agregada al final y esta será nuestra nueva neurona de salida.

En el proceso *Perturbar un peso* se realiza una mutación en la cual se selecciona aleatoriamente una capa y posteriormente se selecciona de igual manera en forma aleatoria una conexión; la cual es modificada añadiendo un valor. Se genera un nuevo número aleatorio entre cero y uno, el cual definirá si se sumara o sustraerá un valor aleatorio entre cero y uno al peso de la conexión. Si el valor es menor a 0.5 se resta y si es mayor a 0.5 se suma. Enseguida se genera el nuevo aleatorio que se añadirá o sustraerá al peso actual de la conexión.

La operación de reemplazo se repite hasta que se llega al número de elementos de la población.

Al finalizar la mutación de elementos, la nueva población es evaluada calculando el MAPE para cada individuo.

Finalmente, en el paso 11 se ordena una vez más la población de menor a mayor y en el paso 12 se selecciona al mejor individuo de toda la población. Esta es la mejor red neuronal para realizar el pronóstico y se obtuvo mediante la evolución tanto de los pesos como de la topología de las redes.

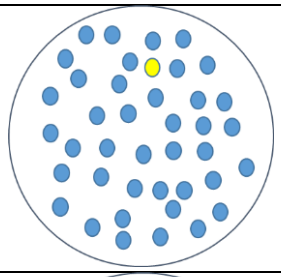
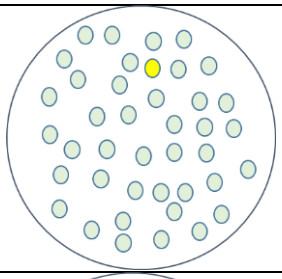
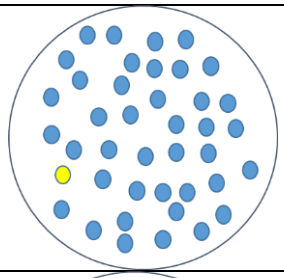
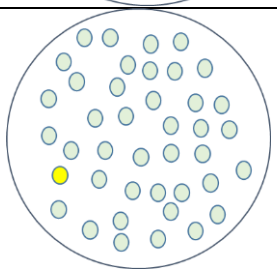
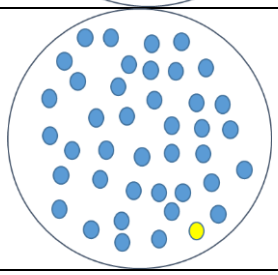
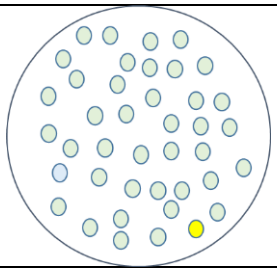
Generación	Poblacion	Nueva poblacion	Mape del mejor individuo
1			45.1
55			29.1
N			14.5

Figura 10 Ejemplo de evolución por mutación

En la figura 8 se muestra un esquema que ejemplifica la evolución de redes durante la ejecución del algoritmo. Se muestra en color amarillo el elemento con el menor MAPE y en un color más claro, las mutaciones del mejor elemento de la población para la generación actual, este proceso se realiza sin cruza.

4.3.2 Algoritmo genético para redes neuronales evolutivas (GAENN)

Este algoritmo realiza la evolución de redes neuronales por medio del uso de la estructura clásica de un algoritmo genético (Holland,1992), con la particularidad de que en el algoritmo propuesto en este trabajo por cada pareja de padres se genera un único descendiente. De manera que la nueva generación se obtiene mediante la competencia de acuerdo al *fitness* de la totalidad de la población formada por padres y descendientes.

La estructura del algoritmo es la siguiente:

	Algoritmo GAENN
1:	Inicializar población *p*
2:	Evaluar población
3:	Repetir
4:	Generar *k* hijos de *p* padres
5:	Evaluar hijos
6:	Reemplazar población con padres e hijos
7:	Hasta número de generaciones
8:	Ordenar población
9:	Seleccionar el mejor individuo

En este algoritmo es inicializada de igual manera que en el algoritmo anterior una población de 100 individuos con topología básica. Para lo cual se crean 100 redes neuronales con 3 neuronas en la capa de entrada, 1 neurona en la capa de salida y sin capa oculta. Esta población inicial fue evaluada por medio del MAPE en el paso 2.

El paso 3 implementa el proceso para obtener los elementos de la siguiente generación como se indica enseguida:

1. La población de la generación actual es desordenada y compiten en pares, mediante torneo binario, para lo cual se comparan en forma ordenada por parejas de redes con el fin de seleccionar 50 padres con el menor MAPE por pareja para formar el conjunto A de padres.
2. Se realiza un segundo desordenamiento de la población y se aplica el mismo torneo binario descrito en el paso 2 para seleccionar los padres que formaran el conjunto B de padres.
3. Una vez seleccionados los padres se procede a la operación de cruza entre padre A_i con padre B_i donde i=1…50 de forma que se obtienen 50 hijos.

En el paso 4 se realiza la operación de cruza. La operación de cruza consiste en comparar las conexiones de cada una de las redes, y cuando existen conexiones con el mismo origen y destino se selecciona al azar una de esas 2 conexiones para formar parte del cromosoma hijo como se ve en la figura 11, cuando una conexión solo exista en una de las dos redes entonces pasa directamente al cromosoma hijo como se ve en la figura 13.

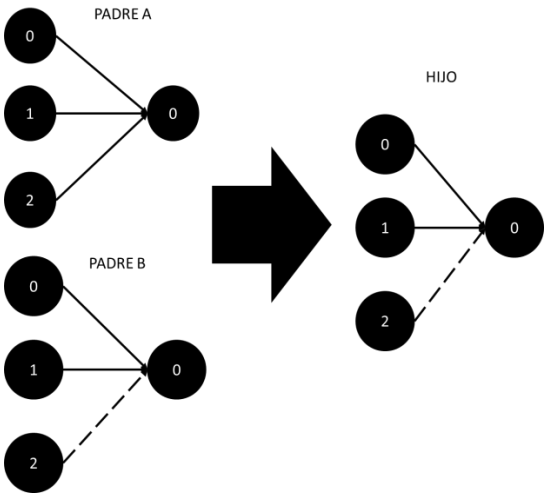


Figura 11 Ejemplo de cruza de redes de topología similar

PADRE A

Entrada: 0 Salida:0 Peso:0.65 Activada=Si	Entrada: 1 Salida:0 Peso:0.85 Activada=Si	Entrada: 2 Salida:0 Peso:0.125 Activada=Si

PADRE B

Entrada: 0 Salida:0 Peso:0.5 Activada=Si	Entrada: 1 Salida:0 Peso:0.333 Activada=Si	Entrada: 2 Salida:0 Peso:0.47 Activada=No

HIJO

Entrada: 0 Salida:0 Peso:0.65 Activada=Si	Entrada: 1 Salida:0 Peso:0.333 Activada=Si	Entrada: 2 Salida:0 Peso:0.47 Activada=No

Figura 12 Ejemplo de cruza de redes de topología similar representada por vector de conexiones

En las figuras 11 y 12 se pueden observar ejemplos de cruza utilizando representación de grafo y cromosómica respectivamente; donde los padres A y B tienen la misma topología con tres neuronas en la capa de entrada, y una neurona en la capa de salida. En la figura 12, el cromosoma en color blanco es el padre A y en color negro el padre B, como se puede observar el cromosoma hijo hereda la primera conexión del padre A y las otras dos conexiones del padre B. Este proceso se realiza para generar las conexiones en el hijo seleccionando de cada una de las conexiones en los padres aquellas que comparten misma entrada y salida en forma aleatoria una de las 2 hasta llegar al final de la lista de conexiones; sin importar si la conexión está habilitada o deshabilitada.

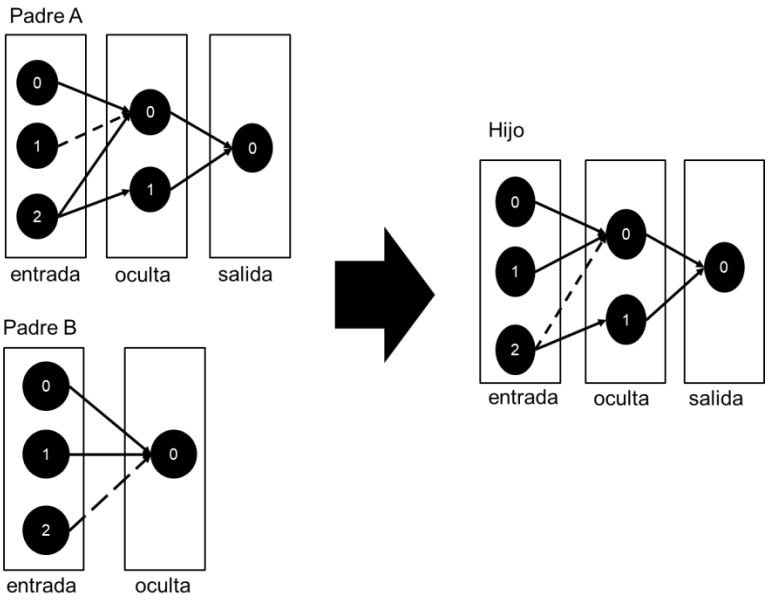


Figura 13 Ejemplo de cruza de redes de topología diferente

	Conexiones de capa de entrada a capa oculta				Conexiones de capa oculta a salida	
Padre A	Entrada: 0 Salida:0 Peso:0.474 Activa=Si	Entrada: 1 Salida:0 Peso:0.632 Activa=No	Entrada: 2 Salida:0 Peso:0.47 Activa=Si	Entrada: 2 Salida:1 Peso:0.11 Activa=Si	Entrada: 0 Salida:0 Peso:0.554 Activa=Si	Entrada: 1 Salida:0 Peso:0.13 Activa=Si

	Conexiones de capa de entrada a capa oculta		
Padre B	Entrada: 0 Salida:0 Peso:0.5 Activa=Si	Entrada: 1 Salida:0 Peso:0.333 Activa=Si	Entrada: 2 Salida:0 Peso:0.7 Activa=No

	Conexiones de capa de entrada a capa oculta				Conexiones de capa oculta a salida	
Hijo	Entrada: 0 Salida:0 Peso:0.474 Activa=Si	Entrada: 1 Salida:0 Peso:0.333 Activa=Si	Entrada: 2 Salida:0 Peso:0.7 Activa=No	Entrada: 2 Salida:1 Peso:0.11 Activa=Si	Entrada: 0 Salida:0 Peso:0.554 Activa=Si	Entrada: 1 Salida:0 Peso:0.13 Activa=Si

Figura 14 Ejemplo de cruza de redes de topología diferente representada por vector de conexiones

En las figuras 13 y 14 se presentan ejemplos de cruza entre redes de topología diferente utilizando la representación de grafo y cromosómica respectivamente, el padre A se compone de 3 neuronas en la capa de entrada, 2 neuronas en la capa oculta y una neurona en la capa de salida, el padre B se conforma de 3 neuronas de capa de entrada y una neurona en la capa de salida sin capa oculta, para realizar este tipo de cruza, de los dos individuos a cruzar, se asocia en un primer paso una jerarquía con base en el número de capas, la red que tenga el mayor número de capas es la que tiene mayor jerarquía y por lo tanto ese es el número de capas que tendrá el hijo, un segundo criterio que se considera analiza el número de neuronas por capas. El análisis se realiza considerando en primer la jerarquía por capas, entonces el individuo con mayor jerarquía determina el número de capas del hijo y además se

utiliza como referencia para el análisis de conexiones por capa iniciando desde la capa 0 hasta la capa de mayor nivel. Como segundo criterio una vez que se está analizando las capas, se aplica un segundo criterio para ver cuál de los 2 padres tiene mayor número de neuronas en la correspondiente capa, el máximo número de neuronas de los 2 padres determinará el número de neuronas en la capa correspondiente en el hijo. El análisis por capa consiste en identificar en primer término las conexiones que tengan mismo origen y mismo destino en ambos individuos como conexiones comunes. Para cada conexión común en ambos individuos selecciona una en forma aleatoria para pasar a formar parte del cromosoma hijo. Por otro lado, las conexiones no comunes de ambos padres pasan directamente al hijo.

En el ejemplo que se muestra en las figuras 13 y 14 caso se puede observar que las conexiones de los padres A y B que compiten para formar parte del cromosoma hijo son: del padre A, conexiones 0 a 0, 1 a 0, y 2 a 0; en este caso se debe notar que la conexión 2 a 1 no existe en padre B por lo que esta conexión pasa directamente al cromosoma hijo. Este mismo proceso se realiza con las conexiones de las siguientes capas. Debido a que el padre A tiene una capa más que padre B, todas las conexiones de la capa no común pasan directamente al cromosoma hijo.

Al finalizar el proceso de cruza se aplica mutación. Para cada red resultante se lanza un numero aleatorio, si este número es menor al 10% se aplica una de las 4 estrategias de mutación que se describen en la tabla 2. Para seleccionar el tipo de mutación a aplicar, se obtiene un segundo número aleatorio. Si el valor es menor a 0.5 se aplica mutación tipo 1, si el valor está dentro del rango 0.5 y menor a 0.7 se aplica 2, si el valor es mayor o igual a 0.7 y menor a 0.98 se aplica 3, y si es mayor a 0.98 se aplica 4.

Tabla 2 Distribución de las probabilidades de mutación de red neuronal

Numero	Mutación	Probabilidad
1	Perturbar peso	50%
2	Agregar/eliminar conexión	20%
3	Agregar Neurona	28%
4	Agregar Capa	2%
	Total	100%

Como podemos observar en la tabla 2, existe más probabilidad de perturbar un peso que agregar capas a la red, esto tiene la finalidad de evitar la creación de redes con muchas capas ocultas y de esta manera evitar el fenómeno de sobre entrenamiento.

En el paso 5 se evalúan los hijos, y se ordena la población de padres e hijos en forma ascendente. Este proceso descrito en los pasos 1 a 5 se repite hasta finalizar el número de generaciones. Finalmente se realiza un ordenamiento ascendente para seleccionar el mejor elemento, es decir la red con el menor MAPE de todas las generaciones.

4.3.3 Algoritmo genético con recocido Simulado para redes neuronales evolutivas (GASAENN)

En este caso se trata de un algoritmo genético que evoluciona topologías utilizando un algoritmo genético clásico, sin embargo, este algoritmo tiene la particularidad de que después de aplicar el proceso de cruza de la misma manera que el algoritmo GAENN se aplica un proceso de ajuste de pesos utilizando un algoritmo recocido simulado. La estructura de este algoritmo es la siguiente:

	Algoritmo GASAENN
1:	Inicializar población *p*
2:	Evaluar población
3:	Repetir
4:	Generar *k* hijos de *p* padres
5:	Ajustar Pesos
6:	Evaluar hijos
7:	Reemplazar población con padres e hijos
8:	Hasta número de generaciones
9:	Ordenar población
10:	Seleccionar el mejor individuo

De igual forma que GAENN, el algoritmo GASAENN inicializa redes con una topología de 3 neuronas de capa de entrada y una neurona de capa de salida. El MAPE constituye el fitness de estas redes. La selección, la cruza y la mutación se implementan de la misma manera que el algoritmo GAENN, posteriormente a la mutación se aplica un proceso de ajuste de pesos en el paso 5 mediante un algoritmo de tipo recocido simulado.

Para el ajuste de pesos se utilizó el algoritmo recocido simulado (Kirkpatrick et al,1983), en el cual se utilizó una temperatura inicial de 100 grados, α=0.95 y un ciclo de metrópolis de longitud 10; para la función de decremento de temperatura se utiliza la función geométrica:

$$tem = \alpha * temp \tag{19}$$

En este algoritmo se siguen los pasos del algoritmo clásico de Kirkpatrick, es decir se tiene el ciclo de temperatura y el ciclo de metrópolis. La solución obtenida después haber efectuado la cruza de los padres y haber mutado al hijo será definida como la solución inicial del ciclo de temperatura, y es la primera solución que será perturbada dentro del ciclo de metrópolis.

En el ciclo de metrópolis se obtiene la solución vecina de la solución actual realizando una perturbación aleatoria de pesos. En primer término se selecciona aleatoriamente una capa, posteriormente se selecciona aleatoriamente un número para decidir si se va a perturbar con adición o sustracción al peso de una conexión de dicha capa, cabe mencionar que la conexión también se elige aleatoriamente generando un número entre 1 y el número de conexiones existentes en la capa. Se describe el proceso de ajuste con recocido simulado en el algoritmo siguiente:

Algoritmo recocido simulado

```
1:  Inicializar best=current=hijo, temp=100, coolingRate=0.95, tempF=0.3
2:  Repetir
3:      Repetir
4:          Perturbar current
5:          si candidato<current
                current = current
            Sino
                Aceptar candidato si probabilidad Boltzman
6:      Hasta 10
7:      Si current<best
8:          best=current
9:      temp = temp * coolingRate
10: Mientras (temperatura>tempF)
11: retornar best
```

El paso 1 del algoritmo inicializa los parámetros de temperatura inicial, final y la tasa de enfriamiento. El ciclo de temperatura será controlado mediante la temperatura final establecida al inicio del proceso. Mientras que el ciclo interno denominado ciclo de metrópolis es controlado por un número límite de iteraciones que en este caso es 10 y se obtuvo mediante prueba y error para mantener tiempos de ejecución aceptables.

En el ciclo de metrópolis se implementa el ajuste de pesos, donde la solución inicial es el vector de pesos del hijo, cada vecino que se explora en una iteración del ciclo de metrópolis se obtiene aplicando una perturbación aleatoria de pesos, si la nueva solución es mejor que la solución actual, esta nueva solución es aceptada, si no, de manera clásica en SA se analiza la posibilidad de aceptarla utilizando la probabilidad de Boltzman. Al salir del ciclo de metrópolis se actualiza *mejor solución y* se decrementa la temperatura utilizando la tasa de decremento correspondiente. De esta manera al terminar la última iteración del ciclo de temperatura se tiene la *mejor solución* con el menor MAPE

Capítulo 5

Experimentación

En esta sección se detallan las condiciones de experimentación y se muestran los resultados experimentales.

5.1 Equipo utilizado

La evaluación experimental fue realizada en una computadora con arquitectura x64, procesador AMD A10-8700 Radeon R6, 10 Compute cores 4C+6G a 1.8 GHz y 12 GB en RAM. El sistema operativo utilizado fue Windows 10. El lenguaje utilizado para implementar los algoritmos fue Java utilizando el jdk 9 y el IDE de NetBeans.

5.2 Instancias

En1969, Reid (1969, 1975) y Newbold y Granger (1974) compararon un gran número de series para determinar su exactitud de pronóstico post-muestra. Sin embargo, estos primeros estudios de precisión basaron sus comparaciones en un número limitado de métodos. Makridakis y Hibon (1979) fue el primer esfuerzo para comparar un gran número de métodos de series de tiempo mayores a través de series múltiples. (Makridakis,2000).

M-Competition contiene un conjunto de 3003 de series temporales (micro industria, macro, etc.) y diferentes intervalos de tiempo entre observaciones sucesivas (anual, trimestral, etc.). Con el fin de garantizar que se tengan datos suficientes para elaborar un modelo de predicción adecuado, estableció un número mínimo de observaciones. Este mínimo se estableció como 14 observaciones para las series anuales (el promedio para 645 series anuales es de 19 observaciones), 16 para los trimestres (el promedio de las 756 series trimestrales es de 44 observaciones), 48 para el promedio para la serie mensual de 1428 series es de 115 observaciones) y 60 para la categoría de serie otras (la longitud promedio de las 174 series de la categoría otras son de 63 observaciones) (Makridakis,2000).

Las series utilizadas fueron las siguientes:

Tabla 3 Series utilizadas de Makridakis M3

Series	
N1402	N2525
N1403	N2526
N1404	N2527
N1405	N2528
N1406	N2529
N1407	N2530
N1408	N2531
N1409	N2532
N2522	N2533
N2523	N2534
N2524	

Adicionalmente se realizaron pruebas de series de la bolsa mexicana de valores, las series fueron tomadas de yahoo finance y se utilizaron los datos de cierre(close), del 1 enero del 2000 al 8 de agosto del 2016 para LIVERPOOL y CEMEX, para FEMSA se tomaron los datos del 6 de enero 2003 al 4 de abril 2013, para estas series de tiempo se tomaron el 90% para entrenamiento y el 10% para validación.

Se realizaron también pruebas para las series de la competencia Makridakis M4, esta competencia se conforma de 100,000 series de tiempo de diferentes sectores las cuales se dividen en intervalos anuales, cuatrimestrales, mensuales, semanales, diarios y por hora, para los cuales era necesario pronosticar 6, 8, 18, 13, 14, 48 valores correspondientemente, en esta experimentación solo se seleccionaron un total de 17 series de tiempo de maneara aleatoria del intervalo diario para las cuales era necesario realizar un pronóstico de 14 puntos de las series de tiempo, las series seleccionadas fueron las siguientes:

Tabla 4 Series utilizadas de M4

Series	
D2061	D2188
D2085	D2206
D2088	D2209
D2092	D2211
D2105	D2214
D2140	D2228
D2168	D2371
D2186	D3299
D2187	

5.3 Experimentación

Tabla 5 Métodos seleccionados para comparación de competencia Makridakis M3

Método	Competidor	Descripción
NAIVE2	M. Hibon	Deseasonalized Naive(Random Walk)
B-J auto	M. Hibon	Metodología Box Jenkins de bussiness forecast system.
ARARMA	N. Meade	Metodología de Parzen automatizada con filtro auto regresivo.
SMARTFCS	C. Smart	Sistema experto de pronóstico automático que realiza un torneo de pronóstico entre cuatro métodos de suavizado exponencial y dos de promedio móvil
THETAsm	V. Assimakopoulos	Suavizado sucesivo más un conjunto de reglas para amortiguar la tendencia
ForcX	J. Galt	Ejecuta la prueba de estacionalidad y valores atípicos y selecciona entre varios métodos: Suavizamiento exponencial, Box Jenkins y Cronston.
AAM1	G. Melard	Modelado automático de ARIMA con análisis de intervención.
AAM2	J.M. Pasteels	Modelado automático de ARIMA con análisis de intervención.

En la tabla 5 se muestran los 8 mejores métodos de 24 disponibles para realizar la comparación porque esos son los que obtuvieron un MAPE menor para esas series, todos los métodos se describen en el artículo de Makridakis del año 2000, la mayoría de los métodos son variantes de métodos clásicos.

Los resultados experimentales obtenidos por los métodos desarrollados se presentan en la tabla 6.

En esta tabla se pueden observar con subrayado y fondo gris los mejores resultados de los métodos de la competencia, esto quiere decir los resultados que obtuvieron el menor MAPE, en negrita se muestran los resultados de los métodos de pronóstico que obtuvieron el menor MAPE de toda la tabla, como puede observarse los 3 métodos desarrollados en este trabajo obtuvieron un MAPE menor que los métodos usados en la competencia en 16 de las 21 instancias utilizadas en M3.

Los resultados de la tabla 6 se compararon estadísticamente utilizando el test de Friedman Alineado donde la hipótesis nula asume que no hay diferencias entre las muestras

Tabla 6 Comparación de métodos con los algoritmos desarrollados

Serie	NAIVE2	B-J auto	ARARMA	SMARTFCS	THETAsm	ForcX	AAM1	AAM2	ARIMA	FFM	GAE-NN	GAESA-NN
N1402	132.377	210.723	231.899	248.444	164.663	224.326	212.246	212.246	77.625	**34.703**	35.685	35.323
N1403	47.853	62.092	62.092	41.921	61.685	34.786	44.578	44.578	116.101	37.287	**22.698**	23.182
N1404	34.93	36.013	34.456	45.156	26.97	44.958	38.321	38.321	34.181	30.936	**17.662**	17.695
N1405	41.654	39.809	51.343	34.338	40.332	34.678	54.209	54.209	82.571	28.495	20.655	**20.557**
N1406	28.428	26.267	51.531	34.034	48.116	25.308	43.542	43.542	40.782	32.461	16.483	**16.323**
N1407	293.735	134.661	240.84	179.358	238.688	156.013	106.234	106.234	91.007	**47.375**	50.892	50.775
N1408	100.703	28.826	28.826	34.001	37.245	34.699	32.205	32.205	40.597	34.445	**25.404**	25.489
N1409	72.069	38.039	41.615	42.069	68.313	34.055	33.203	33.203	82.736	36.345	**31.068**	31.49
N2522	10.237	5.752	6.547	10.289	10.966	2.160	7.811	**2.081**	2.528	7.454	6.057	5.842
N2523	1.259	2.680	6.382	4.152	1.604	1.941	1.497	6.990	4.413	2.288	1.269	**1.258**
N2524	3.163	3.766	10.577	2.815	5.696	4.330	3.765	10.551	8.785	3.151	1.927	**1.889**
N2525	6.619	13.268	20.060	6.392	10.113	13.691	8.844	23.838	14.189	4.268	2.970	**2.891**
N2526	2.955	4.187	6.105	11.927	3.779	9.764	**2.637**	3.807	11.238	3.846	3.257	3.345
N2527	7.538	6.714	**2.288**	6.537	6.327	4.721	5.782	4.458	3.491	3.748	3.355	3.238
N2528	7.441	4.918	3.949	5.240	5.488	4.489	7.849	3.285	5.402	2.747	2.436	**2.394**
N2529	9.228	8.039	6.806	6.489	9.225	6.888	9.964	5.893	7.604	3.189	2.521	**2.489**
N2530	10.097	7.814	5.921	6.060	6.183	10.787	9.834	6.187	7.187	2.760	2.443	**2.424**
N2531	7.441	4.918	3.949	5.240	5.488	4.489	7.849	3.285	5.402	2.713	2.462	**2.389**
N2532	10.798	15.892	16.586	14.469	12.767	13.637	**10.776**	14.825	12.903	16.140	15.492	15.556
N2533	7.235	12.609	18.515	16.119	**6.604**	12.929	11.307	10.043	13.611	12.748	11.821	11.719
N2534	10.578	12.797	11.501	12.124	8.561	11.742	10.753	11.729	12.450	4.791	4.124	**4.113**
Promedios	40.302	32.371	41.038	36.532	37.086	32.876	31.581	31.977	32.133	16.757	13.366	13.351

Tabla 7 Resultados estadísticos Friedman alineado con método de control FFM

Comparación	Resultado
FFM vs ARARMA	H0 se rechaza
FFM vs ARIMA	H0 se rechaza
FFM vs THETAsm	H0 se rechaza
FFM vs SMARTFCS	H0 se rechaza
FFM vs NAIVE	H0 se rechaza
FFM vs AAM1	H0 se rechaza
FFM vs B-J auto	H0 se rechaza
FFM vs AAM2	H0 se rechaza
FFM vs ForcX	H0 se rechaza
FFM vs GAESA-NN	H0 no se rechaza
FFM vs GAE-NN	H0 no se rechaza

En la tabla 7 muestra el resultado de aplicar el método Friedman alineado con una prueba post-hoc de Holm en la cual el método de control fue el algoritmo FFM, este fue comparado con los demás algoritmos contra la hipótesis nula de que no existen diferencias significativas, la cual es rechazada para todos los métodos comparados de la competencia M3 ya que como puede observarse en la tabla 6, en FFM existen diferencias obtuvo un MAPE menor en 9 de las 21 series en comparación con los métodos de M3.

Tabla 8 Resultados estadísticos Friedman alineado con método de control GAE-NN

Comparación	Resultado
GAE-NN vs ARARMA	H0 se rechaza
GAE-NN vs ARIMA	H0 se rechaza
GAE-NN vs THETAsm	H0 se rechaza
GAE-NN vs SMARTFCS	H0 se rechaza
GAE-NN vs NAIVE	H0 se rechaza
GAE-NN vs AAM1	H0 se rechaza
GAE-NN vs B-J auto	H0 se rechaza
GAE-NN vs AAM2	H0 se rechaza
GAE-NN vs ForcX	H0 se rechaza
GAE-NN vs FFM	H0 no se rechaza
GAE-NN vs GAESA-NN	H0 no se rechaza

En la tabla 8 muestra el resultado de aplicar el método Friedman alineado con una prueba post-hoc de Holm en la cual el método de control fue el algoritmo GAE-NN, este fue

comparado con los demás algoritmos contra la hipótesis nula de que no existen diferencias significativas, la cual es rechazada para todos los métodos comparados de la competencia M3 ya que como puede observarse en la tabla 6, en GAE-NN existen diferencias obtuvo un MAPE menor en 15 de las 21 series en comparación con los métodos de M3.

Tabla 9 Resultados Friedman alineado con método de control GAESA-NN

Comparación	Resultado
GAESA-NN vs ARARMA	H0 se rechaza
GAESA-NN vs ARIMA	H0 se rechaza
GAESA-NN vs THETAsm	H0 se rechaza
GAESA-NN vs SMARTFCS	H0 se rechaza
GAESA-NN vs NAIVE	H0 se rechaza
GAESA-NN vs AAM1	H0 se rechaza
GAESA-NN vs B-J auto	H0 se rechaza
GAESA-NN vs AAM2	H0 se rechaza
GAESA-NN vs ForcX	H0 se rechaza
GAESA-NN vs FFM	H0 no se rechaza
GAESA-NN vs GAE-NN	H0 no se rechaza

En la tabla 9 muestra el resultado de aplicar el método Friedman alineado con una prueba post-hoc de Holm en la cual el método de control fue el algoritmo GAESA-NN, este fue comparado con los demás algoritmos contra la hipótesis nula de que no existen diferencias significativas, la cual es rechazada para todos los métodos comparados de la competencia M3 ya que como puede observarse en la tabla 6, en GAE-NN existen diferencias obtuvo un MAPE menor en 16 de las 21 series en comparación con los métodos de M3.

Los resultados obtenidos para las series de tiempo de la bolsa mexicana de valores fueron los siguientes:

Tabla 10 Comparación con series de la BMV

Series	FFM	GAE-NN	GAESA-NN	SAP-SVR	ARIMA
CEMEX	1.704	**1.471321**	1.55185181	5.977396	39.27661
FEMSA	4.023	**1.212924**	1.25148118	15.74609	2.099846
LIVERPOOL	1.375	1.277194	**1.27242788**	7.195112	20.07579

En la Tabla 10, el resultado con el MAPE más bajo está marcado en negrita, para las series temporales CEMEX y LIVERPOOL FFM, GAE-NN, GASAE-NN, tuvieron mejores resultados que ARIMA y SAP-SVR, para FEMSA GAE-NN, GASAE-NN tuvo mejores resultados que ARIMA y SAP-SVR, FFM tuvo un MAPE más bajo que ARIMA, pero no más bajo que SAP-SVR.

En la tabla 11 se presentan los algoritmos benchmark ganadores de la competencia M4 Makridakis, así como una descripción breve de cada uno.

Tabla 11 Algoritmos Benchmark de M4

Nombre	Descripción
Naïve 1	$F_t+1 = Y_t$ i = 1, 2, 3, … , m
Seasonal Naïve	El pronóstico es igual a la última observación observada en el mismo periodo de tiempo
Naïve 2	Como Naïve 1 con la diferencia que los datos son ajustados estacionalmente de ser necesario aplicando descomposición multiplicativa
Simple Exponential Smoothing (S)	Función ses () del paquete v8.2 de R
Holt´s Exponential Smoothing (H)	Función holt () del paquete v8.2 de R
Dampen Exponential Smoothing (D)	Función holt () del paquete v8.2 de R
Combining S-H-D	El promedio aritmético de los métodos S-H-D
Theta	Como fue aplicado a la competencia M3 (θ=2, realiza ajuste estacional de la misma manera que Naïve2 aplicando la función ses () del paquete v8.2 de R
MLP	Un perceptrón de estructura y parametrización básica desarrollada en Python con la librería Scikit v0.19.1
RNN	Red neuronal recurrente de estructura y parametrización básica desarrollada en Python utilizando Keras v2.0.9 y tensorflow v1.4.0

Se puede observar que son en total 10 algoritmos que sirven de punto de partida para comparar métodos de pronóstico clásicos utilizando las series de la competencia M4 Makridakis. Se agregaron los algoritmos MLP y RNN como métodos de computo inteligente los cuales son códigos de Python utilizando librerías Scikit, Keras y Tensorflow.

A continuación, se presentan los resultados obtenidos ejecutando los Algoritmos FFM, GAENN y GASAENN para las series de tiempo de la competencia M4 comparándolos con los métodos benchmarking.

Tabla 12 Resultados de algoritmos benchamark de M4

Serie	FFM	GAENN	GASAENN	ARIMA	Comb	Damped	Holt	MLP	Naive	Naive2	RNN	SES	sNaive	Theta
D2061	5.227	4.804	4.774	**0.963**	0.994	0.964	1.115	3.358	**0.963**	**0.963**	4.813	0.964	**0.963**	1.025
D2085	1.724	1.667	**1.637**	3.891	4.749	4.825	6.336	23.577	3.891	3.891	34.845	3.891	3.891	4.100
D2088	2.011	1.960	**1.911**	4.132	2.740	3.171	3.122	6.775	2.503	2.503	5.018	2.502	2.503	2.559
D2092	12.268	9.922	10.127	3.551	2.892	2.557	3.843	2.360	2.295	2.295	**2.119**	2.295	2.295	2.819
D2105	5.453	4.176	4.091	1.089	1.095	1.125	1.080	3.435	1.089	1.089	7.437	1.089	1.089	**1.049**
D2140	18.056	**13.683**	13.765	183.083	133.984	129.290	185.427	1201.701	517.435	517.435	1165.128	517.447	517.435	494.347
D2168	18.333	13.145	**12.975**	229.510	190.288	189.140	173.179	509.492	208.544	208.544	257.755	208.545	208.544	212.857
D2186	10.345	9.400	9.249	3.889	3.880	3.778	4.096	6.942	3.889	3.889	6.838	**3.766**	3.889	4.138
D2187	8.366	8.134	8.070	0.725	0.679	0.725	**0.626**	3.929	0.725	0.725	4.654	0.725	0.725	0.885
D2188	10.405	8.609	8.772	0.793	0.786	0.793	**0.772**	3.827	0.793	0.793	4.852	0.793	0.793	0.845
D2206	24.203	**15.093**	16.584	101.803	126.016	111.148	95.868	314.546	171.030	171.030	303.044	171.031	171.030	173.208
D2209	8.411	7.242	7.350	6.700	4.955	5.242	**4.197**	11.442	5.425	5.425	7.525	5.425	5.425	4.901
D2211	3.852	3.855	3.795	**1.955**	2.684	1.966	4.133	7.388	**1.955**	**1.955**	5.465	**1.955**	**1.955**	2.402
D2214	20.548	15.625	**15.130**	185.781	376.267	392.092	119.608	1283.867	617.095	617.095	1244.488	617.102	617.095	639.541
D2228	4.519	3.576	3.578	1.953	1.238	0.987	1.777	10.374	0.979	0.979	2.786	0.979	0.979	**0.905**
D2371	0.832	0.660	0.679	0.648	0.610	0.653	**0.523**	1.946	0.648	0.648	2.070	0.654	0.648	0.559
D3299	3.1195	2.923	2.961	1.173	1.143	1.195	1.130	1.139	1.104	1.104	**0.851**	1.104	1.104	1.117
Promedio	9.275	7.322	7.379	43.038	50.294	49.979	35.696	199.770	90.610	90.610	179.982	90.604	90.610	91.015

En la tabla 12 podemos observar subrayado y con fondo gris los mejores resultados de los métodos de la competencia M4, esto quiere decir los resultados que obtuvieron el menor MAPE, en negrita los resultados de los métodos de pronóstico que obtuvieron el menor MAPE de toda la tabla, posteriormente se compararon estadísticamente utilizando el test de Friedman Alineado donde la hipótesis nula considera que no hay diferencia significativa entre los resultados muestras con lo cual se obtuvieron los siguientes resultados

Tabla 13 Ejecución de Friedman Alineado para métodos M4 con algoritmo de control FFM

Comparación	Resultado
FFM y ARIMA	H0 no se rechaza
FFM y Com	H0 no se rechaza
FFM y Damped	H0 no se rechaza
FFM y Holt	H0 no se rechaza
FFM y MLP	H0 no se rechaza
FFM y Naive	H0 no se rechaza
FFM y Naive2	H0 no se rechaza
FFM y RNN	H0 no se rechaza
FFM y SES	H0 no se rechaza
FFM y sNaive	H0 no se rechaza
FFM y Theta	H0 no se rechaza

Tabla 14 de Friedman Alineado para métodos M4 con algoritmo de control GAENN

Comparación	Resultado
GAENN y ARIMA	H0 no se rechaza
GAENN y Com	H0 no se rechaza
GAENN y Damped	H0 no se rechaza
GAENN y Holt	H0 no se rechaza
GAENN y MLP	H0 no se rechaza
GAENN y Naive	H0 no se rechaza
GAENN y Naive2	H0 no se rechaza
GAENN y RNN	H0 no se rechaza
GAENN y SES	H0 no se rechaza
GAENN y sNaive	H0 no se rechaza
GAENN y Theta	H0 no se rechaza

Tabla 15 de Friedman Alineado para métodos M4 con algoritmo de control GASAENN

Comparación	Resultado
GASAENN y ARIMA	H0 no se rechaza
GASAENN y Com	H0 no se rechaza
GASAENN y Damped	H0 no se rechaza
GASAENN y Holt	H0 no se rechaza
GASAENN y MLP	H0 no se rechaza
GASAENN y Naive	H0 no se rechaza
GASAENN y Naive2	H0 no se rechaza
GASAENN y RNN	H0 no se rechaza
GASAENN y SES	H0 no se rechaza
GASAENN y sNaive	H0 no se rechaza
GASAENN y Theta	H0 no se rechaza

Como puede observarse en las tablas 13-15 los algoritmos FFM, GAENN, y GASAENN tienen un desempeño similar a los algoritmos de benchmarking de la competencia M4 debido a que la hipótesis nula de Friedman alineado no se rechaza.

Capítulo 6

Conclusiones y trabajos futuros

6.1 Conclusiones

Se cumplieron satisfactoriamente con todos los objetivos que se plantearon, se realizó la implementación de 3 algoritmos neuroevolutivos (FFM. GAENN, GASAENN) y se probaron con series de tiempo de M3 contra los métodos NAIVE2, B-J auto, ARARMA, SMARTFCS, THETAsm, ForcX, AAM1, AAM2, y adicionalmente ARIMA, de los cuales lo algoritmos desarrollados en este trabajo fueron superiores estadísticamente a los métodos de la competencia de acuerdo al test de Friedman alineado, esto es notorio debido a que obtuvieron un MAPE más pequeño para ese conjunto de instancias de la competencia. Para las series de tiempo de la competencia M4 comparados contra los métodos de benchmarking Comb S-H-D, Damped, Holt, MLP, Naive, Naive2, RNN, SES, sNaive, Theta, y adicionalmente ARIMA, los algoritmos desarrollados en este trabajo tienen estadísticamente un desempeño similar de acuerdo al test de Friedman alineado sin embargo es de notar que para series en las cuales los métodos *benchamark* y ARIMA obtuvieron un MAPE muy elevado, este método obtuvo un MAPE en promedio no superior al 21%. Analizando los resultados de los algoritmos *Benchmark* de las competencias M3 Y M4 se puede concluir que los algoritmos desarrollados en este trabajo de tesis tienen un desempeño similar o superior al método clásico ARIMA el cual fue el objetivo de comparación en este trabajo de investigación. Adicionalmente en la comparación con los algoritmos *benchmark* de m4 se obtiene un desempeño estadísticamente similar con los 3 algoritmos propuestos en este trabajo

6.2 Principales contribuciones

A continuación, se describen las principales aportaciones de este proyecto de tesis:

- Un modelo evolutivo que utiliza únicamente mutación con el cual obtiene un mejor desempeño que ARIMA
- Un modelo neuroevolutivo que incluyendo cruza.
- Un nuevo modelo neuroevolutivo que ajusta pesos por medio del algoritmo recocido simulado.
- Una nueva forma de cruza por capas de redes de diferente topología con restricción de redes feedforward

6.3 Trabajos futuros

Algunos trabajos futuros con los cuales se puede continuar este trabajo de investigación son los siguientes:

- Realizar pruebas de evolución incluyendo evolución de función de activación en capa oculta y capa de salida donde estas sean diferentes.
- Evolucionar pesos con diferentes métodos metaheurísticos.
- Modificar la forma de selección de elementos de la siguiente generación para aceptar peores soluciones.
- Comparación con métodos ganadores de competencia M4.

Apéndice A Código de métodos neuroevolutivos en JAVA

//Se implementan las clases y los métodos para Neurona

```
public class Neuron {
   int id;
   private String activation;
   private double net;
   private double out;

   public Neuron() {
      activation = "sigmoid";
      net = 0;
      out = 0;
   }

   public Neuron(String activation,int id) {
      this.activation = activation;
      net = 0;
      out = 0;
      this.id=id;
   }
   public Neuron(int id) {
      activation = "sigmoid";
      net = 0;
      out = 0;
      this.id=id;

   }

   public double netValue() {
      return net;
   }

   public double outValue() {
      return out;
   }
   public void setNet(double net) {
      this.net=net;
   }

   public void setOut(double out) {
      this.out=out;
   }

   public void activate() {
      switch (activation) {
         case "sigmoid":
            out = sigmoid(net);
```

```
            case "linear":
                out = net;
            default:
                out = net;
        }
    }

    private double sigmoid(double v) {
        return (1 / (1 + Math.pow(Math.E, (-1 * v))));
    }

    private double htan(double v){return 0.0;}
    private double sigmoidDerivate(double v) {
        return (v * (1 - v));
    }
}
```

//Se implementan las clases y los métodos para Capa

```
public class Layer {

    Neuron[] neurons;
    Edge[] edge;

    public Layer() {
        neurons = new Neuron[1];
        edge = new Edge[1];
    }

    public Layer(int nNeurons) {
        neurons = new Neuron[nNeurons];
        edge = new Edge[nNeurons];
        for (int i = 0; i < nNeurons; i++) {
            neurons[i] = new Neuron();
            edge[i] = new Edge(0, i, Math.random());
        }

    }

    public Layer(Layer l2) {
        int n2, e2;
        n2 = l2.neurons.length;
        e2 = l2.edge.length;
        this.neurons = new Neuron[n2];
        this.edge = new Edge[e2];
        for (int i = 0; i < n2; i++) {
            this.neurons[i] = new Neuron();
        }
        for (int j = 0; j < l2.edge.length; j++) {
            this.edge[j] = new Edge(l2.edge[j].getIn(), l2.edge[j].getOut(), l2.edge[j].getWeight());
            this.edge[j].enabled=l2.edge[j].enabled;
        }

    }
```

```
public void restoreValues() {
   double cero = 0.0;
   for (Neuron neuron : neurons) {
      neuron.setNet(cero);
      neuron.setOut(cero);
   }
}

public void activateNeurons() {
   for (Neuron neuron : neurons) {
      neuron.activate();
   }
}

public void perturbWeight() {
   Random r = new Random(System.currentTimeMillis());
   double factor;
   factor = r.nextDouble() < 0.5 ? -1.0 : 1.0;
   factor *= factor * r.nextDouble();
   int k = (int) (Math.random() * edge.length);
   edge[k].setWeight(edge[k].getWeight() + factor);
}

public void addNeuron() {
   neurons = new Neuron[neurons.length + 1];
   for (int i = 0; i < neurons.length; i++) {
      neurons[i] = new Neuron();
   }
}

public void addEdge(Edge e) {
   Edge aux[] = new Edge[edge.length + 1];
   for (int i = 0; i < edge.length; i++) {
      aux[i] = new Edge(edge[i]);
   }
   aux[edge.length] = new Edge(e);
   edge = aux;

}

public void disableEnableEdge() {
   Random r = new Random(System.currentTimeMillis());
   int n = r.nextInt(edge.length);
   if (edge[n].enabled == true) {
      edge[n].enabled = false;
   } else {
      edge[n].enabled = true;
   }

}

public void disableEdge() {
   edge[(int) (Math.random() * edge.length)].enabled = false;
}
```

```
    public void sortEdges(int lowerIndex, int higherIndex) {
        int i = lowerIndex;
        int j = higherIndex;
        int pivot = lowerIndex + (higherIndex - lowerIndex) / 2;
        while (i <= j) {
            while (edge[i].getLabel().compareTo(edge[pivot].getLabel()) < 0) {
                i++;
            }
            while (edge[j].getLabel().compareTo(edge[pivot].getLabel()) > 0) {
                j--;
            }
            if (i <= j) {
                Edge temp = edge[i];
                edge[i] = edge[j];
                edge[j] = temp;
                i++;
                j--;
            }
        }
        if (lowerIndex < j) {
            sortEdges(lowerIndex, j);
        }
        if (i < higherIndex) {
            sortEdges(i, higherIndex);
        }

    }

}
```

//Se implementan las clases y los métodos para conexiones

```
public class Edge {

    private int in;
    private int out;
    private double w;
    private String label;
    public boolean enabled;

    public Edge(int in, int out, double w) {
        this.in = in;
        this.out = out;
        this.w = w;
        this.label=this.in+"->"+this.out;
        enabled=true;
    }
    public Edge(Edge e)
    {
    this.in=e.in;
    this.out=e.out;
    this.w=e.w;
    this.label=this.in+"->"+this.out;
    this.enabled=e.enabled;
    }
```

```
    public void setWeight(double w) {
        this.w = w;
    }

    public double getWeight() {
        return w;
    }

    public int getIn() {
        return in;
    }

    public int getOut() {
        return out;
    }
    public String getLabel() {
        return label;
    }
}
```

//Se implementan los métodos para Red

```
public class Network {

    int cNeuron;
    double MAPE;
    double MASE;
    double SMAPE;
    Layer layers[];
    double expected;

    public Network() {
        layers = new Layer[2];
        layers[0] = new Layer(3);
        layers[1] = new Layer(1);
        layers[0].edge = new Edge[1];
        layers[0].edge[0] = new Edge(0, 0, 0);
        layers[1].edge = new Edge[3];
        for (int i = 0; i < layers[1].edge.length; i++) {
            layers[1].edge[i] = new Edge(i, 0, miRandom());

        }
    }

    public Network(int l) {
        Random r = new Random(System.currentTimeMillis());
        int el;
        layers = new Layer[l];
        for (int i = 0; i < l; i++) {
            if (i < l - 1) {
                layers[i] = new Layer(1 + r.nextInt(4));
            } else {
                layers[i] = new Layer(1);
```

```
        }

        if (i == 0) {
            layers[i].edge = new Edge[1];
            layers[i].edge[i] = new Edge(0, 0, 0.001);
        } else if (i > 0) {
            el = layers[i].neurons.length * layers[i - 1].neurons.length;
            layers[i].edge = new Edge[el];
            int c = 0;
            for (int j = 0; j < layers[i - 1].neurons.length; j++) {
                for (int k = 0; k < layers[i].neurons.length; k++) {
                    layers[i].edge[c++] = new Edge(j, k, miRandom());
                }

            }
        }
    }
}

public Network(Network n) {
  this.MAPE = n.MAPE;
    int ed, nn;
    this.layers = new Layer[n.layers.length];
    for (int i = 0; i < n.layers.length; i++) {
        this.layers[i] = new Layer(n.layers[i]);
    }
}

private void propagateLayer(Layer layer, Layer layer2) {
    layer2.restoreValues();
    for (Edge edge : layer2.edge) {
        if (edge.enabled == true) {
            layer2.neurons[edge.getOut()].setNet(layer2.neurons[edge.getOut()].netValue()
                    + (edge.getWeight() * layer.neurons[edge.getIn()].outValue()));
        }
    }
    layer2.activateNeurons();

}

public void propagateNet() {

    for (int i = 1; i < layers.length; i++) {
        propagateLayer(layers[i - 1], layers[i]);
    }
}

public void printNetwork() {
    for (int i = 0; i < layers.length; i++) {
        if (i > 0) {
            for (Edge edge : layers[i].edge) {
                System.out.print(edge.getLabel() + ":" + edge.getWeight() + "_" + edge.enabled + " | ");
            }
            System.out.println();
        }
    }
```

```
}

public void setInput(double[] in) {
    for (int i = 0; i < layers[0].neurons.length; i++) {
        layers[0].neurons[i].setOut(in[i]);
    }
}

public void AddLayer() {
    Layer[] aux = new Layer[layers.length + 1];
    for (int i = 0; i < layers.length; i++) {
        aux[i] = new Layer(layers[i]);
    }
    aux[layers.length] = new Layer(1);
    layers = aux;
}

private double miRandom() {
    return (double) Math.random();// * (1 - -1) + -0.99;
}

public double getOutput() {
    return layers[layers.length - 1].neurons[0].outValue();
}

public void Mutate() {
    Random r = new Random(System.currentTimeMillis());
    int lyr = r.nextInt(layers.length);
    double opcion = r.nextDouble();
    if (opcion <= 0.5) {
        if (lyr == 0) {
            lyr = 1;
        }
        layers[lyr].perturbWeight();

    } else if (opcion > 0.5 && opcion <= 0.7) {
        if (lyr == 0) {
            lyr = 1;
        }
        layers[lyr].disableEnableEdge();
    } else if (opcion > 0.7 && opcion <= 0.98) {
        if (lyr == layers.length - 1) {
            lyr = lyr - 1;
        }
        layers[lyr].addNeuron();
        for (int i = 0; i < layers[lyr + 1].neurons.length; i++) {
            layers[lyr + 1].addEdge(new Edge(layers[lyr].neurons.length - 1, i, miRandom()));
        }
        if (lyr >= 1) {
            for (int i = 0; i < layers[lyr - 1].neurons.length; i++) {
                layers[lyr].addEdge(new Edge(i,layers[lyr].neurons.length - 1,  miRandom()));
            }
        }
        // System.out.println("_______agrego Neurona");
    } else if (opcion > 0.98) {
        AddLayer();
```

```
            //System.out.println("_____agrego capa");
        }
    }

    public void adjustWeights(){
        double w[];
        int n=0;
        for (int i = 1; i < layers.length; i++) {
            n+=layers[i].edge.length;
        }
    }

    public void setWeights(double []w){
        int c=0;
    for (int i = 1; i < layers.length; i++) {
            for (Edge edge : layers[i].edge) {
                edge.setWeight(w[c++]);
            }
        }
    }
}

// implementación de los tres métodos de pronóstico
public class ANNForecastingMoises {

    static Scanner sc;
    static PrintWriter w;
    static double data[];
    static double normalized[];
    static Network population[];
    static Network newPop[];
    static Network parents1[];
    static Network parents2[];
    static Network childs[];
    static double max = Double.MIN_VALUE, min = Double.MAX_VALUE;
    static ArrayList<Double> dataArray;
    static double trainPC = 0.75;
    static double testPC = 0.25;
    static int trainV;
    static int testV;
    static int Horizon=0;
    static void printData() {
        for (int i = 0; i < data.length; i++) {
            System.out.println("   " + data[i]);
        }
    }

    static void readData(String s) throws FileNotFoundException {
        sc = new Scanner(new File(s));
        dataArray = new ArrayList<>();
        while (sc.hasNext()) {
            dataArray.add(sc.nextDouble());
        }
        data = new double[dataArray.size()];
        for (int i = 0; i < dataArray.size(); i++) {
            data[i] = (double) dataArray.get(i);
```

```
        if (data[i] < min) {
            min = data[i];
        }
        if (data[i] > max) {
            max = data[i];
        }
    }
    if (Horizon>0) {
        System.out.println("trainPC=(("+data.length+"-"+Horizon+") * 1)/"+data.length);
        System.out.println("testPC= ("+Horizon +"* 1)/"+data.length+")");
        trainPC= ((data.length-Horizon) * 1)/(double)data.length;
        testPC= (Horizon * 1)/(double)data.length;
    }

    testV = (int) (data.length * testPC);
    trainV = (int) (data.length * trainPC);

    System.out.println(s);
    System.out.println("Data lenght = " + data.length);
    System.out.println("Train pc = " + trainPC);
    System.out.println("Test pc = " + testPC);
    System.out.println("data.length*testPC= " + data.length * testPC);
    System.out.println("data.length*trainPC= " + data.length * trainPC);

    w.println(s);
    w.println("Data lenght = " + data.length);
    w.println("Train pc = " + trainPC);
    w.println("Test pc = " + testPC);
    w.println("data.length*testPC= " + data.length * testPC);
    w.println("data.length*trainPC= " + data.length * trainPC);
}

//normalize de data in range[0,1]
static void normalizeData() {
    normalized = new double[data.length];
    for (int i = 0; i < data.length; i++) {
        normalized[i] = ((data[i] - min) / (max - min)) * (0.9 - 0.1) + 0.1;
    }
}

//denormalize a value
static double denormalizeValue(double v) {
    v = ((v - 0.1) / (0.9 - 0.1)) * (max - min) + min;
    return v;
}

static void startPopulation(int count) {
    population = new Network[count];
    newPop = new Network[population.length + count / 2];
    for (int i = 0; i < count; i++) {
        population[i] = new Network();
    }

}

static void evaluatePopulation() {
```

```
        int ca;
        double[] in;
        for (Network population1 : population) {
            ca = 0;
            population1.MAPE = 0.0;
            for (int i = population1.layers[0].neurons.length; i < normalized.length - 1; i++) {
                ca++;
                population1.expected = normalized[i];
                in = new double[population1.layers[0].neurons.length];
                System.arraycopy(normalized, i - population1.layers[0].neurons.length, in, 0,
population1.layers[0].neurons.length);
                population1.setInput(in);
                population1.propagateNet();
                population1.MAPE += Math.abs(population1.getOutput() - population1.expected) /
Math.abs(population1.expected);
            }
            population1.MAPE *= (100.0 / ((double) ca));
        }
    }

    static void evaluatePopulation2(Boolean Training) {

        int ca;
        double[] in;
        int dStart, dEnd;
        if (Training == true) {
            dStart = 0;
            dEnd = trainV - 1;
        } else {
            dStart = trainV + 1;
            dEnd = normalized.length - 1;
        }
        for (Network population1 : population) {
            ca = 0;
            population1.MAPE = 0.0;
            population1.SMAPE=0.0;
            population1.MASE=0.0;
            if (Training==false && Horizon>0) {
                dStart = trainV + 1;
                dStart=dStart-population1.layers[0].neurons.length;
            }
            for (int i = (dStart + population1.layers[0].neurons.length); i < dEnd; i++) {
                ca++;
                population1.expected = normalized[i];
                in = new double[population1.layers[0].neurons.length];
                System.arraycopy(normalized, i - population1.layers[0].neurons.length, in, 0,
population1.layers[0].neurons.length);
                population1.setInput(in);
                population1.propagateNet();
                population1.MAPE += Math.abs(population1.getOutput() - population1.expected) /
Math.abs(population1.expected);
            }
            population1.MAPE *= (100.0 / ((double) ca));
            if (Training == false) {
            }
        }
```

```
    }

    static void evaluatePopulation3(Boolean Training) {

        int ca;
        double[] in;
        int dStart, dEnd;
        if (Training == true) {
            dStart = 0;
            dEnd = trainV - 1;
        } else {
            dStart = trainV + 1;
            dEnd = normalized.length - 1;
        }
        for (Network population1 : population) {
            ca = 0;
            population1.MAPE = 0.0;
            if (Training==false) {
                dStart=dStart-population1.layers[0].neurons.length;
            }
            for (int i = (dStart + population1.layers[0].neurons.length); i < dEnd; i++) {
                ca++;
                population1.expected = normalized[i];
                in = new double[population1.layers[0].neurons.length];
                System.arraycopy(normalized, i - population1.layers[0].neurons.length, in, 0,
population1.layers[0].neurons.length);
                population1.setInput(in);
                population1.propagateNet();
                population1.MAPE += Math.abs(population1.getOutput() - population1.expected) /
Math.abs(population1.expected);
            }
            population1.MAPE *= (100.0 / ((double) ca));
            if (Training == false) {
            }
        }

    }

    static void evaluateNewPop(Boolean Training) {
        int ca;
        double[] in;
        int dStart, dEnd;
        if (Training == true) {
            dStart = 0;
            dEnd = trainV - 1;
        } else {
            dStart = trainV + 1;
            dEnd = normalized.length - 1;
        }
        for (Network newP : newPop) {
            ca = 0;
            newP.MAPE = 0.0;
            for (int i = (dStart + newP.layers[0].neurons.length); i < dEnd; i++) {
                ca++;
                newP.expected = normalized[i];
                in = new double[newP.layers[0].neurons.length];
```

```
            System.arraycopy(normalized, i - newP.layers[0].neurons.length, in, 0, newP.layers[0].neurons.length);
            newP.setInput(in);
            newP.propagateNet();
            newP.MAPE += Math.abs(newP.getOutput() - newP.expected) / Math.abs(newP.expected);
            //System.out.println(              denormalizeValue(population1.getOutput())              +              "_"              +
denormalizeValue(population1.expected));
         }
         newP.MAPE *= (100.0 / ((double) ca));
         if (Training == false) {
         }
      }

   }

   public static Network[] quickSort(Network[] pop, int lowerIndex, int higherIndex) {
      int i = lowerIndex;
      int j = higherIndex;
      Network pivot = pop[lowerIndex + (higherIndex - lowerIndex) / 2];
      while (i <= j) {

         while (pop[i].MAPE < pivot.MAPE) {
            i++;
         }
         while (pop[j].MAPE > pivot.MAPE) {
            j--;
         }
         if (i <= j) {
            pop = exchangeNumbers(pop, i, j);
            i++;
            j--;
         }
      }
      if (lowerIndex < j) {
         pop = quickSort(pop, lowerIndex, j);
      }
      if (i < higherIndex) {
         pop = quickSort(pop, i, higherIndex);
      }
      return pop;
   }

   public static Network[] exchangeNumbers(Network[] a, int i, int j) {
      Network temp = new Network(a[i]);
      a[i] = new Network(a[j]);
      a[j] = new Network(temp);
      return a;
   }

   public static void shufflePop() {
      Random random = new Random(System.currentTimeMillis());
      for (int i = population.length; i > 1; i--) {
         population = exchangeNumbers(population, i - 1, random.nextInt(i));
      }
   }

   public static void printPopulation() {
```

```
    for (Network population1 : population) {
        System.out.println("MAPE = " + population1.MAPE);
        population1.printNetwork();
    }
}

public static void selectParents() {
    //tournament selection
    int c = 0;
    parents1 = new Network[population.length / 2];
    parents2 = new Network[population.length / 2];
    for (int i = 0; i < population.length; i += 2) {
        if (population[i].MAPE < population[i + 1].MAPE) {
            parents1[c++] = population[i];
        } else {
            parents1[c++] = population[i + 1];
        }
    }
    shufflePop();
    evaluatePopulation2(true);
    c = 0;
    for (int i = 0; i < population.length - 1; i += 2) {
        if (population[i].MAPE < population[i + 1].MAPE) {
            parents2[c++] = population[i];
        } else {
            parents2[c++] = population[i + 1];
        }
    }
}

public static void crossOver() {
    Random r = new Random(System.currentTimeMillis());
    double prob;
    childs = new Network[parents1.length];
    for (int i = 0; i < parents1.length; i++) {
        childs[i] = new Network(networkCrossover(parents1[i], parents2[i]));
        prob = r.nextDouble();
        if (prob < 0.1) {
            childs[i].Mutate();
        }
    }
}

public static Network networkCrossover(Network parent1, Network parent2) {
    int g = (parent1.layers.length > parent2.layers.length) ? 1 : 2;
    int l = (parent1.layers.length > parent2.layers.length) ? parent1.layers.length : parent2.layers.length;
    int e = 1;
    int n = 1;
    String nLbl, p1Lbl, p2Lbl;
    Network na = new Network(l);
    //na.printNetwork();
    for (int i = 0; i < l; i++) {
        int c = 0;
        if (g == 1) {
            if (i < parent2.layers.length) {
```

```
            n = (parent1.layers[i].neurons.length > parent1.layers[i].neurons.length) ? parent2.layers[i].neurons.length :
parent2.layers[i].neurons.length;
         } else {
            n = parent1.layers[i].neurons.length;
         }

      } else if (g == 2) {
         if (i < parent1.layers.length) {
            //    parent1.layers[i].sortEdges(0, parent1.layers[i].edge.length-1);
            n = (parent1.layers[i].neurons.length > parent1.layers[i].neurons.length) ? parent2.layers[i].neurons.length :
parent2.layers[i].neurons.length;
         } else {
            n = parent2.layers[i].neurons.length;
         }
      }
      na.layers[i].neurons = new Neuron[n];
      for (int j = 0; j < n; j++) {
         na.layers[i].neurons[j] = new Neuron();
      }
      if (i == 0) {
         na.layers[i].edge = new Edge[1];
         na.layers[i].edge[0] = new Edge(0, 0, Math.random());
      } else {
         na.layers[i].edge = new Edge[na.layers[i].neurons.length * na.layers[i - 1].neurons.length];

         for (int j = 0; j < na.layers[i].neurons.length; j++) {
            for (int k = 0; k < na.layers[i - 1].neurons.length; k++) {
               na.layers[i].edge[c] = new Edge(k, j, Math.random());
               na.layers[i].edge[c].enabled = false;
               c++;
            }
         }
      }

      for (int j = 0; j < na.layers[i].edge.length; j++) {
         int i1 = -1;
         int i2 = -1;
         if (i < parent1.layers.length) {
            for (int k = 0; k < parent1.layers[i].edge.length; k++) {
               if (na.layers[i].edge[j].getLabel().compareTo(parent1.layers[i].edge[k].getLabel()) == 0) {
                  i1++;
                  break;
               }
            }
         }
         if (i < parent2.layers.length) {
            for (int k = 0; k < parent2.layers[i].edge.length; k++) {
               if (na.layers[i].edge[j].getLabel().compareTo(parent2.layers[i].edge[k].getLabel()) == 0) {
                  i2++;
                  break;
               }
            }
         }
         if (i2 > -1 || i2 > -1) {
            if (i1 > -1 && i2 > -1) {
```

```
                na.layers[i].edge[j] = (Math.random() < 0.5) ? new Edge(parent1.layers[i].edge[i1]) : new
Edge(parent2.layers[i].edge[i2]);
            }
            if (i1 > -1 && i2 == -1) {
                na.layers[i].edge[j] = new Edge(parent1.layers[i].edge[i1]);
            }
            if (i1 == -1 && i2 > -1) {
                na.layers[i].edge[j] = new Edge(parent2.layers[i].edge[i2]);
            }

        }
      }
    }
    return na;
  }

  public static double noCrossOver(int pop, int gen) {
    int nes = pop; //population number
    int generation = gen;//generation number
    Network[] net = new Network[nes];
    startPopulation(nes);
    evaluatePopulation2(true);//training
    for (int j = 0; j < generation; j++) {
      population = quickSort(population, 0, population.length - 1);
      for (int i = 0; i < population.length; i++) {
        net[i] = new Network(population[0]);
        if (i > 0) {
          net[i].Mutate();
        }
        // net[i].printNetwork();
      }

      System.arraycopy(net, 0, population, 0, net.length);
      evaluatePopulation2(true);//training
    }
    evaluatePopulation2(false);//evaluation
    population = quickSort(population, 0, population.length - 1);
    return population[0].MAPE;
  }

  public static double crossoverProcedure(int nes, int generation) {
    startPopulation(nes);
    evaluatePopulation2(true);//training
    for (int j = 0; j < generation; j++) {
      //System.out.println("***************Generacion = " + j + "*****************************");
      selectParents();
      crossOver();
      newPop = new Network[population.length + childs.length];
      for (int i = 0; i < newPop.length; i++) {
        if (i < population.length) {
          newPop[i] = new Network(population[i]);
        }
        if (i >= population.length) {
          newPop[i] = new Network(childs[i - population.length]);
        }
      }
```

```
            evaluateNewPop(true);
            newPop = quickSort(newPop, 0, newPop.length - 1);
            for (int i = 0; i < population.length; i++) {
                population[i] = new Network(newPop[i]);
            }
        }
        evaluatePopulation2(false);//evaluation
        population = quickSort(population, 0, population.length - 1);
        return population[0].MAPE;
    }

    public static Network getNeighbor(Network n) {
        Random r = new Random(System.currentTimeMillis());
        int lyr = r.nextInt(n.layers.length);
        double opcion = r.nextDouble();
        if (lyr == 0) {
            lyr = 1;
        }
        n.layers[lyr].perturbWeight();
        return n;
    }

    public static Network adjustWeights(Network n) {
        Network best = new Network(n);
        Network current = new Network(n);
        Network neighbor;
        double temp = 100;
        double coolingRate = 0.95;
        while (temp > 0.3) {
            for (int i = 0; i < 10; i++) {
                neighbor = new Network(getNeighbor(current));
                neighbor.MAPE = EvaluateNetwork(true, neighbor);

                if (acceptanceProbability(current.MAPE, neighbor.MAPE, temp) > Math.random()) {
                    current = new Network(neighbor);
                }
            }
            if (current.MAPE < best.MAPE) {
                best = new Network(current);

            }
            temp = temp * coolingRate;
        }
        return best;
    }

    public static Network adjustWeights2(Network Si) {
        double deltha;
        int Lk = 10;
        Network Sj = new Network(Si);
        //initial temperature
        double temp = 100;
        double alpha = 0.95;
        while (temp >= 0.3) {
            for (int i = 0; i < Lk; i++) {
                Sj = new Network(getNeighbor(Si));
```

```
            Sj.MAPE = EvaluateNetwork(true, Sj);
            deltha = Si.MAPE - Sj.MAPE;
            if (deltha <= 0) {
                Si = new Network(Sj);
            } else if (Math.random()<Math.exp((-1 * deltha)/ temp) ) {
                Si = new Network(Sj);
            }
        }
        // Keep track of the best solution found
        if (Sj.MAPE < Si.MAPE) {
            Si = new Network(Sj);
        }
        temp = temp * alpha;
    }
    return Si;
}

public static double acceptanceProbability(double energy, double newEnergy, double temperature) {

    if (newEnergy < energy) {
        return 1.0;
    }
    return Math.exp((energy - newEnergy) / temperature);
}

public static double EvaluateNetwork(boolean Training, Network population1) {
    double ca = 0.0;
    double[] in;
    int dStart, dEnd;
    if (Training == true) {
        dStart = 0;
        dEnd = trainV - 1;
    } else {
        dStart = trainV + 1;
        dEnd = normalized.length - 1;
    }
    population1.MAPE = 0.0;
    for (int i = (dStart + population1.layers[0].neurons.length); i < dEnd; i++) {
        ca++;
        population1.expected = normalized[i];
        in = new double[population1.layers[0].neurons.length];
        System.arraycopy(normalized, i - population1.layers[0].neurons.length, in, 0,
population1.layers[0].neurons.length);
        population1.setInput(in);
        population1.propagateNet();
        population1.MAPE += Math.abs(population1.getOutput() - population1.expected) /
Math.abs(population1.expected);
    }
    population1.MAPE *= (100.0 / ((double) ca));
    return population1.MAPE;
}

public static double noCrossOver2(int pop, int gen) {
    int nes = pop; //population number
    int generation = gen;//generation number
    Network[] net = new Network[nes];
```

```
    startPopulation(nes);
    evaluatePopulation2(true);//training
    for (int i = 0; i < population.length; i++) {
        population[i] = new Network(adjustWeights(population[i]));
    }

    for (int j = 0; j < generation; j++) {
        population = quickSort(population, 0, population.length - 1);
        for (int i = 0; i < population.length; i++) {
            net[i] = new Network(population[0]);
            if (i > 0) {
                net[i].Mutate();
                net[i] = new Network(adjustWeights(net[i]));
            }
        }

        System.arraycopy(net, 0, population, 0, net.length);
        evaluatePopulation2(true);//training
    }
    evaluatePopulation2(false);//evaluation
    population = quickSort(population, 0, population.length - 1);
    return population[0].MAPE;
}

public static double crossoverProcedure2(int nes, int generation) {
    startPopulation(nes);
    evaluatePopulation2(true);//training
    for (int i = 0; i < population.length; i++) {
        population[i] = new Network(adjustWeights(population[i]));
    }
    for (int j = 0; j < generation; j++) {
        selectParents();
        crossOver();
        for (int i = 0; i < childs.length; i++) {
            childs[i] = new Network(adjustWeights(childs[i]));
        }
        newPop = new Network[population.length + childs.length];
        for (int i = 0; i < newPop.length; i++) {
            if (i < population.length) {
                newPop[i] = new Network(population[i]);
            }
            if (i >= population.length) {
                newPop[i] = new Network(childs[i - population.length]);
            }
        }
        evaluateNewPop(true);
        newPop = quickSort(newPop, 0, newPop.length - 1);
        for (int i = 0; i < population.length; i++) {
            population[i] = new Network(newPop[i]);
        }
    }
    evaluatePopulation2(false);//evaluation
    population = quickSort(population, 0, population.length - 1);

    return population[0].MAPE;
}
```

```
    public static void doProcedure(String s, int executions) {

        dataArray = new ArrayList<>();
        try {
            readData("series/" + s);//02 TO 09
            //readData("cemex.txt");
            // readData("LIVERPOOL.txt");
            //readData("FEMSA.txt");
        } catch (FileNotFoundException ex) {
            Logger.getLogger(ANNForecastingMoises.class.getName()).log(Level.SEVERE, null, ex);
        }
        normalizeData();
        double m;
        System.out.println("SERIE,TOTAL,HORIZONTE,METODO,MAPE");
        System.out.print(s+","+dataArray.size()+","+testV+",FFM");
        w.println(s+","+dataArray.size()+","+testV+",FFM");
        for (int i = 0; i < executions; i++) {
            m = noCrossOver(100, 100);
            System.out.print("," + m);
            w.print("," + m);
        }
System.out.println();
        System.out.print(s+","+dataArray.size()+","+testV+","+"GAENN");
        w.println();
        w.println(s+","+dataArray.size()+","+testV+","+"GAENN");
        for (int i = 0; i < executions; i++) {
            m = crossoverProcedure(100, 100);  //network evolution trough crossover
            System.out.print("," + m);
            w.print("," + m);
        }
        System.out.println();
        System.out.print(s+","+dataArray.size()+","+testV+","+"GASAENN");
        w.println();
        w.println(s+","+dataArray.size()+","+testV+","+"GASAENN");
        for (int i = 0; i < executions; i++) {
            m = crossoverProcedure2(100, 100); //network evolution trough crossover
            System.out.print("," + m);
            w.print("," + m);
        }
        System.out.println("\n\n");

    }

    public static void doProcedure2(String s) {
        try {
            readData("series/" + s);//02 TO 09
        } catch (FileNotFoundException ex) {
            Logger.getLogger(ANNForecastingMoises.class.getName()).log(Level.SEVERE, null, ex);
        }
        normalizeData();
        double m;
Network otp[]=new Network[100];
        for (int i = 0; i < 100; i++) {
            otp[i]=new Network(3);
            EvaluateNetwork(true, otp[i]);
```

```
        }
        for (int i = 0; i < 30; i++) {
            m = crossoverProcedure2(100, 100);  //network evolution trough crossover
            System.out.print("   " + m);
        }
        System.out.println("\n\nTermino ");



    }

    public static void main(String[] args) throws FileNotFoundException {
         String da;
        Date date = new Date(System.currentTimeMillis());
         SimpleDateFormat dt = new SimpleDateFormat("yyyymmdd_hh_mm_ss");
         da=dt.format(date);

        File f = new File("series/");
        ArrayList<String> names = new ArrayList<>(Arrays.asList(f.list()));
        String nombre = "s1402.txt";
        String [] s1;
       // System.out.println("args.length = " + args.length);
        if (args.length>0) {
            if (args.length==3) {
               trainPC = Double.parseDouble(args[1]);
               testPC = Double.parseDouble(args[2]);
            }
            if (args.length==2) {
                Horizon=Integer.parseInt(args[1]);
            }
            s1=args[0].split("[.]");
            System.out.println("sl.length = " + s1.length);
            w = new PrintWriter(new File("output/" + s1[0]+"_out_"+da+"."+s1[1]));
            doProcedure(args[0],30);

        }else{
            s1=nombre.split("[.]");
            System.out.println("sl.length = " + s1.length);
           w = new PrintWriter(new File("output/" + s1[0]+"_out_"+da+"."+s1[1]));
        doProcedure(nombre,30);

        }
        w.flush();
        w.close();
    }
}
```

Bibliografía

[Ariyo et al.,2014] Ariyo, A. A., Adewumi, A. O., & Ayo, C. K. (2014, March). Stock price prediction using the ARIMA model. In Computer Modelling and Simulation (UKSim), 2014 UKSim-AMSS 16th International Conference on (pp. 106-112). IEEE.

[Bollerslev, 1986] Bollerslev, T. Generalized autoregressive conditional heteroskedasticity. Journal of econometrics, 31(3), 307-327. (1986).

[Box & Jenkins, 2008] Box P. E.G. ,Jenkins G.M.,Reinsel C. G.:Time Series Analysis Forecasting and Control(2008)

[Brabazon 2015] Brabazon, A., O'Neill, M., & McGarraghy, S. Natural computing algorithms. Berlin: Springer. (2015).

[Chatfield ,2000] Chatfield, C. Time-series forecasting. Chapman and Hall/CRC Press. (2000).

[Chen & Chen, 1995] Chen, T., & Chen, H. (1995). Universal approximation to nonlinear operators by neural networks with arbitrary activation functions and its application to dynamical systems. IEEE Transactions on Neural Networks, 6(4), 911-917.

[Chen & Chang, 2009] Chen, Y. H., & Chang, F. J. Evolutionary artificial neural networks for hydrological systems forecasting. Journal of Hydrology, 367(1), 125-137. (2009).

[Ciumac, 2011] Ciumac S.,Financial Predictor, https://www.codeproject.com/Articles/175777/Financial-predictor-via-neural-network (2011)

Coelo et al.,2004 Coello, C. A. C., & Zacatenco, C. S. P. (2004). Introducción a la computación evolutiva (Notas de curso). CINVESTAV-IPN, Departamento de Ingeniería Eléctrica, Sección de Computación. México, DF

[De Arce et al., 2003] De Arce, R., & Mahía, R. Modelos Arima. Programa CITUS: Técnicas de Variables Financieras(2003).

[Eaton, 2017] Eaton J. W., Octave, https://www.gnu.org/software/octave/ (2017)

[Engle, 1982] Engle, R. F. Autoregressive conditional heteroscedasticity with estimates of the variance of United Kingdom inflation. Econometrica: Journal of the Econometric Society, 987-1007. (1982).

[Escobar e al., 2009] Escobar, L., Valdes, J., & Zapata, S. Redes Neuronales Artificiales en predicción de Series de Tiempo. Una aplicación a la Industria. (2009).

[Faruk, 2010] Faruk, D. Ö A hybrid neural network and ARIMA model for water quality time series prediction. Engineering Applications of Artificial Intelligence, 23(4), 586-594 . (2010).

[Atsalakis et al.,2011] G.S. Atsalakis, E.M Dimitrakakis. and C.D. Zopounidis, "Elliot Wave Theory and neuro-fuzzy systems, stock market prediction: The WASP system", Expert Systems with Applications, vol. 38, pp.9196–9206,(2011).

[Giraldo, 2006] Giraldo, G.N.: Notas de clase Series de Tiempo con R, Universidad Nacional de Colombia (2006)

[Glover et al., 2000] Glover, F., Laguna, M., & Martí, R. Fundamentals of scatter search and path relinking. Control and cybernetics, 29(3), 653-684. (2000).

[Glover et al., 2003] Glover, F., Melián-Batista, B. Búsqueda Tabú. Inteligencia Artificial, Revista Iberoamericana de Inteligencia Artificial, (2003).

[Gonzalez et al., 2012] Gonzalez, B. P., Sánchez, G. G., Donate, J. P., Cortez, P., & de Miguel, A. S. Parallelization of an evolving artificial neural networks system to forecast time series using openmp and mpi. In Evolving and Adaptive Intelligent Systems (EAIS), 2012 IEEE Conference on (pp. 186-191). IEEE. (2012, May).

González et al.,2017 González-Mancha, J. J., Frausto-Solís, J., Valdez, G. C., Terán-Villanueva, J. D., & Barbosa, J. J. G. (2017). Financial time series forecasting using Simulated Annealing and Support Vector Regression. International Journal of Combinatorial Optimization Problems and Informatics, 8(2), 10-18.

[Hanke, 2010] Hanke J.E.; Reitsch A.G.:Pronóstico en los negocios(2010).

[Hanke et al., 2006] Hanke, J. E., & Wichern, D. W. Pronósticos en los negocios. Pearson Educación. (2006).

[Harris ,2003] Harris R., Sollis R.: Applied Time Series Modelling and Forecasting(2003).

[Henao ,2013] Henao Parra, J. S. Las redes neuronales y su desempeño bajo la estrategia de Neuroevolución (Master's thesis). (2013).

[Hernandez, 1997] Hernandez M.R.: El uso del computador en la aplicación de técnicas para el pronóstico de series de tiempo, económicas y financieras, (1997).

[Holland, 1992] Holland, J. H. Genetic algorithms. Scientific american, 267(1), 66-72. (1992).

[Hyndman & Athanasopoulos, 2014] Hyndman, R. J., & Athanasopoulos, G. Forecasting: principles and practice. OTexts (2014).

[Wang et al., 2012] J.J. Wang, J.Z. Wang, Z.G. Zhang and S.P. Guo, "Stock index forecasting based on a hybrid model", Omega vol.40 pp.758-766,(2012).

[Jimenez et al.,2006] Jimenez, J.F.; Gázquez, J.C; Sánchez, R: La capacidad predictiva en los métodos Box- Jenkins y Holt-Winters: una aplicación al sector turístico.(2006)

[Kirkpatrick et al, 1987] Kirkpatrick, S., Gelatt, C. D., & Vecchi, M. P. (1983). Optimization by simulated annealing. science, 220(4598), 671-680.

[Kriesel ,2007] Kriesel D., A Brief Introduction to Neural Networks(2007) http://www.dkriesel.com

[L.Y. Wei, 2013] L.Y. Wei, "A hybrid model based on ANFIS and adaptive expectation genetic algorithm to forecast TAIEX", Economic Modelling vol. 33 pp. 893-899, (2013).

[Makridakis & Hibon,2000] Makridakis, S., & Hibon, M. The M3-Competition: results, conclusions and implications. International journal of forecasting, 16(4), 451-476. (2000).

[Martín, 2003] Martín, I. G. Análisis y predicción de la serie de tiempo del precio externo del café colombiano utilizando redes neuronales artificiales. Universitas Scientiarum, 8, 45-50. (2003).

[Matich ,2001] Matich, D. J. Redes Neuronales: Conceptos básicos y aplicaciones. Cátedra de Informática Aplicada a la Ingeniería de Procesos–Orientación I. (2001).

[Mitchell ,1999] Mitchell, M. An Introduction to Genetic Algorithms. Massachusetts: Bradford Book. (1999).

[Montenegro ,2010] Montenegro, R.: Medición de la volatilidad en series de tiempo financieras (2010)

[Morales,2008] Morales, A. Pronósticos mediante redes neuronales artificiales y modelos arima: el caso de los cetes en México. UNAM, 794pp. (2008)

[Pai et al. 2005] P. Pai and C. Lin, "A hybrid ARIMA and support vector machines model in stock price prediction", Omega vol.33 pp. 497-505(2005)

[Peña,2010] Peña D., Análisis de series temporales, Alianza Editorial,(2010)

[Rumelhart et al., 1985] Rumelhart, D. E., Hinton, G. E., & Williams, R. J. Learning internal representations by error propagation (No. ICS-8506). California University, San Diego La Jolla Inst. for Cognitive Science. (1985).

[S. Panigrahi, 2013] S. Panigrahi, Y. Karali, H. S. Behera, Normalize Time Series and Forecast using Evolutionary Neural Network, International Journal of Engineering Research & Technology, IJERT, Vol. 2 Issue 9, September (2013)

[Srinivasan, 1998] Srinivasan, D. Evolving artificial neural networks for short term load forecasting Neurocomputing, 23(1), 265-276.(1998).

[Stanley et al., 2002] Stanley, K. O., & Miikkulainen, R. (2002). Evolving neural networks through augmenting topologies. Evolutionary computation, 10(2), 99-127.

[Tsay, 2010] Tsay R.S.: Analysis of Financial Time Series(2010).

[Tsay ,2005] Tsay, R. S. Analysis of financial time series (Vol. 543). John Wiley & Sons. (2005).

[Van Laarhoven et al., 1987] Van Laarhoven, P. J., & Aarts, E. H. Simulated annealing. In Simulated Annealing: Theory and Applications (pp. 7-15). Springer Netherlands. (1987).

Wang et al.,2003] Wang, J. J., Wang, J. Z., Zhang, Z. G., & Guo, S. P. Stock index forecasting based on a hybrid model. Omega, 40(6), 758-766. (2012).

[Werbos, 1990] Werbos, P. J. Backpropagation through time: what it does and how to do it. Proceedings of the IEEE, 78(10), 1550-1560. (1990).

[Wong ,2014] Wong, B. Introduction to (Generalized) Autoregressive Conditional Heteroskedasticity Models in Time Series Econometrics. (2014)

[Zhang,2003] Zhang, G. P. Time series forecasting using a hybrid ARIMA and neural network model. Neurocomputing, 50, 159-175. (2003).

[Zilouchian et al.,200] Zilouchian, A., & Jamshidi, M. Intelligent control systems using soft computing methodologies. CRC Press, Inc. (2000).