UNIVERSIDAD CATÓLICA SANTA MARÍA

FACULTAD DE CIENCIAS E INGENIERÍAS FÍSICAS Y FORMALES

PROGRAMA PROFESIONAL DE INGENIERÍA ELECTRÓNICA

Diseño de un Algoritmo para Evolucionar Redes Neuronales Artificiales mediante Algoritmos Genéticos

Tesis presentada por el Bachiller:

Jhonatan Calle Cardenas

Para optar el Título Profesional de Ingeniero Electrónico.

2014

- AREQUIPA -

Diseño de un Algoritmo para Evolucionar Redes Neuronales Artificiales mediante Algoritmos Genéticos

por Jhonatan Calle Cárdenas

1. Presentación

El siguiente trabajo de investigación se centra en el tema de Neuro-Evolución (NE), el cual abarca un conjunto de técnicas y métodos para Evolucionar Redes Neuronales Artificiales (ANNs), estos métodos de Evolución pueden considerar a los pesos de las conexiones entre los nodos de una Red Neuronal Artificial, o pueden considerar la existencia de dichas conexiones, o pueden también considerar la arquitectura de la red, es decir, su topología.

Esta investigación estudia los diversos factores que alteran el desempeño de las Redes Neuronales Artificiales, más específicamente, cómo su topología puede alterar drásticamente el dominio del problema para el que la Red está mejor preparada. Además también se discute el problema de cómo codificar el genoma de una Red Neuronal efectivamente, de manera que nos permita alterar su estructura, reproducirla y mutarla, manteniendo durante este proceso los bloques básicos que le dan características típicas a dicha Red.

Para poder estudiar dichos aspectos, nos apoyamos en numerosos antecedentes directa e indirectamente relacionados con la temática de Inteligencia Artificial, Aprendizaje Máquina, Algoritmos Genéticos, Control Automático, etc.

Todo esto con el fin de diseñar e implementar un algoritmo que resuelva la problemática más común en Neuro-Evolución. Dicho algoritmo toma la forma de un programa diseñado para la plataforma de Matlab. El programa desarrollado, como se puede ver posteriormente en la evaluación, es capaz de evolucionar Redes Neuronales con estructura mínima, capaces de resolver problemas de Regresión, Clasificación y Control.

2. Dedicatoria

Le dedico la presente investigación a mi familia, ya que sin su apoyo e infinita paciencia, ésta hubiera sido imposible.

3. Índice

4. Índice de Figuras

5. Índice de Tablas

6. Resumen

La presente Investigación propone un algoritmo novedoso basado en técnicas Neuro-Evolutivas, el cual es capaz de Evolucionar Redes Neuronales Artificiales hacia topologías óptimas para resolver problemas de, por ejemplo, Regresión, Clasificación, Control, etc.

En primer lugar, se presenta un detallado marco teórico, que abarca Redes Neuronales Artificiales, Algoritmos Genéticos y Neuro-Evolución. Aquí se presenta toda la información que se ha considerado relevante, las características más importantes, el funcionamiento y la puesta en marcha de sistemas basados en dichos métodos.

A continuación se hace referencia a numerosas investigaciones que se han hecho en este campo, se estudia dichos antecedentes y se resaltan sobretodo los obstáculos y problemas que se tuvo al momento de elaborar y desarrollar las investigaciones, así como también se pone atención a las alternativas de soluciones que plantean los diferentes autores, se analizan sus ventajas y desventajas, y se yuxtapone todos los trabajos para tener una visión general de la problemática y del estado actual de desarrollo sobre el tema.

Luego de tener un sólido marco teórico y de haber estudiado los antecedentes, se pudo identificar ciertos aspectos de la Neuro-Evolución que pasamos a clasificar como la problemática más común en dicho tema, estos aspectos serían:

- Genotipo en el marco TWEANNs
- Innovar Estructura
- Características de la Población Inicial
- Permutación

Es así que podemos establecer como criterios de diseño para nuestro Algoritmo, la resolución de dicha problemática, y es de esta manera que las siguientes características se proponen para la implementación de un método Neuro-Evolutivo:

- Codificación del Genotipo Neuronal directa
- Marcadores Históricos
- Especiación
- Estructura mínima

Es a partir de éstas características que se diseña un algoritmo genético que sea capaz de evolucionar una población finita de redes neuronales hacia soluciones topológicamente mínimas y que aún así, sean eficaces en su desempeño.

Como parte de la implementación del algoritmo, se presentan las estructuras de datos utilizadas y los diagramas de flujo para las funciones más importantes del programa. Con esto concluye la etapa de diseño e implementación.

Por último, se diseñan y evalúan una series de experimentos que ponen a prueba la capacidad del programa para evolucionar redes neuronales exitosas. Estos experimentos cubren los siguientes dominios:

- Regresión Lineal
- Regresión No Lineal
- Clasificación
- Control

Luego de la evaluación del programa se presentan las conclusiones del trabajo, que en este caso son favorables, ya que las pruebas demuestran que el método es capaz de resolver la problemática planteada.

7. Abstract

This research examines the various factors that affect the performance of Artificial Neural Networks, more specifically, how their topology can drastically alter the problem domain for which the network is better prepared. In addition, we discuss the problem of how to code the genome of a neural network effectively so that we can alter its structure, reproduce and mutate it, while at the same time, maintaining the building blocks that give the Network it's typical characteristics.

All this, in order to design and implement an algorithm to solve the most common problems in Neuro-Evolution:

- Genotype Encoding
- Innovation
- Initial Population Characteristics
- Permutation

This algorithm takes the form of a program designed for the Matlab platform. The program developed is able to evolve neural networks implementing:

- Direct Genotype Encoding
- Historical Markings
- Speciation
- Minimal Structure

Finally, in the evaluation stage we test the algorithm successfully in problems of:

- Regression
- Classification
- Control

The implementation of the algorithm proves satisfactory, as we explain in the conclusions.

8. Identificación del Problema

Las Redes Neuronales Artificiales juegan un papel de importancia entre los métodos de Control Avanzados que se utilizan hoy en día, más aún si consideramos que las aplicaciones de campo que usan dichos métodos aumentan todos los años. Es así que estamos viendo un incremento en la necesidad de implementar dichos agentes para aplicaciones reales, sin embargo, a pesar de ésta necesidad, no se cuenta actualmente con métodos canónicos formales que nos permitan diseñar eficientemente Redes Neuronales Artificiales (RNA o ANN por sus siglas en inglés).

La norma de facto actual para el diseño de la topología de ANNs consiste básicamente en métodos de prueba y error, donde el diseñador determina arbitrariamente, y en la mayoría de los casos, basándose empíricamente en experiencias pasadas, el número de capas escondidas (o Hidden Layers en inglés) y el número de neuronas por capa que conformarán la red.

La necesidad del diseñador de alterar iterativamente la topología de la red para mejorar el rendimiento de la misma demuestra que dicha topología está directamente relacionada con la capacidad de la red para poder encontrar el mínimo óptimo en el espacio de pesos correspondientes a la red.

Por lo tanto, podemos decir que existe un problema real en la falta de una metodología que nos permita diseñar la topología de una ANN independientemente de la aplicación para la que estemos diseñando dicha red.

9. Hipótesis

La presente investigación tiene como Hipótesis principal demostrar que:

"Es posible implementar un algoritmo que diseñe automática y eficazmente la topología de una red neuronal, independientemente del dominio del problema a resolver."

Como Hipótesis secundaria se tiene:

"La incorporación de marcadores históricos, especiación y estructura inicial mínima, en el diseño de un sistema Neuro-Evolutivo, hacen más eficientes el proceso de Evolución y Aprendizaje."

10. Variables

Todas las variables mencionadas en esta sección se detallan más adelante.

10.1. Independientes

- Cantidad de Individuos en Población Inicial.
- Probabilidad de cruce o apareamiento
- Probabilidad de mutación
- Ponderación de Nodos Exceso.
- Ponderación de Nodos Disjuntos.

- Ponderación de Diferencia de Peso Promedio.
- Umbral de distancia para especiación.
- Umbral de estancamiento.
- Umbral de reenfoque.

10.2. Dependientes

- Cantidad de conexiones activas.
- Cantidad de nodos ocultos.
- Aptitud.
- Cantidad de Individuos por Especie.
- Promedio de Cuadrados de Errores (MSE).

11. Objetivos

11.1. Objetivos Generales

Examinar la capacidad de métodos algorítmicos, más específicamente algoritmos genéticos, para poder diseñar eficazmente la topología o arquitectura de una Red Neuronal Artificial independientemente de la aplicación en la que dicha red desee ser implementada.

11.2. Objetivos Específicos

Diseñar e Implementar una aplicación en Matlab que retorne una estructura representativa de la Red Neuronal.

Examinar la eficacia de dicho algoritmo para entrenar la red además de diseñarla topológicamente.

12. Enfoque

La presente investigación es de tipo fundamental, ya que al plantear examinar la eficacia de un método, la misma adquiere un carácter exploratorio. Y la metodología es cualitativa, ya que su propuesta se centra en la formulación de un algoritmo y de su posterior modelado sobre una plataforma específica de software.

13. Alcance

La propuesta actual tiene un alcance exploratorio, es así que delimitamos la investigación a hallar pruebas que verifiquen o refuten la factibilidad de que un algoritmo genético pueda diseñar una red neuronal de topología arbitraria, por medio de aprendizaje supervisado, y que dicha red sea capaz de resolver problemas de carácter general, como regresión y clasificación.

De hallarse eficaz el algoritmo planteado, la aplicación del mismo cae en todos aquellos campos en los que hoy en día se utilizan Redes Neuronales Artificiales, como:

- Aproximación de Funciones
- Análisis de Regresión
- Predicción de Series de Tiempo
- Clasificación
- Reconocimiento de Patrones
- Procesamiento de Datos
- Clustering
- Robótica
- Control
- etc.

14. Marco Teórico

14.1. Redes Neuronales Artificiales

En ciencias de la computación y afines, las redes neuronales artificiales son modelos computacionales inspirados en el sistema nervioso central de animales (especialmente el cerebro) que son capaces de aprendizaje automático y reconocimiento de patrones. Por lo general, se presentan como sistemas de "neuronas" interconectadas que pueden calcular valores a partir de entradas al alimentar de información a la red.

Por ejemplo, en una red neuronal para el reconocimiento de escritura, un conjunto de neuronas de entrada pueden ser activadas por los píxeles de una imagen de entrada que representa una letra o dígito. Las activaciones de estas neuronas se pasan, ponderadas y transformadas por alguna función determinada por el diseñador de la red, a otras neuronas, etc., hasta que finalmente una neurona de salida se activa que determina el caracter que se leyó.

Al igual que otros métodos de Machine Learning, las redes neuronales se han utilizado para resolver una amplia variedad de tareas que son difíciles de resolver utilizando programación basada en reglas, incluyendo visión por ordenador y reconocimiento de voz.

14.1.1. Modelos

Modelos de redes neuronales en inteligencia artificial son referidos generalmente como redes neuronales artificiales (RNA), que son esencialmente simples modelos matemáticos que definen una función f: X -> Y o una distribución sobre X o X e Y, pero a veces los modelos están íntimamente asociados a un algoritmo de aprendizaje en particular o regla de aprendizaje. Un uso común de la frase "modelo de RNA" realmente significa la definición de una clase de tales funciones (donde los miembros de la clase se obtienen variando parámetros como los pesos de conexión, o específicos de la arquitectura, tales como el número de neuronas o su conectividad).

14.1.1.1. Función de Red

La palabra red en el término "red neuronal artificial" se refiere a las interconexiones entre las neuronas en las diferentes capas de cada sistema. Un sistema de ejemplo tiene tres capas. La primera capa tiene neuronas de entrada, que envían datos a través de las sinapsis a la segunda capa de neuronas, y luego a través de más sinapsis a la tercera capa de neuronas de salida. Los sistemas más complejos tendrán más capas de neuronas con algunos que tienen mayores capas de neuronas de entrada y neuronas de salida. Las sinapsis almacenan parámetros llamados "pesos" que manipulan los datos en los cálculos.

Una RNA se define normalmente por tres tipos de parámetros:

- La estructura de interconexión entre diferentes capas de neuronas.

- El proceso de aprendizaje para la actualización de los pesos de las interconexiones .
- La función de activación que convierte la entrada ponderada de una neurona a su salida.

Matemáticamente, la función de red de una neurona f(x) se define como una composición de otras funciones $g_i(x)$, que además se puede definir como una composición de otras funciones. Esto puede ser convenientemente representado como una estructura de red, con flechas que representan las dependencias entre las variables. Un tipo de composición ampliamente utilizado es la suma ponderada no lineal, donde $f(x) = K (\Sigma_i w_i.g_i(x))$, donde K (comúnmente conocida como la función de activación) es una función predefinida, como la tangente hiperbólica. Será conveniente para lo siguiente, referirse a una colección de funciones gi simplemente como un vector $g = (g_1, g_2 , ... , g_n)$.

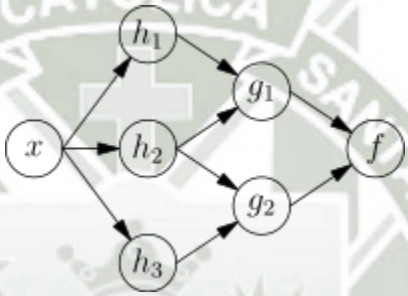

Figura 14-1 Representación gráfica de una RNA

La Figura 14-1 representa tal descomposición de f, con dependencias entre las variables indicadas por flechas. Esto puede ser interpretado de dos maneras.

El primer punto de vista es el funcional: la entrada x se transforma en un vector de 3 dimensiones de h, que luego se transforma en un vector de 2 dimensiones de g, que finalmente se transforma en f. Este punto de vista se encuentra más comúnmente en el contexto de optimización.

El segundo punto de vista es el probabilístico: la variable aleatoria F = f(G) depende de la variable aleatoria G = g(H), que depende de H = h(X), que depende de la variable aleatoria X. Este punto de vista es más comúnmente encontrado en el contexto de modelos gráficos.

Los dos puntos de vista son en gran medida equivalentes. En cualquier caso, para esta arquitectura de red en particular, los componentes de las capas individuales son independientes el uno del otro (por ejemplo, los componentes de g son independientes el uno del otro debido a su entrada h). Esto permite, naturalmente, un grado de paralelismo en la ejecución.

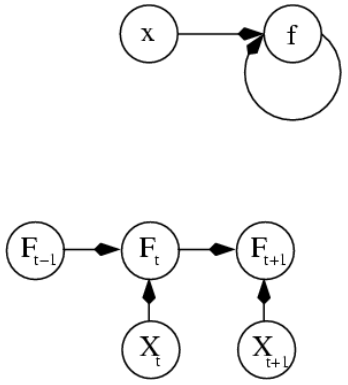

Figura 14-2 Dos representaciones de Redes Recurrentes

Las redes como la anterior se denominan comúnmente feedforward (de alimentación directa), porque su gráfico es un gráfico acíclico dirigido. Redes con ciclos se denominan comúnmente recurrentes. Tales redes se representan comúnmente en la manera mostrada en la parte superior de la Figura 14-2, donde f se muestra como dependiente sobre sí misma. Sin embargo, no se muestra una dependencia temporal implícita.

14.1.1.2. Aprendizaje

Lo que ha atraído el mayor interés en redes neuronales es la posibilidad de aprender. Dada una tarea específica a resolver, y una clase de funciones F, el aprendizaje consiste en utilizar un conjunto de observaciones para encontrar $f^* \in F$ que resuelve la tarea en algún sentido óptimo.

Esto implica la definición de una función de costo C: F -> R tal que, para la solución óptima f^*, $C(f^*) \leq C(f) \; \forall \; f \in F$ - es decir, no hay solución que tenga un costo menor que el costo de la solución óptima.

La función de coste C es un concepto importante en el aprendizaje, ya que es una medida de la distancia a la que una solución particular está de una solución óptima al problema a ser resuelto. Algoritmos de aprendizaje buscan a través del espacio de soluciones para encontrar una función que tiene el menor costo posible.

Para aplicaciones en las que la solución depende de algunos datos, el costo tiene que ser necesariamente una función de las observaciones, de lo contrario no estaríamos modelando nada relacionado con los datos. Se define con frecuencia como una estadística a la que se pueden hacer sólo aproximaciones. Como un simple ejemplo, considerar el problema de encontrar el modelo f, que minimice $C = E[(f(x) - y)^2]$, para pares de datos (x, y) dibujado de alguna distribución D. En situaciones prácticas sólo tendríamos N muestras de D y por lo tanto, para el ejemplo anterior, sólo minimizaríamos $C = 1/N * \sum^{N}_{i=1}(f(x_i) - y_i)^2$. Por lo tanto, el costo se reduce al mínimo sobre una muestra de los datos en lugar de todo el conjunto de datos.

Cuando N -> ∞ algún tipo de Machine Learning (aprendizaje máquina) en línea se debe usar, donde el costo se minimiza parcialmente cuando se ve cada nuevo ejemplo. Mientras que Machine Learning en línea se utiliza a menudo cuando D es fija, es más útil en el caso en el que la distribución cambia lentamente con el tiempo. En métodos de redes neuronales, alguna forma de Machine Learning en línea se utiliza con frecuencia para conjuntos de datos finitos.

14.1.1.2.1. Escogiendo una función Costo

Si bien es posible definir alguna función de coste ad hoc arbitraria, con frecuencia se utilizará un costo en particular, ya sea debido a que tiene propiedades deseables (tales como convexidad) o porque surge naturalmente a partir de una formulación particular del problema (por ejemplo, en un formulación probabilística la probabilidad posterior del modelo puede ser utilizada como un costo inverso). En última instancia, la función de coste dependerá de la tarea deseada. Una visión general de las tres categorías principales de tareas de aprendizaje se proporciona a continuación.

14.1.1.3. Paradigmas de Aprendizaje

Hay tres principales paradigmas de aprendizaje, cada uno correspondiente a una particular tarea de aprendizaje abstracto. Estos son aprendizaje supervisado, aprendizaje no supervisado y aprendizaje por reforzamiento.

14.1.1.3.1. Aprendizaje Supervisado

En el aprendizaje supervisado, se nos da una serie de ejemplos de pares (x , y) , x Є X, y Є Y, y el objetivo es encontrar una función f: X -> Y en la clase permitida de funciones, que corresponda con los ejemplos. En otras palabras, deseamos inferir el mapeo implícito en los datos, la función de costo está relacionada con la falta de coincidencia entre nuestro mapeo y los datos, y contiene implícitamente conocimiento previo sobre el problema.

Un costo comúnmente utilizado es el error cuadrático medio (mean squared error MSE), que trata de minimizar el error cuadrático medio entre la salida de la red, f(x), y el valor objetivo y, sobre todos los pares de ejemplo. Cuando uno trata de minimizar este costo usando descenso de gradiente para la clase de redes neuronales llamados perceptrones multicapa, se obtiene el común y bien conocido algoritmo de retropropagación para entrenar redes neuronales.

Tareas que caen dentro del paradigma de aprendizaje supervisado son el reconocimiento de patrones (también conocido como clasificación) y la regresión (también conocida como aproximación de funciones). El paradigma de aprendizaje supervisado es también aplicable a datos secuenciales (por ejemplo, para el reconocimiento de habla y gestos). Podemos pensar en esto como el aprender con un "maestro", en la forma de una función que proporciona

información continua sobre la calidad de las soluciones obtenidas hasta el momento.

14.1.1.3.2. Aprendizaje No Supervisado

En el aprendizaje no supervisado, se dan algunos datos x y la función de costo a minimizar, que puede ser cualquier función de los datos x y salida de la red, f.

La función de costo depende de la tarea (lo que estamos tratando de modelar) y los supuestos a priori (las propiedades implícitas de nuestro modelo, los parámetros y las variables observadas).

Como un ejemplo trivial, tengamos en cuenta el modelo de f(x) = a, donde a es una constante y el costo C = E[(x - f(x))^2]. Minimizar este costo nos dará un valor de a igual a la media de los datos. La función de coste puede ser mucho más complicada. Su forma depende de la aplicación: por ejemplo, en compresión podría estar relacionada con la información mutua entre x y f(x), mientras que en modelado estadístico, podría estar relacionada con la probabilidad posterior del modelo dado los datos. (Se debe tener en cuenta que en ambos ejemplos esas cantidades se maximizan en lugar minimizarse).

Tareas que caen dentro del paradigma de aprendizaje no supervisado están en problemas generales de estimación; aplicaciones incluyen clustering, estimación de distribuciones estadísticas, compresión y filtrado.

14.1.1.3.3. Aprendizaje por Reforzamiento

En el aprendizaje por reforzamiento, por lo general no se dan datos x, pero se generan por la interacción del agente con el medio ambiente. En cada punto en el tiempo t, el agente realiza una acción y_t y el medio ambiente genera una observación x_t y un costo instantáneo c_t, de acuerdo con alguna dinámica (por lo general desconocida). El objetivo es descubrir una política de selección de acciones que minimice en alguna medida un costo a largo plazo, es decir, el costo acumulado esperado. La dinámica del medio ambiente y el costo a largo plazo para cada política son generalmente desconocidos, pero se pueden estimar.

Más formalmente, el medio ambiente se modela como un proceso de decisión de Markov (MDP) con los estados s_1, ... , s_n Є S y acciones a_1, ... , a_m Є A con las siguientes distribuciones de probabilidad: la distribución de costos instantáneos P(c_t | s_t), la distribución de observación P(x_t | s_t) y la transición P(s_{t+1} | s_t , a_t), mientras que una política se define como la distribución condicional sobre las acciones dadas las observaciones. Tomados en conjunto, los dos definen una cadena de Markov (MC). El objetivo es descubrir la política que minimice el costo, es decir, la MC para la que el costo es mínimo.

RNAs se utilizan con frecuencia en el aprendizaje por reforzamiento como parte del algoritmo general. La programación dinámica se ha acoplado con RNAs (Neuro programación dinámica) por Bertsekas y Tsitsiklis y aplicado a problemas no lineales multidimensionales tales como los implicados en enrutamiento de vehículos, gestión de recursos naturales o medicina, debido a la capacidad de las RNAs para mitigar las pérdidas de precisión, incluso al reducir la densidad de la malla de discretización para aproximar numéricamente la solución de los problemas de control originales.

Las tareas que están dentro del paradigma del aprendizaje por reforzamiento son problemas de control, juegos y otras tareas de decisión secuencial.

14.1.1.4. Algoritmos de Aprendizaje

El entrenamiento de un modelo de red neuronal esencialmente significa seleccionar un modelo a partir del conjunto de modelos permitidos (o, en un marco bayesiano, la determinación de una distribución sobre el conjunto de modelos permitidos) que minimiza el criterio de costo. Hay numerosos algoritmos disponibles para entrenar modelos de redes neuronales, la mayoría de ellos pueden ser vistos como una aplicación directa de la teoría de optimización y estimación estadística.

La mayoría de los algoritmos utilizados en el entrenamiento de redes neuronales artificiales emplean alguna forma de descenso de gradiente. Esto se hace simplemente tomando la derivada de la función de coste con respecto a los parámetros de la red y a continuación, cambiar los parámetros en una dirección relacionada con el gradiente.

Métodos evolutivos, programación de expresión genética, maximización de expectativa, métodos no paramétricos y optimización de enjambre de partículas son algunos de los métodos utilizados para el entrenamiento de redes neuronales.

14.1.2. Propiedades Teóricas

14.1.2.1. Poder Computacional

El perceptrón multicapa (MLP) es un aproximador de funciones universal, como lo demuestra el teorema de Cybenko. Sin embargo, la prueba no es constructiva con respecto al número de neuronas requeridas o los ajustes de los pesos.

Trabajo por Hava Siegelmann y Eduardo D. Sontag ha proporcionado prueba de que una específica arquitectura recurrente con valores racionales para los pesos (a diferencia de pesos con valores de precisión completa de números reales) tiene toda la potencia de una Máquina de Turing Universal utilizando un número finito de neuronas y conexiones lineales estándar. Ellos han demostrado, además, que el uso de valores irracionales para los pesos resulta en una máquina de Turing con súper potencia.

14.1.2.2. Capacidad

Los modelos de redes neuronales artificiales tienen una propiedad llamada "capacidad", que corresponde aproximadamente a su capacidad para modelar cualquier función dada. Se relaciona con la cantidad de información que puede ser almacenada en la red y a la noción de complejidad.

14.1.2.3. Convergencia

No se puede decir nada en general sobre la convergencia ya que depende de un número de factores. En primer lugar, pueden existir muchos mínimos locales. Esto depende de la función de costo y el modelo. En segundo lugar, el método de optimización utilizado puede no garantizar el converger cuando se encuentra lejos de un mínimo local. En tercer lugar, para una cantidad muy grande de datos o parámetros, algunos de los métodos se vuelven poco práctico. En general, se ha encontrado que las garantías teóricas con respecto a la convergencia no son un indicador fiable de la aplicación práctica.

14.1.2.4. Generalización y Estadística

En aplicaciones donde el objetivo es crear un sistema que generaliza también en ejemplos no vistos, ha surgido el problema de exceso de entrenamiento. Esto surge en sistemas complicados o sobre - especificados cuando la capacidad de la red supera significativamente los parámetros libres necesarios. Hay dos escuelas de pensamiento para evitar este problema: La primera es utilizar la validación cruzada y técnicas similares para detectar la presencia de exceso de entrenamiento y de manera óptima seleccionar hiperparámetros tales que minimicen el error de generalización. La segunda es utilizar algún tipo de regularización. Este es un concepto que surge de manera natural en un marco probabilístico (Bayesiano), donde la regularización puede llevarse a cabo mediante la selección de una probabilidad a priori más grande sobre modelos más simples, pero también en teoría del aprendizaje estadístico, donde el objetivo es minimizar sobre dos cantidades: el "riesgo empírico" y el "riesgo estructural", que corresponden aproximadamente al error sobre el conjunto de entrenamiento y el error de predicción en datos que no se ven debido a exceso de entrenamiento.

Redes neuronales supervisadas que utilizan una función de coste MSE pueden utilizar métodos estadísticos formales para determinar la confianza del modelo entrenado. El MSE en un conjunto de validación se puede utilizar como una estimación para la varianza. Este valor se puede utilizar para calcular el intervalo de confianza de la salida de la red, suponiendo una distribución normal. Un análisis de confianza hecho de esta manera es estadísticamente válido, siempre y cuando la distribución de probabilidad de salida permanezca igual y la red no sea modificada.

Mediante la asignación de una función de activación softmax, una generalización de la función logística, en la capa de salida de la red neuronal (o un componente softmax

en una red neural basado en componentes) para las variables objetivo categóricas, las salidas pueden ser interpretados como probabilidades a posteriori. Esto es muy útil en clasificación, ya que da una medida de seguridad en las clasificaciones.

La función de activación softmax es:

$$y_i = \frac{e^{x_i}}{\sum_{j=1}^{c} e^{x_j}}$$

14.1.2.5. Propiedades Dinámicas

Diversas técnicas desarrolladas originalmente para el estudio de sistemas magnéticos desordenados (por ejemplo, el vidrio de espín) se han aplicado con éxito a arquitecturas de redes neurales simples, tales como la red de Hopfield. Influyente obra de E. Gardner y B. Derrida ha revelado muchas propiedades interesantes sobre perceptrones con pesos sinápticos de valor real, mientras que más tarde el trabajo de W. y M. Krauth Mézard ha extendido estos principios a las sinapsis de valor binario.

14.2. Algoritmos Genéticos

En el campo de la informática de inteligencia artificial, un algoritmo genético (GA) es una heurística de búsqueda que imita el proceso de selección natural. Esta heurística (también llamada metaheurística) se utiliza rutinariamente para generar soluciones útiles a problemas de búsqueda y optimización. Los algoritmos genéticos pertenecen a la clase más grande de los algoritmos evolutivos (AE), que generan soluciones a problemas de optimización con técnicas inspiradas en la evolución natural, como la herencia, mutación, selección y cruce.

Los algoritmos genéticos encuentran aplicación en bioinformática, filogenética, ciencias de la computación, ingeniería, economía, química, fabricación, matemáticas, física, farmacometría y otros campos.

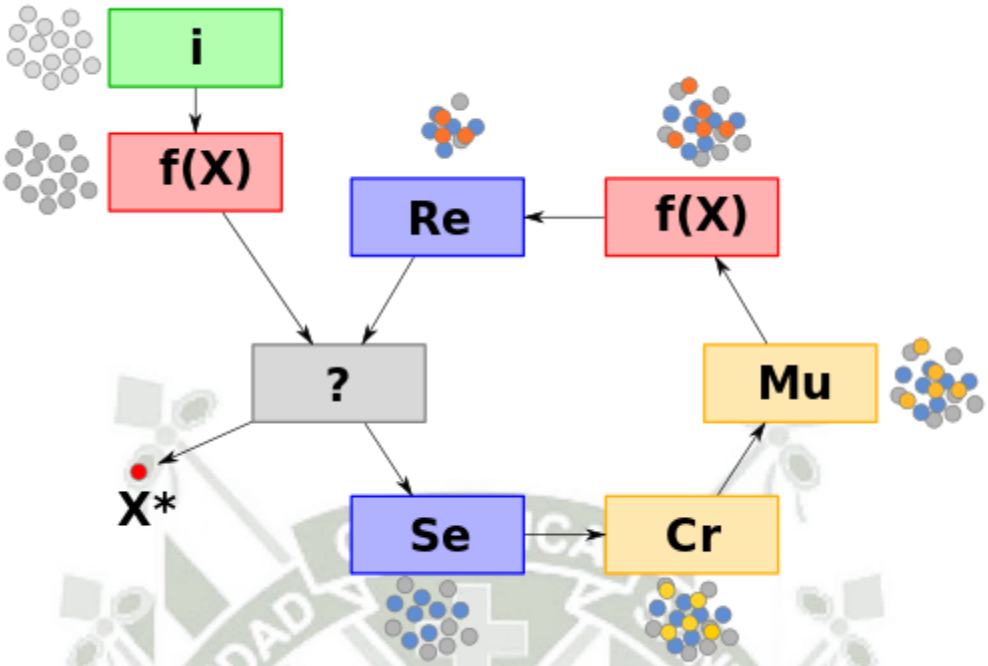

Figura 14-3 Diagrama de Flujo para Algoritmos Genéticos

14.2.1. Metodología

En un algoritmo genético, una población de soluciones candidatas (llamadas individuos, criaturas, o fenotipos) a un problema de optimización, se evolucionan hacia mejores soluciones. Cada solución candidata tiene un conjunto de propiedades (sus cromosomas o genotipo) que pueden ser mutadas y alteradas; tradicionalmente, las soluciones están representados en binario como cadenas de 0s y 1s, pero otras codificaciones son también posibles.

La evolución por lo general comienza a partir de una población de individuos generados al azar, y es un proceso iterativo, a la población en cada iteración se la llama generación. En cada generación, se evalúa la aptitud de cada individuo en la población; la aptitud es por lo general el valor de la función objetivo en el problema de optimización que se resuelve. Los individuos más aptos son estocásticamente seleccionados de la población actual, y el genoma de cada individuo se modifica (recombinado y posiblemente mutado aleatoriamente) para formar una nueva generación. La nueva generación de soluciones candidatas se utiliza a continuación, en la siguiente iteración del algoritmo. Comúnmente, el algoritmo termina cuando se ha producido un número máximo de generaciones, o se ha alcanzado un nivel de aptitud satisfactoria para la población.

Un algoritmo genético típico requiere :

- Una representación genética del dominio de soluciones,

- Una función de aptitud para evaluar el dominio de soluciones.

Una representación estándar de cada solución candidata es como una matriz de bits. Matrices de otros tipos y estructuras se pueden utilizar esencialmente de la misma manera. La propiedad principal que hace que estas representaciones genéticas sean convenientes, es que sus partes se alinean fácilmente debido a su tamaño fijo, lo que facilita las operaciones de cruce simples. Representaciones de longitud variable también se pueden utilizar, pero el cruce es más complejo en este caso. Representaciones de árbol se exploran en programación genética y representaciones de grafo se exploran en programación evolutiva , una mezcla de ambos cromosomas lineales y árboles se explora en la programación de expresión génica.

Una vez que la representación genética y la función de aptitud se definen, un GA procede a inicializar una población de soluciones y luego a mejorarla a través de la aplicación repetitiva de los operadores de mutación, cruce, inversión y selección.

14.2.1.1. Inicialización

Inicialmente muchas soluciones individuales son (por lo general) generadas aleatoriamente para formar una población inicial. El tamaño de la población depende de la naturaleza del problema, pero normalmente contiene varios cientos o miles de posibles soluciones. Tradicionalmente, se genera la población al azar, permitiendo toda la gama de posibles soluciones (el espacio de búsqueda). En ocasiones, las soluciones pueden ser "sembradas" en las zonas donde es probable encontrar soluciones óptimas.

14.2.1.2. Selección

Durante cada generación sucesiva, una proporción de la población existente se selecciona para crear una nueva generación. Las soluciones individuales se seleccionan a través de un proceso basado en aptitud, donde soluciones más adecuadas (tal como se miden por una función de aptitud) son típicamente más probables a ser seleccionadas. Ciertos métodos de selección califican la aptitud de cada solución y preferentemente seleccionan las mejores soluciones. Otros métodos califican sólo una muestra aleatoria de la población, ya que el primer proceso puede tomar mucho tiempo.

La función de aptitud se define sobre la representación genética y mide la calidad de la solución representada. La función de aptitud siempre depende del problema. Por ejemplo, en el problema de la mochila se quiere maximizar el valor total de los objetos que se pueden poner en una mochila de cierta capacidad fija. Una representación de una solución podría ser una matriz de bits, donde cada bit representa un objeto diferente, y el valor del bit (0 o 1) representa ya sea o no que el objeto está en la mochila. No todas tales representaciones son válidas, ya que el tamaño de los objetos puede exceder la capacidad de la mochila. La aptitud de la solución es la suma de los

valores de todos los objetos en la mochila si la representación es válida, o 0 en caso contrario.

En algunos problemas, es difícil o incluso imposible definir la expresión de aptitud, en estos casos, una simulación puede ser utilizada para determinar el valor de la función de aptitud de un fenotipo (por ejemplo, la dinámica de fluidos computacional se utiliza para determinar la resistencia al aire de un vehículo cuya forma se codifica en el fenotipo), o incluso se utilizan algoritmos genéticos interactivos.

14.2.1.3. Operadores Genéticos

El siguiente paso es generar la población de la segunda generación de soluciones de los seleccionados a través de operadores genéticos: cruce (también llamado recombinación), y / o mutación.

Para cada nueva solución a ser producida, se selecciona un par de soluciones "padres" del grupo previamente seleccionado. Al producir una solución "hijo" con los métodos anteriores de cruce y mutación, se crea una nueva solución que normalmente comparte muchas de las características de sus "padres". Los padres son seleccionados para cada nuevo "hijo", y el proceso continúa hasta que se genera una nueva población de soluciones de tamaño adecuado. Aunque los métodos de reproducción que se basan en el uso de dos padres son más "inspirados en biología" , algunas investigaciones sugieren que más de dos "padres" generan cromosomas de mayor calidad.

Estos procesos dan como resultado en última instancia, a la próxima generación de población de cromosomas que es diferente de la generación inicial. En general, la aptitud media habrá aumentado en este procedimiento para la población, ya que sólo los mejores organismos de la primera generación se seleccionan para aparear, junto con una pequeña proporción de soluciones menos aptas. Estas soluciones menos aptas aseguran la diversidad genética dentro de la reserva genética de los padres y por lo tanto garantizan la diversidad genética de la posterior generación de "hijos".

La opinión está dividida sobre la importancia del cruce frente a la mutación. Hay muchas referencias en Fogel (2006) que apoyan la importancia de la búsqueda basada en mutación.

A pesar de que el cruce y la mutación se conocen como los principales operadores genéticos, es posible utilizar otros operadores tales como reagrupamiento, colonización - extinción , o la migración.

Vale la pena ajustar parámetros como la probabilidad de mutación, probabilidad de cruce y el tamaño de la población para encontrar ajustes razonables a la clase de problema que se está trabajando. Una tasa de mutación muy pequeña puede conducir a la deriva genética. Una tasa de recombinación que es demasiado alta puede

conducir a la convergencia prematura del algoritmo genético. Una tasa de mutación que es demasiado alta puede conducir a la pérdida de una buena solución si no hay selección elitista. Existen límites superiores e inferiores teóricos, pero no prácticos de estos parámetros, que pueden ayudar a guiar la selección a través de experimentos.

14.2.1.4. Terminación

Este proceso generacional se repite hasta que se ha alcanzado una condición de terminación. Condiciones de terminación comunes son:

- Se encuentra una solución que satisfaga los criterios mínimos.
- Número fijo de generaciones se alcanzó.
- Presupuesto asignado (tiempo de cálculo / dinero) se alcanzó.
- Idoneidad de la solución de más alto rango se está alcanzando o ha alcanzado un nivel tal que iteraciones sucesivas ya no producen mejores resultados.
- Inspección manual.
- Combinaciones de las anteriores.

14.2.2. Limitaciones

Existen varias limitaciones en la utilización de un algoritmo genético en comparación con algoritmos de optimización alternativos:

Evaluación repetida de la función de aptitud para problemas complejos a menudo es el segmento más prohibitivo y limitante de los algoritmos evolutivos artificiales. Encontrar la solución óptima a problemas complejos, altamente dimensionales y multimodales a menudo requiere muy costosas evaluaciones de la función de aptitud. En problemas del mundo real, tales como problemas de optimización estructural, una única evaluación de la función puede requerir desde varias horas a varios días de simulación para completar. Métodos de optimización típicos no pueden hacer frente a ese tipo de problema. En este caso, puede ser necesario renunciar a una evaluación exacta y utilizar una aptitud aproximada que sea computacionalmente eficiente. Es evidente que la fusión de modelos aproximados puede ser uno de los enfoques más prometedores para utilizar convincentemente GA para resolver problemas complejos de la vida real.

Los algoritmos genéticos no escalan bien con la complejidad. Es decir, donde el número de elementos que están expuestos a la mutación es grande a menudo hay un aumento exponencial de tamaño del espacio de búsqueda. Esto hace que sea muy difícil de usar la técnica en problemas tales como el diseño de un motor, una casa o un avión. Para hacer este tipo de problemas manejables en búsqueda evolutiva, deben ser divididos en la representación más simple posible. Por lo tanto vemos típicamente algoritmos evolutivos que codifican el diseño de las aspas del ventilador en lugar del de motores, la construcción de formas en vez de planos detallados de construcción, perfiles aerodinámicos en lugar de diseños de aeronaves completas. El segundo problema de la complejidad es la cuestión de cómo proteger piezas que se han desarrollado para representar buenas soluciones, de

mutación destructiva, sobre todo cuando su evaluación de aptitud requiere que combinen bien con otras partes. Se ha sugerido por algunos miembros de la comunidad que un enfoque de desarrollo para soluciones evolucionadas, podría superar algunos de los problemas de protección, pero esto sigue siendo un tema de investigación.

La "mejor" solución lo es sólo en comparación con otras soluciones. Como resultado, el criterio de parada no es claro en todos los problemas.

En muchos problemas, GAs pueden tener una tendencia a converger hacia óptimos locales o incluso puntos arbitrarios en lugar del óptimo global del problema. Esto quiere decir que no "sabe cómo" sacrificar aptitud a corto plazo para ganar mejor aptitud en el largo plazo. La probabilidad de que esto ocurra depende de la forma del "terreno de aptitud": ciertos problemas pueden proporcionar un fácil ascenso hacia un óptimo global, otros pueden hacer más fácil para la función el encontrar un óptimo local. Este problema puede ser aliviado mediante el uso de una función de aptitud diferente, el aumento de la tasa de mutación, o mediante el uso de técnicas de selección que mantengan una población de soluciones diversa, aunque el teorema de "No hay almuerzo gratis" demuestra que no existe una solución general a este problema. Una técnica común para mantener la diversidad es imponer una "penalidad de nicho", en la que, cualquier grupo de individuos que tienen suficiente similitud (radio de nicho) tienen una penalización añadida, lo que reducirá la representación de ese grupo en generaciones posteriores, permitiendo a otros (menos similares) individuos mantenerse en la población. Este truco, sin embargo, puede no ser eficaz, en función del problema. Otra técnica posible sería simplemente sustituir una parte de la población con individuos generados al azar, cuando la mayoría de la población es demasiado similar entre sí. La diversidad es importante en algoritmos genéticos (y programación genética) porque el cruce sobre una población homogénea no produce nuevas soluciones. En las estrategias de evolución y programación evolutiva, la diversidad no es esencial debido a una mayor dependencia en la mutación.

Operar en conjuntos de datos dinámicos es difícil, ya que los genomas comienzan a converger desde el principio, hacia soluciones que pueden no ser válidas para datos posteriores. Se han propuesto varios métodos para remediar esto aumentando la diversidad genética de alguna manera y previniendo la convergencia temprana, ya sea mediante el aumento de la probabilidad de mutación cuando la calidad de la soluciones baja (llamada hipermutación gatillada), o por la introducción ocasional de elementos completamente nuevos generados al azar, a la piscina genética (llamados inmigrantes aleatorios). Una vez más, las estrategias de evolución y programación evolutiva pueden implementarse con una llamada "estrategia de coma" en la que los padres no se mantienen y los nuevos padres se seleccionan sólo en la descendencia. Esto puede ser más eficaz en problemas dinámicos.

Los GAs no pueden resolver eficazmente problemas en los que la única medida de aptitud es una medida única de bueno / malo (como problemas de decisión), ya que no hay

manera de converger en la solución (no hay colina que subir). En estos casos, una búsqueda al azar puede encontrar una solución tan rápidamente como un GA. Sin embargo, si la situación permite ensayos de éxito / fracaso repetidos, dando (posiblemente) resultados diferentes, entonces la relación entre éxitos y fallas proporciona una medida de aptitud adecuada.

Para problemas de optimización específicos e instancias de problema, otros algoritmos de optimización pueden encontrar mejores soluciones que los algoritmos genéticos (dada la misma cantidad de tiempo de cálculo). Algoritmos alternativos y complementarios incluyen estrategias de evolución, programación evolutiva, recorrido simulado, adaptación de Gauss, algoritmo de escalada, inteligencia de enjambre, y métodos basados en programación lineal de enteros. La cuestión de qué problema, de serlo alguno, es adecuado para algoritmos genéticos (en el sentido de que tales algoritmos son mejores que otros) es abierta y controvertida.

14.2.3. Variantes

14.2.3.1. Representación del Cromosoma

El algoritmo más simple representa cada cromosoma como una cadena de bits. Típicamente, los parámetros numéricos pueden ser representados por números enteros, aunque es posible utilizar representaciones de punto flotante. La representación de punto flotante es natural para las estrategias de evolución y programación evolutiva. La idea de algoritmos genéticos de valor real se ha ofrecido, pero es realmente un nombre inapropiado, ya que en realidad no representan a la teoría de los bloques que fue propuesto por John Henry Holland en la década de 1970. Esta teoría no está sin apoyo, sobre la base de resultados teóricos y experimentales. El algoritmo básico realiza cruces y mutaciones a nivel de bit. Otras variantes tratan al cromosoma como una lista de números que son índices en una tabla de instrucciones, nodos de una lista enlazada, hashes, objetos o cualquier otra estructura de datos imaginable. Cruce y mutación se realizan con el fin de respetar los límites de elementos de datos. Para la mayoría de los tipos de datos, operadores de variación específicos pueden ser diseñados. Diferentes tipos de datos cromosómicos parecen funcionar mejor o peor para diferentes dominios específicos de problemas.

Cuando se utilizan representaciones de cadena de bits de números enteros, se emplea a menudo código Gray. De esta manera, pequeños cambios en el número entero pueden efectuarse inmediatamente a través de mutaciones o cruces. Se ha encontrado que esto ayuda a prevenir la convergencia prematura en las llamadas paredes Hamming, en el que muchas mutaciones simultáneas (o eventos de cruce) deben ocurrir con el fin de cambiar el cromosoma a una mejor solución.

Otros métodos incluyen el uso de matrices de números con valores reales en lugar de cadenas de bits para representar los cromosomas. Los resultados de la teoría de los esquemas sugieren que, en general, cuanto menor es el alfabeto, mejor será el

rendimiento, pero fue inicialmente sorprendente para los investigadores que se hayan obtenido buenos resultados con el uso de cromosomas con valores reales. Esto se explicó como el conjunto de valores reales en una población finita de cromosomas estaba formando un alfabeto virtual (cuando selección y recombinación son dominantes) con una cardinalidad mucho menor de lo que cabría esperar de una representación de punto flotante.

14.2.3.2. Elitismo

Una (ligera) variante muy exitosa del proceso general de construcción de una nueva población es permitir que algunos de los mejores organismos de la generación actual pasen a la siguiente, sin alteraciones. Esta estrategia se conoce como selección elitista.

14.2.3.3. Implementaciones paralelas

Las implementaciones paralelas de algoritmos genéticos son de dos tipos. Algoritmos genéticos paralelos de "grano grueso" asumen una población en cada uno de los nodos del ordenador y la migración de los individuos entre los nodos. Algoritmos genéticos paralelos de "grano fino" suponen un individuo en cada nodo del procesador que actúa con individuos vecinos para selección y reproducción. Otras variantes, como los algoritmos genéticos para problemas de optimización en línea, presentan dependencia temporal o ruido en la función de aptitud.

14.2.3.4. Algoritmos Genéticos Adaptativos

Los algoritmos genéticos con parámetros adaptativos (algoritmos genéticos adaptativos , AGAs) son una variante importante y prometedora de los algoritmos genéticos. Las probabilidades de cruce (pc) y la mutación (pm) determinan en gran medida el grado de precisión de la solución y la velocidad de convergencia que los algoritmos genéticos pueden obtener. En lugar de utilizar valores fijos de pc y pm, AGAs utilizan la información de la población en cada generación y adaptativamente ajustan el pc y pm con el fin de mantener la diversidad de la población, así como para mantener la capacidad de convergencia. En AGA (algoritmo genético adaptativo), el ajuste de pc y pm depende de los valores de aptitud de las soluciones. En CAGA (clustering-based adaptive genetic algorithm), a través del uso de análisis de grupo se juzgan los estados de optimización de la población, el ajuste de pc y pm depende de la optimización de estos estados. Puede ser muy efectivo combinar GA con otros métodos de optimización. GA tiende a ser bastante bueno en encontrar buenas soluciones globales, pero bastante ineficaz en encontrar las últimas mutaciones para encontrar el óptimo absoluto. Otras técnicas (como simple hill climbing) son muy eficientes en la búsqueda de un óptimo absoluto en una región limitada. Alternando GA y simple hill climbing se puede mejorar la eficiencia del GA, mientras que se supera la falta de solidez de simple hill climbing.

Esto significa que las reglas de variación genética pueden tener un significado diferente en el caso natural. Por ejemplo - a condición de que las medidas se

almacenen en orden consecutivo - cruzar puede sumar una serie de escalones de ADN materno añadiendo una serie de escalones del ADN paterno y así sucesivamente. Esto es como la adición de vectores que más probablemente pueden seguir una cordillera en el paisaje fenotípico. Por lo tanto, la eficiencia del proceso se puede aumentar en varios órdenes de magnitud. Por otra parte, el operador de inversión tiene la oportunidad de colocar escalones en orden consecutivo o en cualquier otro orden adecuado a favor de la supervivencia o la eficiencia.

Una variación, donde la población en su conjunto se evoluciona en lugar de sus miembros individuales, se conoce como recombinación de piscina genética.

Un número de variaciones se han desarrollado para tratar de mejorar el rendimiento de GAs en problemas con un alto grado de epistasis de aptitud, es decir, donde la aptitud de una solución consiste en subconjuntos de sus variables interactuando. Tales algoritmos tienen como objetivo aprender (antes de aprovechar) estas interacciones fenotípicas beneficiosas. Como tales, están alineados con la Hipótesis del bloque fundamental en la reducción de forma adaptativa, de recombinación disruptiva. Ejemplos destacados de este enfoque incluyen mGA , GEMGA y LLGA.

14.3. Neuroevolución

La Neuroevolución es una forma de aprendizaje máquina que utiliza algoritmos evolutivos para entrenar y / o optimizar redes neuronales artificiales. Es útil para aplicaciones tales como video juegos y control de motores en robots, en las que es fácil medir el rendimiento de la red en una tarea, pero difícil o imposible crear un arreglo de pares de entrada-salida correctas para su uso con un algoritmo de aprendizaje supervisado.

14.3.1. Características

Hay muchos algoritmos de neuroevolución. Se hace una distinción entre los que evolucionan los valores de los pesos en las conexiones de una red de topología pre-especificada, frente a aquellos que evolucionan la topología de la red, además de los pesos. Aunque no hay términos estandarizados para esta distinción en su conjunto, agregar o quitar las conexiones de una red durante la evolución puede ser denominado como complejización o simplificación, respectivamente. Las redes que tienen tanto sus pesos de conexión y topología evolucionadas se denominan TWEANNs (Redes Neuronales Artificiales de Topología y Pesos Evolucionados por sus siglas en inglés).

Se hace otra distinción entre los métodos que evolucionan la estructura (topología) de las redes neurales en paralelo a otros parámetros (por ejemplo, pesos sinápticos) y los que los desarrollan por separado.

14.3.2. Codificación Directa e Indirecta de Redes

Los algoritmos evolutivos operan en una población de genotipos (en algunos algoritmos éstos pueden dividirse en especies con diferentes genomas); en neuroevolución, un genotipo es una representación de una red neuronal (un fenotipo).

En esquemas de codificación directos el genotipo es el mismo que el fenotipo, cada neurona y conexión se especifica directamente y de manera explícita en el genotipo. Por el contrario, en los esquemas de codificación indirectos el genotipo especifica reglas o alguna otra estructura para la generación de la red.

Codificaciones indirectas se utilizan a menudo para conseguir varios objetivos:

- Permitir que estructuras recurrentes o funciones de red específicas se formen (modularidad y otras regularidades);
- Compresión del fenotipo a un genotipo más pequeño, proporcionando un espacio de búsqueda más pequeño;
- Mapear el espacio de búsqueda (genoma) a el dominio del problema.

14.3.2.1. Taxonomía de Sistemas Embriogénicos para Codificación Indirecta

Tradicionalmente, codificaciones indirectas que emplean embriogenia artificial (también conocida como desarrollo artificial) se han clasificado análogamente a un enfoque gramatical frente a un enfoque químico celular. Los primeros desarrollan conjuntos de reglas en la forma de sistemas de reescritura gramaticales. Los últimos intentan imitar cómo emergen estructuras físicas en biología a través de la expresión genética. Sin embargo, esta distinción es en gran parte superficial: a pesar de que a menudo se desarrollan de manera diferente en varios aspectos importantes, muchas de estas diferencias sólo existen por razones históricas y no a causa de ningún requisito intrínseco de uno u otro enfoque. Sistemas de codificación indirectos a menudo utilizan aspectos de ambos enfoques.

Stanley y Miikkulainen proponen una taxonomía de sistemas embriogénicos que pretenden reflejar sus propiedades fundamentales. La taxonomía identifica cinco dimensiones continuas a lo largo de las cuales cualquier sistema embrionario puede ser colocado y por lo tanto comparado con los demás:

- Destino Celular (neurona): las características finales y el papel de la célula en el fenotipo maduro. Esta dimensión varía desde tener un único método para la determinación del destino de una célula a tener muchos métodos de determinación.
- Nodo Objetivo: el método por el cual las conexiones se dirigen a partir de células de origen a las células objetivo. Esto va desde focalización específica (origen y destino se identifican explícitamente) a sólo focalización relativa (por ejemplo, sobre la base de la ubicación de células respecto a otras).
- Heterocronía: el tiempo y el orden de los acontecimientos durante la embriogénesis. El rango va de ningún mecanismo para la modificación del itinerario de eventos a muchos mecanismos.

- Canalización: el grado de tolerancia del genoma a mutaciones (fragilidad). Varía desde la necesidad de instrucciones genotípicas precisas hasta una alta tolerancia a la imprecisión o mutación.
- Complejización: la capacidad del sistema (incluyendo algoritmo evolutivo y el mapeo de genotipo a fenotipo) para permitir la complejización del genoma (y por lo tanto del fenotipo) con el tiempo. Su rango va desde permitir sólo genomas de tamaño fijo a permitir genomas de tamaños muy variables.

15. Antecedentes

Neuroevolución (NE), la evolución artificial de redes neuronales mediante algoritmos genéticos, ha demostrado una gran promesa en tareas complejas de aprendizaje por refuerzo (Gómez y Miikkulainen, 1999; Gruau et al., 1996; Moriarty y Miikkulainen, 1997; Potter et al., 1995; Whitley et al., 1993). En neuroevolución se busca a través del espacio de comportamientos de una red que funciona bien en una tarea dada. Este enfoque para resolver problemas de control complejos representa una alternativa a las técnicas estadísticas que intentan estimar la utilidad de acciones particulares en estados particulares del mundo (Kaelbling et al., 1996). NE es un enfoque prometedor para la solución de problemas de aprendizaje por refuerzo por varias razones.

Estudios anteriores han demostrado que NE es más rápida y más eficiente que métodos por refuerzo como Adaptive Heuristic Critic y Q -Learning en el balance de péndulo invertido y el control de brazos robóticos (Moriarty y Miikkulainen, 1996; Moriarty, 1997). Debido a que NE busca un comportamiento en lugar de una función de valor, es eficaz en problemas con espacios de estados continuos y de alta dimensión. Además, se puede representar memoria fácilmente a través de conexiones recurrentes en redes neuronales, haciendo de NE una elección natural para el aprendizaje de tareas no - Markovianas (Gómez y Miikkulainen, 1999, 2002).

En los enfoques tradicionales de NE, se elige una topología para las redes en evolución antes de que comience el experimento. Por lo general, la topología de la red es de una sola capa de neuronas ocultas, con cada neurona oculta conectada a todas las entradas y salidas de la red. La evolución busca en el espacio de pesos de conexión de esta topología totalmente conectada al permitir que las redes de alto rendimiento puedan reproducirse. El espacio de pesos se explora a través del cruce de vectores de pesos de la redes y a través de la mutación de los pesos de redes individuales. Por lo tanto, el objetivo en NE con topología fija es optimizar los pesos de conexión que determinan la funcionalidad de una red.

Sin embargo, los pesos de conexión no son el único aspecto de redes neuronales que contribuyen a su comportamiento. La topología, o estructura, de las redes neuronales también afecta a su funcionalidad. La modificación de la estructura de la red se ha mostrado eficaz como parte de entrenamiento supervisado (Chen et al., 1993). También ha habido un gran interés en la evolución de topologías de red, así como en la de los pesos, durante las últimas décadas (Angeline et al., 1993; Branke, 1995; Gruau et al., 1996; Yao, 1999). La pregunta de fondo, sin embargo, sigue siendo: ¿Puede la evolución de topologías y pesos proporcionar una ventaja sobre la evolución de los pesos en una topología fija? Una red totalmente conectada puede, en principio, aproximar cualquier función continua (Cybenko, 1989). Así que ¿Por qué perder valioso esfuerzo permutando diferentes topologías?

Las respuestas dadas hasta el momento no son concluyentes. Algunos han argumentado que la complejidad de la red puede afectar a la velocidad y precisión del aprendizaje (Zhang y Muhlenbein, 1993). Aunque esta afirmación es cierta para el algoritmo de retropropagación, no está claro si se aplica cuando los pesos son optimizados por evolución y no retropropagación.

16. Problemática

Como hemos visto en los antecedentes a la investigación, existe trabajo previo que explora la naturaleza de la Neuroevolución con técnicas y asunciones particulares, de este trabajo previo podemos extraer la siguiente problemática en común y proponer una estrategia para enfrentarla:

16.1. Codificación TWEANN

La cuestión de cómo codificar redes utilizando una representación genética eficiente debe ser abordada por todas las TWEANNs.

Las TWEANNs se pueden dividir entre aquellas que utilizan una codificación directa, y las que utilizan una indirecta. Esquemas de codificación directa, empleados por la mayoría de TWEANNs, especifican en el genoma cada conexión y nodo que aparecerá en el fenotipo (Angeline et al., 1993; Braun y Weisbrod, 1993; Dasgupta y McGregor, 1992; Fullmer y Miikkulainen, 1992; Krishnan y Ciesielski, 1994; Lee y Kim, 1996; Maniezzo, 1994; Opitz y Shavlik, 1997; Pujol y Poli, 1998; Yao y Liu, 1996; Zhang y Muhlenbein, 1993). En contraste, codificaciones indirectas por lo general sólo especifican normas para la construcción de un fenotipo (Gruau, 1993; Mandischer, 1993). Estas reglas pueden ser especificaciones de capa o reglas de crecimiento a través de división celular. La codificación indirecta permite una representación más compacta que la codificación directa, porque cada nodo y conexión no se especifican en el genoma, a pesar de que se pueden derivar de él.

Aunque es posible evolucionar sistemas de desarrollo, elegimos codificación directa para la investigación porque, como Braun y Weisbrod (1993) sostienen, la codificación indirecta requiere "conocimiento más detallado de los mecanismos genéticos y neuronales." En otras palabras, debido a que las codificaciones indirectas no se correlacionan directamente con sus fenotipos, estas puede sesgar la búsqueda de maneras impredecibles. Para hacer un buen uso de codificaciones indirectas, necesitamos primero entenderlas lo suficientemente bien como para asegurarnos de que no centran la búsqueda en alguna clase subóptima de topologías.

16.2. Problema de Permutación

Este problema tiene que ver con tener más de una manera de expresar la solución a un problema de optimización de pesos en una red neural. Cuando genomas que representan la misma solución no tienen la misma codificación, es probable que su cruce produzca descendencia dañada.

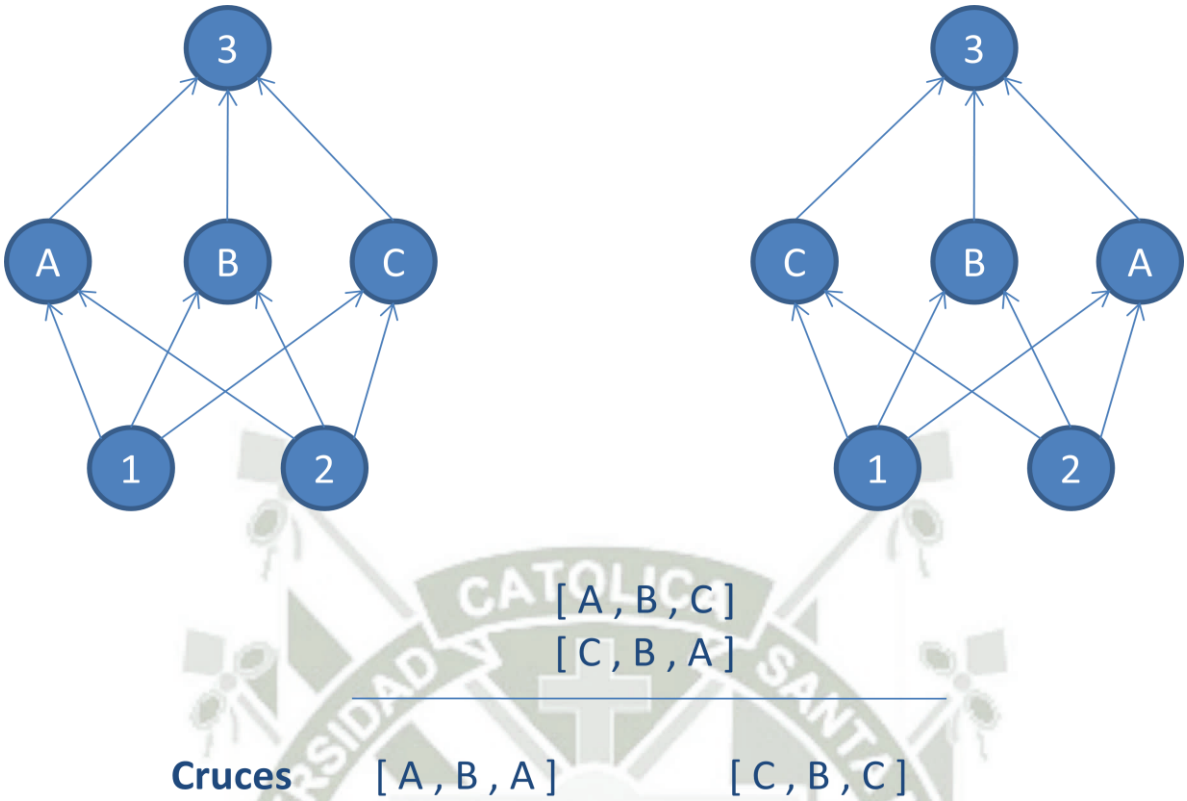

Figura 16-1 Representación gráfica del problema de permutación

La Figura 16-1 representa el problema de una simple red de 3 nodos ocultos. Las tres neuronas ocultas A, B y C, pueden representar la misma solución general de 3! = 6 permutaciones diferentes. Cuando una de estas permutaciones se traspasa con otra, es probable que se pierda información crítica. Por ejemplo, cruzar [A, B, C] y [C, B, A] puede dar lugar a [C, B, C], una representación que ha perdido una tercera parte de la información que ambos padres tenían. En general, para n nodos ocultos, hay n! soluciones funcionalmente equivalentes. El problema puede ser aún más complicado con diferentes arreglos, es decir, [A, B, C] y [D; B, E], que comparten interdependencia funcional en B.

La solución de la naturaleza es homología: dos genes son homólogos si son alelos del mismo rasgo. Homología real entre las redes neuronales no es fácil de determinar por análisis estructural directo (de ahí el problema de permutación). La propuesta es que el origen histórico de dos genes es una evidencia directa de homología si los genes tienen el mismo origen. Por lo tanto, el algoritmo realizará sinapsis artificial basada en marcadores históricos, lo que le permite añadir estructura sin perder de vista cuál gen es cuál en el transcurso de una simulación.

16.3. Innovación

En TWEANNs , la innovación se lleva a cabo mediante la adición de nueva estructura a través de mutaciones. Con frecuencia, la adición de nueva estructura inicialmente hace que la aptitud

de una red disminuya. Por ejemplo, la adición de un nuevo nodo introduce una no linealidad, donde antes no la había, añadir una nueva conexión puede reducir la aptitud antes de que su peso tenga la oportunidad de optimizarse. Es poco probable que un nuevo nodo o conexión exprese una función útil tan pronto como se introduce. Algunas generaciones se requieren para optimizar la nueva estructura y hacer uso de ella. Por desgracia, a causa de la pérdida inicial de aptitud causada por la nueva estructura, es poco probable que la innovación sobreviva en la población lo suficientemente para ser optimizada. Por lo tanto, es necesario proteger de alguna manera a las redes con innovaciones estructurales para que tengan la oportunidad de hacer uso de su nueva estructura.

En la naturaleza, estructuras diferentes tienden a estar en diferentes especies que compiten en diferentes nichos. Por lo tanto, la innovación está protegida implícitamente dentro de un nicho. Del mismo modo, si las redes con estructuras innovadoras pudieran ser aisladas en su propia especie, tendrían la oportunidad de optimizar sus estructuras antes de tener que competir con la población en general.

Debido a que el algoritmo propuesto tiene una solución para el problema de permutación, usando información histórica acerca de los genes, la población puede fácilmente ser clasificada en especies. Utilizaremos aptitud compartida explícita, lo que obliga a los individuos con genomas similares a compartir su aptitud producida (Goldberg y Richardson , 1987). La versión original de aptitud compartida implícita, introducida por Holland (1975), agrupó los individuos por similitud de rendimiento en lugar de similitud genética. La versión explícita es apropiada para TWEANNs, ya que permite agrupar a las redes en base a la topología y las configuraciones de peso. El resultado de compartir aptitud es que el número de redes que pueden existir en la población en un solo pico de aptitud está limitado por el tamaño del pico. Por lo tanto, la población se divide en un número de especies, cada una en un pico diferente, sin la amenaza de que cualquiera de las especies asuma el control. Aptitud compartida explícita hace que la similitud se pueda medir fácilmente sobre la base de la información histórica en los genes. Por lo tanto, las innovaciones se protegen en su propia especie.

16.4. Población Inicial

En muchos sistemas de TWEANN, la población inicial es una colección al azar de topologías. Tal población asegura la diversidad topológica desde el principio. Sin embargo, las poblaciones iniciales aleatorias llegan a producir muchos problemas para TWEANNs. Por ejemplo, en muchos de los esquemas de codificación directos, hay una posibilidad de que una red no tenga ningún camino desde cada una de sus entradas a sus salidas. Estas redes no factibles toman tiempo para eliminar de la población.

Sin embargo, hay un problema más sutil y más grave con el inicio al azar. Es deseable desarrollar soluciones mínimas, de esta manera, se reduce el número de parámetros que tienen que ser buscados. Comenzar con topologías al azar no conduce a la búsqueda de soluciones mínimas, ya que la población comienza con muchos nodos innecesarios y

conexiones que ya están presentes. Ninguno de estos nodos o conexiones han tenido que soportar una sola evaluación, es decir, no hay ninguna justificación para su configuración. Cualquier reducción de las redes tendrá que ser para deshacerse de una estructura que no debería haber estado allí en un primer lugar, y nada en el proceso de recombinación de diferentes topologías conduce hacia esa minimización. Puesto que no hay costo de aptitud en la creación de redes de mayor tamaño, estas dominarán siempre que tengan alta aptitud.

Por lo tanto, comenzar con una población mínima e incrementar estructuras a partir de ahí es un principio de diseño en este proyecto.

17. Metodología Algorítmica

El método propuesto en este documento, como se describe en detalle en esta sección, consiste en poner juntas las ideas desarrolladas en la sección anterior en un solo sistema. Comenzaremos por explicar la codificación genética utilizada en el algoritmo y continuar con la descripción de los componentes que se refieren específicamente a cada uno de los tres problemas de TWEANNs ("*Problemática*" *ver Pág. 34*).

17.1. Codificación Genética

El esquema de codificación genética de este algoritmo está diseñado para permitir que los genes correspondientes sean fácilmente alineados cuando dos genomas se cruzan durante el apareamiento. Los genomas son representaciones lineales de conectividad en una red (Figura 17-1). Cada genoma incluye una lista de genes de conexión, cada uno de los cuales se refiere a dos genes nodo que están siendo conectados. Los genes nodo proporcionan una lista de entradas, nodos ocultos, y las salidas que se pueden conectar. Cada gen de conexión especifica el nodo - entrada, el nodo - salida, el peso de la conexión, ya sea o no que el gen de conexión está habilitado (un bit de habilitación), y un número de innovación, que permite la búsqueda de genes correspondientes (como se explicará más adelante).

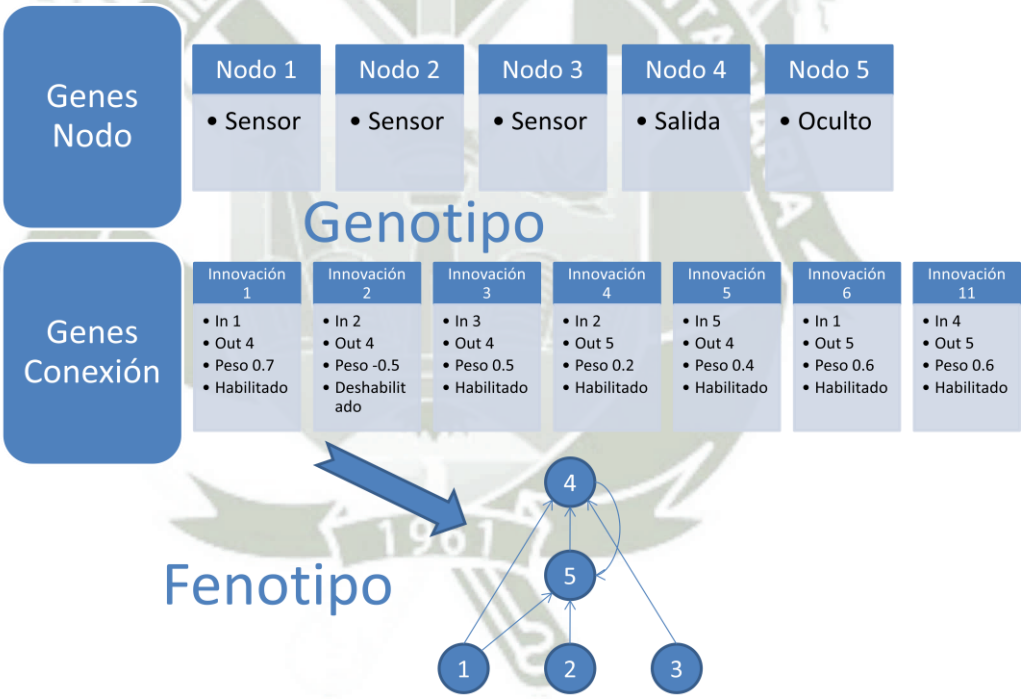

Figura 17-1 Genotipo Vs Fenotipo

La mutación en este algoritmo puede cambiar ambos, pesos de conexión y estructuras de red. Los pesos de conexión mutan como en cualquier sistema de NE, con cada conexión ya sea que esté deshabilitada o no en cada generación. Mutaciones estructurales ocurren en dos formas (Figura 17-2). Cada mutación expande el tamaño del genoma mediante la adición de uno o más genes. En la mutación de conexión, se añade un único nuevo gen de

conexión con un peso al azar que conecta dos nodos previamente desconectados. En la mutación de adición de nodos, una conexión existente se divide y el nuevo nodo se sitúa donde la antigua conexión solía estar. La conexión antigua se desactiva y dos conexiones nuevas se añaden al genoma. La nueva conexión hacia el nuevo nodo recibe un peso de 1, y la nueva conexión de salida recibe el mismo peso que la conexión antigua. Este método de adición de nodos fue elegido con el fin de minimizar el efecto inicial de la mutación. La nueva no linealidad en la conexión cambia la función ligeramente, pero nuevos nodos se pueden integrar inmediatamente a la red, en lugar de la adición de estructuras extrañas que tendrían que ser integradas a la red más adelante. De esta forma, a causa de la especiación, la red tendrá tiempo para optimizarse y hacer uso de su nueva estructura.

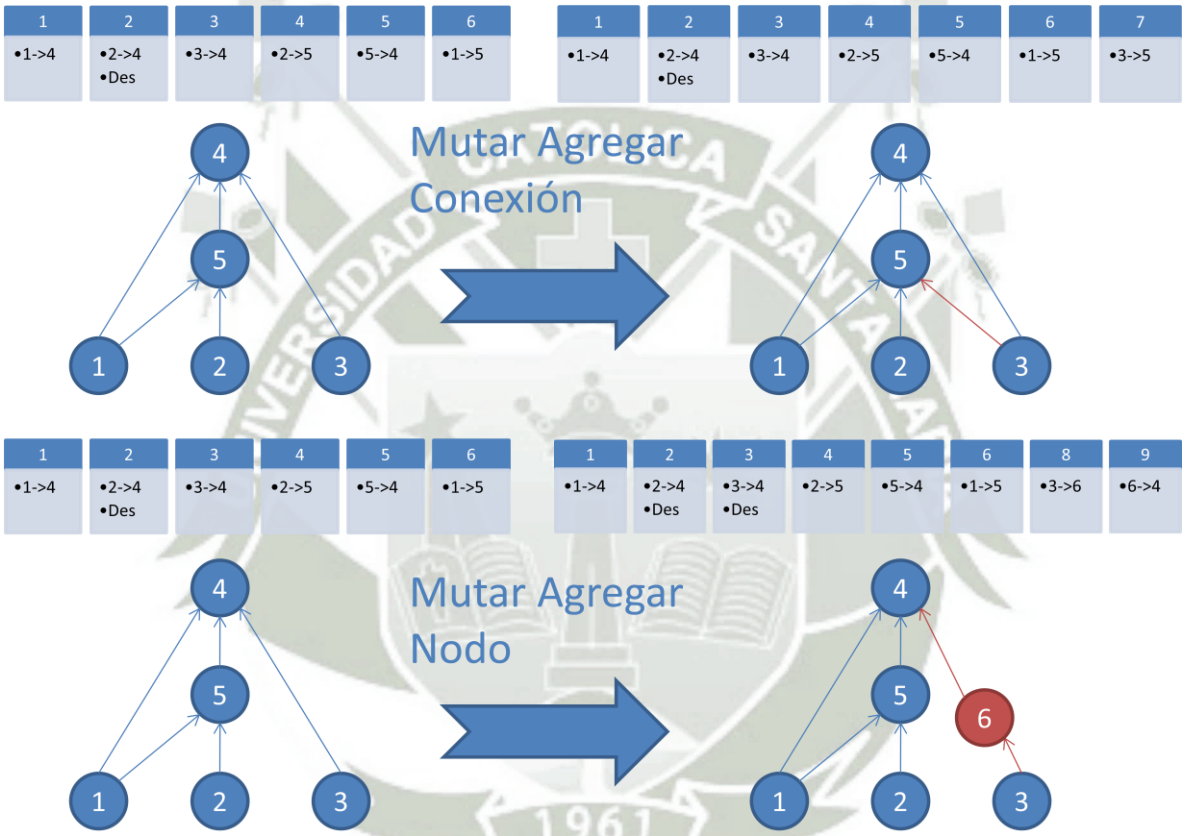

Figura 17-2 Tipos de Mutaciones

A través de la mutación, los genomas de este algoritmo poco a poco se hacen más grandes. Resultarán genomas de diferentes tamaños, a veces con diferentes conexiones en las mismas posiciones. La forma más compleja del problema de permutación, con numerosas diferentes topologías y combinaciones de peso, es un resultado inevitable de permitir que los genomas crezcan sin límites. ¿Cómo puede NE cruzar genomas de diferentes tamaños en una forma sensata? La siguiente sección explica cómo este algoritmo resuelve este problema.

17.2. Rastreando Genes con Marcadores Históricos

Hay información sin explotar en la evolución que nos dice exactamente cuáles genes coinciden con cuáles entre los individuos de una población topológicamente diversa. Esta información es el origen histórico de cada gen. Dos genes con el mismo origen histórico deben representar la misma estructura (aunque posiblemente con diferentes pesos), ya que son ambos derivados del mismo gen ancestral en algún momento del pasado. Por lo tanto, todo lo que el sistema debe hacer para saber cuáles genes alinear con cuáles, es seguir la pista del origen histórico de cada gen en el sistema.

El seguimiento de los orígenes históricos requiere muy poco cálculo. Cada vez que aparece un nuevo gen (a través de mutación estructural), un número global de innovación se incrementa y se asigna a ese gen. Los números de innovación por lo tanto representan una cronología de la aparición de cada gen en el sistema. Como un ejemplo, digamos que las dos mutaciones en la Figura 17-2 se produjeron una tras otra en el sistema. Al nuevo gen de conexión creado en la primera mutación se le asigna el número 7, y a los dos nuevos genes de conexión adicionales en la nueva mutación de nodo se les asignan los números 8 y 9. En el futuro, cuando estos genomas se apareen, los hijos heredarán los mismos números de innovación de cada gen, los números de innovación no se cambian. Por lo tanto, el origen histórico de cada gen en el sistema es conocido en toda la evolución.

Un posible problema es que la misma innovación estructural recibirá números diferentes de innovación en la misma generación, si se produce por azar más de una vez. Sin embargo, al mantener una lista de las innovaciones que se produjeron en la generación actual, es posible asegurar que cuando la misma estructura se presenta más de una vez a través de mutaciones independientes en la misma generación, a cada mutación idéntica se le asigna el mismo número de la innovación. Por lo tanto, no se produce una explosión de los números de innovación.

Los marcadores históricos dan al algoritmo una nueva y poderosa capacidad. Ahora, el sistema sabe exactamente cuáles genes coinciden con cuáles (Figura 17-3). Al cruzarse, los genes de ambos genomas con los mismos números de innovación están alineados. Estos genes se llaman genes coincidentes. Genes que no coinciden, o son disjuntos o son exceso, dependiendo de si se producen dentro o fuera del rango de números de innovación del otro padre. Estos representan estructura que no está presente en el otro genoma. En la composición de la descendencia, los genes son seleccionados al azar de cualquiera de los padres en los genes coincidentes, mientras que los genes exceso o disjuntos siempre son heredados del padre más apto. De esta manera, los marcadores históricos nos permiten ejecutar cruces con genomas lineales sin la necesidad de un análisis topológico costoso.

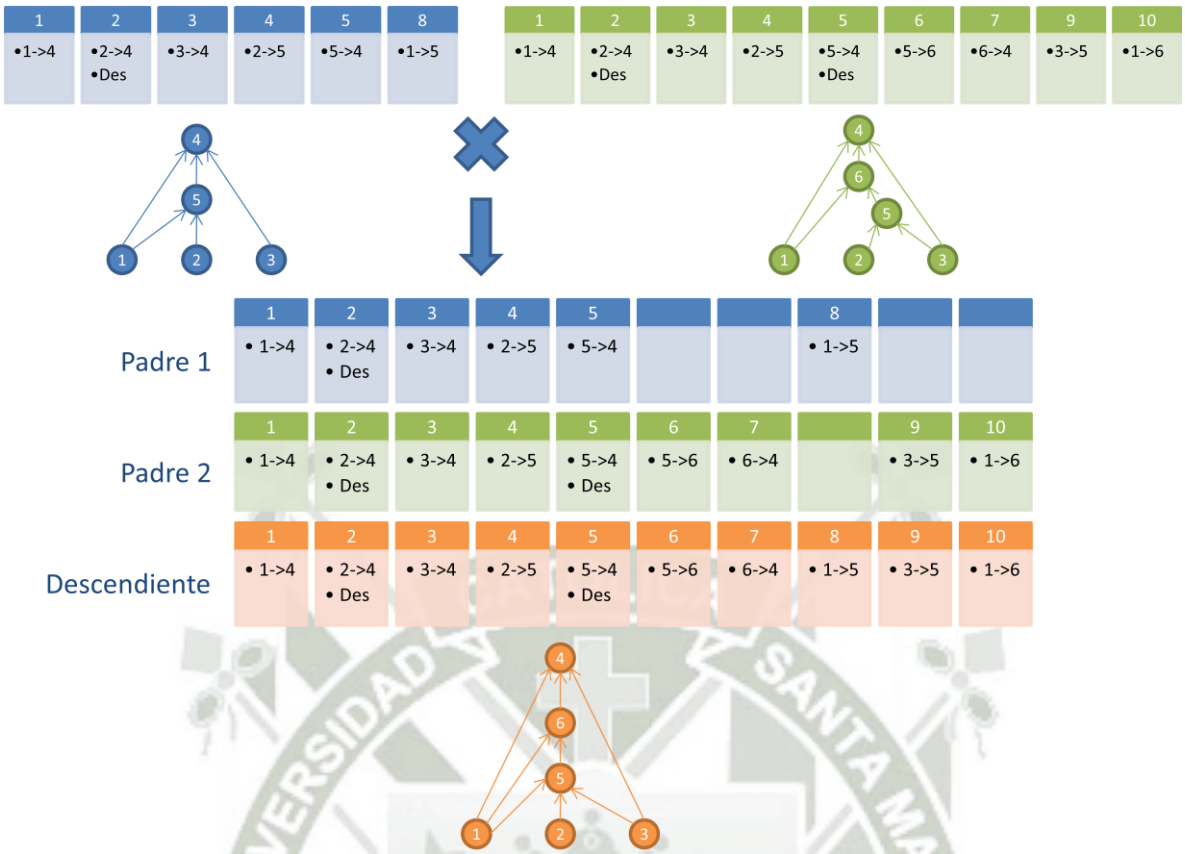

Figura 17-3 Reproducción por Cruce

Mediante la adición de nuevos genes a la población y el apareamiento sensible de genomas que representan diferentes estructuras, el sistema puede formar una población de diversas topologías. Sin embargo, resulta que tal población por sí sola no puede mantener innovaciones topológicas. Dado que estructuras más pequeñas se optimizan más rápido que estructuras más grandes, y la adición de nodos y conexiones generalmente disminuye la aptitud de la red al principio, las estructuras recientemente aumentadas tienen pocas esperanzas de sobrevivir por más de una generación a pesar de que las innovaciones que ellas representan pueden ser cruciales para resolver la tarea en el largo plazo. La solución es proteger la innovación separando a la población en especies, como se explica en la siguiente sección.

17.3. Protegiendo Innovación con Especiación

La especiación de la población permite a los organismos competir principalmente en sus propios nichos en lugar de con la población en general. De esta manera, las innovaciones topológicas están protegidas en un nuevo nicho en el que tienen tiempo para optimizar su estructura a través de competencia en dicho nicho. La idea es dividir la población en especies tales que, topologías similares se encuentren en la misma especie. Esta tarea parece ser un problema de hacer coincidir topologías. Sin embargo, una vez más resulta que los marcadores históricos ofrecen una solución eficiente.

El número de genes exceso y disjuntos entre un par de genomas, es una medida natural de su distancia en compatibilidad. Mientras más disjuntos dos genomas son, comparten menos historia evolutiva, y por lo tanto son menos compatibles. Es así que podemos medir la distancia en compatibilidad δ de diferentes estructuras, como una simple combinación lineal del número de genes exceso E y disjuntos D, así como el promedio de las diferencias de peso en los genes coincidentes $\bar{W}$, incluidos genes deshabilitados:

$$\delta = \frac{c_1 E}{N} + \frac{c_2 D}{N} + c_3 . \bar{W} \qquad (1)$$

Los coeficientes c1, c2 y c3 nos permiten ajustar la importancia de los tres factores, y el factor N, el número de genes en el genoma más grande, normaliza el tamaño del genoma (N se puede ajustar a 1 si ambos genomas son pequeños, es decir, consisten de menos de 20 genes).

La medida de distancia δ nos permite asignar especies utilizando un umbral de compatibilidad δ_t. Se mantiene una lista ordenada de las especies. En cada generación, los genomas se colocan de forma secuencial en especies. Cada especie existente está representada por un genoma al azar dentro de las especies de la generación anterior. Dado un genoma g de la generación actual se lo coloca en la primera especie en la que g sea compatible con el genoma representante de dicha especie. De esta manera, las especies no se superponen. Si g no es compatible con alguna de las especies existentes, se crea una nueva especie con g como su representante.

Como mecanismo de reproducción para este algoritmo, utilizamos aptitud explícita compartida (Goldberg y Richardson, 1987), donde los organismos de la misma especie deben compartir la aptitud de su nicho. Por lo tanto, una especie no puede permitirse el lujo de llegar a ser demasiado grande, aunque muchos de sus organismos sean muy aptos. Por lo tanto, es poco probable que cualquier especie se apodere de toda la población, lo cual es crucial para que la evolución por especies funcione. La aptitud ajustada f'_i para el organismo i se calcula de acuerdo a su distancia δ de cada otro organismo j en la población:

$$f_i' = \frac{f_i}{\sum_{j=1}^{n} sh(\, \delta(i,j)\,)} \qquad (2)$$

La función de intercambio de sh se hace 0 cuando la distancia δ(i; j) está por encima del umbral de δ_t, de lo contrario, sh(δ(i; j)) se hace 1 (Spears, 1995). Por lo tanto, $\Sigma^{n}_{j=1}$ sh(δ(i; j)) se reduce a el número de organismos de la misma especie que el organismo i. Esta reducción es natural, ya que las especies ya están agrupadas por compatibilidad con el umbral δ_t. A cada especie se le asigna un número potencialmente diferente de descendencia en proporción a la suma de la aptitud ajustada f'_i de sus organismos miembros. A continuación las especies se reproducen, primero eliminando los miembros de más bajo rendimiento en la población. Luego toda la población es reemplazada por los descendientes de los organismos restantes de cada especie.

El efecto neto deseado de la especiación de la población es el de proteger la innovación topológica. El objetivo final del sistema, es entonces, llevar a cabo la búsqueda de una solución de la forma más eficientemente posible. Este objetivo se logra al minimizar la dimensionalidad del espacio de búsqueda.

17.4. Minimizando Dimensionalidad mediante el Crecimiento Incremental de Estructuras Mínimas

Como se discutió en las secciones anteriores, TWEANNs suelen comenzar con una población inicial de topologías al azar con el fin de introducir la diversidad desde el principio. Por el contrario, nosotros orientamos la búsqueda hacia espacios de dimensiones mínimas, comenzando con una población uniforme de redes con cero nodos ocultos (es decir, todas las entradas se conectan directamente a las salidas). Nueva estructura se introduce progresivamente a medida que se producen mutaciones estructurales, y sólo sobreviven las estructuras que se encuentran útiles a través de evaluaciones de aptitud. En otras palabras, los incrementos estructurales que se producen siempre están justificados. Dado que la población comienza mínimamente, la dimensionalidad del espacio de búsqueda se reduce al mínimo, y el algoritmo siempre está buscando a través de un menor número de dimensiones que en otros TWEANNs y sistemas NE de topología fija. Minimizar dimensionalidad nos da una ventaja de rendimiento en comparación con otros enfoques.

18. Implementación

En esta sección se presentan las estructuras de datos y los diagramas de flujo que conforman la implementación propia del Algoritmo propuesto. La implementación se hizo sobre Matlab, así que las estructuras de datos más utilizadas son Matrices, al mismo tiempo se desarrollaron arreglos de estructuras, cuyos miembros son Matrices, o en algunos casos, otras Estructuras.

18.1. Estructuras de Datos

Estas son las variables más utilizadas a lo largo de la implementación, así que son transversales a la mayoría de funciones.

18.1.1. NodeGenes

Matriz que contiene el detalle de todos los nodos del individuo, cada uno está conformado en parte por una de éstas matrices. La matriz se detalla a continuación (Figura 18-1):

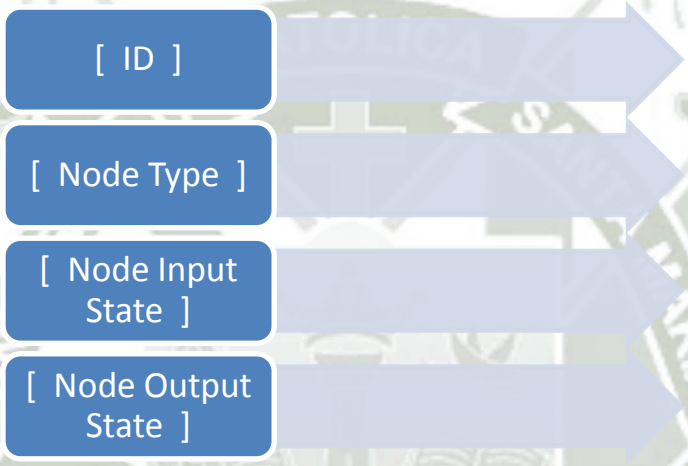

Figura 18-1 NodeGenes

Donde cada fila es:

- ID: Número entero que Identifica cada nodo individualmente.
- Node Type: Tipo de Nodo (1=input, 2=output, 3=hidden, 4=bias).
- Node Input State: Estado de Activación de la entrada del Nodo.
- Node Output State: Estado de la salida del Nodo.

Y se extiende por un número de Columnas igual a la cantidad de nodos del Individuo.

18.1.2. ConnectionGenes

Matriz que detalla las características de cada Conexión del Individuo, cada uno está conformado en parte por una de éstas matrices. La matriz se detalla a continuación (Figura 18-2):

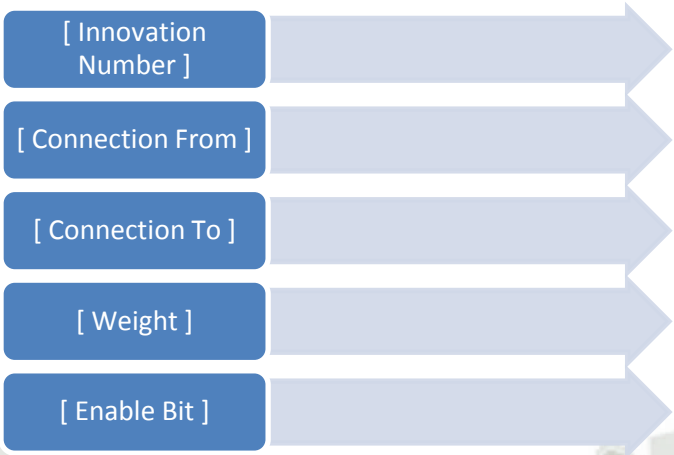

Figura 18-2 ConnectionGenes

Donde cada fila es:

- Innovation Number: Identificador numérico de la innovación a la que pertenece cada conexión.
- Connection From: Nodo Origen de la Conexión.
- Connection To: Nodo Destino de la Conexión.
- Weight: Peso de la Conexión.
- Enable Bit: Bit de Habilitación de la Conexión.

Y se extiende por un número de Columnas igual a la cantidad de Conexiones del Individuo.

18.1.3. Innovation_Record

Matriz que guarda información de las Innovaciones que ocurren durante la Ejecución. La matriz se detalla a continuación (Figura 18-3):

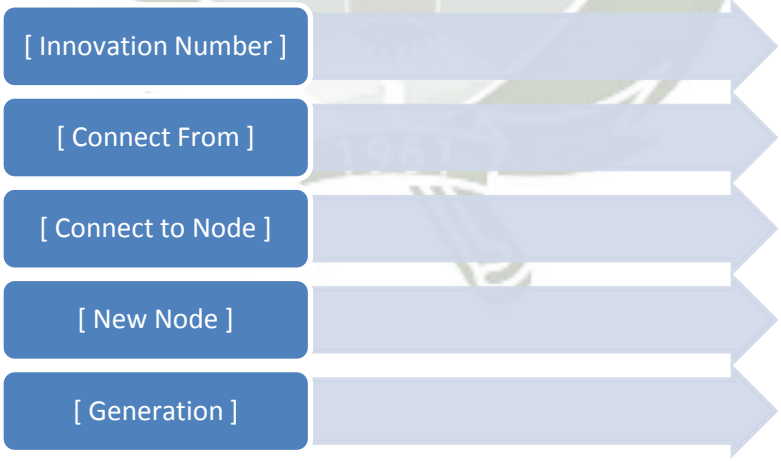

Figura 18-3 Innovation_Record

Donde cada fila es:

- Innovation Number: Identificador numérico de la innovación a la que pertenece cada conexión.
- Connection From: Nodo Origen de la Conexión.
- Connection To: Nodo Destino de la Conexión.
- New Node: ID de Nodo nuevo cuando se muta uno.
- Generation: Generación en la que ocurrió la Innovación.

Y se extiende por un número de Columnas igual a la cantidad de Innovaciones.

18.1.4. Population

Arreglo de estructuras que representan a todos los individuos de la Población. La estructura se detalla a continuación (Figura 18-4):

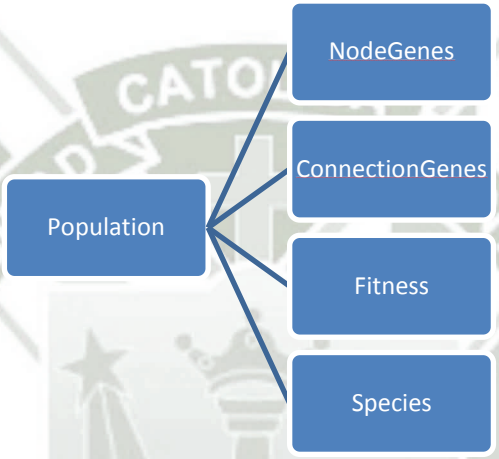

Figura 18-4 Population

Donde cada miembro es:

- NodeGenes: Matriz que contiene todos los Nodos del Individuo.
- ConnectionGenes: Matriz que contiene todas las Conexiones del Individuo.
- Fitness: Aptitud del Individuo.
- Species: Especie a la que pertenece el Individuo.

18.1.5. Generation_Record

Matriz que almacena detalles de Aptitud de los Individuos a lo largo de las generaciones. La matriz se detalla a continuación (Figura 18-5):

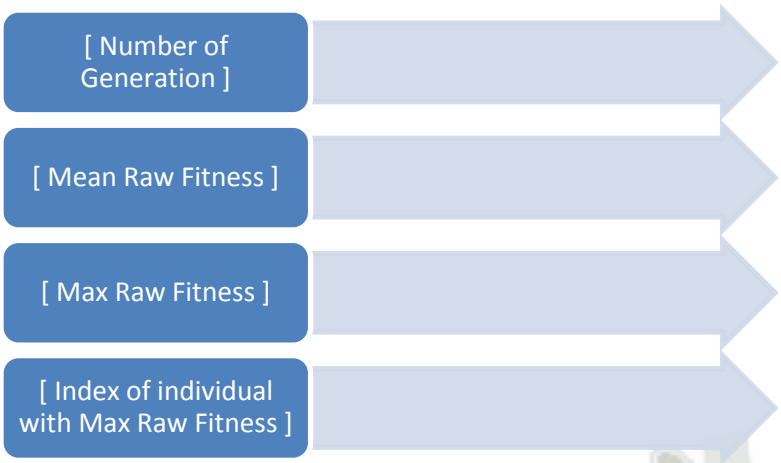

Figura 18-5 Generation_Record

Donde cada fila es:

- Number of Generation: Identificador de la Generación.
- Mean Raw Fitness: Aptitud Media Generacional.
- Max Raw Fitness: Aptitud Máxima Generacional.
- Index of Individual with Max Raw Fitness: Índice del Individuo con la mayor Aptitud.

Y se extiende por un número de Columnas igual a la cantidad de Generaciones.

18.1.6. Species_Record

Estructura que almacena detalles acerca de cada Especie. La estructura se detalla a continuación (Figura 18-6):

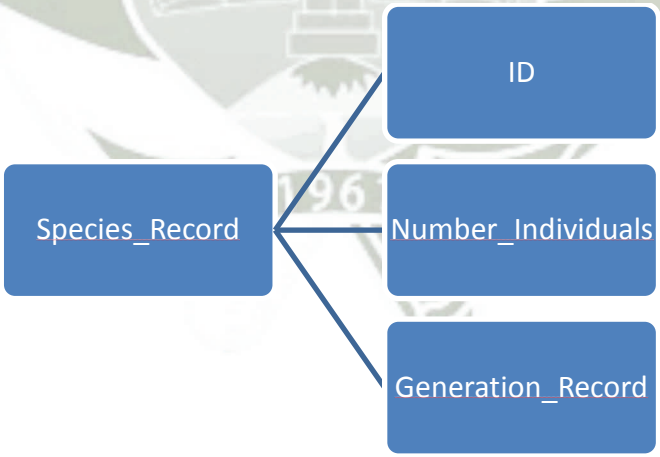

Figura 18-6 Species_Record

Donde cada miembro es:

- ID: Identificador de cada Especie.

- Number_Individuals: Número de Individuos en cada Especie.
- Generation_Record: Matriz que almacena detalles de Aptitud de los Individuos a lo largo de las generaciones

18.1.7. Speciation

Estructura que almacena Parámetros acerca de Especiación. La estructura se detalla a continuación (Figura 18-7):

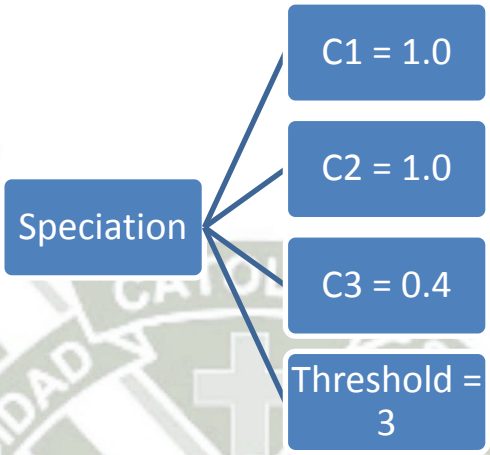

Figura 18-7 Speciation

Donde cada miembro es:

- C1: Constante de Ponderación de Nodos Exceso.
- C2: Constante de Ponderación de Nodos Disjuntos.
- C3: Constante de Ponderación de Diferencia de Peso Promedio.
- Threshold: Umbral de distancia máxima para estar en la misma Especie.

18.1.8. Stagnation

Estructura que almacena Parámetros acerca de Estancamiento. La estructura se detalla a continuación (Figura 18-8):

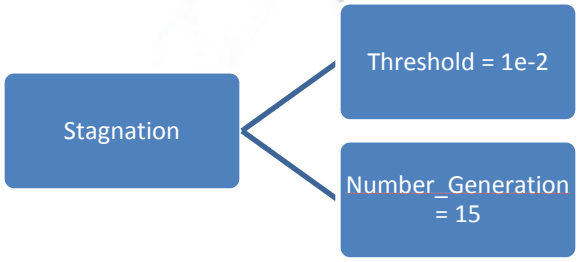

Figura 18-8 Stagnation

Donde cada miembro es:

- Threshold: Umbral bajo el cual se considera la Especie, estancada.
- Number_Generation: Número de Generaciones luego de las cuales se deja morir la especie.

18.1.9. Refocus

Estructura que almacena Parámetros acerca de Reenfoque, el cual se da si la Aptitud de la Población no mejora por un número determinado de Generaciones. La estructura se detalla a continuación (Figura 18-9):

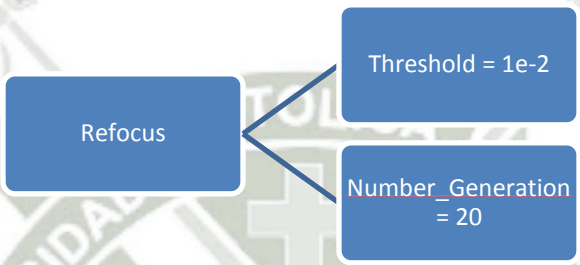

Figura 18-9 Refocus

Donde cada miembro es:

- Threshold: Umbral bajo el cual se considera la Población, estancada.
- Number_Generation: Número de Generaciones luego de las cuales se deja morir todas menos dos Especies.

18.1.10. Crossover

Estructura que almacena Parámetros acerca del Cruce, el cual es parte de la reproducción de la Población. La estructura se detalla a continuación (Figura 18-10):

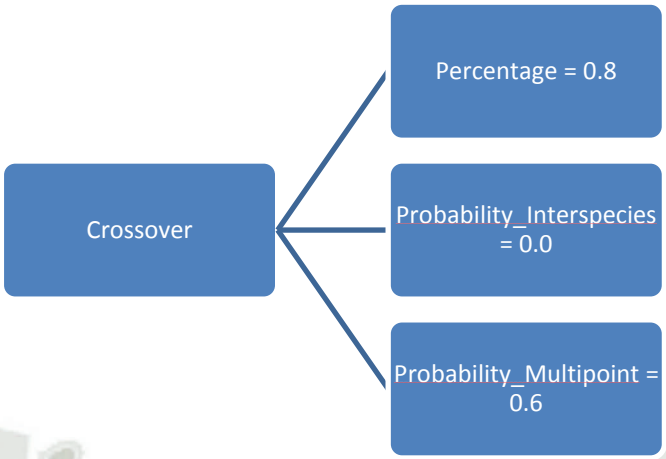

Figura 18-10 Crossover

Donde cada miembro es:

- Percentage: Probabilidad de Cruce.
- Probability_Interspecies: Probabilidad de que, de haber cruce, este sea de Especies diferentes.
- Probability_Multipoint: Probabilidad de heredar Genes de Conexión aleatoriamente de alguno de los padres.

18.1.11. Mutation

Estructura que almacena Parámetros acerca de la Mutación, la cual es parte de la reproducción de la Población. La estructura se detalla a continuación (Figura 18-11):

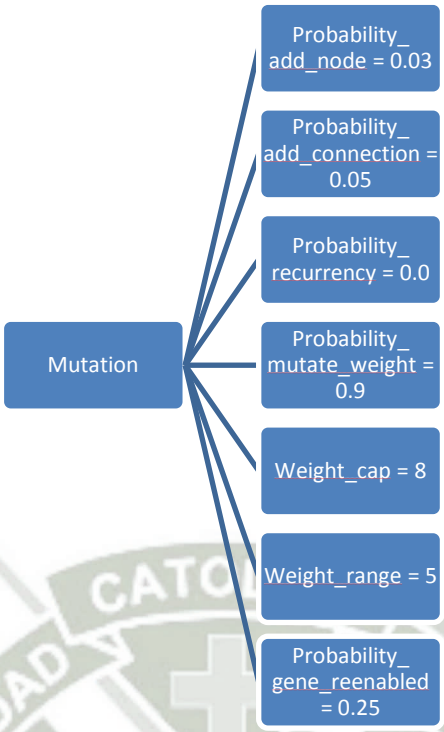

Figura 18-11 Mutation

Donde cada miembro es:

- Probability_add_node: Probabilidad de que se añada un nodo al Individuo.
- Probability_add_connection: Probabilidad de que se añada una conexión al Individuo.
- Probability_recurrency: Probabilidad de que existan conexiones recurrentes.
- Probability_mutate_weight: Probabilidad de que mute alguna de las conexiones del Individuo.
- Weight_cap: Restricción límite para los pesos de conexiones.
- Weight_range: Amplitud de la distribución aleatoria de los pesos.
- Probability_gene_reenabled: Probabilidad de reactivar una conexión que había sido inhabilitada.

18.1.12. Matrix_existing_and_propagating_species

Matriz que almacena detalles de la cantidad de descendencia por cada Especie. La matriz se detalla a continuación (Figura 18-12):

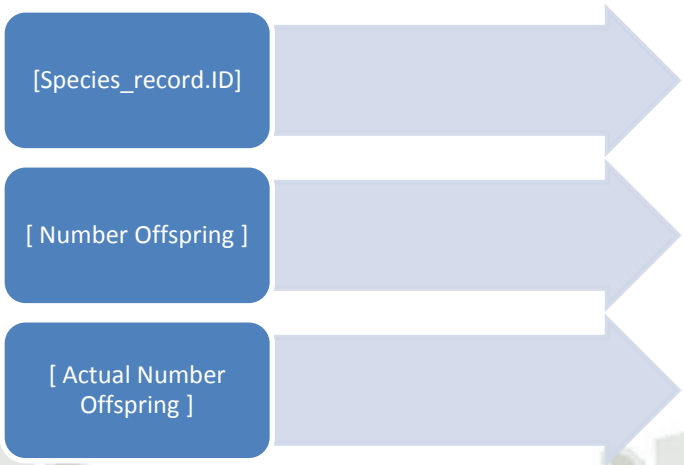

Figura 18-12 Matrix_existing_and_propagating_species

Donde cada fila es:

- Species_record.ID: Identificador de la Especie.
- Number Offspring: Número de Descendientes Correspondientes a la Especie.
- Actual Number Offspring: Contador Real de Descendientes por Especie.

Y se extiende por un número de Columnas igual a la cantidad de Especies en la generación actual.

18.2. Diagramas de Flujo

A continuación se muestran los Diagramas de Flujo para las principales funciones que componen el Programa, dichas funciones utilizan las estructuras de datos, o parte de ellas, detalladas en la sección anterior, como argumentos o variables de retorno. Se empieza por el nivel de abstracción más alto con el diagrama completo del programa, y luego se van detallando las funciones más importantes que éste comprende.

18.2.1. Flujo General

El siguiente esquema (Figura 18-13) muestra el flujo de información para el algoritmo propuesto:

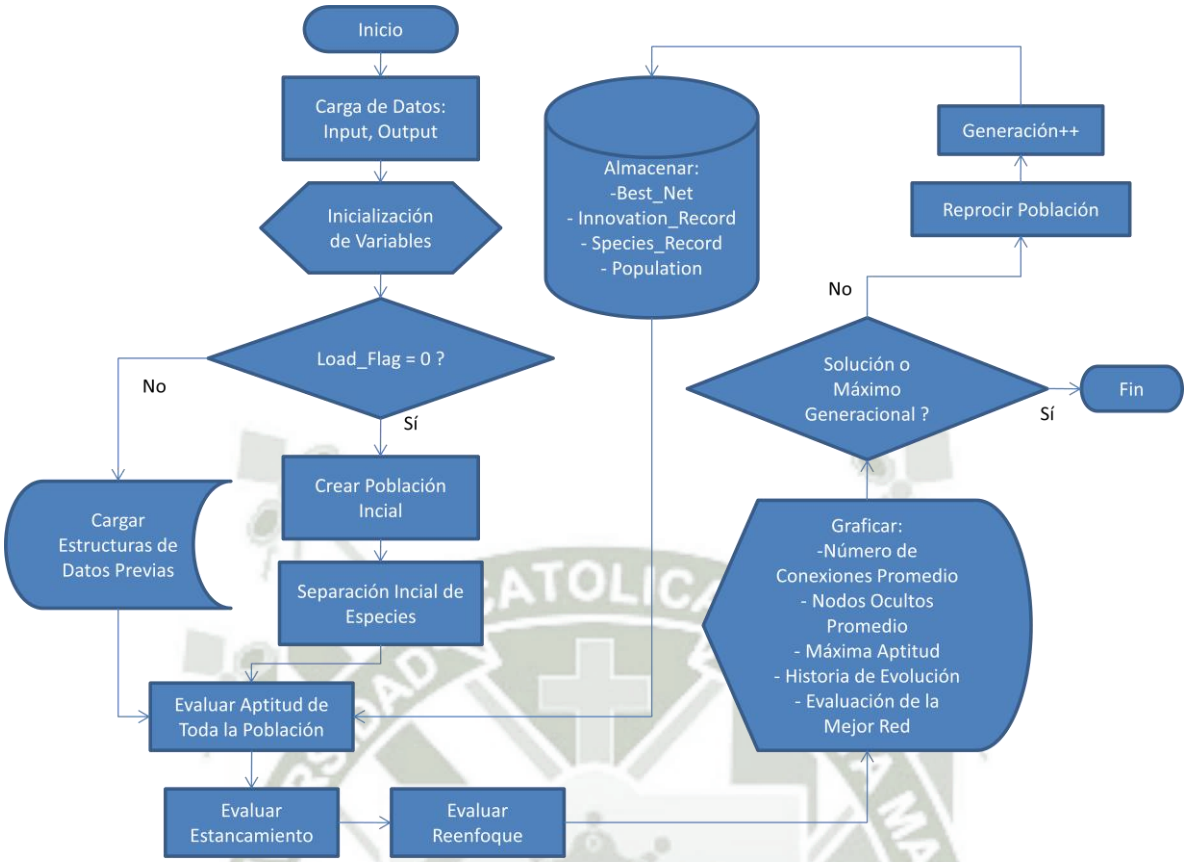

Figura 18-13 Diagrama de Flujo General

La carga de datos de entrada/salida es particular para cada problema, ya que el Paradigma utilizado para esta implementación en particular es de Aprendizaje Supervisado. Por lo tanto, para evaluar un problema específico con este método, necesitamos generar un set de datos de entrada/salida, que sea representativo del dominio del mismo.

La inicialización de Variables se refiere a los parámetros de Cruce, Mutación, Especiación, etc. Para fines de Visualización, con cada generación que pasa se va actualizando el display de:

- Número de Conexiones Promedio
- Número de Nodos Ocultos Promedio
- Máxima Aptitud
- Historia de Evolución
- Evaluación Gráfica de la mejor Red Neuronal

De modo que podemos ir supervisando el progreso del programa al resolver el problema propuesto, más aun podemos ver cómo es que la adición de nodos y conexiones afecta el desempeño y la salida de la Red Neuronal. Así como también, comparar la cantidad de Individuos por Especie durante todo el proceso de evolución, y estudiar cómo su desempeño afecta la cantidad de Descendencia que se le asigna a determinada especie.

El Algoritmo itera sobre el loop descrito hasta que la Aptitud de alguno de los individuos sobrepasa un Umbral predeterminado, o hasta que el número máximo de generaciones se ha alcanzado. Esto evita que el Algoritmo itere infinitamente buscando una solución. En cualquiera de los casos, el programa guarda en disco una estructura conteniendo detalles de la evolución, la última generación de la población, y más importante aún, la Mejor Red Neuronal Obtenida.

18.2.2. Población Inicial

El siguiente esquema (Figura 18-14) muestra el flujo de información para la función que produce la Población Inicial:

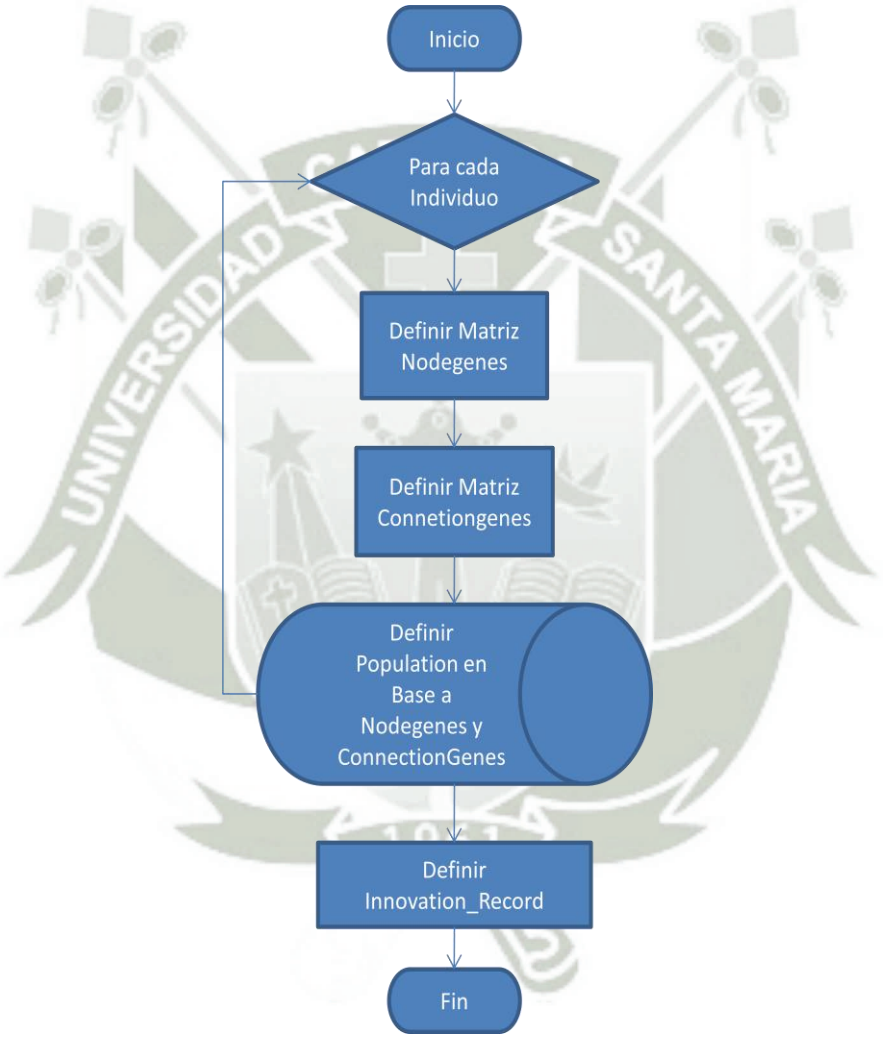

Figura 18-14 Población Inicial

Como se discutió previamente, el problema de la población inicial no es trivial. Es así que en esta función se define a todos los individuos de la población inicial con una topología mínima, es decir, el genoma de todos ellos incluye sólo los nodos de entrada necesarios

según el set de datos entrada/salida, un nodo de parcialidad (bias) y el/los nodos de salida, además todas las entradas están directamente conectadas al/los nodos de salida.

De la misma manera, en esta función se definen las bases genéticas de todos los individuos de la población, es decir, las matrices de NodeGenes y ConnectionGenes, que conforman el genoma de cada individuo. Además se Inicializa la matriz de Innovación.

18.2.3. Evaluar Red Neuronal

El siguiente esquema (Figura 18-15) muestra la lógica de evaluación de las Redes Neuronales:

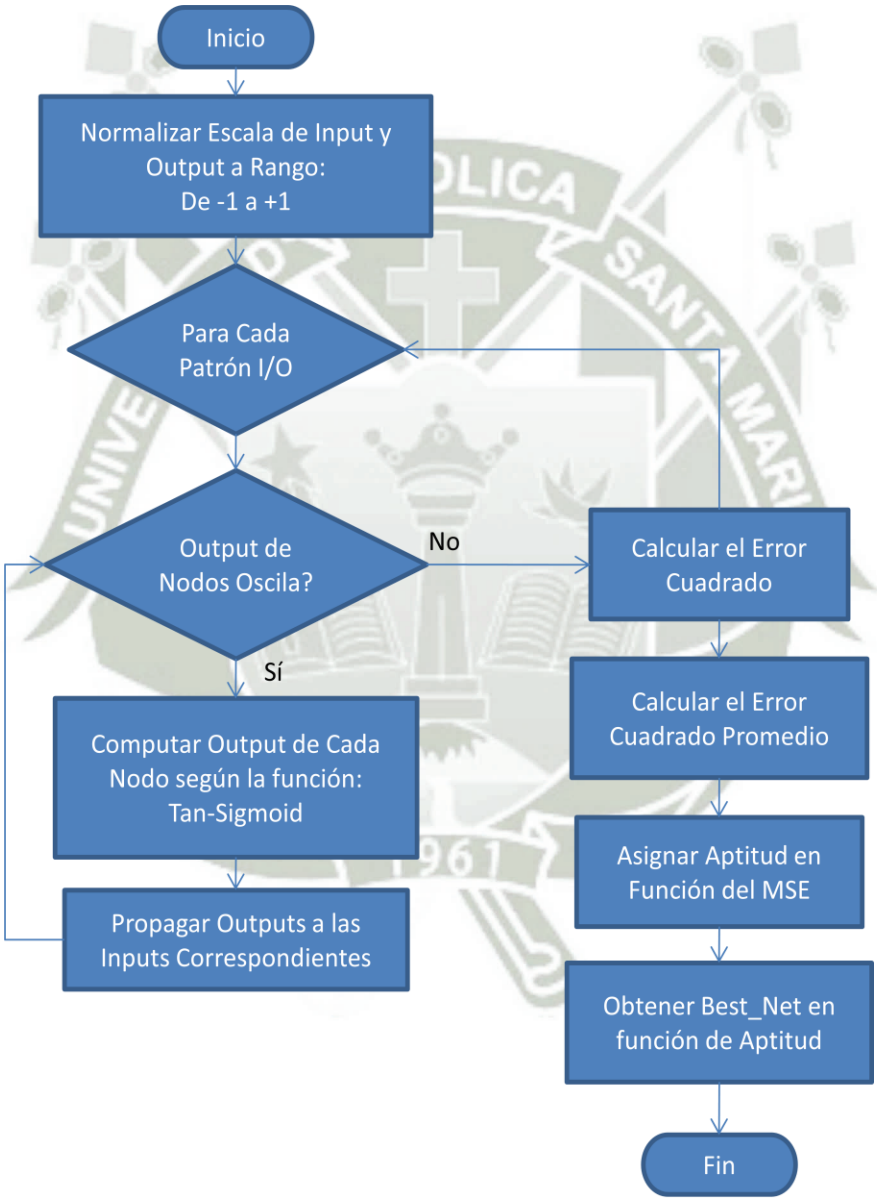

Figura 18-15 Evaluación de RNA

Ya que las estructuras de datos que definen las redes son particulares al algoritmo propuesto, no fue posible hacer uso del Toolkit de Redes Neuronales de Matlab, por lo que se diseñó esta función para evaluar la aptitud de cada red.

La función importa el set completo de entradas/salida e itera sobre cada patrón. Para esto, primero tenemos que normalizar el set de datos, es decir, convertir los datos a una misma escala. Luego procedemos a evaluar cada patrón sobre la red, para esto se utiliza la función Tan-Sigmoid como función de activación de todos los nodos ocultos, mientras que se utiliza una función lineal para los nodos de entrada y los de salida. Se propaga el patrón de muestra por toda la red hasta que la salida deja oscilar, y entonces se itera sobre el siguiente patrón, hasta que se han evaluado todos los patrones.

En esta función también se calcula la aptitud de cada red, para ello utilizamos el método de "Least Mean Square Error" o LMS, por lo que necesitamos calcular, primero, el Cuadrado del Error sobre todos los patrones, y luego obtener la media de ellos. Así obtenemos una aproximación del desempeño de cada red sobre el problema en cuestión.

18.2.4. Reproducción

El siguiente esquema (Figura 18-16) muestra la lógica de Reproducción de las Redes Neuronales:

Figura 18-16 Reproducción

Primero debemos computar la Matriz de Especies existentes y en Propagación, esto con el fin de determinar cuáles Especies se deben dejar morir, y cuanta descendencia asignar a las Especies que aún quedan, luego ejercemos Elitismo al copiar a los individuos más Aptos de cada Especie a la nueva población y eliminamos los individuos menos aptos.

Para fines de Reproducción, ya sea por Cruce o Mutación, debemos obtener una muestra de la población que no esté parcializada por motivos de Aptitud, y para esto utilizamos el método conocido como Muestreo Estocástico Universal. Luego procedemos a Cruzar y Mutar la población, ambas funciones se detallan luego. Cuando todas las Especies tienen completo su lote de Descendencia asignado, se da a lugar la especiación, así podemos clasificar a toda la descendencia obtenida en el procedo de reproducción, ya sea en Especies preexistentes, o en nuevas Especies. Por último se actualizan Species_Record e Innovation_Record con los datos obtenidos.

18.2.5. Cruce

El siguiente esquema (Figura 18-17) muestra la lógica del Cruce entre Redes Neuronales:

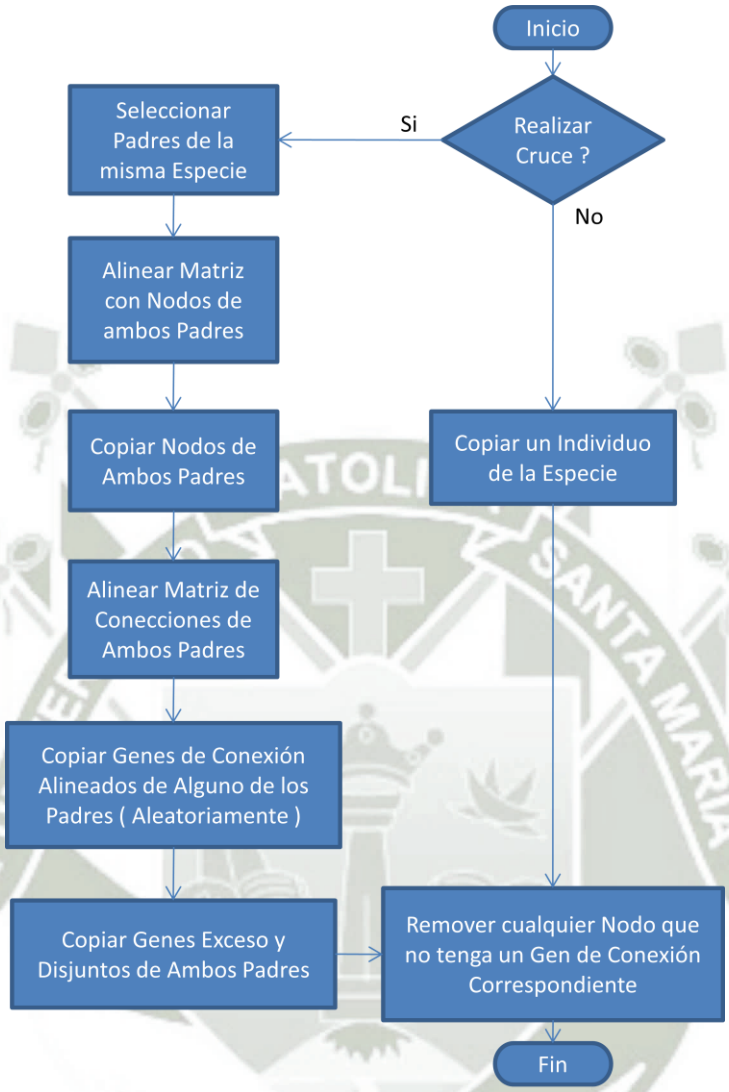

Figura 18-17 Cruce

Luego de decidir a favor de realizar el cruce se seleccionan dos padres de la misma especie a partir del muestreo estocástico realizado previamente. En seguida se alinean sus NodeGenes y se copian todos los nodos de ambos padres en el nuevo individuo, también se alinean los ConnectionGenes de ambos padres, pero sólo se copian los genes de Conexión alineados de alguno de los padres aleatoriamente, luego se copian todos los genes de Exceso y Disjuntos de ambos padres. Por último, independientemente de si se realizó o no el cruce, se remueve cualquier Nodo que no tenga asociado un Gen de Conexión.

18.2.6. Mutar Conexión

El siguiente esquema (Figura 18-18) muestra la lógica de Mutación de Conexión en una Red Neuronal:

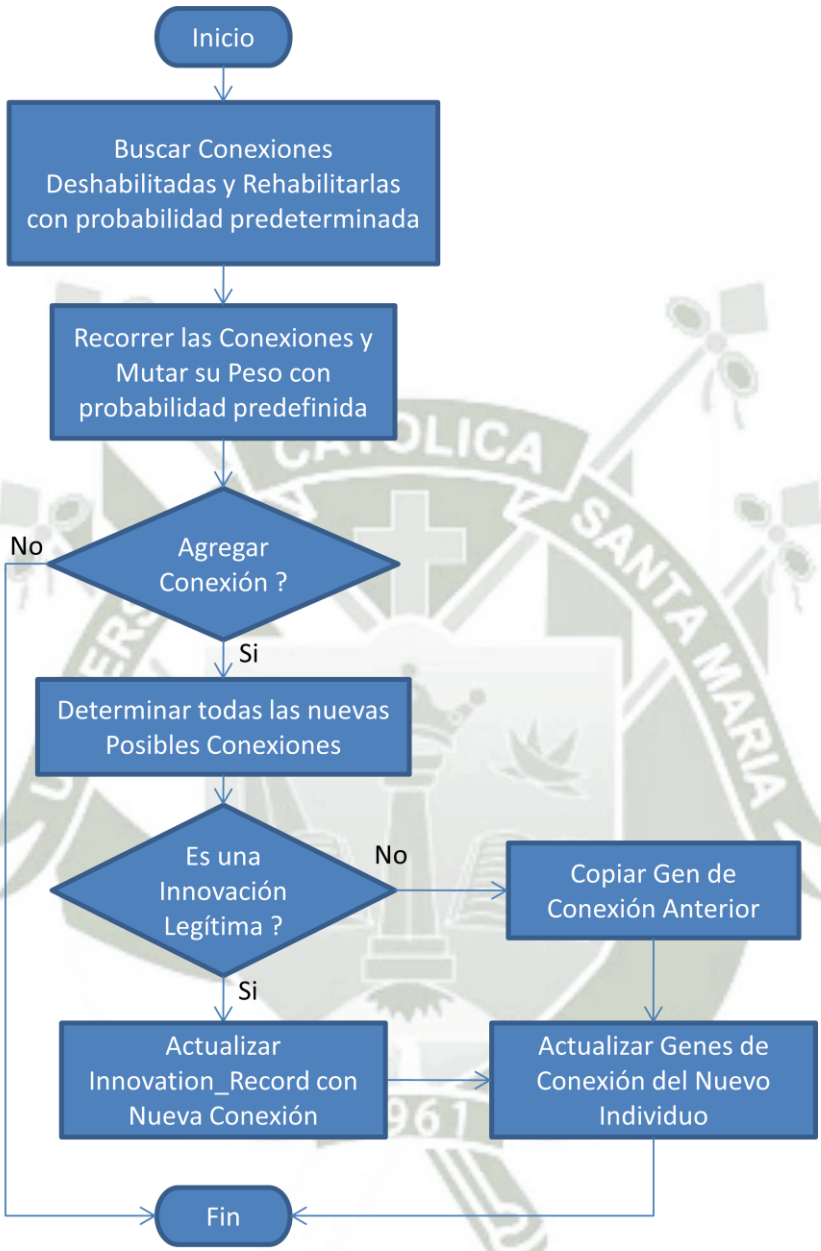

Figura 18-18 Mutación de Conexión

Inicialmente se recorren las conexiones del genoma y de acuerdo a las probabilidades Probability_gene_reenabled y Probability_mutate_weight decidimos si reactivar alguna conexión que previamente haya sido deshabilitada, o si alteramos el peso de alguna Conexión, respectivamente.

En seguida, si decidimos Agregar una Conexión, obtenemos una matriz con todas las conexiones posibles y tomamos una, evaluamos si dicha conexión sería un Innovación legítima, es decir, si no existía previamente una conexión que uniera los mismos nodos con anterioridad. De ser legítima la nueva conexión, se actualiza Innovation_Record con la Conexión y se la agrega al nuevo individuo actualizando su genoma. De lo contrario, se copia la Conexión de la Innovación anterior y actualiza el genoma del Individuo.

18.2.7. Mutar Nodo

El siguiente esquema (Figura 18-19) muestra la lógica de Mutación de Nodo en una Red Neuronal:

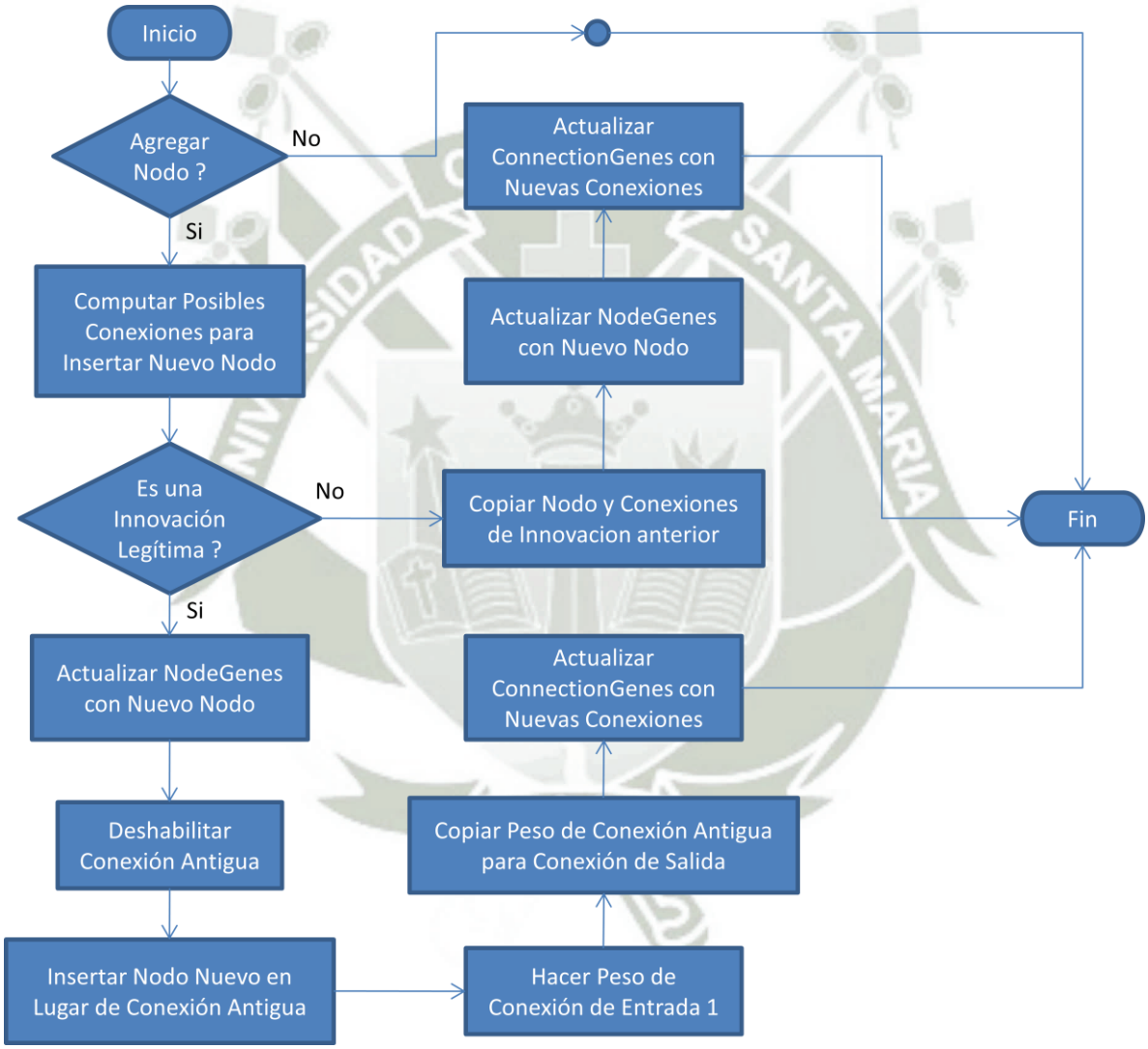

Figura 18-19 Mutación de Nodo

Al agregar un nodo en la estructura de la red podemos alterar gravemente el desempeño de ésta, es por esto que el Algoritmo propuesto plantea una metodología para hacerlo mediante la cual el ruido ocasionado por dicho evento sea el menor posible. El método

utiliza una conexión ya existente en la topología, y la divide en dos, insertando el nodo en el medio, así que para poder hacer esto primero necesitamos computar las posibles conexiones donde podríamos insertar el nodo, luego tomamos una de ellas e investigamos si es una Innovación legítima, es decir, si insertar dicho nodo no resulta en una cadena de conexiones preexistentes. De ser así, actualizamos Innovation_Record con el nuevo nodo y conexiones, a continuación, deshabilitamos la conexión preexistente e insertamos el nuevo nodo en lugar de la misma, luego hacemos que su conexión de entrada tenga peso unitario (de nuevo para alterar en lo menos posible el desempeño de la red), copiamos la antigua conexión para agregarla como conexión de salida y actualizamos el genoma. Si la innovación no es legítima, copiamos las conexiones y nodo de las Innovaciones anteriores y actualizamos el genoma.

18.2.8. Especiación

El siguiente esquema (Figura 18-20) muestra la lógica de Especiación para la Población:

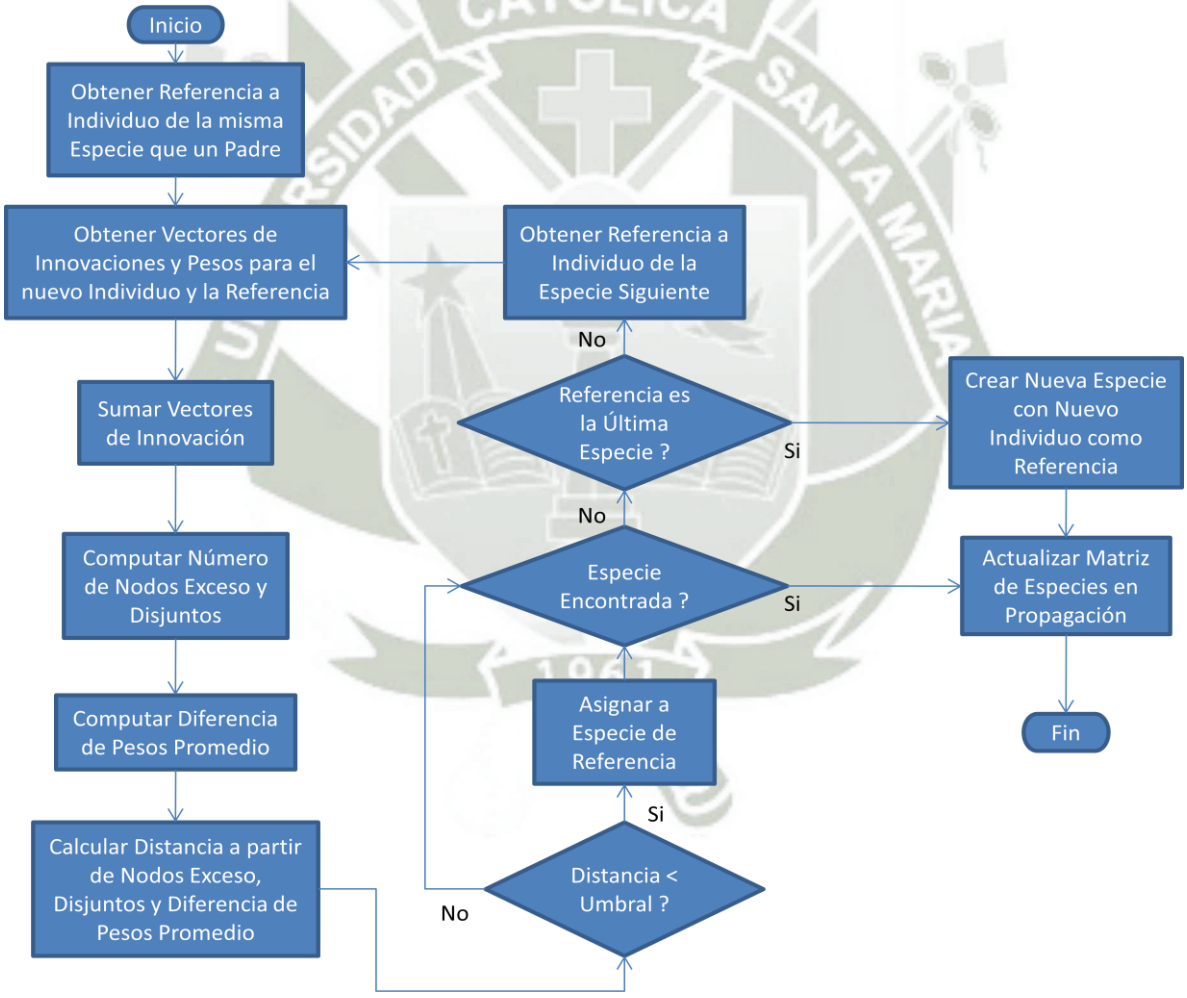

Figura 18-20 Especiación

En términos generales, para poder clasificar a un Individuo en una determinada especie, lo que hacemos es compararlo con un Individuo referencial de dicha especie, la comparación se hace en términos de lo que llamamos "distancia" que es el resultado de un cálculo que compara las diferencias topológicas y los pesos de las conexiones.

La primera comparación la hacemos con una Referencia de la misma especie que alguno de los padres, ya que lo más probable es que el nuevo individuo pertenezca a dicha especie. Luego obtenemos Vectores de Innovación y de Pesos, para poder analizar las diferencias topológicas y los pesos por separado. A continuación sumamos los Vectores de Innovación obtenidos, esto nos permite diferenciar claramente los nodos que ambos individuos tienen en común, los nodos exceso y los nodos disjuntos, también calculamos la diferencia del promedio de los pesos de ambos individuos y con esto calculamos la distancia, de la forma:

$$\delta = \frac{c_1 E}{N} + \frac{c_2 D}{N} + c_3 . \overline{W} \qquad (1)$$

Si la distancia es menor al umbral predeterminado para especiación, asignamos el nuevo individuo a la especie de referencia y actualizamos la matriz de especies existentes y en propagación. Si la distancia es mayor, continuamos iterando sobre las especies restantes en la población, comparando el individuo nuevo con individuos de las demás especies. Si luego de evaluar todas las especies existentes no se ha encontrado pertenencia en ninguna de ellas, procedemos a crear una nueva especie y hacemos al nuevo individuo la referencia de la especie recién creada, por último, actualizamos la matriz de especies existentes y en propagación.

19. Evaluación

Luego de diseñar e implementar el algoritmo, pasamos a evaluar si este método (*Metodología Algorítmica ver Pág. 38*) en realidad es capaz de construir redes neuronales que soluciones problemas tipo.

Para esto, primero se cargan Tablas de Datos ("*Tablas de Datos por Experimento*" *ver Pág. 99*) previamente elaboradas, estas tablas corresponden a la data particular de cada uno de los casos en los que se pretende evaluar el algoritmo. Una vez cargados los datos procedemos según se indica en el diagrama de "*Flujo General*" (*ver Pág. 52*), así que a continuación, luego de inicializar las variables, procedemos a la creación de la "*Población Inicial*" (*ver Pág. 54*), dicha población se crea con estructuras lo más pequeñas posibles, de esta manera estamos "*Minimizando Dimensionalidad mediante el Crecimiento Incremental de Estructuras Mínimas*" (*ver Pág. 43*) lo cual contribuirá a que las posibles soluciones sean redes con el menor número de nodos y conexiones posibles.

Además, el genotipo de todos los individuos durante la totalidad del proceso utilizará una "*Codificación Genética*" (*ver Pág. 38*) "Directa" ("*Codificación Directa e Indirecta de Redes*" *ver Pág. 30*), para poder tener mayor control sobre el proceso evolutivo, al mismo tiempo, se introduce en dicha codificación el concepto de "*Rastreando Genes con Marcadores Históricos*" (*ver Pág. 40*) a través de una variable denominada "Innovación"("*Innovation_Record*" *ver Pág. 45*), ambas características pueden ser observadas en las estructura de datos "*NodeGenes*", que se enfoca en la representación de los nodos individuales de la red y en sus características; y "*ConnectionGenes*" que se enfoca en las conexiones entre nodos y sus características (*ver Pág. 44*).

Una vez que tenemos la población inicial, se procede a una primera especiación, la cual por lo general produce ente 1 a 3 especies iniciales, esto se debe a que la población inicial es muy homogénea. A continuación, se procede a "*Evaluar Red Neuronal*" (*ver Pág. 55*) y así obtenemos la "Aptitud" ("*Selección*" *ver Pág. 24*) de cada red y también de cada especie, con este dato, y considerando el número de generaciones que han transcurrido, podemos evaluar el "Estancamiento" ("*Stagnation*" *ver Pág. 48*) de alguna especie, y decidir si proceder con su extinción, o de ser el número de especies estancadas alto, con el "Reenfoque" ("*Refocus*" *ver Pág. 49*), que ocasiona una eliminación masiva de la mayoría de especies.

Luego de esta evaluación podemos decir si hemos alcanzado una solución, de no ser el caso procedemos con la "*Reproducción*" (*ver Pág. 56*) de los individuos en la población. Es como parte de ésta función, que los procesos de "*Cruce*" (*ver Pág. 58*), "*Mutar Conexión*" (*ver Pág. 59*) y "*Mutar Nodo*" (*ver Pág. 60*) permiten que los individuos más aptos ("*Selección*" *ver Pág. 24*) dentro de cada especie produzcan descendencia, mientras que los menos aptos son eliminados de la población ("*Elitismo*" *ver Pág. 29*).

Por último, la nueva población es sometida al proceso de "*Especiación*" (*ver Pág. 61*), siguiendo la metodología explicada en "*Protegiendo Innovación con Especiación*" (*ver Pág. 41*) para luego

volver a evaluar a los individuos e iterar sobre este loop hasta que se encuentre una solución o hasta que se alcance el número máximo de generaciones.

Es con la aplicación de este método, que se pretende atender a la "*Problemática*" (*ver Pág. 34*) planteada anteriormente.

A continuación se presentan los resultados de evaluar la eficacia del método para resolver una serie de problemas que van desde Regresión Lineal hasta Control Automático. Para todas las evaluaciones se utilizó un límite de poblacional de 150 individuos.

19.1. Regresión Lineal

El problema de regresión lineal consiste en obtener, a partir del set de datos proporcionado, un modelo de las relaciones subyacentes entre las variables de entrada y de salida. En este caso en particular, los datos describen una función lineal de la forma:

$$f(x) = 3x + 11$$

Sin embargo, el programa no tiene conocimiento del dominio del problema, lo que trata de hacer es estimar esta relación, y producir una estructura neuronal que aproxime dicha función.

Se prepara una tabla con el set de datos (Tabla 23-1) que se alimentarán a las redes neuronales y se la importa al Espacio de Trabajo de Matlab para que el algoritmo disponga de la data en memoria.

Los resultados de la evaluación son los siguientes:

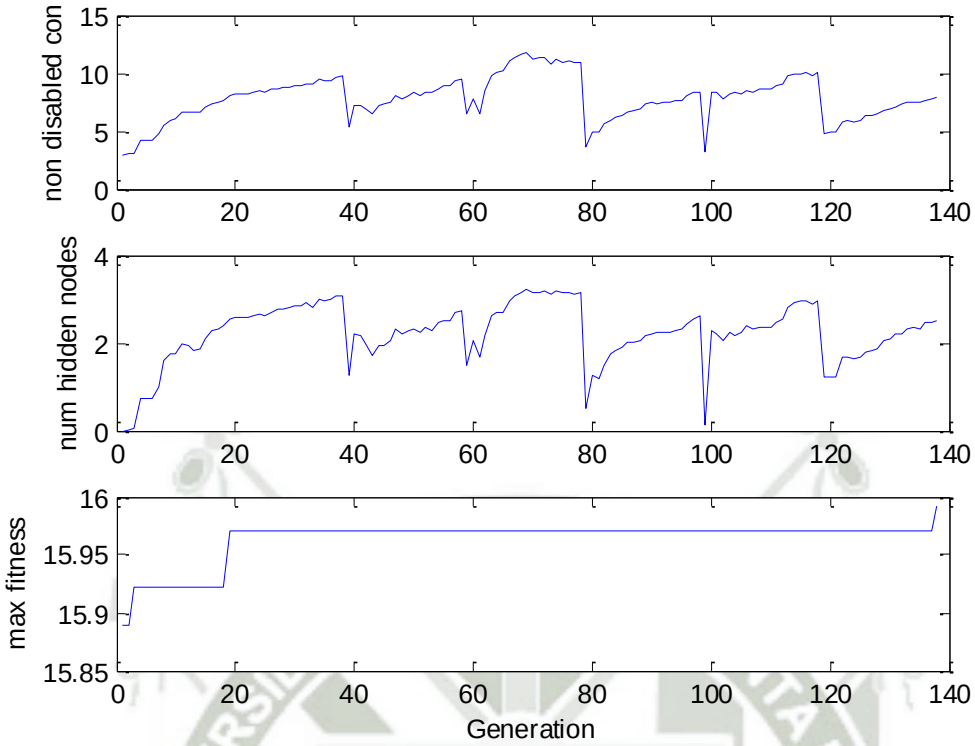

Figura 19-1 Número de conexiones y nodos, grado de Aptitud (Regresión Lineal)

En la Figura 19-1 podemos comparar el número de conexiones promedio, el número de nodos promedio y la máxima aptitud alcanzada en la población, versus las generaciones. Podemos ver claramente cómo el algoritmo encuentra soluciones con aptitudes bastante altas dentro de las primeras 20 generaciones, en este punto la solución que se tiene es lo suficientemente capaz de modelar la data, sin embargo se dejó que algoritmo continuara optimizando las redes, y si vemos las últimas generaciones podemos ver que al final se hizo un gran avance.

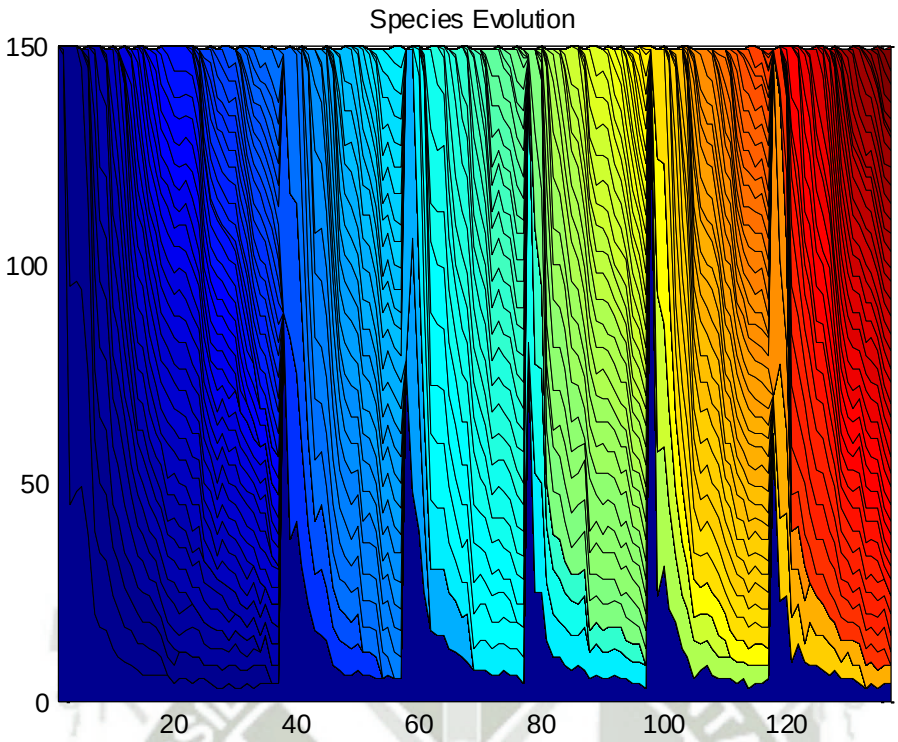

Figura 19-2 Individuos por especies (Regresión Lineal)

En la Figura 19-2 podemos ver la superposición de la cantidad de individuos por especie en un gráfico de áreas, donde el área dedicada a cada tono es directamente proporcional con el número de individuos por especie en dicha generación. Los colores más fríos representan las especies más antiguas, mientras que los colores más cálidos representan especies más jóvenes.

Podemos notar que mientras la aptitud máxima de la población se mantiene estática la cantidad de individuos en ciertas especies cambia drásticamente con cierto periodo generacional. Esto es a causa del reenfoque que el algoritmo fuerza sobre la población, cada 20 generaciones que la aptitud máxima de la población no mejora se eliminan todas menos las dos especies más aptas. Podemos ver este fenómeno repetirse una y otra vez mientras que la aptitud máxima no mejore, y como es de esperarse, el número promedio de conexiones y nodos ocultos también sufren cambios drásticos a cause del reenfoque.

En este caso en particular, podemos ver que es una de las especies más antiguas la que sobrevive los eventos de reenfoque, esto se refleja en el tono azul que persiste y coexiste hacia el final de la evolución con colores mucho más cálidos.

Al final de la evolución, la mejor de la redes tiene la siguiente estructura (Figura 19-3):

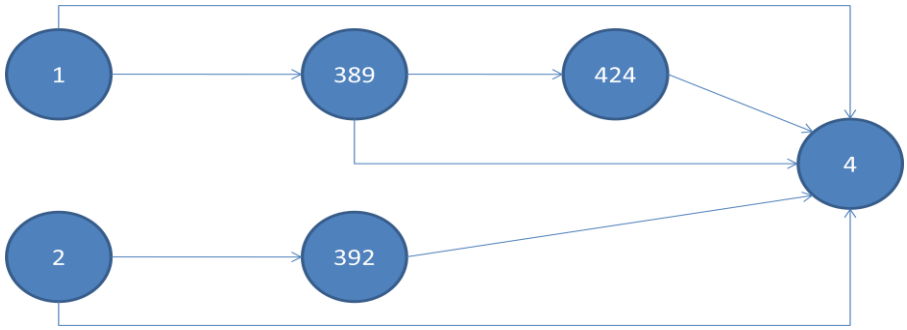

Figura 19-3 Mejor RNA (Regresión Lineal)

Y su genoma de conexiones es:

	Innovaciones								
Innovation Number	1	2	3	1285	1286	1295	1296	1417	1418
Connection From	1	2	3	1	389	2	392	389	424
Connection To	4	4	4	389	4	392	4	424	4
Weight	3.853	-1.114	-0.616	1.388	-1.452	2.984	0.246	-0.455	0.622
Enable Bit	1	1	0	1	1	1	1	1	1

Tabla 19-1 Genoma de Conexiones (Regresión Lineal)

Como resultado, al evaluar esta red obtenemos el siguiente gráfico (Figura 19-4):

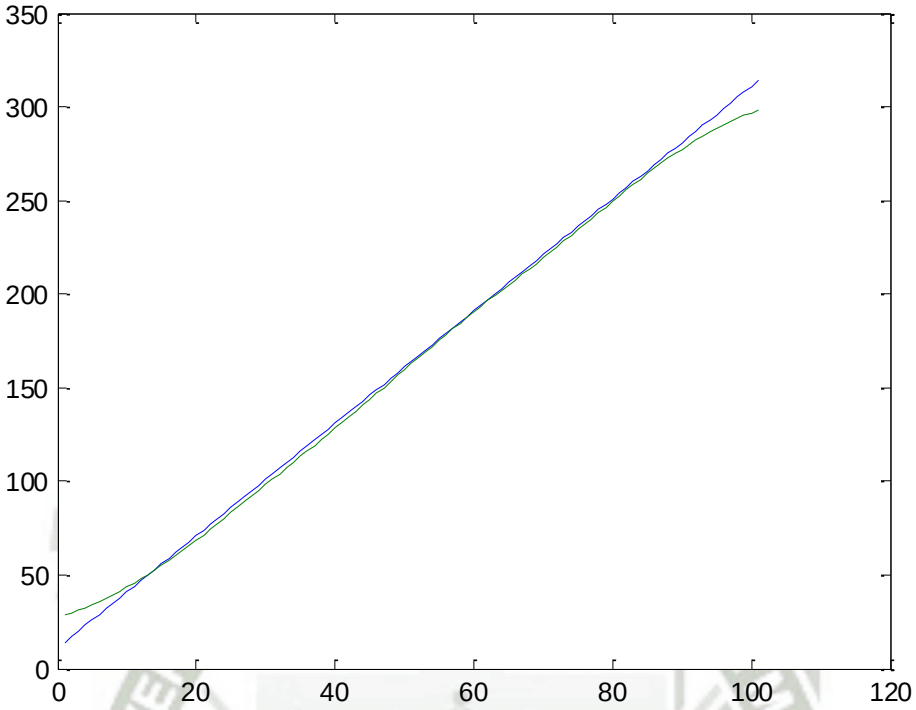

Figura 19-4 Evaluación de la RNA (Regresión Lineal)

Donde la Línea azul representa el set de datos objetivo, es decir, los datos originales alimentados a la red durante su entrenamiento y evolución. Y la línea verde representa la salida de la red luego de iterar sobre el set de datos de entrada.

Como podemos ver, ambas curvas están alineadas, especialmente al centro de la gráfica, y difieren ligeramente en los extremos de estas, probablemente como resultado de la naturaleza no lineal de la red neuronal. En todo caso, podemos decir que el programa produjo una red capaz de aproximar la función propuesta en los datos.

19.2. Regresión No Lineal

El problema de regresión no lineal consiste en obtener, a partir del set de datos proporcionado, un modelo de las relaciones subyacentes entre las variables de entrada y de salida. En este caso en particular, los datos describen una función no lineal de la forma:

$$f(x) = x^2 + 4x + 8$$

Sin embargo, el programa no tiene conocimiento del dominio del problema, lo que trata de hacer es estimar esta relación, y producir una estructura neuronal que aproxime dicha función.

Se prepara una tabla con el set de datos (Tabla 23-2) que se alimentarán a las redes neuronales y se la importa al Espacio de Trabajo de Matlab para que algoritmo disponga de la data en memoria.

Los resultados de la evaluación son los siguientes:

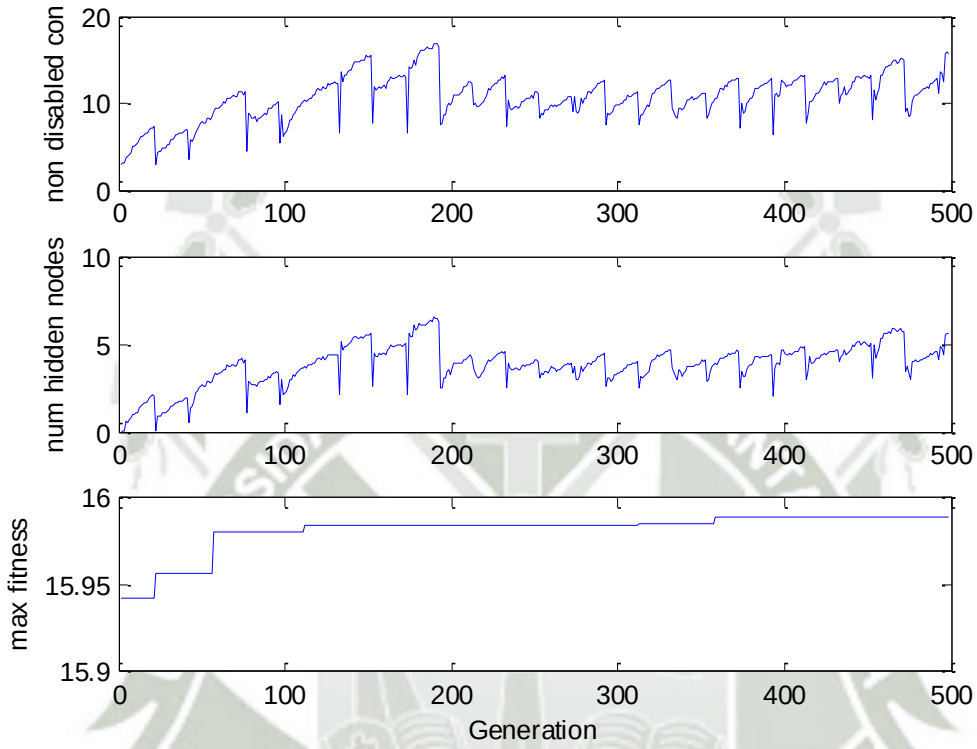

Figura 19-5 Número de conexiones y nodos, grado de Aptitud (Regresión No Lineal)

En la Figura 19-5 podemos comparar el número de conexiones promedio, el número de nodos promedio y la máxima aptitud alcanzada en la población, versus las generaciones. Podemos ver claramente cómo el algoritmo encuentra soluciones con aptitudes bastante altas dentro de las primeras 100 generaciones, en este punto la solución que se tiene es lo suficientemente capaz de modelar la data, sin embargo se dejó que algoritmo continuara optimizando las redes, y como vemos al final se hizo un gran avance.

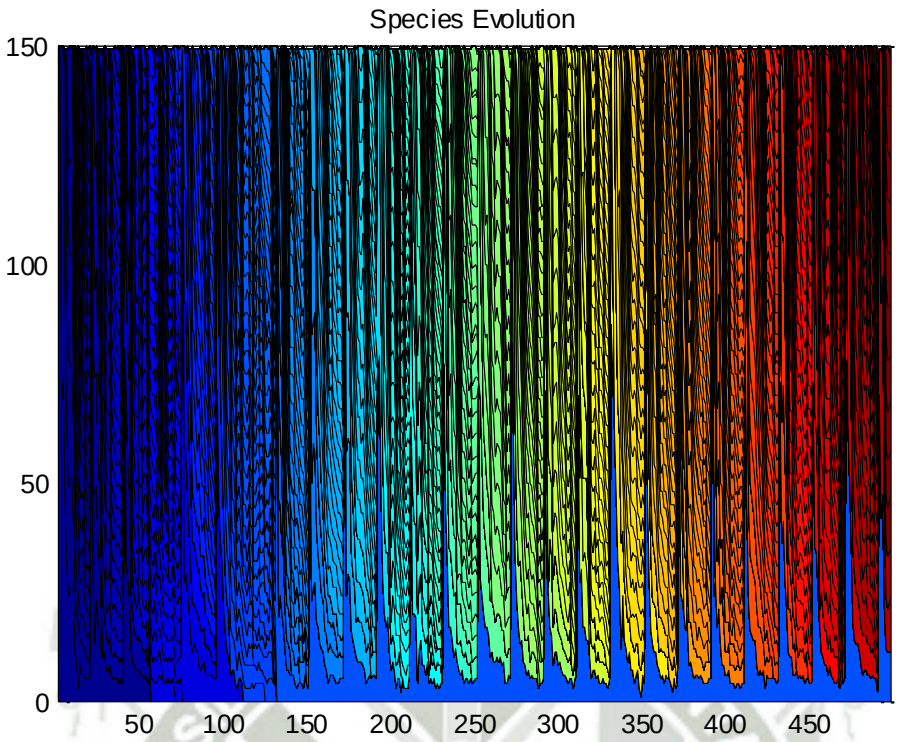

Figura 19-6 Individuos por especies (Regresión No Lineal)

En la Figura 19-6 podemos ver la superposición de la cantidad de individuos por especie en un gráfico de áreas, donde el área dedicada a cada tono es directamente proporcional con el número de individuos por especie en dicha generación. Los colores más fríos representan las especies más antiguas, mientras que los colores más cálidos representan especies más jóvenes.

Podemos notar que mientras la aptitud máxima de la población se mantiene estática la cantidad de individuos en ciertas especies cambia drásticamente con cierto periodo generacional. Esto es a causa del reenfoque que el algoritmo fuerza sobre la población, cada 20 generaciones que la aptitud máxima de la población no mejora se eliminan todas menos las dos especies más aptas. Podemos ver este fenómeno repetirse una y otra vez mientras que la aptitud máxima no mejore, y como es de esperarse, el número promedio de conexiones y nodos ocultos también sufren cambios drásticos a cause del reenfoque.

En este caso en particular, podemos ver que es una de las especies más antiguas la que sobrevive los eventos de reenfoque, esto se refleja en el tono azul que persiste y coexiste hacia el final de la evolución con colores mucho más cálidos.

Al final de la evolución, la mejor de la redes tiene la siguiente estructura (Figura 19-7):

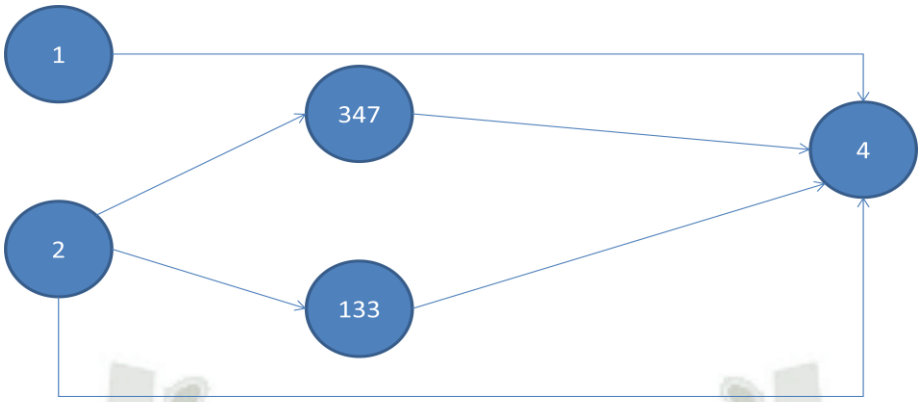

Figura 19-7 Mejor RNA (Regresión No Lineal)

Y su genoma de conexiones es:

	Innovaciones						
Innovation Number	1	2	3	451	452	1200	1201
Connection From	1	2	3	2	133	2	347
Connection To	4	4	4	133	4	347	4
Weight	0.246	5.825	2.121	-0.587	1.649	-0.775	5.345
Enable Bit	1	1	0	1	1	1	1

Tabla 19-2 Genoma de Conexiones (Regresión No Lineal)

Como resultado, al evaluar esta red obtenemos el siguiente gráfico (Figura 19-8):

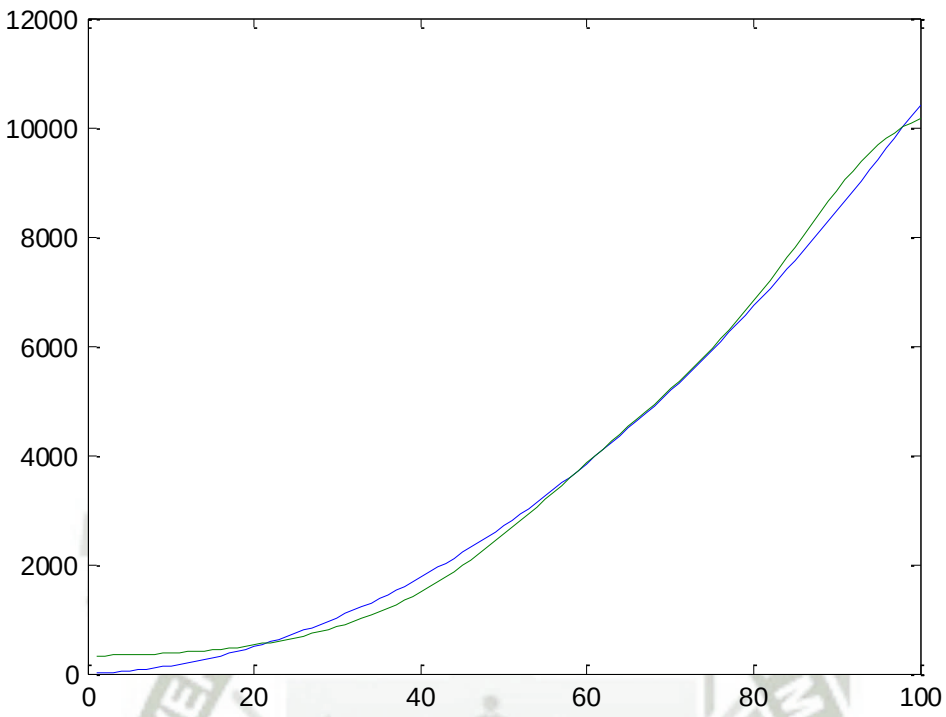

Figura 19-8 Evaluación de la RNA (Regresión No Lineal)

Donde la Línea azul representa el set de datos objetivo, es decir, los datos originales alimentados a la red durante su entrenamiento y evolución. Y la línea verde representa la salida de la red luego de iterar sobre el set de datos de entrada.

Como podemos ver, ambas curvas están alineadas, especialmente al centro de la gráfica, y difieren en los extremos de estas. En todo caso, podemos decir que el programa produjo una red capaz de aproximar la función propuesta en los datos.

19.3. Clasificación: El Problema XOR

El problema XOR es un problema lógico, derivado del operador Disyunción Exclusiva en teoría de conjuntos. Está caracterizado por la siguiente Tabla 19-3 de verdad:

Xi	Yi	Zo
0	0	0
0	1	1
1	0	1
1	1	0

Tabla 19-3 Tabla de verdad XOR

Debido a que XOR no es linealmente separable, una red neuronal requiere nodos ocultos para resolverlo. Las dos entradas se deben combinar en algún nodo oculto, en contraposición a sólo en el nodo de salida, porque no hay ninguna función sobre una combinación lineal de las entradas que pueda separar los elementos de entrada en las clases adecuadas. Estos requisitos estructurales hacen XOR adecuado para probar la capacidad del algoritmo para evolucionar estructura. Por ejemplo, este método para la adición de nuevos nodos podría ser demasiado destructivo para permitir que nuevos nodos entren en la población. O bien, podría encontrar un campeón local con un tipo equivocado de conectividad que domina la población tanto que los sistemas no logran desarrollar la conectividad adecuada. En tercer lugar, tal vez la estructura cambiante hace valores de peso de conexión pasados obsoletos. Si es así, el algoritmo tendría problemas para la ampliación de topologías que ya son en gran parte especializadas. Este experimento está destinado a mostrar que el programa no se ve impedido por dichos obstáculos potenciales, pero puede evolucionar estructura eficiente y consistente cuando sea necesario.

Los resultados de la evaluación son los siguientes:

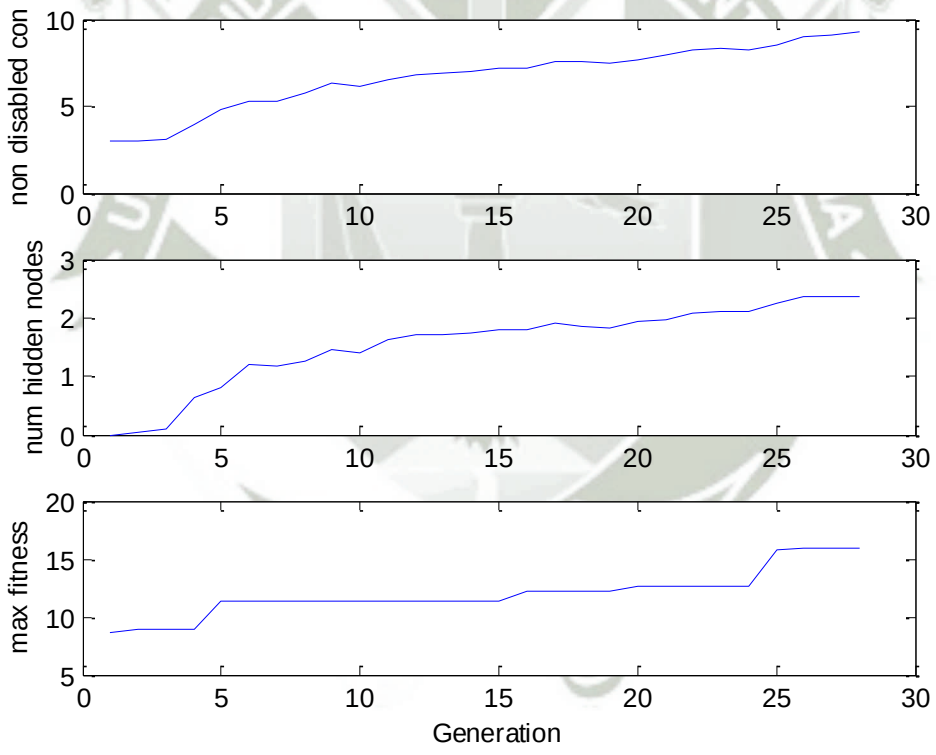

Figura 19-9 Número de conexiones y nodos, grado de Aptitud (XOR)

En la Figura 19-9 podemos comparar el número de conexiones promedio, el número de nodos promedio y la máxima aptitud alcanzada en la población, versus las generaciones. Podemos ver claramente cómo el algoritmo encuentra soluciones con aptitudes bastante altas dentro

de las primeras 30 generaciones. De hecho, el algoritmo es capaz de evolucionar una red con aptitud cercana a la máxima posible en esta implementación, es decir, una aptitud de 15.9998.

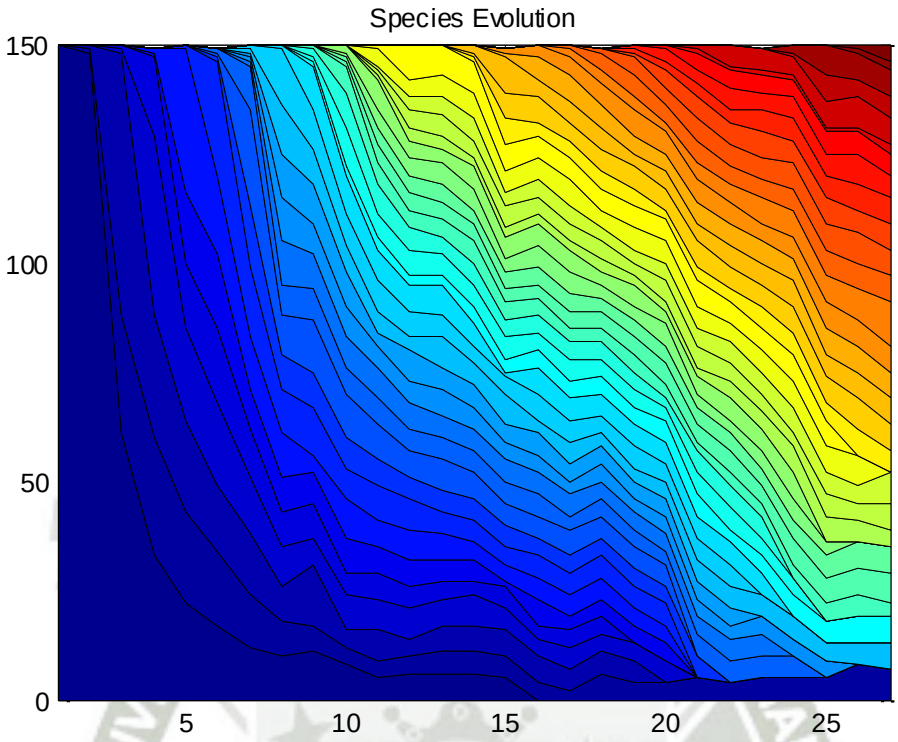

Figura 19-10 Individuos por especies (XOR)

En la Figura 19-10 podemos ver la superposición de la cantidad de individuos por especie en un gráfico de áreas, donde el área dedicada a cada tono es directamente proporcional con el número de individuos por especie en dicha generación. Los colores más fríos representan las especies más antiguas, mientras que los colores más cálidos representan especies más jóvenes.

En este caso en particular, podemos ver que es una de las especies más antiguas la que sobrevive, esto se refleja en el tono azul que persiste y coexiste hacia el final de la evolución con colores más cálidos.

Al final de la evolución, la mejor de la redes tiene la siguiente estructura (Figura 19-11):

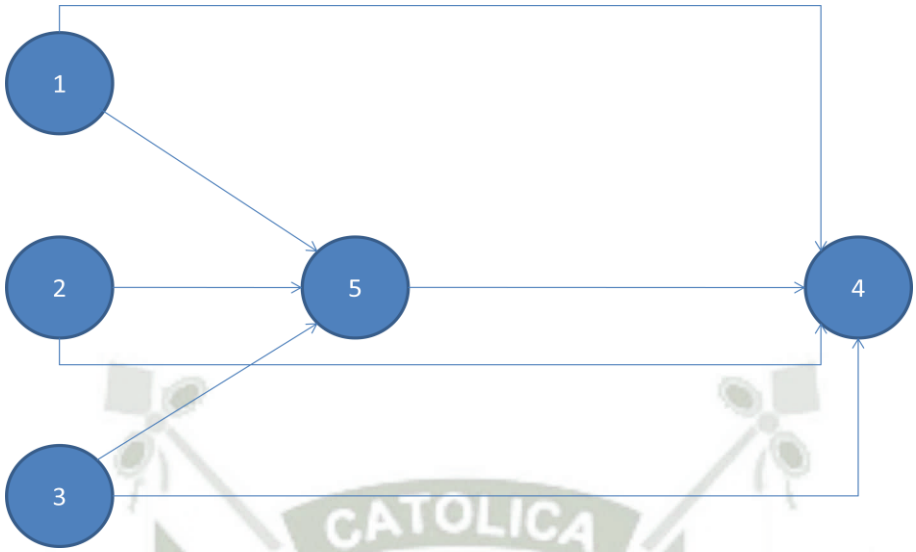

Figura 19-11 Mejor RNA (XOR)

Y su genoma de conexiones es:

	Innovaciones						
Innovation Number	1	2	3	4	5	10	18
Connection From	1	2	3	1	5	2	3
Connection To	4	4	4	5	4	5	5
Weight	3.282	3.605	-3.793	4.203	-8.000	5.347	-7.640
Enable Bit	1	1	1	1	1	1	1

Tabla 19-4 Genoma de Conexiones (XOR)

Como resultado, al evaluar esta red obtenemos el siguiente gráfico:

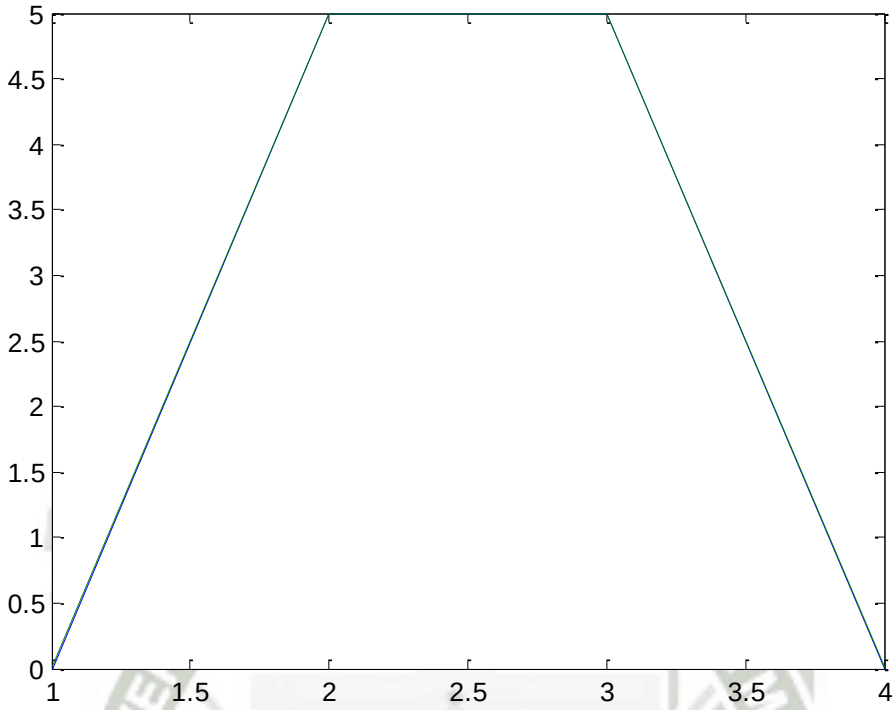

Figura 19-12 Evaluación de la RNA (XOR)

Donde la Línea azul representa el set de datos objetivo, es decir, los datos originales alimentados a la red durante su entrenamiento y evolución. Y la línea verde representa la salida de la red luego de iterar sobre el set de datos de entrada.

Como podemos ver, ambas curvas están alineadas casi perfectamente. En todo caso, podemos decir que el programa produjo una red capaz de aproximar la función propuesta en los datos.

19.4. Control: Péndulo Invertido

El Control Automático es un área que se ha visto beneficiada de desarrollos en Inteligencia Artificial, Sistemas Expertos, Fuzzy Logic, y por supuesto Redes Neuronales. En este caso tratamos de resolver una versión simplificada de un problema tipo en teoría de control conocido como, Péndulo Invertido. Es importante notar que el análisis del sistema del Péndulo Invertido es en sí mismo un tema amplio de investigación, así que en esta sección nos limitaremos a describir el problema sin entrar en detalles.

El sistema de péndulo invertido es un problema de control clásico que se utiliza en universidades de todo el mundo. Es un procedimiento adecuado para probar controladores prototipo debido a sus altas no linealidades y su falta de estabilidad. El sistema consta de un péndulo invertido unido con bisagras a un carro que es libre de moverse en la dirección x. En

esta sección, se muestran las ecuaciones dinámicas del sistema, el modelo será desarrollado en Simulink y se desarrollará un controlador PID básico. El objetivo de desarrollar un péndulo invertido en Simulink es que el modelo desarrollado tendrá las mismas características que el proceso real. Será posible probar el controlador neuronal prototipo en el medio ambiente Simulink. Antes de que el modelo de péndulo invertido pueda ser desarrollado en Simulink, las ecuaciones dinámicas del sistema se obtienen a partir de 'Ecuaciones de Newton'. Las ecuaciones de Newton son uno de los muchos métodos de determinación de las ecuaciones del sistema. El uso de este método, hace posible derivar las ecuaciones del sistema dinámico para un sistema mecánico complicado como el péndulo invertido. Las siguientes Figura 19-13 y Figura 19-14 son diagramas de cuerpo libre del sistema de péndulo.

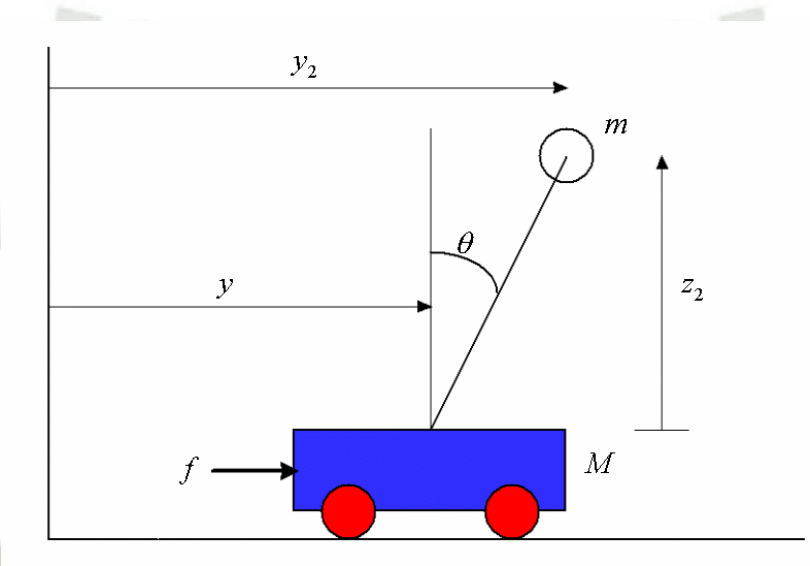

Figura 19-13 Arreglo del Péndulo Invertido

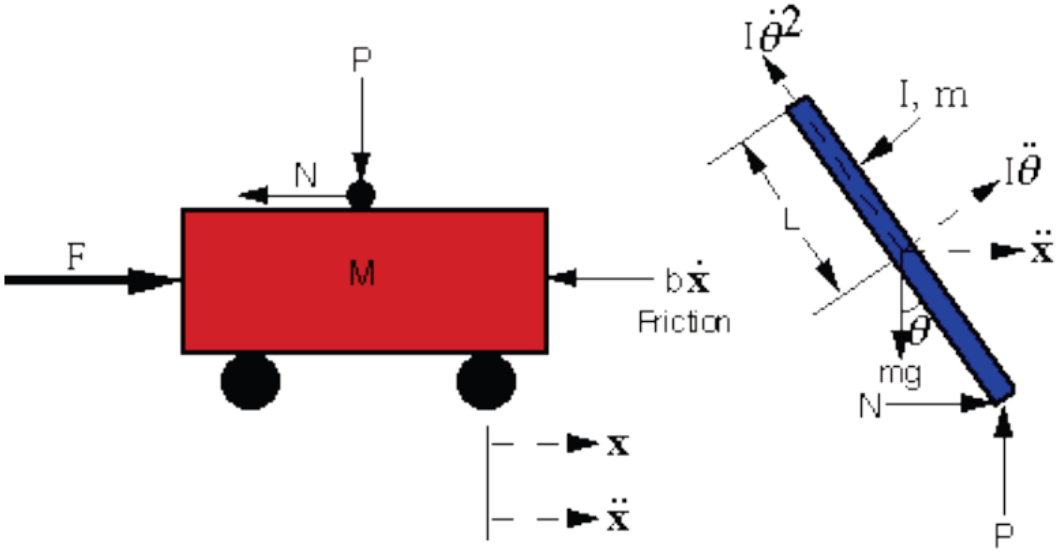

Figura 19-14 DLCs para el Carro y el Péndulo

Para este ejemplo asumiremos:

- M = 0.5 kg — Masa del Carro.
- m = 0.2 kg — Masa del Péndulo.
- b = 0.1 N/m/seg — Fricción del Carro.
- l = 0.3 m — Largo hasta el centro de Masa del Péndulo.
- I = 0.006 kg*m2 — Inercia del Péndulo.
- F = ? — Fuerza aplicada al Carro.
- X = ? — Coordenada de Posición del Carro.
- Theta = ? — Ángulo del Péndulo respecto a la vertical.

En este ejemplo, vamos a implementar un controlador PID que sólo puede aplicarse a un sistema de una sola salida y de una sola entrada (SISO), por lo que estaremos más interesados en el control del ángulo del péndulo. Por lo tanto, ninguno de los criterios de diseño considera la posición del carro. Vamos a suponer que el sistema se inicia en el equilibrio, y experimenta una fuerza de perturbación en forma de ruido blanco. El péndulo debe permanecer en su posición vertical, y nunca moverse más de 0.35 radianes de la vertical.

Ambos, el carro y el péndulo tienen un grado de libertad, X y Theta respectivamente. Entonces se modelan las Ecuaciones de Newton para ambos grados de libertad:

$$\frac{d^2x}{dt^2} = \frac{1}{M}\sum_{carro} F_x = \frac{1}{M}\left(F - N - b\frac{dx}{dt}\right)$$

$$\frac{d^2\theta}{dt^2} = \frac{1}{I}\sum_{pend} \tau = \frac{1}{I}(NL\cos(\theta) + PL\sin(\theta))$$

Sin embargo es necesario que incluyamos las fuerzas N y P que interactúan entre el carro y el péndulo para poder modelar la dinámica.

$$N = m\frac{d^2x_p}{dt^2}$$

$$P = m\left(\frac{d^2y_p}{dt^2} + g\right)$$

Adicionalmente, xp y yp son funciones exactas de Theta, así que podemos representar sus derivadas en términos de la derivada de Theta.

$$\frac{d^2x_p}{dt^2} = \frac{d^2x}{dt^2} + L\sin\theta\left(\frac{d\theta}{dt}\right)^2 - L\cos\theta\left(\frac{d^2\theta}{dt^2}\right)$$

$$\frac{d^2y_p}{dt^2} = -L\cos\theta\left(\frac{d\theta}{dt}\right)^2 - L\sin\theta\left(\frac{d^2\theta}{dt^2}\right)$$

Estas son las ecuaciones que usaremos para modelar el sistema en Simulink, es así que resulta el siguiente modelo (Figura 19-15):

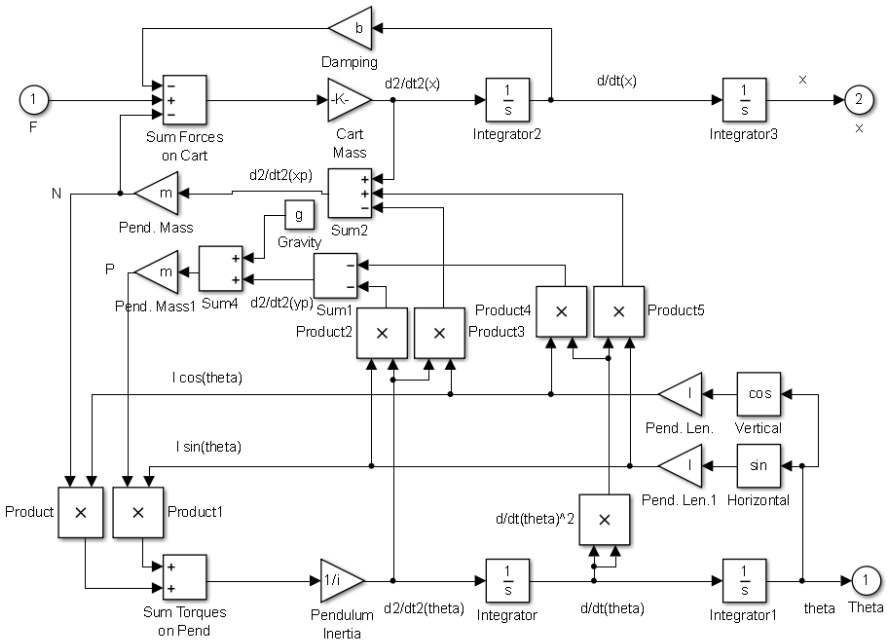

Figura 19-15 Modelo Simulink para el Péndulo Invertido

Para poder utilizar este modelo en nuestro programa se lo redujo a un componente (Inverted Pendulum) donde la entrada F es la Fuerza, y las salidas X y Theta son la Posición y el Ángulo respectivamente. Ya que estamos interesados en controlar el Ángulo, se generó un bloque extra llamado Limit_Angle, la única función de este bloque es poner límites inferiores y superiores sobre la salida Theta, en 0 y 2π radianes respectivamente.

Ahora, evaluamos la respuesta de lazo abierto del sistema del Péndulo que acabamos de modelar (Figura 19-15), para esto hacemos uso de Simulink (Figura 19-16).

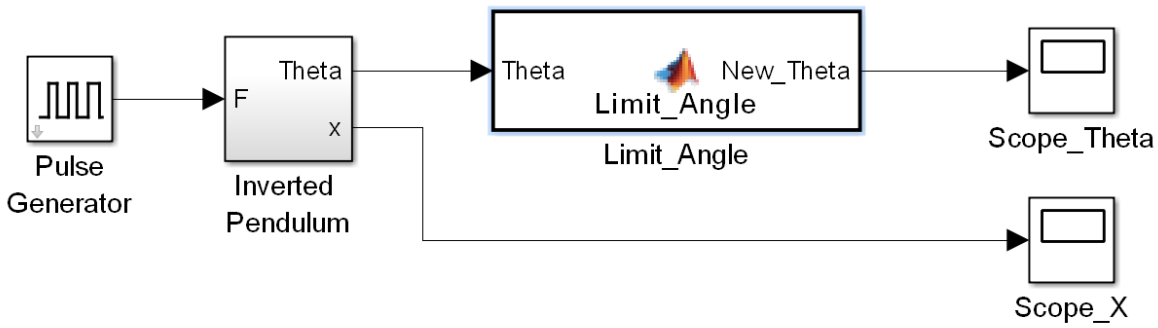

Figura 19-16 Arreglo para Respuesta de Lazo Abierto

Luego de correr la simulación el Ángulo se comporta de la siguiente manera:

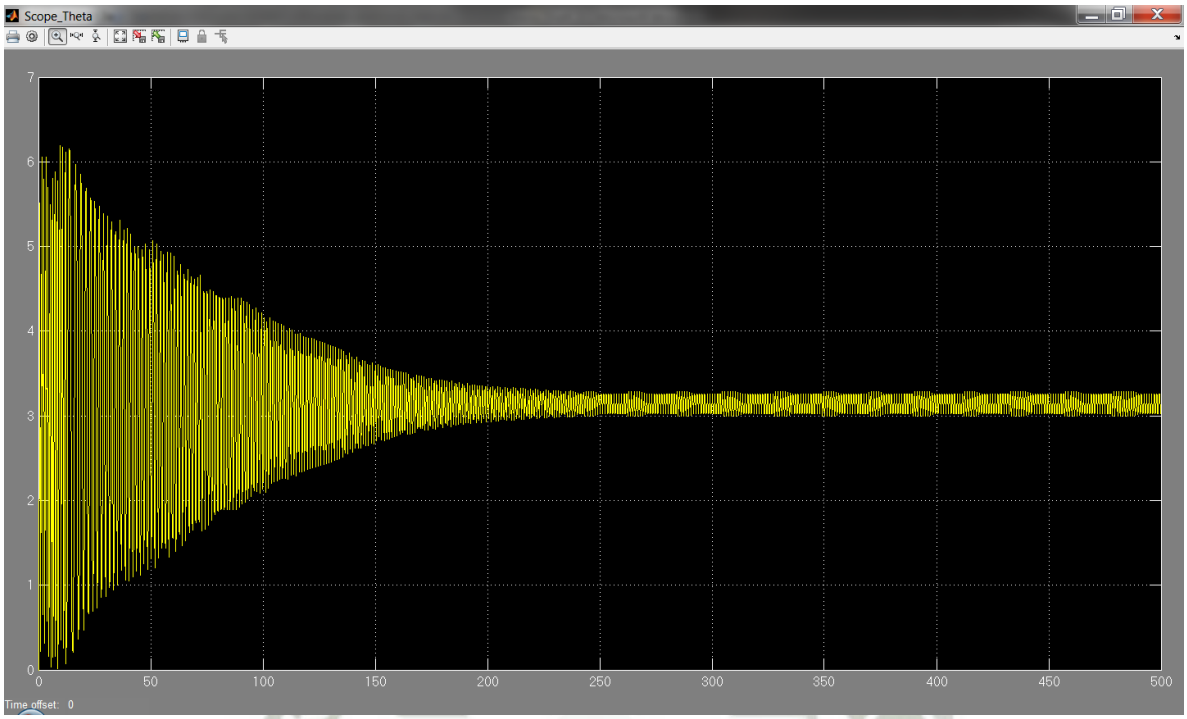

Figura 19-17 RLA para Theta

Como podemos ver en la Figura 19-17, después de la perturbación inicial el péndulo es sacado del estado de equilibrio y empieza a oscilar entre 0 y 2π radianes, es decir, el péndulo da varias vueltas para luego oscilar alrededor de π radianes, es decir, completamente hacia abajo.

Luego de verificar que el sistema se comporta como esperábamos, podemos pasar a la etapa de control. No debemos olvidar que el objetivo es diseñar un Controlador Neuronal que pueda aprender del Controlador PID, para poder evolucionar y entrenar la red neuronal necesitamos un set de datos que modele la relación entrada/salida del Controlador PID. Con este objetivo, diseñamos una simulación que capture los datos de entrada y salida del Controlador mientras éste trata de balancear el Péndulo ante una señal de ruido.

Para poder obtener los parámetros de dicho controlador, debemos primero analizar la función de transferencia del péndulo considerando el ángulo:

$$\frac{\theta_{(s)}}{U_{(s)}} = \frac{\frac{ml}{q}s}{s^3 + \frac{b(I+ml^2)}{q}s^2 - \frac{(M+m)mgl}{q}s - \frac{bmgl}{q}}$$

donde:

$$q = [\,(M+m)(I+ml^2) - (ml)^2\,]$$

Ésta función de transferencia puede ser visualizada con un diagrama del tipo (Figura 19-18):

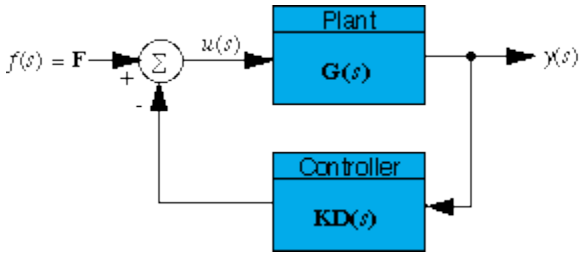

Figura 19-18 Lazo de Control

De este modo, ahora podemos analizar la respuesta del sistema a un impulso, para esto nos ayudamos de Matlab con el siguiente Script:

```
M = 0.5;
m = 0.2;
b = 0.1;
i = 0.006;
g = 9.8;
l = 0.3;

q = (M+m)*(i+m*l^2)-(m*l)^2;

num = [m*l/q  0];
den = [1  b*(i+m*l^2)/q  -(M+m)*m*g*l/q  -b*m*g*l/q];
pend=tf(num,den);
Kd = 1;
Kp = 1;
Ki = 1;
contr=tf([Kd Kp Ki],[1 0]);
sys_cl=feedback(pend,contr);
t=0:0.01:5;
impulse(sys_cl,t)
axis([0 1.5 0 40])
```

La respuesta del sistema, como se ve en la Figura 19-19, no es estable.

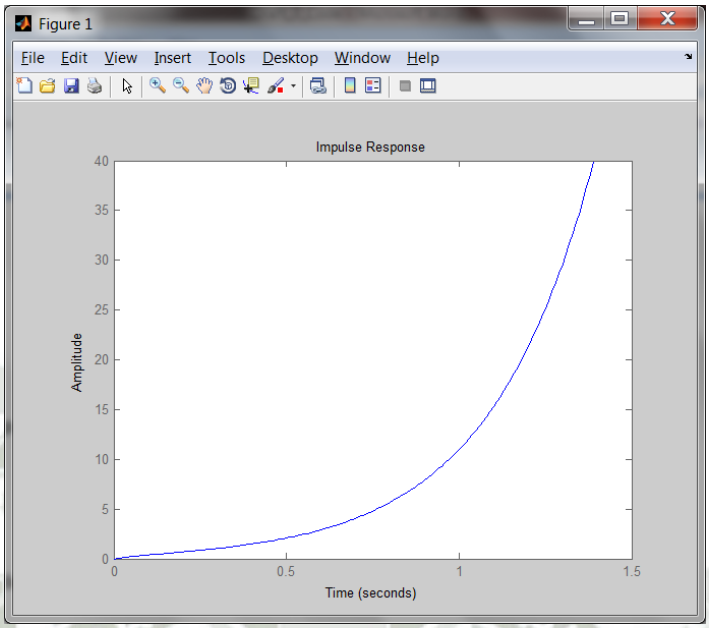

Figura 19-19 Respuesta Impulso Kd=1, Kp=1, Ki=1

Sin embargo, si incrementamos el componente proporcional a Kp =100, el sistema se estabiliza y el error en estado estable se hace cero (Figura 19-20):

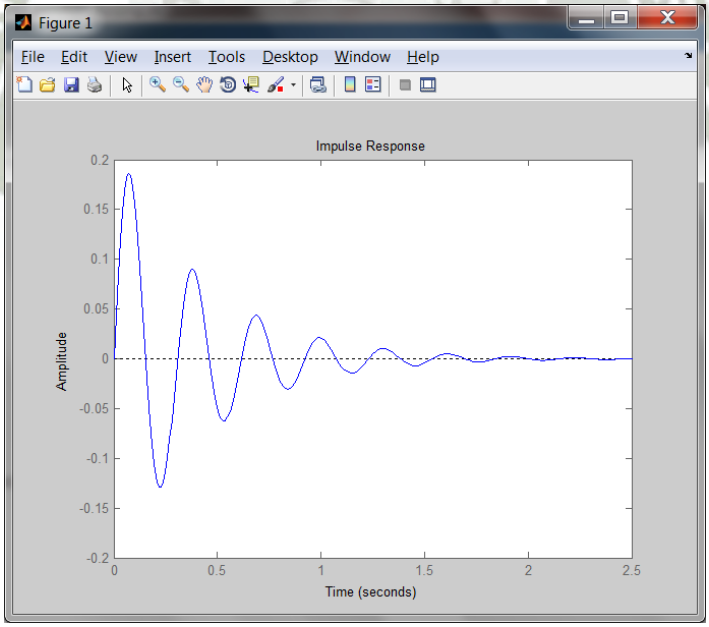

Figura 19-20 Respuesta Impulso Kd=1, Kp=100, Ki=1

Sin embargo el sobreimpulso inicial todavía es muy alto, así que para aliviar esto podemos incrementar el componente derivativo a Kd=20, y así obtener la respuesta que se muestra en la Figura 19-21:

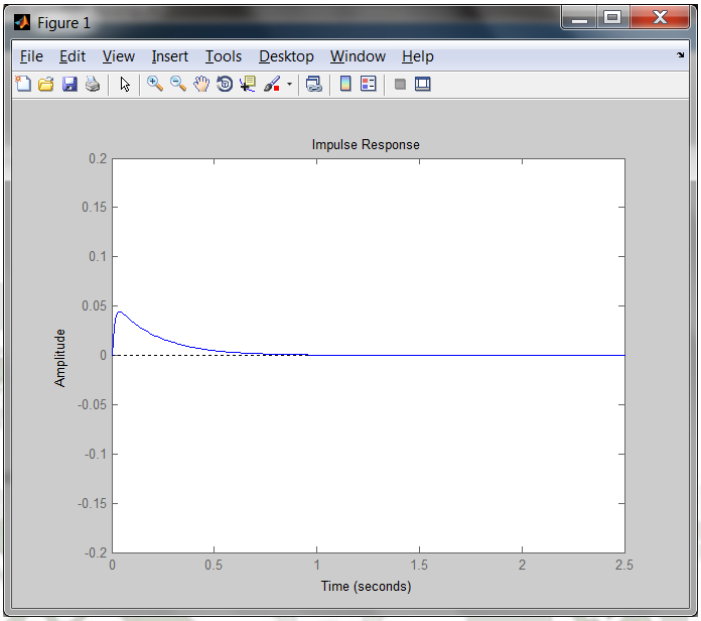

Figura 19-21 Respuesta Impulso Kd=20, Kp=100, Ki=1

Con éste cambio tenemos los parámetros que necesitábamos para el controlador PID, así que procedemos con la construcción del modelo en Simulink como se ve a continuación (Figura 19-22):

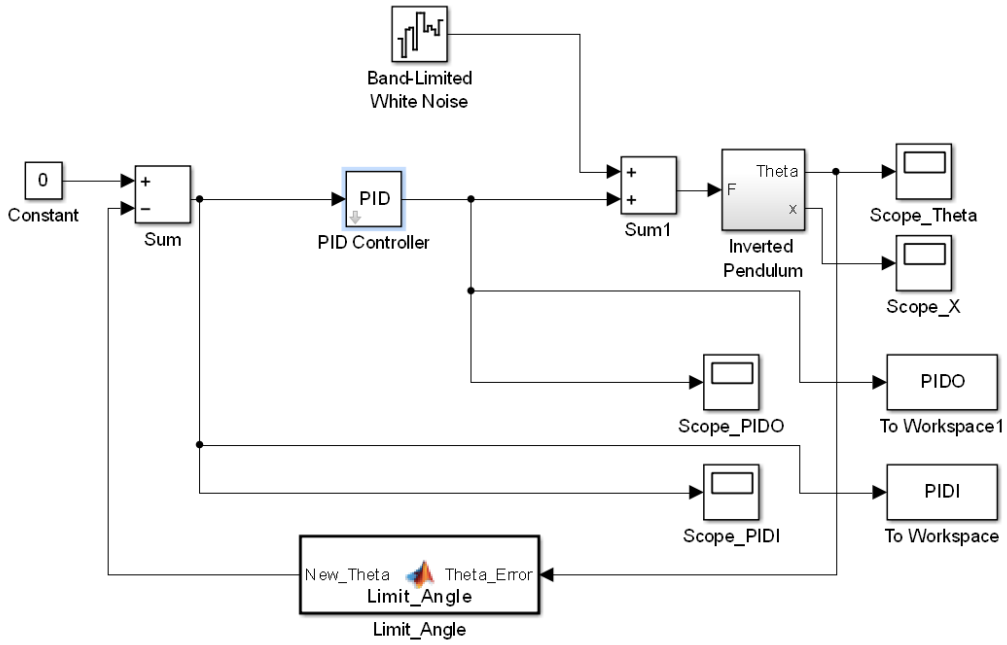

Figura 19-22 Modelo con Controlador PID

Los parámetros proporcional, integral y derivativo del controlador están dados por la Figura 19-23:

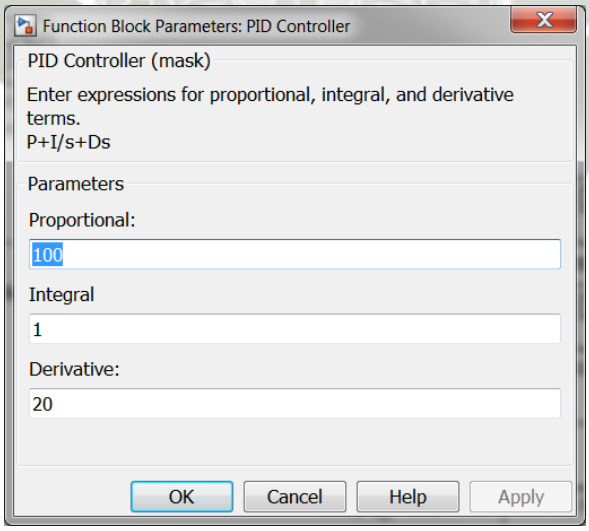

Figura 19-23 Parámetros PID

Luego de ejecutar la simulación tenemos los siguientes resultados:

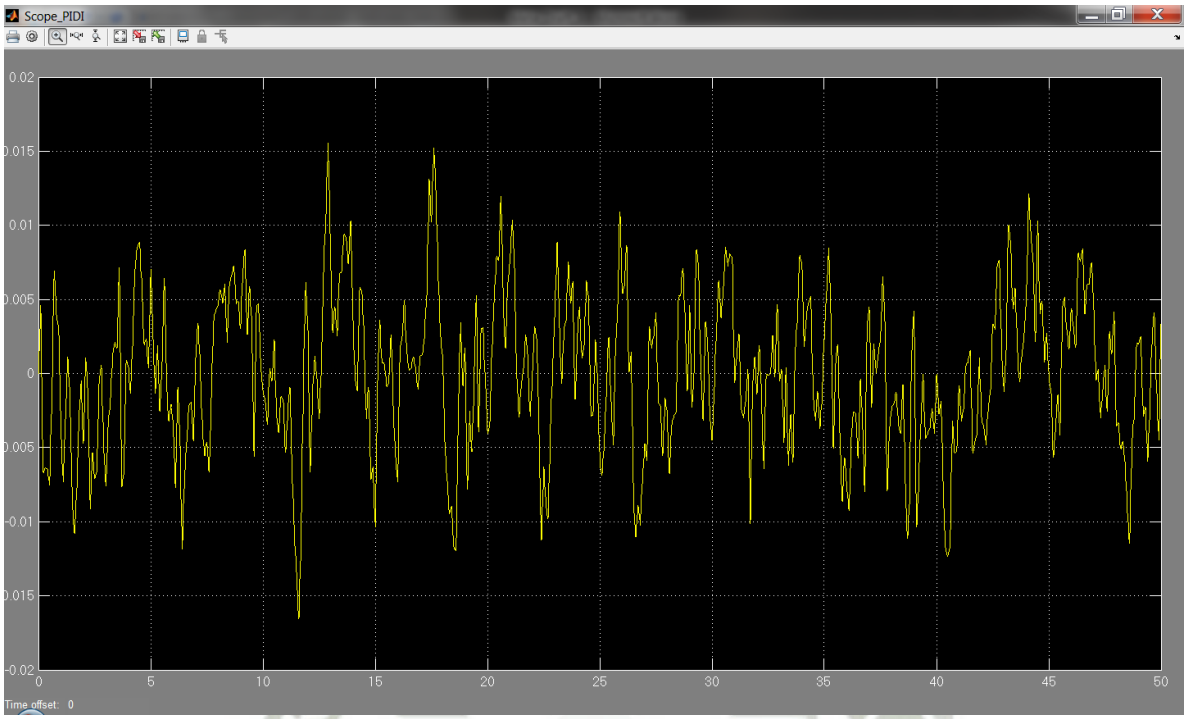

Figura 19-24 Entrada al Controlador PID

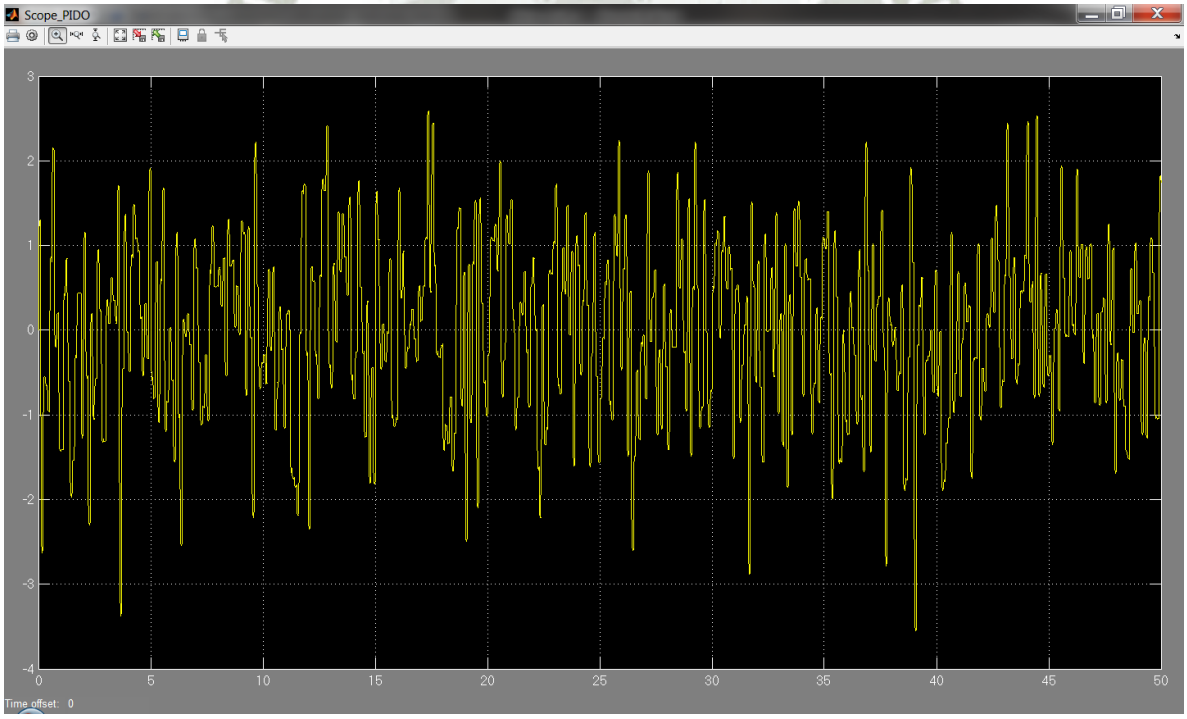

Figura 19-25 Salida del Controlador PID

Ambas señales, la entrada (Figura 19-24) y la salida (Figura 19-25) del Controlador, serán las que alimentaremos posteriormente al Algoritmo Genético para evolucionar y entrenar la red neuronal.

El siguiente, es el ángulo del péndulo controlado por PID:

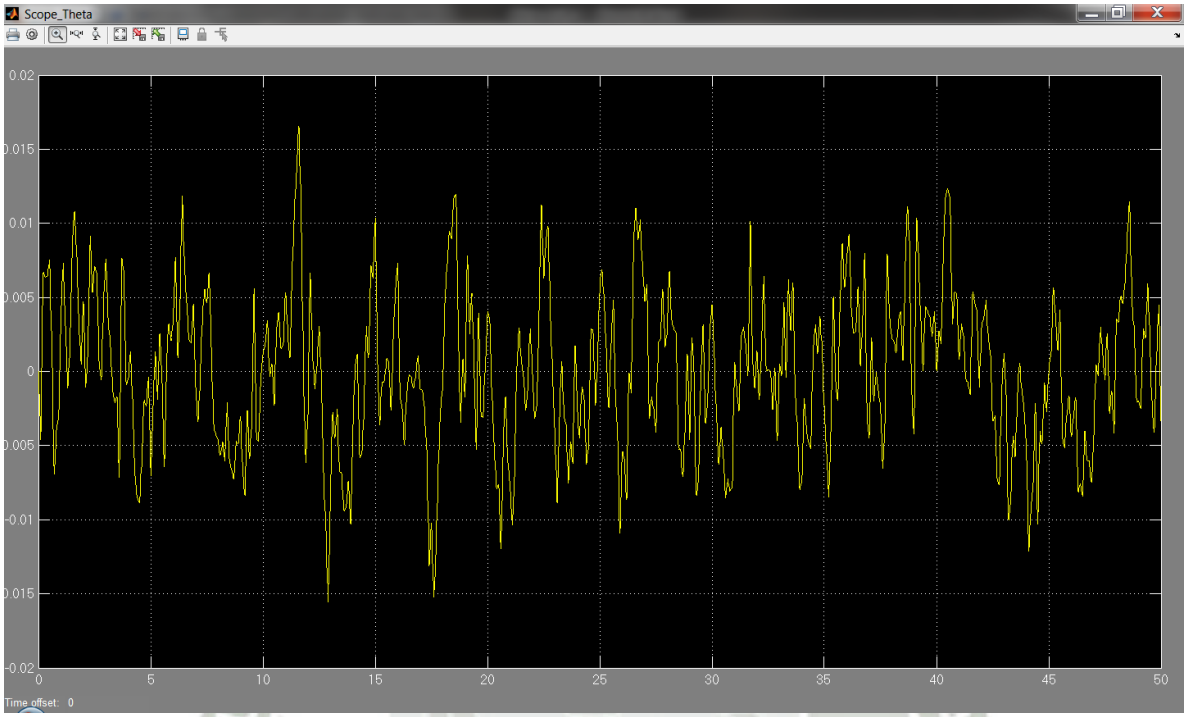

Figura 19-26 Theta controlado por PID

Como podemos ver en la Figura 19-26, el ángulo nunca sobrepasa los 0.02 radianes, lo cual satisface los requerimientos de diseño.

Una vez que tenemos los datos con qué entrenar nuestra población de redes neuronales (Tabla 23-4) procedemos a importar los datos al Espacio de Trabajo de Matlab y corremos el programa.

Los resultados de la evaluación son los siguientes:

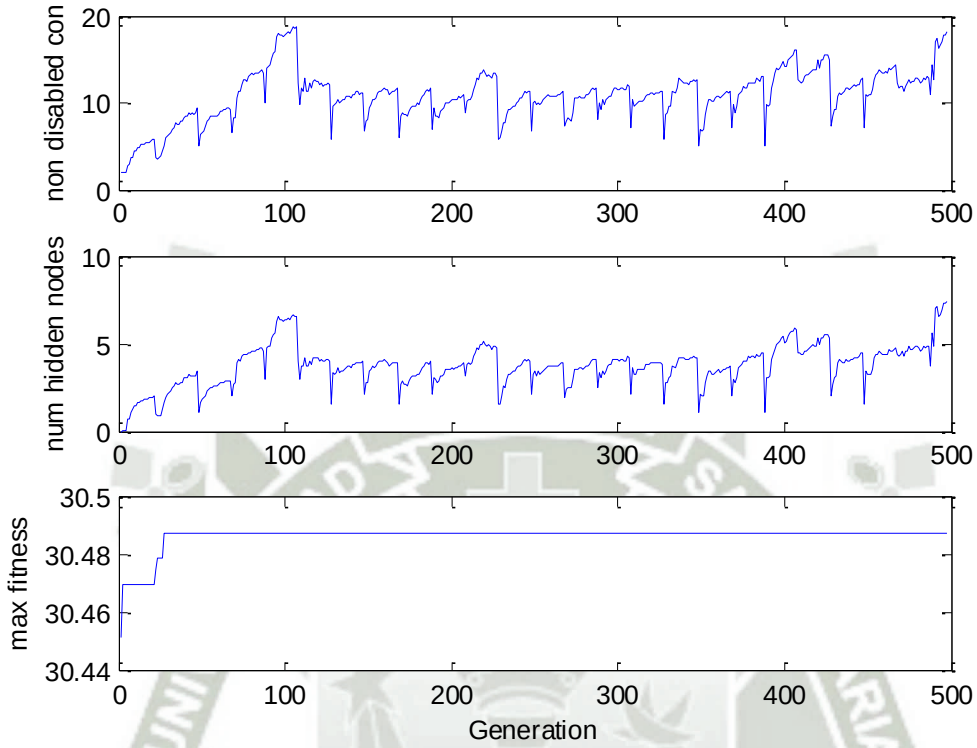

Figura 19-27 Número de conexiones y nodos, grado de Aptitud (Control)

En la Figura 19-27 podemos comparar el número de conexiones promedio, el número de nodos promedio y la máxima aptitud alcanzada en la población, versus las generaciones. Podemos ver claramente cómo la aptitud máxima de la población crece en las primeras 50 generaciones, sin embargo, esta se estanca y le es imposible encontrar una mejor solución, así que el algoritmo alcanza el número máximo de generaciones y deja de buscar.

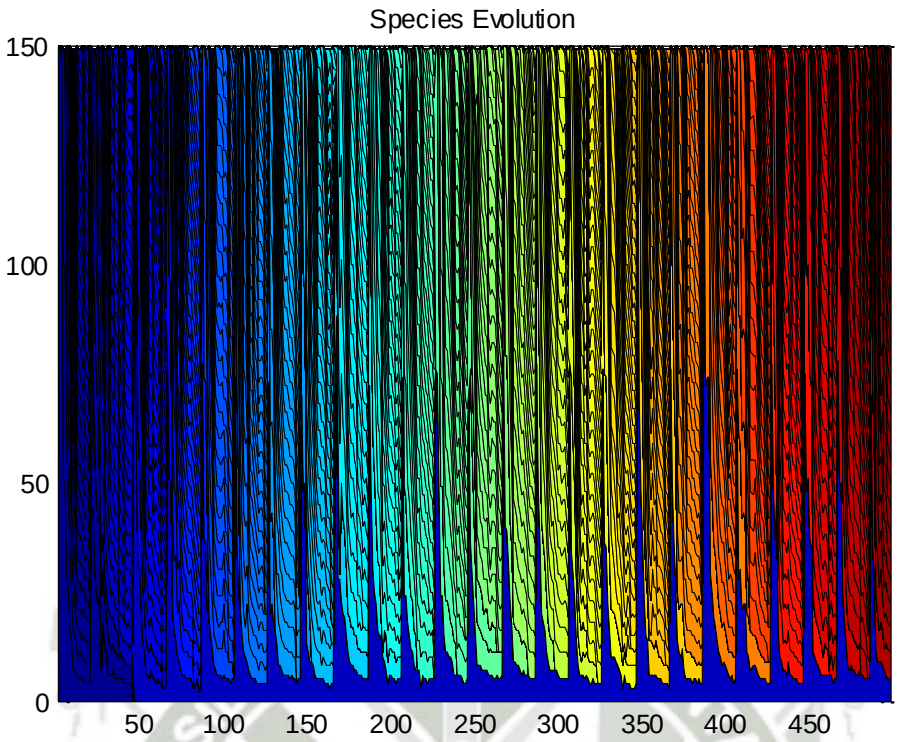

Figura 19-28 Individuos por especies (Control)

En la Figura 19-28 podemos ver la superposición de la cantidad de individuos por especie en un gráfico de áreas, donde el área dedicada a cada tono es directamente proporcional con el número de individuos por especie en dicha generación. Los colores más fríos representan las especies más antiguas, mientras que los colores más cálidos representan especies más jóvenes.

Podemos notar que mientras la aptitud máxima de la población se mantiene estática la cantidad de individuos en ciertas especies cambia drásticamente con cierto periodo generacional. Esto es a causa del reenfoque que el algoritmo fuerza sobre la población, cada 20 generaciones que la aptitud máxima de la población no mejora se eliminan todas menos las dos especies más aptas. Podemos ver este fenómeno repetirse una y otra vez mientras que la aptitud máxima no mejore, y como es de esperarse, el número promedio de conexiones y nodos ocultos también sufren cambios drásticos a cause del reenfoque.

En este caso en particular, podemos ver que es una de las especies más antiguas la que sobrevive, esto se refleja en el tono azul que persiste y coexiste hacia el final de la evolución con colores más cálidos.

Al final de la evolución, la mejor de la redes tiene la siguiente estructura (Figura 19-29):

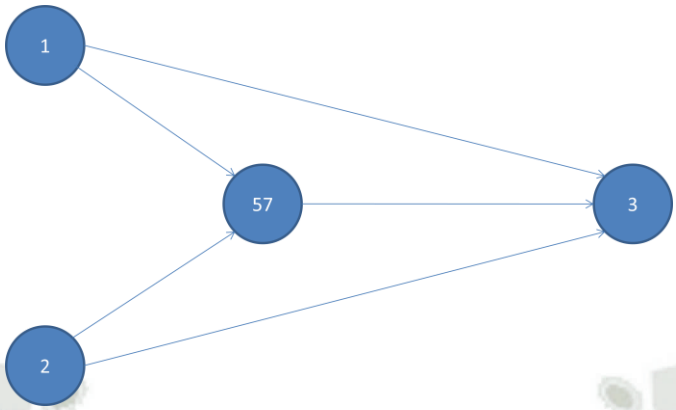

Figura 19-29 Mejor RNA (Control)

Y su genoma de conexiones es:

	Innovaciones				
Innovation Number	1	2	175	176	183
Connection From	1	2	2	57	1
Connection To	3	3	57	3	57
Weight	0.419	1.743	-2.521	1.604	-0.246
Enable Bit	1	1	1	1	1

Tabla 19-5 Genoma de Conexiones (Control)

Como resultado, al evaluar esta red obtenemos el siguiente gráfico (Figura 19-30):

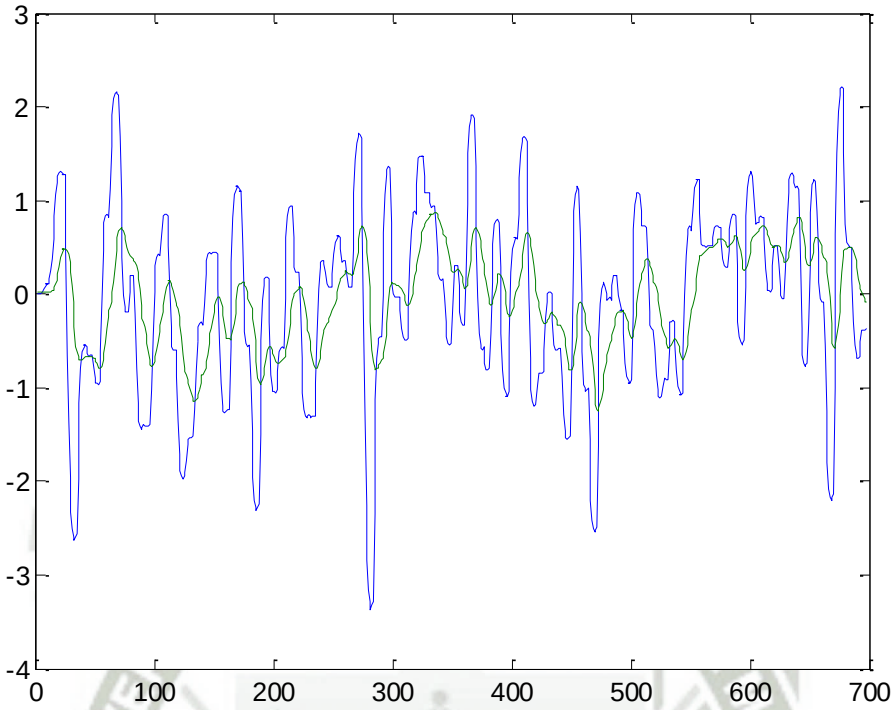

Figura 19-30 Evaluación de la RNA (Control)

Donde la Línea azul representa el set de datos objetivo, es decir, los datos originales alimentados a la red durante su entrenamiento y evolución. Y la línea verde representa la salida de la red luego de iterar sobre el set de datos de entrada.

En este caso en particular, las dos curvas difieren a lo largo de toda la iteración, y la salida de la red no parece tener la misma forma que la curva objetivo, sin embargo, es posible notar que aunque ambas curvas son muy diferentes, la red parece estar siguiendo el sentido de cambio de la curva objetivo, es decir, cuando la curva objetivo sube, la red aumenta su salida y viceversa, además, la intensidad de los cambios también parece estar relacionada con la de la curva objetivo.

Para poder evaluar si es que la red resultante es capaz de controlar el sistema del péndulo invertido, tenemos que diseñar un modelo (Figura 19-31) que reemplace el controlador PID por uno Neuronal, que evalúe el Error del Ángulo y produzca una señal de Fuerza a la salida. Tal modelo se muestra a continuación:

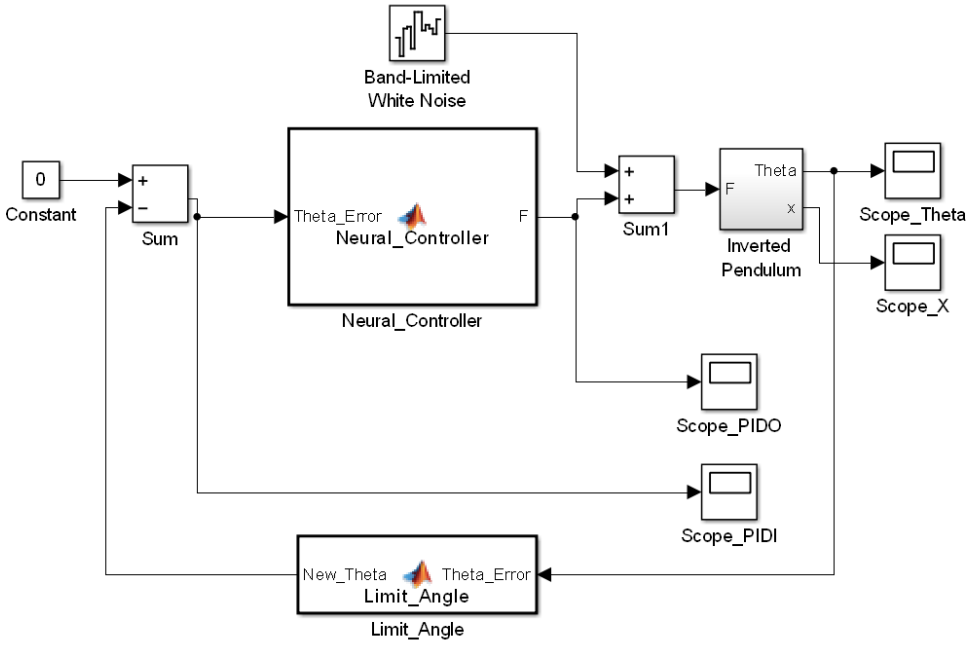

Figura 19-31 Modelo con Controlador Neuronal

En éste último modelo el controlador neuronal es representado por el bloque Neural_Controller, éste es un bloque de código script procesado en tiempo de ejecución por Simulink, para insertar este tipo de bloque debemos agregarlo desde "User Defined Functions" y seleccionar "MATLAB Function" desde la librería de Simulink.

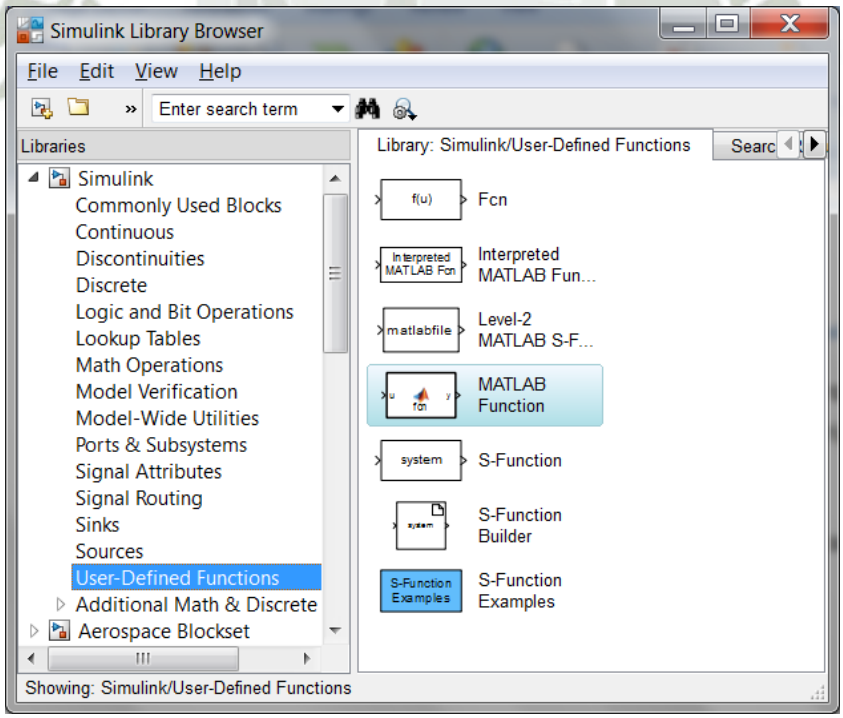

Figura 19-32 Librería de Simulink

Luego podemos cambiar la configuración de entradas y salidas de la función editando el script de la siguiente manera:

```
function F = Neural_Controller( Theta_Error,
Best_Net)
%#codegen

Y = Best_Net_Output(Theta_Error);
F = Y;
```

E insertando el código que computa el proceso por el cual, la red neuronal obtenida mediante el proceso de evolución (Best_Net), evalúa a Theta_Error para obtener una fuerza (representado en el ejemplo como Best_Net_Output); como el cuerpo de la función (*Código Fuente ver Pág. 108*).

En seguida, corremos la simulación, los resultados son los siguientes:

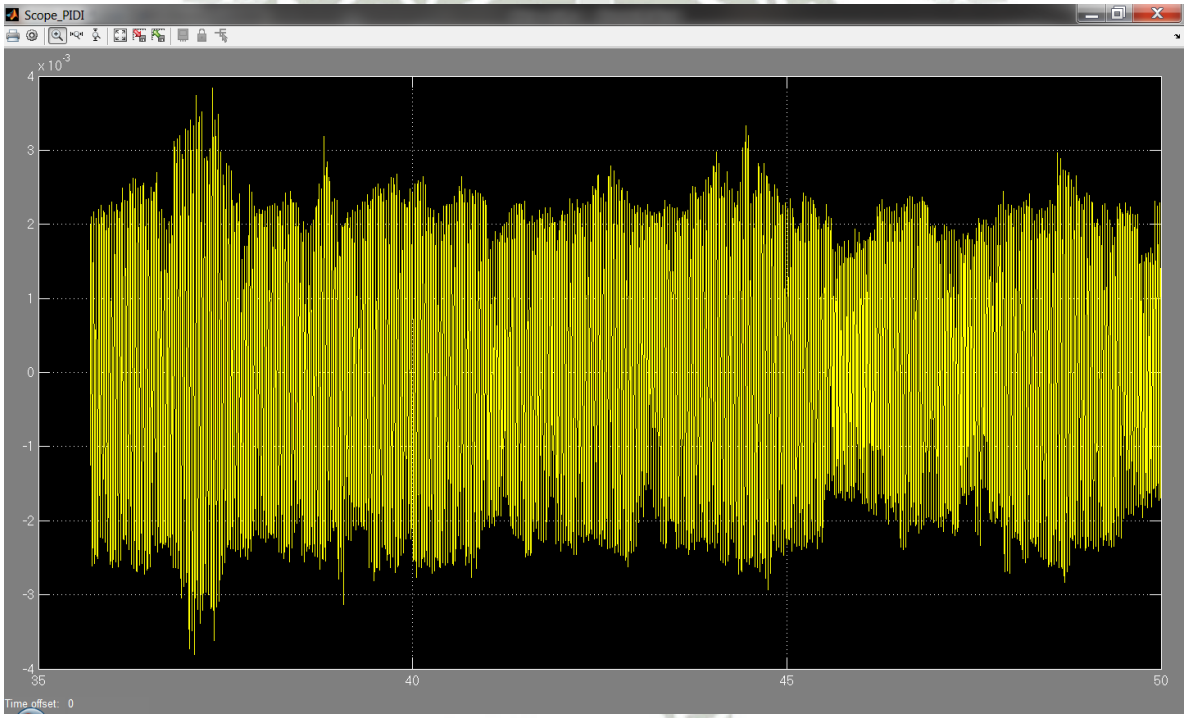

Figura 19-33 Entrada al Controlador Neuronal

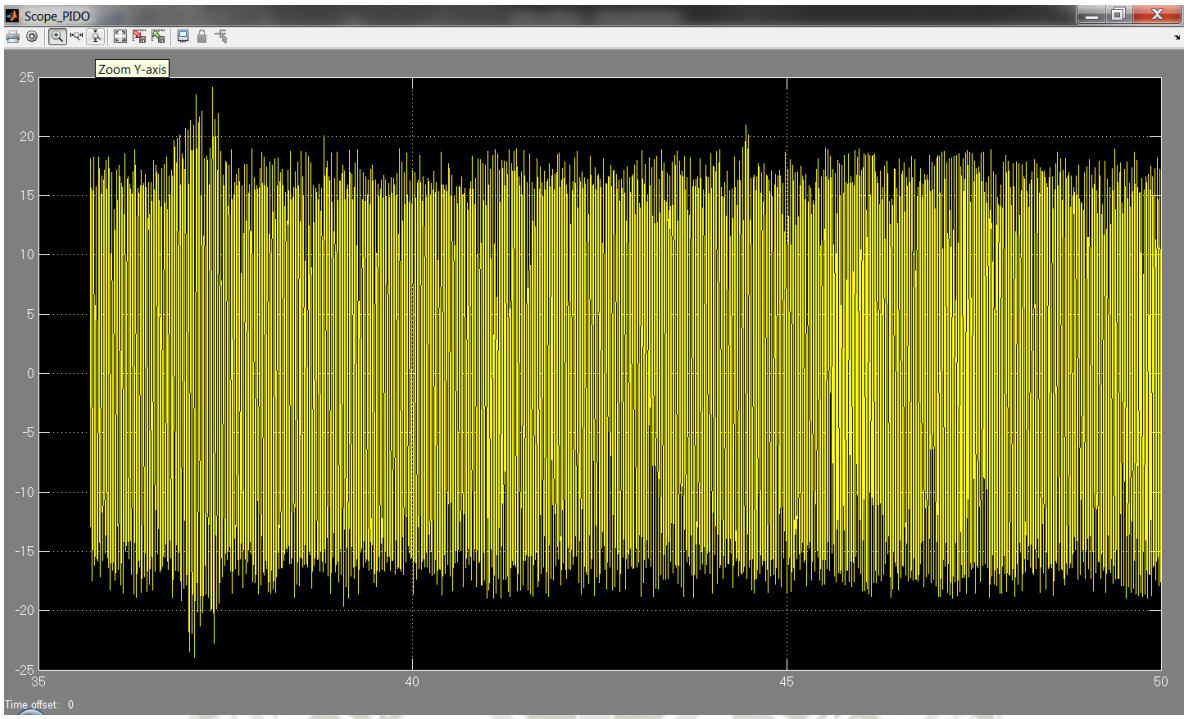

Figura 19-34 Salida del Controlador Neuronal

En esta oportunidad, ambas señales, de entrada (Figura 19-33) y salida (Figura 19-34) al controlador son mucho más ruidosas que en el caso anterior. Además, el orden de magnitud de la entrada al controlador se ha reducido significativamente, lo que quiere decir que el error es de menor magnitud, por otro lado, la salida del controlador ejerce fuerzas mucho mayores que las del controlador PID. Al parecer, la red neuronal ejerce un control mucho más brusco sobre el sistema.

La siguiente es la gráfica del ángulo del péndulo:

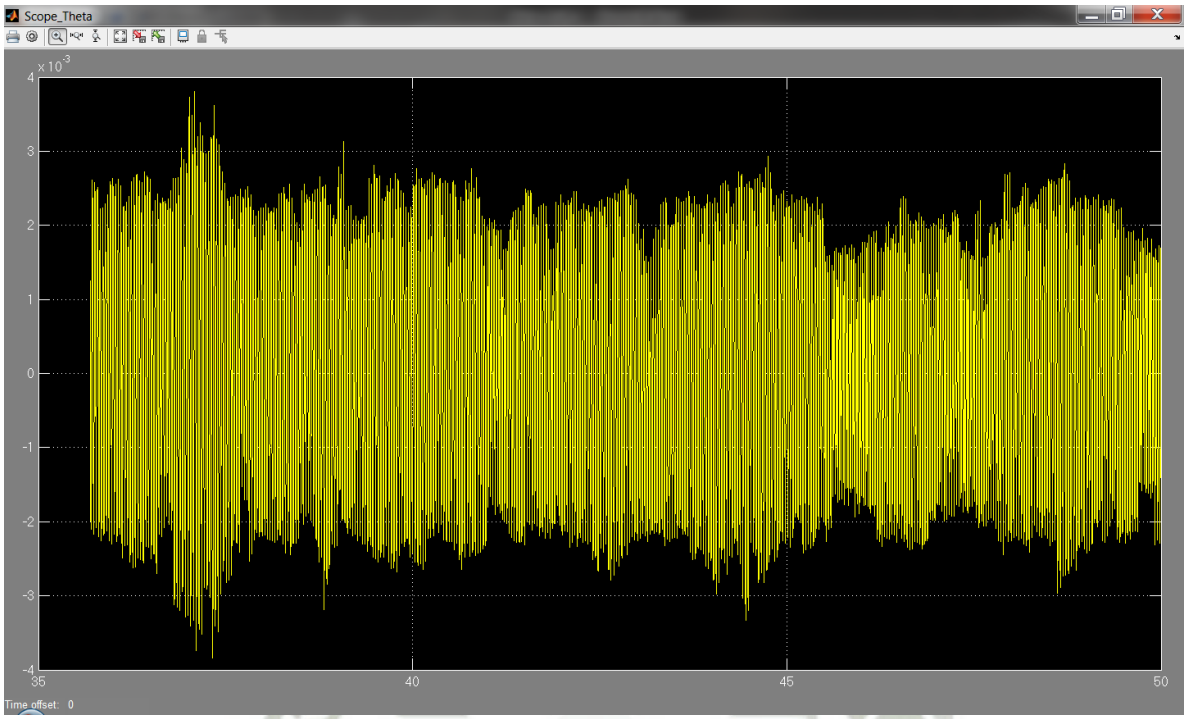

Figura 19-35 Theta controlado por RNA

Como podemos ver en la Figura 19-35, el ángulo nunca supera los 0.004 radianes a pesar del ruido introducido en el sistema, es decir que el controlador cumple con los requisitos de diseño, es capaz de mantener el péndulo en una posición vertical incluso cuando se perturba el carro.

20. Conclusiones

Luego de implementar el algoritmo y de evaluar sus capacidades para resolver problemas tipo del dominio de Redes Neuronales, podemos concluir que este método es capaz de evolucionar Redes Neuronales empezando con topologías mínimas y haciendo más compleja la estructura conforme la circunstancias lo necesitan, como consecuencia, el Espacio de Pesos de las redes se ve reducido y esto resulta en evaluaciones más rápidas, que a su vez se refleja en aprendizaje más rápido. Al mismo tiempo, conforme se va incrementando la estructura de las redes, los pesos de sus conexiones se van cambiando, así que, no sólo optimizamos estructura, sino que también entrenamos a las redes mientras ellas evolucionan.

21. Recomendaciones

Todo esto enmarca un método que dentro del alcance planteado resuelve la problemática satisfactoriamente. Sin embargo, también podemos notar algunas mejoras que se podrían hacer al sistema en futuras revisiones. Por ejemplo, la generalización de la implementación para que instancias posteriores puedan hacer uso de otros tipos de paradigmas de aprendizaje, como No Supervisado, Por Reforzamiento, e inclusive podríamos considerar una implementación que sea capaz de correr en línea, y esté constantemente evolucionando mejores soluciones (sistema adaptativo), en fin, las posibilidades de extender la implementación son amplias.

Otro aspecto que podría mejorarse en una implementación posterior es el grado de sofisticación del sistema, ya que hoy en día existen sistemas para el entrenamiento de redes neuronales capaces de detectar el sobre-entrenamiento de la red, así como ofrecen la posibilidad de escoger sobre diversas funciones de aptitud, etc.

Una de las mejoras que, personalmente creo, sería más prometedora para mejorar el desempeño del sistema, sería el de ampliar el Genotipo de las Redes para incluir también la función de activación, de este modo el algoritmo podría no solamente mutar los pesos de las conexiones, sino que también iterar sobre un número finito de funciones de activación, haciendo que cada nodo implemente su propia función, y ofreciendo un grado más de optimización para el sistema. Dicha mejora implica un cambio drástico en el diseño del programa, pero es uno de los que más directamente afecta los resultados.

22. Bibliografía

- Bhadeshia H. K. D. H. (1999). "Neural Networks in Materials Science". ISIJ International 39 (10): 966–979. doi:10.2355/isijinternational.39.966.
- Bishop, C.M. (1995) Neural Networks for Pattern Recognition, Oxford: Oxford University Press. ISBN 0-19-853849-9 (hardback) or ISBN 0-19-853864-2 (paperback)
- Cybenko, G.V. (1989). Approximation by Superpositions of a Sigmoidal function, Mathematics of Control, Signals, and Systems, Vol. 2 pp. 303–314. electronic version
- Duda, R.O., Hart, P.E., Stork, D.G. (2001) Pattern classification (2nd edition), Wiley, ISBN 0-471-05669-3
- Egmont-Petersen, M., de Ridder, D., Handels, H. (2002). "Image processing with neural networks - a review". Pattern Recognition 35 (10): 2279–2301. doi:10.1016/S0031-3203(01)00178-9.
- Gurney, K. (1997) An Introduction to Neural Networks London: Routledge. ISBN 1-85728-673-1 (hardback) or ISBN 1-85728-503-4 (paperback)
- Haykin, S. (1999) Neural Networks: A Comprehensive Foundation, Prentice Hall, ISBN 0-13-273350-1
- Fahlman, S, Lebiere, C (1991). The Cascade-Correlation Learning Architecture, created for National Science Foundation, Contract Number EET-8716324, and Defense Advanced Research Projects Agency (DOD), ARPA Order No. 4976 under Contract F33615-87-C-1499. electronic version
- Hertz, J., Palmer, R.G., Krogh. A.S. (1990) Introduction to the theory of neural computation, Perseus Books. ISBN 0-201-51560-1
- Lawrence, Jeanette (1994) Introduction to Neural Networks, California Scientific Software Press. ISBN 1-883157-00-5
- Masters, Timothy (1994) Signal and Image Processing with Neural Networks, John Wiley & Sons, Inc. ISBN 0-471-04963-8
- Ripley, Brian D. (1996) Pattern Recognition and Neural Networks, Cambridge
- Siegelmann, H.T. and Sontag, E.D. (1994). Analog computation via neural networks, Theoretical Computer Science, v. 131, no. 2, pp. 331–360. electronic version
- Sergios Theodoridis, Konstantinos Koutroumbas (2009) "Pattern Recognition", 4th Edition, Academic Press, ISBN 978-1-59749-272-0.
- Smith, Murray (1993) Neural Networks for Statistical Modeling, Van Nostrand Reinhold, ISBN 0-442-01310-8
- Wasserman, Philip (1993) Advanced Methods in Neural Computing, Van Nostrand Reinhold, ISBN 0-442-00461-3
- Computational Intelligence: A Methodological Introduction by Kruse, Borgelt, Klawonn, Moewes, Steinbrecher, Held, 2013, Springer, ISBN 9781447150121
- Neuro-Fuzzy-Systeme (3rd edition) by Borgelt, Klawonn, Kruse, Nauck, 2003, Vieweg, ISBN 9783528252656

- Banzhaf, Wolfgang; Nordin, Peter; Keller, Robert; Francone, Frank (1998). Genetic Programming – An Introduction. San Francisco, CA: Morgan Kaufmann. ISBN 978-1558605107.
- Bies, Robert R; Muldoon, Matthew F; Pollock, Bruce G; Manuck, Steven; Smith, Gwenn and Sale, Mark E (2006). "A Genetic Algorithm-Based, Hybrid Machine Learning Approach to Model Selection". Journal of Pharmacokinetics and Pharmacodynamics (Netherlands: Springer): 196–221.
- Cha, Sung-Hyuk; Tappert, Charles C (2009). "A Genetic Algorithm for Constructing Compact Binary Decision Trees". Journal of Pattern Recognition Research 4 (1): 1–13.
- Fraser, Alex S. (1957). "Simulation of Genetic Systems by Automatic Digital Computers. I. Introduction". Australian Journal of Biological Sciences 10: 484–491.
- Goldberg, David (1989). Genetic Algorithms in Search, Optimization and Machine Learning. Reading, MA: Addison-Wesley Professional. ISBN 978-0201157673.
- Goldberg, David (2002). The Design of Innovation: Lessons from and for Competent Genetic Algorithms. Norwell, MA: Kluwer Academic Publishers. ISBN 978-1402070983.
- Fogel, David. Evolutionary Computation: Toward a New Philosophy of Machine Intelligence (3rd ed.). Piscataway, NJ: IEEE Press. ISBN 978-0471669517.
- Holland, John (1992). Adaptation in Natural and Artificial Systems. Cambridge, MA: MIT Press. ISBN 978-0262581110.
- Koza, John (1992). Genetic Programming: On the Programming of Computers by Means of Natural Selection. Cambridge, MA: MIT Press. ISBN 978-0262111706.
- Michalewicz, Zbigniew (1996). Genetic Algorithms + Data Structures = Evolution Programs. Springer-Verlag. ISBN 978-3540606765.
- Mitchell, Melanie (1996). An Introduction to Genetic Algorithms. Cambridge, MA: MIT Press. ISBN 9780585030944.
- Poli, R., Langdon, W. B., McPhee, N. F. (2008). A Field Guide to Genetic Programming. Lulu.com, freely available from the internet. ISBN 978-1-4092-0073-4.
- Rechenberg, Ingo (1994): Evolutionsstrategie '94, Stuttgart: Fromman-Holzboog.
- Schmitt, Lothar M; Nehaniv, Chrystopher L; Fujii, Robert H (1998), Linear analysis of genetic algorithms, Theoretical Computer Science 208: 111–148
- Schmitt, Lothar M (2001), Theory of Genetic Algorithms, Theoretical Computer Science 259: 1–61
- Schmitt, Lothar M (2004), Theory of Genetic Algorithms II: models for genetic operators over the string-tensor representation of populations and convergence to global optima for arbitrary fitness function under scaling, Theoretical Computer Science 310: 181–231
- Schwefel, Hans-Paul (1974): Numerische Optimierung von Computer-Modellen (PhD thesis). Reprinted by Birkhäuser (1977).
- Vose, Michael (1999). The Simple Genetic Algorithm: Foundations and Theory. Cambridge, MA: MIT Press. ISBN 978-0262220583.
- Whitley, Darrell (1994). "A genetic algorithm tutorial". Statistics and Computing 4 (2): 65–85. doi:10.1007/BF00175354.

- Hingston, Philip; Barone, Luigi; Michalewicz, Zbigniew (2008). Design by Evolution: Advances in Evolutionary Design. Springer. ISBN 978-3540741091.
- Eiben, Agoston; Smith, James (2003). Introduction to Evolutionary Computing. Springer. ISBN 978-3540401841.
- Stanley, Miikkulainen. A Taxonomy for Artificial Embryogeny. The MIT Press Journals, 2003.
- Gomez, F. y Miikkulainen, R. (1997). Incremental evolution of complex general behavior. Adaptive Behavior, 5:317–342.
- Gruau, F., Whitley, D., y Pyeatt, L. (1996). A comparison between cellular encoding and direct encoding for genetic neural networks. En Koza, J. R. et al., Genetic Programming 1996: Proceedings of the First Annual Conference, páginas 81–89, MIT Press, Cambridge, Massachusetts.
- Moriarty, D. E. (1997). Symbiotic Evolution of Neural Networks in Sequential Decision Tasks. Ph.D. thesis, Department of Computer Sciences, The University of Texas at Austin. Technical Report UT-AI97-257.
- Potter, M. A. y De Jong, K. A. (1995). Evolving neural networks with collaborative species. En Oren, T. I. y Birta, L. G., Proceedings of the 1995 Summer Computer Simulation Conference, páginas 340–345, Society for Computer Simulation, San Diego, California.
- Whitley, D. et al. (1993). Genetic reinforcement learning for neurocontrol problems. Machine Learning, 13:259–284.
- Kaelbling, L. P., Littman, M., y Moore, A.W. (1996). Reinforcement learning: A survey. Journal of Artificial Intelligence, 4:237–285.
- Chen, D. et al. (1993). Constructive learning of recurrent neural networks. En Petsche, T., Judd, S., y Hanson, S., Computational Learning Theory and Natural Learning Systems III. MIT Press, Cambridge, Massachusetts.
- Angeline, P. J., Saunders, G. M., y Pollack, J. B. (1993). An evolutionary algorithm that constructs recurrent neural networks. IEEE Transactions on Neural Networks, 5:54–65.
- Branke, J. (1995). Evolutionary algorithms for neural network design and training. En Alander, J. T., Proceedings First Nordic Workshop on Genetic Algorithms and their Applications, páginas 145–163, University of Vaasa Press, Vaasa, Finland.
- Yao, X. (1999). Evolving artificial neural networks. Proceedings of the IEEE, 87(9):1423–1447.
- Kenneth O. Stanley et al. (2002). Evolving Neural Networks through Augmenting Topologies. Evolutionary Computation 10(2): 99-127, Massachusetts Institute of Technology, Cambridge, Massachusetts.

23. Anexos

23.1. Tablas de Datos por Experimento

23.1.1. Regresión Lineal

X-2	X-1	X
8	11	14
11	14	17
14	17	20
17	20	23
20	23	26
23	26	29
26	29	32
29	32	35
32	35	38
35	38	41
38	41	44
41	44	47
44	47	50
47	50	53
50	53	56
53	56	59
56	59	62
59	62	65
62	65	68
65	68	71
68	71	74
71	74	77
74	77	80
77	80	83
80	83	86
83	86	89
86	89	92
89	92	95
92	95	98
95	98	101
98	101	104
101	104	107
104	107	110
107	110	113
110	113	116
113	116	119
116	119	122
119	122	125
122	125	128
125	128	131
128	131	134
131	134	137
134	137	140
137	140	143
140	143	146
143	146	149
146	149	152
149	152	155
152	155	158
155	158	161
158	161	164
161	164	167
164	167	170
167	170	173
170	173	176
173	176	179
176	179	182
179	182	185
182	185	188
185	188	191
188	191	194
191	194	197
194	197	200
197	200	203
200	203	206
203	206	209
206	209	212
209	212	215
212	215	218
215	218	221
218	221	224
221	224	227
224	227	230
227	230	233
230	233	236
233	236	239
236	239	242
239	242	245
242	245	248
245	248	251
248	251	254
251	254	257
254	257	260
257	260	263
260	263	266
263	266	269
266	269	272
269	272	275
272	275	278
275	278	281
278	281	284
281	284	287
284	287	290
287	290	293
290	293	296
293	296	299
296	299	302
299	302	305
302	305	308
305	308	311
308	311	314

Tabla 23-1 Datos de Regresión Lineal

23.1.2. Regresión No Lineal

X-2	X-1	X
5	8	13
8	13	20
13	20	29
20	29	40
29	40	53
40	53	68
53	68	85
68	85	104
85	104	125
104	125	148
125	148	173
148	173	200
173	200	229
200	229	260
229	260	293
260	293	328
293	328	365
328	365	404
365	404	445
404	445	488
445	488	533
488	533	580
533	580	629
580	629	680
629	680	733
680	733	788
733	788	845
788	845	904
845	904	965
904	965	1028
965	1028	1093
1028	1093	1160
1093	1160	1229
1160	1229	1300
1229	1300	1373
1300	1373	1448
1373	1448	1525
1448	1525	1604
1525	1604	1685
1604	1685	1768
1685	1768	1853
1768	1853	1940
1853	1940	2029
1940	2029	2120
2029	2120	2213
2120	2213	2308
2213	2308	2405
2308	2405	2504
2405	2504	2605
2504	2605	2708
2605	2708	2813
2708	2813	2920
2813	2920	3029
2920	3029	3140
3029	3140	3253
3140	3253	3368
3253	3368	3485
3368	3485	3604
3485	3604	3725
3604	3725	3848
3725	3848	3973
3848	3973	4100
3973	4100	4229
4100	4229	4360
4229	4360	4493
4360	4493	4628
4493	4628	4765
4628	4765	4904
4765	4904	5045
4904	5045	5188
5045	5188	5333
5188	5333	5480
5333	5480	5629
5480	5629	5780
5629	5780	5933
5780	5933	6088
5933	6088	6245
6088	6245	6404
6245	6404	6565
6404	6565	6728
6565	6728	6893
6728	6893	7060
6893	7060	7229
7060	7229	7400
7229	7400	7573
7400	7573	7748
7573	7748	7925
7748	7925	8104
7925	8104	8285
8104	8285	8468
8285	8468	8653
8468	8653	8840
8653	8840	9029
8840	9029	9220
9029	9220	9413
9220	9413	9608
9413	9608	9805
9608	9805	10004
9805	10004	10205
10004	10205	10408

Tabla 23-2 Datos de Regresión No Lineal

23.1.3. Clasificación

X	Y	Z
0	0	0
0	5	5
5	0	5
5	5	5

Tabla 23-3 Tabla de verdad XOR

23.1.4. Control

PIDI	PIDO
2.74E-59	1.74E-28
6.66E-25	2.71E-11
6.66E-25	5.27E-11
3.67E-09	2.01E-03
1.32E-07	1.40E-02
1.65E-06	5.42E-02
2.69E-06	9.54E-02
2.69E-06	1.07E-01
2.69E-06	1.07E-01
2.69E-06	1.06E-01
1.05E-05	1.57E-01
2.30E-05	2.52E-01
5.70E-05	3.75E-01
1.66E-04	5.85E-01
3.82E-04	8.41E-01
7.38E-04	1.06E+00
1.21E-03	1.20E+00
1.80E-03	1.27E+00
2.57E-03	1.30E+00
3.50E-03	1.30E+00
4.26E-03	1.30E+00
4.56E-03	1.28E+00
4.56E-03	1.27E+00
4.56E-03	1.28E+00
4.57E-03	4.94E-01
4.34E-03	-6.52E-01
3.88E-03	-1.37E+00
3.05E-03	-1.95E+00
1.99E-03	-2.33E+00
6.21E-04	-2.51E+00
-1.10E-03	-2.59E+00
-3.51E-03	-2.63E+00
-5.38E-03	-2.60E+00
-5.96E-03	-2.55E+00
-6.46E-03	-1.97E+00
-6.64E-03	-1.16E+00
-6.70E-03	-7.87E-01
-6.67E-03	-6.21E-01
-6.61E-03	-5.66E-01
-6.52E-03	-5.52E-01
-6.40E-03	-5.50E-01
-6.37E-03	-5.56E-01
-6.36E-03	-6.25E-01
-6.38E-03	-6.66E-01
-6.40E-03	-6.54E-01
-6.41E-03	-6.57E-01
-6.42E-03	-6.55E-01
-6.48E-03	-7.67E-01
-6.60E-03	-9.00E-01
-6.78E-03	-9.44E-01
-7.02E-03	-9.56E-01
-7.43E-03	-9.60E-01
-7.52E-03	-9.40E-01
-7.48E-03	-6.01E-01
-7.32E-03	-5.85E-02
-6.65E-03	4.68E-01
-5.99E-03	7.61E-01
-5.07E-03	8.27E-01
-3.68E-03	8.56E-01
-2.49E-03	8.43E-01
-1.94E-03	8.23E-01
-1.58E-03	1.12E+00
-1.10E-03	1.57E+00
-5.53E-05	1.92E+00
1.01E-03	2.09E+00
2.44E-03	2.14E+00
4.26E-03	2.16E+00
5.73E-03	2.15E+00
6.64E-03	2.13E+00
6.87E-03	1.68E+00
6.94E-03	9.95E-01
6.80E-03	3.88E-01
6.52E-03	-7.07E-03
6.08E-03	-1.37E-01
5.55E-03	-1.82E-01
4.77E-03	-1.97E-01
4.08E-03	-1.92E-01
3.82E-03	1.54E-02
3.70E-03	1.94E-01
3.59E-03	2.00E-01
3.41E-03	1.98E-01
3.21E-03	1.98E-01
3.05E-03	-2.00E-01
2.71E-03	-7.97E-01
2.04E-03	-1.16E+00
1.27E-03	-1.35E+00
2.82E-04	-1.41E+00
-1.43E-03	-1.45E+00
-2.72E-03	-1.42E+00
-2.97E-03	-1.39E+00
-3.45E-03	-1.41E+00
-4.27E-03	-1.42E+00
-5.16E-03	-1.41E+00
-5.97E-03	-1.41E+00
-6.73E-03	-1.41E+00
-7.20E-03	-1.40E+00
-7.28E-03	-1.02E+00
-7.21E-03	-4.42E-01
-6.85E-03	2.62E-02
-6.38E-03	3.12E-01
-5.78E-03	3.97E-01
-4.98E-03	4.25E-01
-3.98E-03	4.33E-01
-3.36E-03	4.14E-01
-2.75E-03	6.26E-01
-2.05E-03	8.24E-01
-1.32E-03	8.38E-01
-3.94E-04	8.48E-01
4.60E-04	8.44E-01
1.04E-03	8.36E-01
1.14E-03	5.18E-01
1.13E-03	5.32E-02
8.63E-04	-3.44E-01
5.43E-04	-5.55E-01
1.14E-04	-5.91E-01
-5.50E-04	-6.05E-01
-1.22E-03	-6.03E-01
-1.43E-03	-8.87E-01
-1.77E-03	-1.36E+00
-2.55E-03	-1.71E+00
-3.37E-03	-1.89E+00
-4.45E-03	-1.94E+00
-6.23E-03	-1.97E+00
-7.51E-03	-1.95E+00
-7.78E-03	-1.91E+00
-8.21E-03	-1.79E+00

-8.58E-03	-1.61E+00
-8.98E-03	-1.55E+00
-9.43E-03	-1.53E+00
-9.94E-03	-1.53E+00
-1.05E-02	-1.53E+00
-1.07E-02	-1.52E+00
-1.08E-02	-1.23E+00
-1.07E-02	-8.10E-01
-1.04E-02	-5.11E-01
-1.01E-02	-3.53E-01
-9.60E-03	-3.14E-01
-8.90E-03	-2.99E-01
-8.24E-03	-3.03E-01
-8.15E-03	-3.30E-01
-7.89E-03	-1.01E-01
-7.50E-03	2.10E-01
-6.91E-03	3.61E-01
-6.20E-03	4.21E-01
-5.16E-03	4.44E-01
-4.19E-03	4.38E-01
-3.57E-03	4.27E-01
-2.84E-03	4.41E-01
-2.08E-03	4.43E-01
-1.42E-03	4.40E-01
-8.36E-04	4.41E-01
-5.38E-04	4.29E-01
-5.08E-04	5.87E-02
-6.24E-04	-5.02E-01
-1.10E-03	-9.53E-01
-1.61E-03	-1.20E+00
-2.31E-03	-1.25E+00
-3.31E-03	-1.28E+00
-4.26E-03	-1.27E+00
-4.66E-03	-1.25E+00
-4.66E-03	-1.24E+00
-4.66E-03	-1.24E+00
-4.70E-03	-5.68E-01
-4.45E-03	3.40E-01
-3.81E-03	8.22E-01
-3.09E-03	1.06E+00
-2.20E-03	1.13E+00
-8.09E-04	1.16E+00
3.98E-04	1.15E+00

8.23E-04	1.12E+00
8.23E-04	1.10E+00
8.23E-04	1.10E+00
1.02E-03	5.87E-01
9.59E-04	-9.31E-02
6.93E-04	-3.96E-01
3.79E-04	-5.26E-01
-3.46E-05	-5.55E-01
-7.37E-04	-5.67E-01
-1.12E-03	-5.50E-01
-1.33E-03	-9.10E-01
-1.67E-03	-1.47E+00
-2.66E-03	-1.96E+00
-3.58E-03	-2.21E+00
-4.84E-03	-2.27E+00
-6.92E-03	-2.31E+00
-8.40E-03	-2.28E+00
-8.88E-03	-2.25E+00
-9.11E-03	-1.50E+00
-8.95E-03	-4.90E-01
-8.59E-03	-7.42E-02
-8.11E-03	1.01E-01
-7.50E-03	1.65E-01
-6.66E-03	1.87E-01
-5.77E-03	1.87E-01
-5.46E-03	1.60E-01
-5.34E-03	-3.03E-01
-5.49E-03	-8.56E-01
-5.71E-03	-9.95E-01
-6.01E-03	-1.04E+00
-6.43E-03	-1.05E+00
-6.92E-03	-1.05E+00
-7.01E-03	-1.03E+00
-7.08E-03	-8.28E-01
-7.03E-03	-6.12E-01
-6.94E-03	-5.78E-01
-6.82E-03	-5.70E-01
-6.63E-03	-5.69E-01
-6.60E-03	-5.80E-01
-6.34E-03	-5.20E-02
-5.80E-03	6.08E-01
-5.12E-03	8.34E-01
-4.27E-03	9.17E-01

-3.09E-03	9.49E-01
-1.87E-03	9.49E-01
-8.59E-04	9.43E-01
-4.65E-04	6.13E-01
-2.76E-04	2.83E-01
-1.22E-04	2.32E-01
7.27E-05	2.26E-01
3.11E-04	2.26E-01
5.25E-04	2.25E-01
4.70E-04	-1.46E-01
2.61E-04	-6.84E-01
-2.61E-04	-1.05E+00
-8.51E-04	-1.24E+00
-1.62E-03	-1.30E+00
-2.80E-03	-1.32E+00
-3.84E-03	-1.31E+00
-4.19E-03	-1.29E+00
-4.86E-03	-1.31E+00
-5.74E-03	-1.32E+00
-6.48E-03	-1.31E+00
-7.11E-03	-1.31E+00
-7.54E-03	-1.31E+00
-7.58E-03	-9.46E-01
-7.49E-03	-4.10E-01
-7.03E-03	5.00E-02
-6.54E-03	2.92E-01
-5.88E-03	3.40E-01
-4.84E-03	3.62E-01
-3.95E-03	3.52E-01
-3.76E-03	3.23E-01
-3.47E-03	2.36E-01
-3.22E-03	1.14E-01
-2.95E-03	8.15E-02
-2.64E-03	7.66E-02
-2.28E-03	7.76E-02
-1.94E-03	7.69E-02
-1.79E-03	2.19E-01
-1.57E-03	4.37E-01
-1.16E-03	5.61E-01
-6.71E-04	6.12E-01
4.04E-06	6.26E-01
6.89E-04	6.25E-01
1.05E-03	6.12E-01

1.32E-03	4.74E-01
1.47E-03	3.48E-01
1.63E-03	3.51E-01
1.88E-03	3.54E-01
2.08E-03	3.52E-01
2.07E-03	1.90E-01
1.98E-03	6.95E-02
1.89E-03	8.27E-02
1.79E-03	7.98E-02
1.69E-03	8.06E-02
1.79E-03	4.88E-01
2.07E-03	1.08E+00
2.65E-03	1.44E+00
3.33E-03	1.62E+00
4.20E-03	1.68E+00
5.60E-03	1.71E+00
6.75E-03	1.69E+00
7.08E-03	1.67E+00
7.12E-03	9.42E-01
6.95E-03	-2.75E-01
6.40E-03	-1.35E+00
5.50E-03	-2.26E+00
4.15E-03	-2.84E+00
2.50E-03	-3.15E+00
4.13E-04	-3.29E+00
-3.07E-03	-3.37E+00
-5.73E-03	-3.32E+00
-6.78E-03	-3.28E+00
-7.46E-03	-2.35E+00
-7.62E-03	-1.13E+00
-7.58E-03	-6.90E-01
-7.45E-03	-5.22E-01
-7.26E-03	-4.66E-01
-7.02E-03	-4.54E-01
-6.68E-03	-4.52E-01
-6.48E-03	-1.18E-02
-6.07E-03	6.54E-01
-5.25E-03	1.07E+00
-4.33E-03	1.27E+00
-3.12E-03	1.33E+00
-1.03E-03	1.37E+00
4.95E-04	1.34E+00
8.69E-04	7.59E-01
8.98E-04	1.33E-01
8.58E-04	3.33E-03
7.86E-04	-2.67E-02
6.86E-04	-3.25E-02
5.69E-04	-3.28E-02
5.69E-04	-2.60E-02
5.69E-04	-2.59E-02
4.82E-04	-1.73E-01
3.24E-04	-3.63E-01
2.80E-05	-4.52E-01
-3.38E-04	-4.83E-01
-8.77E-04	-4.90E-01
-1.32E-03	-4.85E-01
-1.32E-03	-4.66E-01
-1.32E-03	-1.36E-01
-1.17E-03	3.54E-01
-7.55E-04	6.71E-01
-2.60E-04	8.30E-01
3.83E-04	8.71E-01
1.56E-03	8.91E-01
2.16E-03	8.60E-01
2.77E-03	1.17E+00
3.52E-03	1.45E+00
4.28E-03	1.47E+00
5.35E-03	1.48E+00
6.33E-03	1.48E+00
6.85E-03	1.46E+00
7.26E-03	1.27E+00
7.50E-03	1.08E+00
7.74E-03	1.08E+00
8.11E-03	1.09E+00
8.48E-03	1.08E+00
8.61E-03	9.81E-01
8.67E-03	9.24E-01
8.75E-03	9.39E-01
8.84E-03	9.37E-01
8.87E-03	9.34E-01
8.80E-03	7.02E-01
8.62E-03	3.85E-01
8.27E-03	2.24E-01
7.84E-03	1.58E-01
7.27E-03	1.44E-01
6.59E-03	1.41E-01
6.30E-03	1.60E-01
6.08E-03	-3.04E-02
5.77E-03	-2.96E-01
5.18E-03	-4.53E-01
4.49E-03	-5.18E-01
3.51E-03	-5.36E-01
2.57E-03	-5.32E-01
2.11E-03	-5.15E-01
1.91E-03	-1.28E-01
1.96E-03	2.66E-01
2.03E-03	3.05E-01
2.14E-03	3.11E-01
2.32E-03	3.13E-01
2.26E-03	8.42E-02
2.08E-03	-1.89E-01
1.79E-03	-2.95E-01
1.42E-03	-3.30E-01
8.90E-04	-3.39E-01
3.66E-04	-3.37E-01
4.10E-04	1.90E-01
6.64E-04	9.50E-01
1.38E-03	1.50E+00
2.17E-03	1.79E+00
3.21E-03	1.88E+00
4.72E-03	1.91E+00
6.15E-03	1.90E+00
6.86E-03	1.88E+00
7.03E-03	1.37E+00
7.01E-03	5.93E-01
6.66E-03	-3.57E-02
6.15E-03	-4.24E-01
5.50E-03	-5.39E-01
4.64E-03	-5.77E-01
3.38E-03	-5.92E-01
2.79E-03	-5.56E-01
1.85E-03	-7.35E-01
1.09E-03	-8.11E-01
1.13E-04	-8.03E-01
-6.73E-04	-7.96E-01
-1.23E-03	-7.92E-01
-1.31E-03	-4.17E-01
-1.24E-03	1.22E-01
-8.88E-04	5.25E-01

-4.60E-04	7.40E-01
1.04E-04	7.89E-01
9.48E-04	8.07E-01
1.76E-03	8.04E-01
1.85E-03	7.69E-01
1.88E-03	2.88E-01
1.69E-03	-4.19E-01
1.23E-03	-8.15E-01
6.53E-04	-1.01E+00
-9.39E-05	-1.07E+00
-1.21E-03	-1.09E+00
-2.23E-03	-1.09E+00
-2.42E-03	-1.05E+00
-2.52E-03	-4.69E-01
-2.33E-03	2.37E-01
-1.93E-03	4.86E-01
-1.46E-03	5.81E-01
-8.43E-04	6.03E-01
-1.03E-05	6.10E-01
3.65E-04	5.86E-01
6.13E-04	8.51E-01
9.72E-04	1.24E+00
1.74E-03	1.51E+00
2.58E-03	1.63E+00
3.71E-03	1.67E+00
5.04E-03	1.68E+00
6.17E-03	1.67E+00
6.30E-03	1.62E+00
6.40E-03	9.68E-01
6.21E-03	-3.05E-02
5.62E-03	-6.80E-01
4.90E-03	-1.03E+00
3.95E-03	-1.15E+00
2.71E-03	-1.19E+00
1.21E-03	-1.20E+00
1.97E-04	-1.18E+00
-4.76E-04	-1.00E+00
-9.97E-04	-8.42E-01
-1.51E-03	-8.39E-01
-2.18E-03	-8.46E-01
-2.86E-03	-8.45E-01
-3.13E-03	-8.28E-01
-3.22E-03	-5.96E-01

-3.20E-03	-2.67E-01
-3.07E-03	-8.02E-02
-2.87E-03	6.21E-03
-2.62E-03	2.23E-02
-2.15E-03	2.82E-02
-2.08E-03	6.12E-04
-2.11E-03	-2.71E-01
-2.29E-03	-5.51E-01
-2.52E-03	-5.90E-01
-2.85E-03	-6.01E-01
-3.29E-03	-6.03E-01
-3.39E-03	-5.84E-01
-3.39E-03	-5.84E-01
-3.39E-03	-5.83E-01
-3.60E-03	-8.88E-01
-3.96E-03	-1.28E+00
-4.53E-03	-1.46E+00
-5.23E-03	-1.53E+00
-6.22E-03	-1.55E+00
-7.20E-03	-1.55E+00
-7.70E-03	-1.53E+00
-7.70E-03	-1.51E+00
-7.66E-03	-6.72E-01
-7.17E-03	4.49E-01
-6.45E-03	8.83E-01
-5.55E-03	1.06E+00
-4.39E-03	1.12E+00
-2.39E-03	1.15E+00
-1.19E-03	1.11E+00
-9.94E-04	6.65E-01
-9.68E-04	-9.45E-04
-1.28E-03	-6.21E-01
-1.68E-03	-9.61E-01
-2.23E-03	-1.01E+00
-3.04E-03	-1.03E+00
-3.86E-03	-1.03E+00
-4.00E-03	-9.96E-01
-4.52E-03	-1.56E+00
-5.21E-03	-2.20E+00
-6.27E-03	-2.42E+00
-7.47E-03	-2.50E+00
-9.55E-03	-2.54E+00
-1.11E-02	-2.52E+00

-1.16E-02	-2.49E+00
-1.18E-02	-1.61E+00
-1.15E-02	-4.88E-01
-1.10E-02	-9.37E-02
-1.04E-02	5.39E-02
-9.60E-03	1.04E-01
-8.14E-03	1.29E-01
-7.48E-03	8.33E-02
-6.78E-03	8.22E-03
-6.29E-03	-6.65E-02
-5.79E-03	-4.55E-02
-5.06E-03	-4.01E-02
-4.77E-03	-6.29E-02
-4.31E-03	9.42E-02
-3.80E-03	1.98E-01
-3.29E-03	1.87E-01
-2.76E-03	1.90E-01
-2.25E-03	1.90E-01
-2.09E-03	-2.06E-02
-2.07E-03	-1.82E-01
-2.03E-03	-1.72E-01
-1.98E-03	-1.72E-01
-1.94E-03	-1.73E-01
-2.03E-03	-4.21E-01
-2.24E-03	-7.44E-01
-2.61E-03	-8.82E-01
-3.06E-03	-9.32E-01
-3.70E-03	-9.45E-01
-4.35E-03	-9.44E-01
-4.53E-03	-9.22E-01
-4.41E-03	-2.01E-01
-3.89E-03	6.87E-01
-3.24E-03	9.52E-01
-2.40E-03	1.04E+00
-1.25E-03	1.07E+00
-9.52E-07	1.08E+00
7.80E-04	1.06E+00
1.33E-03	8.86E-01
1.75E-03	7.29E-01
2.15E-03	7.29E-01
2.70E-03	7.35E-01
3.25E-03	7.34E-01
3.37E-03	7.14E-01

3.36E-03	3.05E-01
3.12E-03	-1.92E-01
2.77E-03	-3.31E-01
2.32E-03	-3.73E-01
1.67E-03	-3.85E-01
9.76E-04	-3.85E-01
7.52E-04	-5.62E-01
4.35E-04	-8.56E-01
-1.65E-04	-1.02E+00
-8.64E-04	-1.10E+00
-1.84E-03	-1.12E+00
-2.82E-03	-1.11E+00
-3.37E-03	-1.10E+00
-3.86E-03	-9.92E-01
-4.24E-03	-9.01E-01
-4.63E-03	-9.10E-01
-5.15E-03	-9.15E-01
-5.54E-03	-9.09E-01
-5.55E-03	-5.86E-01
-5.37E-03	-2.98E-01
-5.20E-03	-2.94E-01
-4.87E-03	-2.88E-01
-4.64E-03	-2.93E-01
-4.65E-03	-5.15E-01
-4.77E-03	-8.18E-01
-5.05E-03	-9.82E-01
-5.39E-03	-1.05E+00
-5.84E-03	-1.07E+00
-6.44E-03	-1.07E+00
-6.66E-03	-1.05E+00
-6.65E-03	-6.32E-01
-6.46E-03	-6.71E-03
-5.92E-03	4.16E-01
-5.29E-03	6.35E-01
-4.48E-03	6.98E-01
-3.22E-03	7.23E-01
-2.10E-03	7.13E-01
-1.88E-03	6.76E-01
-1.36E-03	8.91E-01
-7.26E-04	1.13E+00
9.51E-05	1.20E+00
1.28E-03	1.23E+00
2.43E-03	1.22E+00

3.39E-03	1.22E+00
3.72E-03	8.68E-01
3.83E-03	5.42E-01
3.92E-03	5.13E-01
4.05E-03	5.12E-01
4.21E-03	5.13E-01
4.25E-03	5.06E-01
4.36E-03	5.19E-01
4.50E-03	5.22E-01
4.65E-03	5.22E-01
4.65E-03	5.14E-01
4.65E-03	5.14E-01
4.73E-03	6.06E-01
4.86E-03	7.06E-01
5.05E-03	7.30E-01
5.33E-03	7.35E-01
5.65E-03	7.36E-01
5.65E-03	7.17E-01
5.65E-03	7.17E-01
5.66E-03	5.71E-01
5.58E-03	3.82E-01
5.43E-03	3.09E-01
5.24E-03	2.85E-01
4.96E-03	2.82E-01
4.74E-03	2.85E-01
4.74E-03	4.61E-01
4.83E-03	6.89E-01
5.02E-03	8.00E-01
5.27E-03	8.44E-01
5.60E-03	8.51E-01
6.00E-03	8.52E-01
6.05E-03	8.33E-01
5.90E-03	3.02E-01
5.54E-03	-2.94E-01
4.96E-03	-4.61E-01
4.28E-03	-5.17E-01
3.25E-03	-5.36E-01
2.33E-03	-5.29E-01
2.14E-03	-4.98E-01
2.09E-03	7.33E-02
2.34E-03	8.33E-01
2.78E-03	1.13E+00
3.33E-03	1.25E+00

4.06E-03	1.29E+00
5.31E-03	1.31E+00
5.76E-03	1.26E+00
6.05E-03	1.01E+00
6.16E-03	7.59E-01
6.26E-03	7.54E-01
6.42E-03	7.56E-01
6.59E-03	7.56E-01
6.77E-03	8.10E-01
6.96E-03	8.28E-01
7.14E-03	8.20E-01
7.22E-03	8.22E-01
7.18E-03	5.87E-01
7.02E-03	2.73E-01
6.69E-03	1.13E-01
6.29E-03	4.72E-02
5.76E-03	3.31E-02
5.12E-03	3.02E-02
4.86E-03	4.92E-02
4.73E-03	3.03E-01
4.77E-03	5.24E-01
4.80E-03	5.14E-01
4.85E-03	5.16E-01
4.90E-03	5.15E-01
4.82E-03	3.19E-01
4.65E-03	7.47E-02
4.36E-03	-1.69E-02
4.00E-03	-4.66E-02
3.45E-03	-5.42E-02
3.03E-03	-4.76E-02
3.03E-03	2.88E-01
3.17E-03	7.62E-01
3.56E-03	1.07E+00
4.04E-03	1.23E+00
4.66E-03	1.27E+00
5.81E-03	1.29E+00
6.37E-03	1.26E+00
6.90E-03	1.19E+00
7.27E-03	1.13E+00
7.65E-03	1.15E+00
8.27E-03	1.16E+00
8.34E-03	1.12E+00
8.15E-03	4.27E-01

7.70E-03	-3.74E-01
6.86E-03	-6.46E-01
5.99E-03	-7.40E-01
4.75E-03	-7.67E-01
3.52E-03	-7.63E-01
2.78E-03	-7.45E-01
2.63E-03	-3.21E-01
2.65E-03	3.09E-01
3.02E-03	8.70E-01
3.45E-03	1.16E+00
4.04E-03	1.20E+00
5.06E-03	1.22E+00
5.80E-03	1.21E+00
5.89E-03	8.91E-01
5.84E-03	4.43E-01
5.60E-03	1.22E-01
5.27E-03	-4.50E-02
4.85E-03	-8.09E-02
4.20E-03	-9.33E-02

3.64E-03	-8.72E-02
3.40E-03	-5.39E-01
2.97E-03	-1.25E+00
1.91E-03	-1.79E+00
8.14E-04	-2.08E+00
-6.32E-04	-2.16E+00
-3.19E-03	-2.21E+00
-5.02E-03	-2.17E+00
-5.43E-03	-2.13E+00
-5.58E-03	-1.00E+00
-5.19E-03	6.38E-01
-4.36E-03	1.51E+00
-3.34E-03	1.95E+00
-2.01E-03	2.13E+00
-3.15E-04	2.20E+00
1.91E-03	2.22E+00
3.81E-03	2.20E+00
4.36E-03	1.51E+00
4.52E-03	7.46E-01

4.57E-03	5.58E-01
4.61E-03	5.09E-01
4.64E-03	4.98E-01
4.69E-03	4.98E-01
4.70E-03	4.96E-01
4.59E-03	1.64E-01
4.33E-03	-2.99E-01
3.83E-03	-5.44E-01
3.22E-03	-6.53E-01
2.43E-03	-6.82E-01
1.36E-03	-6.92E-01
6.58E-04	-6.74E-01
2.46E-04	-5.17E-01
-4.05E-05	-3.82E-01
-3.26E-04	-3.87E-01
-7.50E-04	-3.92E-01
-1.11E-03	-3.88E-01
-1.11E-03	-3.72E-01

Tabla 23-4 Datos I/O de Control

23.2. Código Fuente

```
%% Genetic Evolution of Neural Networks

clear;
tic;
%lista de parámetros

    scrsz = get(0,'ScreenSize');
    TT = [];
    I = [];
    O = [];
    TT = xlsread( 'Logic_Truth_Table.xlsx' , 'XOR' );
    I = TT( : , 1:(size( TT , 2 ) - 1) );
    O = TT( : , size( TT , 2 ) );

    gens_since_lastrefocus = 0;


  %parámetros principales
    maxgeneration=500;
    load_flag=0;
    save_flag=1;
    average_number_non_disabled_connections=[];
    average_number_hidden_nodes=[];
    max_overall_fitness=[];

  %parámetros de población inicial
    population_size=150;
    number_input_nodes = size( I , 2 );
    number_output_nodes=1;
    vector_connected_input_nodes=1:size( I , 2 );

  % parámetros de speciación
    species_record(1).ID=0;
    species_record(1).number_individuals=0;
    species_record(1).generation_record=[];

    speciation.c1=1.0;
    speciation.c2=1.0;
    speciation.c3=0.4;
    speciation.threshold=3;

  % parámetros de reproducción
    %estancamiento y reenfoque
    stagnation.threshold=1e-2;
    stagnation.number_generation=15;
    refocus.threshold=1e-2;
    refocus.number_generation=20;

    % setup inicial
```

```
    initial.kill_percentage=0.2;
    initial.number_for_kill=5;
    initial.number_copy=0;

    %selección (ranqueo y muestreo estocástico universal)
    selection.pressure=2;

    %crossover
    crossover.percentage=0.8;
    crossover.probability_interspecies=0.0;
    crossover.probability_multipoint=0.6;

    %mutación
    mutation.probability_add_node=0.03;
    mutation.probability_add_connection=0.05;
    mutation.probability_recurrency=0.1;
    mutation.probability_mutate_weight=0.9;
    mutation.weight_cap=8;
    mutation.weight_range=5;
    mutation.probability_gene_reenabled=0.25;

%%%%%%%%%%%%%%%% algoritmo principal
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

if load_flag==0

[population,innovation_record]=initial_population(population_size,
number_input_nodes, number_output_nodes,
vector_connected_input_nodes);

   % speciación inicial

number_connections=(length(vector_connected_input_nodes)+1)*number
_output_nodes;
population(1).species=1;

matrix_reference_individuals=population(1).connectiongenes(4,:);
   species_record(1).ID=1;
   species_record(1).number_individuals=1;

   Best_Net = population(1);
   Historic_Species = [];
   for index_individual=2:size(population,2);
      assigned_existing_species_flag=0;
      new_species_flag=0;
      index_species=1;
      while assigned_existing_species_flag==0 &
new_species_flag==0

distance=speciation.c3*sum(abs(population(index_individual).connec
tiongenes(4,:)-
```

```
matrix_reference_individuals(index_species,:)))/number_connections
;
         if distance<speciation.threshold
            population(index_individual).species=index_species;
            assigned_existing_species_flag=1;

species_record(index_species).number_individuals=species_record(in
dex_species).number_individuals+1;
         end
         index_species=index_species+1;
         if index_species>size(matrix_reference_individuals,1) &
assigned_existing_species_flag==0
            new_species_flag=1;
         end
      end
      if new_species_flag==1
population(index_individual).species=index_species;

matrix_reference_individuals=[matrix_reference_individuals;
population(index_individual).connectiongenes(4,:)];
species_record(index_species).ID=index_species;
species_record(index_species).number_individuals=1;

      end
   end
   generation=1;
else % Empezar con la versión guardada
   load 'neatsave'
end

%%%
%%%
%%%  Loop Generacional
%%%
%%%

flag_solution=0;
while generation<maxgeneration & flag_solution==0

   [ population , Best_Net ] = My_experiment_vec( population , I ,
O );

   generation

max_fitnesses_current_generation=zeros(1,size(species_record,2));

   for index_species=1:size(species_record,2)
      if species_record(index_species).number_individuals>0

[max_fitness,index_individual_max]=max(([population(:).species]==i
ndex_species).*[population(:).fitness]);
```

```
mean_fitness=sum(([population(:).species]==index_species).*[popula
tion(:).fitness])/species_record(index_species).number_individuals
;

         if
size(species_record(index_species).generation_record,2)>stagnation
.number_generation-2

stagnation_vector=[species_record(index_species).generation_record
(3,size(species_record(index_species).generation_record,2)-
stagnation.number_generation+2:size(species_record(index_species).
generation_record,2)),max_fitness];
            if sum(abs(stagnation_vector-
mean(stagnation_vector))<stagnation.threshold)==stagnation.number_
generation
               mean_fitness=0.01;
            end
         end

species_record(index_species).generation_record=[species_record(in
dex_species).generation_record,[generation;mean_fitness;max_fitnes
s;index_individual_max]];
max_fitnesses_current_generation(1,index_species)=max_fitness;
      end
   end

[top_fitness,index_top_species]=max(max_fitnesses_current_generati
on);

mean_fitness1=sum(([population(:).species]==index_top_species).*[p
opulation(:).fitness])/species_record(index_top_species).number_in
dividuals;
   species_record(index_top_species).generation_record( 2 ,
size(species_record(index_top_species).generation_record,2) ) =
mean_fitness1;

   %verificar reenfoque
   if gens_since_lastrefocus >= refocus.number_generation
       if
size(species_record(index_top_species).generation_record,2)>refocu
s.number_generation

index1=size(species_record(index_top_species).generation_record,2)
-refocus.number_generation;

index2=size(species_record(index_top_species).generation_record,2)
;
          if
sum(abs(species_record(index_top_species).generation_record(3,inde
x1:index2)-
```

```
mean(species_record(index_top_species).generation_record(3,index1:
index2)))<refocus.threshold)>=refocus.number_generation
[discard,vector_cull]=sort(-max_fitnesses_current_generation);

vector_cull=vector_cull(1,3:sum(max_fitnesses_current_generation>0
));
              for index_species=1:size(vector_cull,2)
                 index_cull=vector_cull(1,index_species);

species_record(index_cull).generation_record(2,size(species_record
(index_cull).generation_record,2))=0.01;
                 gens_since_lastrefocus = 0;
              end
           end
        end
    end
    gens_since_lastrefocus = gens_since_lastrefocus + 1;

    %visualización de aptitud y species
    Y = Eval_Net_vec( Best_Net , I , min(I(:,1)) , max(I(:,1)) ,
min(O) , max(O) );

    a=0;
    b=0;
    for index_individual=1:size(population,2)

a=a+sum(population(index_individual).connectiongenes(5,:)==1);
       b=b+sum(population(index_individual).nodegenes(2,:)==3);
    end

average_number_non_disabled_connections=[average_number_non_disabl
ed_connections,[a/population_size;generation]];

average_number_hidden_nodes=[average_number_hidden_nodes,[b/popula
tion_size;generation]];
    c=[];
    for index_species=1:size(species_record,2)

c=[c,species_record(index_species).generation_record(1:3,size(spec
ies_record(index_species).generation_record,2))];
    end

max_overall_fitness=[max_overall_fitness,[max(c(3,:).*(c(1,:)==gen
eration));generation]];
    maximale_fitness=max(c(3,:).*(c(1,:)==generation))
    if maximale_fitness>15.99
       flag_solution=1;
     end

    figure(1);
    subplot(3,1,1);
```

```
plot(average_number_non_disabled_connections(2,:),average_number_n
on_disabled_connections(1,:));
   ylabel('non disabled con');
   subplot(3,1,2);

plot(average_number_hidden_nodes(2,:),average_number_hidden_nodes(
1,:));
   ylabel('num hidden nodes');
   subplot(3,1,3);
   plot(max_overall_fitness(2,:),max_overall_fitness(1,:));
   ylabel('max fitness');
   xlabel('Generation');
   figure(2);
   if size( Historic_Species , 1 ) >= 2
   area( Historic_Species( 2:size(Historic_Species,1) ,:) )
   title 'Species Evolution'
   end

   figure(3);
   plot([1:size(O,1)],O(:,1),[1:size(Y,1)],Y(:,1));

   drawnow;



   if flag_solution==0

   %llmar a función de reproducción

[population,species_record,innovation_record]=reproduce(population
, species_record, innovation_record, initial, selection,
crossover, mutation, speciation, generation, population_size);
   toc;
   end
   generation=generation+1;
   Historic_Species = Expand_Matrix( Historic_Species ,
species_record );

   if save_flag==1 % Grabar copias de la presente generación
      save 'neatsave' population generation innovation_record
species_record Best_Net Historic_Species
average_number_non_disabled_connections
average_number_hidden_nodes max_overall_fitness
   end
end
```

```
%%%
%%%
%%%  initial_population
%%%
%%%

function
[population,innovation_record]=initial_population(number_individua
ls,number_input_nodes,number_output_nodes,vector_connected_input_n
odes);

% nodegenes es un array de 4 filas * ( number_input_nodes +
number_output_nodes + hidden_nodes ( inexistentes en la población
inicial ) + 1 ( bias_node ) ) columnas
% nodegenes contiene consecutivamente, ID de nodo ( primera fila
), tipo de nodo ( segunda fila ) 1 = input, 2 = output, 3 =
hidden, 4 = bias; estado de entrada en nodo, y estado de salida en
nodo ( para la evaluación, todos los estados de entrada son cero,
excepto el nodo bias, que siempre es 1 )
% connectiongenes es un array de 5 filas * number_connections
columnas
% de arriva a abajo, esas filas contienen: número de innovación,
connexión desde, connexión hacia, peso, bit de habilitación
% el resto de los elementos en la estructura de los individuos se
explica por sí misma

% innovation_record rastrea innovaciones en una matriz de 5 filas
por ( número de innovaciones ) columnas, contiene número de
innovación, connect_from_node, así como connect_to_node para ésta
innovación )
% el nuevo nodo ( si es un mutación de nuevo nodo, entonces éste
nodo aparecerá en la 4ta fila cuando se conecte por primera vez.
Siempre habrá dos innovaciones con una mutación de nodo, ya que
existen conexiones hacia y desde el nuevo nodo.
% En la población inicial, esto será abreviado al Nodo con el
número más alto apareciendo en la última columna del registro, ya
que sólo esto se necesita como punto de partida para el resto del
algoritmo ),
% y la 5ta fila es la generación en la que esta innovación ocurrió
( la generación se asume cero para las innovaciones en la
población inicial )

% computar el número y matriz de conexiones iniciales ( todas las
conexiones entre nodos de salida y los nodos listados en
vector_connected_input_nodes )

number_connections=(length(vector_connected_input_nodes)+1)*number
_output_nodes;
vector_connection_from=rep([vector_connected_input_nodes,number_in
put_nodes+1],[1 number_output_nodes]);
vector_connection_to=[];
```

```
for
index_output_node=(number_input_nodes+2):(number_input_nodes+1+num
ber_output_nodes)

vector_connection_to=[vector_connection_to,index_output_node*ones(
1,length(vector_connected_input_nodes)+1)];
end
connection_matrix=[vector_connection_from;
                   vector_connection_to];

for index_individual=1:number_individuals;

population(index_individual).nodegenes=[1:(number_input_nodes+1+nu
mber_output_nodes);

ones(1,number_input_nodes),4,2*ones(1,number_output_nodes);

zeros(1,number_input_nodes),1,zeros(1,number_output_nodes);

zeros(1,number_input_nodes+1+number_output_nodes)];

population(index_individual).connectiongenes=[1:number_connections
;

connection_matrix;

rand(1,number_connections)*2-1;

ones(1,number_connections)]; % todos los pesos uniformemente
distribuidos en [-1 +1], todas las conexiones habilitadas
   population(index_individual).fitness=0;
   population(index_individual).species=0;
end
innovation_record=[population(index_individual).connectiongenes(1:
3,:);zeros(size(population(index_individual).connectiongenes(1:2,:
)))];
innovation_record(4,size(innovation_record,2))=max(population(1).n
odegenes(1,:)); % el node ID más alto para la población inicial
```

```
%%%
%%%
%%%  My_experiment_vec
%%%
%%%

function [ population_plus_fitnesses , Best_Net
]=My_experiment_vec( population , I , O )
population_plus_fitnesses=population;
no_change_threshold=1e-3; % umbral para determinar si el estado de
un nodo ha cambiado significativamente desde la última iteración
number_individuals=size(population,2);

I = ReScale( I , min(I(:,1)) , max(I(:,1)) , -1 , 1 );

Omin = min(O);
Omax = max(O);
O = ReScale( O , min(O) , max(O) , -1 , 1 );

sf = 1.0;

for index_individual=1:number_individuals
   number_nodes=size(population(index_individual).nodegenes,2);

number_connections=size(population(index_individual).connectiongen
es,2);
   y = [];
   SE=[];
   for index_pattern=1:size( I , 1 )
      % setear el valor de entrada de los nodos para la primera
iteracion
      population(index_individual).nodegenes(3, find( (
population(index_individual).nodegenes(2,:) == 2 ) | (
population(index_individual).nodegenes(2,:) == 3 ) ) ) = 0; %
hacer cero las salidas de todos los nodos
      population(index_individual).nodegenes(3, find(
population(index_individual).nodegenes(2,:) == 4 ) ) = 1; % hacer
la entrada de nodo bias 1
      population(index_individual).nodegenes(3, find(
population(index_individual).nodegenes(2,:) == 1 ) ) =
I(index_pattern,:); %% el nodo entrada tiene el valor de
Theta_Error

      % setear los valores de salida dependiendo de las entradas
      population(index_individual).nodegenes(4, find( (
population(index_individual).nodegenes(2,:) == 1 ) | (
population(index_individual).nodegenes(2,:) == 4 ) ) ) =
population(index_individual).nodegenes(3, find( (
population(index_individual).nodegenes(2,:) == 1 ) | (
population(index_individual).nodegenes(2,:) == 4 ) ) );
    population(index_individual).nodegenes(4, find( (
population(index_individual).nodegenes(2,:) == 2 ) | (
```

```
population(index_individual).nodegenes(2,:) == 3 ) ) ) = (
exp(sf*population(index_individual).nodegenes(3, find( (
population(index_individual).nodegenes(2,:) == 2 ) | (
population(index_individual).nodegenes(2,:) == 3 ) ) )) - exp(-
sf*population(index_individual).nodegenes(3, find( (
population(index_individual).nodegenes(2,:) == 2 ) | (
population(index_individual).nodegenes(2,:) == 3 ) ) )) )./(
exp(sf*population(index_individual).nodegenes(3, find( (
population(index_individual).nodegenes(2,:) == 2 ) | (
population(index_individual).nodegenes(2,:) == 3 ) ) )) + exp(-
sf*population(index_individual).nodegenes(3, find( (
population(index_individual).nodegenes(2,:) == 2 ) | (
population(index_individual).nodegenes(2,:) == 3 ) ) )) );

      no_change_count=0;
      index_loop=0;
      while (no_change_count<number_nodes) &
index_loop<3*number_connections
         index_loop=index_loop+1;

vector_node_state=population(index_individual).nodegenes(4,:);

         %%
         % empezar vectorización
         % primero remodelar connection_gene de ID de nodo a
índice en node_gene

max_node_ID=max(population(index_individual).nodegenes(1,:));

max_node_num=size(population(index_individual).nodegenes,2);
         vector_remodel=zeros(1,max_node_ID);

vector_remodel(population(index_individual).nodegenes(1,:))=[1:max
_node_num];

vector_nodes_from=vector_remodel(population(index_individual).conn
ectiongenes(2,:));

vector_nodes_to=vector_remodel(population(index_individual).connec
tiongenes(3,:));

         matrix_compute=zeros(max_node_num,number_connections);
         % computación vectorizada real
         matrix_compute(([1:number_connections]-
1)*max_node_num+vector_nodes_to(1,:))=population(index_individual)
.nodegenes(4,vector_nodes_from(:)).*population(index_individual).c
onnectiongenes(4,:).*population(index_individual).connectiongenes(
5,:);


population(index_individual).nodegenes(3,:)=ones(1,number_connecti
ons)*matrix_compute';
```

```
        % finalizar vectorización
        %%

        population(index_individual).nodegenes(4, find( (
population(index_individual).nodegenes(2,:) == 2 ) | (
population(index_individual).nodegenes(2,:) == 3 ) ) ) = (
exp(sf*population(index_individual).nodegenes(3, find( (
population(index_individual).nodegenes(2,:) == 2 ) | (
population(index_individual).nodegenes(2,:) == 3 ) )) - exp(-
sf*population(index_individual).nodegenes(3, find( (
population(index_individual).nodegenes(2,:) == 2 ) | (
population(index_individual).nodegenes(2,:) == 3 ) )) )./(
exp(sf*population(index_individual).nodegenes(3, find( (
population(index_individual).nodegenes(2,:) == 2 ) | (
population(index_individual).nodegenes(2,:) == 3 ) )) + exp(-
sf*population(index_individual).nodegenes(3, find( (
population(index_individual).nodegenes(2,:) == 2 ) | (
population(index_individual).nodegenes(2,:) == 3 ) )) );

        % Reinicializar las entradas para la siguiente iteración
        population(index_individual).nodegenes(3, find( (
population(index_individual).nodegenes(2,:) == 2 ) | (
population(index_individual).nodegenes(2,:) == 3 ) ) ) = 0; %
hacer todas las entradas de nodos ocultos y de salida cero

no_change_count=sum(abs(population(index_individual).nodegenes(4,:
)-vector_node_state)<no_change_threshold); % buscar a todos los
nodos cuya salida no haya cambiado más del umbral desde la última
iteración, a lo largo de conncection_genes
      end

      SE = [ SE ; ( O(index_pattern,1) -
population(index_individual).nodegenes(4, find(
population(index_individual).nodegenes(2,:) == 2 ) ) ).^2 ];

   end

   MSE = sum(SE)/size( I , 1 );
   population_plus_fitnesses(index_individual).fitness=(4-MSE)^2;
% función de aptitud

end

[ MaxFit , MaxIndex ] = max( [
population_plus_fitnesses(:).fitness ] );
Best_Net = population_plus_fitnesses( MaxIndex );

end
```

```
%%%
%%%
%%%  reproduce
%%%
%%%

function
[new_population,updated_species_record,updated_innovation_record]=
reproduce(population, species_record, innovation_record, initial,
selection, crossover, mutation, speciation, generation,
population_size);

% el siguiente loop "for" tiene tres objetivos:

% 1.computar la matriz de especies existentes y en propagación a
partir de species_record ( primera fila ), asignarles su número de
descendencia designado a partir de su aptitud compartida ( segunda
fila ), y hacer su número de descendencia real cero ( tercera fila
) ( éste será incrementado según se vayan creando individuos de
ésta especie )

% 2.copiar al individuo más apto en toda especie con más de
initial.number_copy individuos, intactos, a la nueva generación (
elitismo ) ( pero sólo si la especie no está muriendo, es decir,
tiene descendencia asignada en la nueva generación )
% se utiliza data de species_record.generation_record ( índice del
individuo con la aptitud más alta )

% 3.borrar el porcentaje más bajo ( initial.kill_percentage ) en
especies con más de initial.number_for_kill individuos para evitar
que se reproduzcan
% En realidad Matlab no ofrece la facilidad de redireccionar los
punteros que enlazan elementos en una estructura con la siguiente,
así que éste individuo no puede ser borrado de la estructura de la
población. En vez de eso, su ID de especie se hará cero,
% lo cual tiene el mismo efecto que removerlo del resto del ciclo
de reproducción, ya que todas las funciones de reproducción
acceden a la estructura de población a través del ID de especie, y
ninguna especie tiene ID cero

% Computar sum_average_fitnesses
sum_average_fitnesses=0;
for index_species=1:size(species_record,2)

sum_average_fitnesses=sum_average_fitnesses+species_record(index_s
pecies).generation_record(2,size(species_record(index_species).gen
eration_record,2))*(species_record(index_species).number_individua
ls>0);
end
% Las siguientes dos líneas sólo inicializan la estructura de la
nueva población. Ya que su especie se hace cero, el resto del
```

```
algoritmo la ignorará. Se sobrescribe tan pronto como el primer
nuevo individuo es creado

new_population(1)=population(1);
new_population(1).species=0;


overflow=0;
index_individual=0;
matrix_existing_and_propagating_species=[];
for index_species=1:size(species_record,2)
   if species_record(index_species).number_individuals>0 % probar
si las especies existen en la antigua generación

number_offspring=species_record(index_species).generation_record(2
,size(species_record(index_species).generation_record,2))/sum_aver
age_fitnesses*population_size; % computar el número de
descendencia en la nueva generación
      overflow=overflow+number_offspring-floor(number_offspring);
      if overflow>=1 % Como el tamaño de las nuevas especies son
fracciones, overflow suma la diferencia entre size y floor(size),
y siempre que overflow está sobre 1, la especie gana un individuo
         number_offspring=ceil(number_offspring);
         overflow=overflow-1;
      else
         number_offspring=floor(number_offspring);
      end
      if number_offspring>0 % Verificar si la especie está
muriendo, sólo añadir a matrix_existing_and_propagating_species
especies que tengan descendencia asignada

matrix_existing_and_propagating_species=[matrix_existing_and_propa
gating_species,[species_record(index_species).ID;number_offspring;
0]]; % matriz ( objetivo 1 )
         if
species_record(index_species).number_individuals>=initial.number_c
opy % verificar condición para objetivo 2
            index_individual=index_individual+1;

new_population(index_individual)=population(species_record(index_s
pecies).generation_record(4,size(species_record(index_species).gen
eration_record,2))); % Objetivo 2

matrix_existing_and_propagating_species(3,size(matrix_existing_and
_propagating_species,2))=1; % Actualizar
matrix_existing_and_propagating_species
         end
      end

      if
(species_record(index_species).number_individuals>initial.number_f
or_kill) &
```

```
(ceil(species_record(index_species).number_individuals*(1-
initial.kill_percentage))>2) % verificar condición para objetivo 3

matrix_individuals_species=[find([population(:).species]==index_sp
ecies);[population(find([population(:).species]==index_species)).f
itness]];

[sorted_fitnesses_in_species,sorting_vector]=sort(matrix_individua
ls_species(2,:));

matrix_individuals_species=matrix_individuals_species(:,sorting_ve
ctor);
         sorting_vector=matrix_individuals_species(1,:);

         for
index_kill=1:floor(species_record(index_species).number_individual
s*initial.kill_percentage)
            population(sorting_vector(index_kill)).species=0; %
objetivo 3
         end
      end

   end
end

% generar referencia de individuos al azar de todas las especies
de la población antigua
% iterar sobre ID's de especies, y añadir referencias de
individuos de la población antigua, las nuevas especies de la
nueva población serán agregadas durante la reproducción
index_ref=0;
for index_species_ref=1:size(species_record,2)
   if sum([population(:).species]==index_species_ref)>0 %
Verificar si la especie existe en la población antigua
      index_ref=index_ref+1;

[discard,index_ref_old]=max(([population(:).species]==index_specie
s_ref).*rand(1,size(population,2)));
      population_ref(index_ref)=population(index_ref_old);
   end
end
matrix_existing_and_propagating_species

%% Reproducción Estándar ( Principal )
%% Iterar sobre todas las especies existentes
for
index_species=1:size(matrix_existing_and_propagating_species,2)
   count_individuals_species=0;

species_ID=matrix_existing_and_propagating_species(1,index_species
); % este es el ID de la especie que se reproucirá este ciclo
```

```
   % IMPORTANTE: index_species sólo tiene relevancia para
matrix_existing_and_propagating_species, todos los otros
mecanismos que usen especies de alguna manera deben usar
species_ID

   % Rankeo Lineal y Muestreo Universal Estocástico
   % Rankeo con selection.pressure

fitnesses_species=[population(find([population(:).species]==specie
s_ID)).fitness];

index_fitnesses_species=find([population(:).species]==species_ID);
   [discard,sorted_fitnesses]=sort(fitnesses_species);
   ranking=zeros(1,size(fitnesses_species,2));
   ranking(sorted_fitnesses)=1:size(fitnesses_species,2);
   if size(fitnesses_species,2)>1
       FitnV=(2-selection.pressure+2*(selection.pressure-
1)/(size(fitnesses_species,2)-1)*(ranking-1))';
   else
       FitnV=2;
   end
   % Muestreo Universal Estocástico
   % Primero se computa el número de individuos a ser
seleccionados ( se requieren dos padres para procrear por cruce,
uno por mutación )

number_overall=matrix_existing_and_propagating_species(2,index_spe
cies)-matrix_existing_and_propagating_species(3,index_species);
   number_crossover=round(crossover.percentage*number_overall);
   number_mutate=number_overall-number_crossover;
   Nind=size(fitnesses_species,2);
   Nsel=2*number_crossover+number_mutate;

   if Nsel==0
      Nsel=1;
   end

   % Realizar muestreo universal estocástico
   cumfit = cumsum(FitnV);
   trials = cumfit(Nind) / Nsel * (rand + (0:Nsel-1)');
   Mf = cumfit(:, ones(1, Nsel));
   Mt = trials(:, ones(1, Nind))';
   [NewChrIx, ans] = find(Mt < Mf & [ zeros(1, Nsel); Mf(1:Nind-1,
:) ] <= Mt);
   % Mezclar individuos seleccionados
   [ans, shuf] = sort(rand(Nsel, 1));
   NewChrIx = NewChrIx(shuf);
   % relacionar a índices de individuos en población
   NewChrIx = index_fitnesses_species(NewChrIx);
```

```
   while
matrix_existing_and_propagating_species(3,index_species)<matrix_ex
isting_and_propagating_species(2,index_species) % iterar hasta que
el número asignado de descendencia se complete
      index_individual=index_individual+1;
      count_individuals_species=count_individuals_species+1;
      % Crossover
      if count_individuals_species<=number_crossover %O.k: se
realizará cruce
         % Seleccionar Padres
         % Seleccionar parent1
         parent1=population(NewChrIx(2*count_individuals_species-
1));
         % Seleccionar parent2
         found_parent2=0;
         if (rand<crossover.probability_interspecies) &
(size(matrix_existing_and_propagating_species,2)>1) % Seleccionar
parent2 de otra especie ( sólo si existe más de una especie en la
población antigua )
            while found_parent2==0;

[discard,index_parent2]=max(rand(1,size(population,2)));
               parent2=population(index_parent2);
               found_parent2=((parent2.species~=0) &
(parent2.species~=parent1.species)); % verificar si
parent2.species no es 0 o de la misma especie que parent1
            end
            parent2.fitness=parent1.fitness; % igualar aptitudes
para asegurar que genes disjuntos y exceso se hereden
completamente de ambos padres
         else % O.K. tomamos a parent2 de la misma especie que
parent1

parent2=population(NewChrIx(2*count_individuals_species));
         end

         % Crossover
         % heredar nodos de ambos padres
         new_individual.nodegenes=[];

matrix_node_lineup=[[parent1.nodegenes(1,:);1:size(parent1.nodegen
es,2);zeros(1,size(parent1.nodegenes,2))],[parent2.nodegenes(1,:);
zeros(1,size(parent2.nodegenes,2));1:size(parent2.nodegenes,2)]];
         [discard,sort_node_vec]=sort(matrix_node_lineup(1,:));
         matrix_node_lineup=matrix_node_lineup(:,sort_node_vec);
         node_number=0;
         for index_node_sort=1:size(matrix_node_lineup,2)
            if node_number~=matrix_node_lineup(1,index_node_sort)
               if matrix_node_lineup(2,index_node_sort)>0

new_individual.nodegenes=[new_individual.nodegenes,parent1.nodegen
es(:,matrix_node_lineup(2,index_node_sort))];
```

```
                else

new_individual.nodegenes=[new_individual.nodegenes,parent2.nodegen
es(:,matrix_node_lineup(3,index_node_sort))];
                end
                node_number=matrix_node_lineup(1,index_node_sort);
             end
          end
          % Cruce de genes de conexión
          % primero alinear los genes

matrix_lineup=[[parent1.connectiongenes(1,:);1:size(parent1.connec
tiongenes,2);zeros(1,size(parent1.connectiongenes,2))],[parent2.co
nnectiongenes(1,:);zeros(1,size(parent2.connectiongenes,2));1:size
(parent2.connectiongenes,2)]];
          [discard,sort_vec]=sort(matrix_lineup(1,:));
          matrix_lineup=matrix_lineup(:,sort_vec);
          final_matrix_lineup=[];
          innovation_number=0;
          for index_sort=1:size(matrix_lineup,2)
             if innovation_number~=matrix_lineup(1,index_sort)

final_matrix_lineup=[final_matrix_lineup,matrix_lineup(:,index_sor
t)];
                innovation_number=matrix_lineup(1,index_sort);
             else

final_matrix_lineup(2:3,size(final_matrix_lineup,2))=final_matrix_
lineup(2:3,size(final_matrix_lineup,2))+matrix_lineup(2:3,index_so
rt);
             end
          end
          % O.K. los genes están alineados, empezar con el cruce
          new_individual.connectiongenes=[];

          for index_lineup=1:size(final_matrix_lineup,2)
             if (final_matrix_lineup(2,index_lineup)>0) &
(final_matrix_lineup(3,index_lineup)>0) % buscar genes
emparejados, cruzar
                if rand<0.5 % cruce aleatorio para genes
emparejados

new_individual.connectiongenes=[new_individual.connectiongenes,par
ent1.connectiongenes(:,final_matrix_lineup(2,index_lineup))];
                else

new_individual.connectiongenes=[new_individual.connectiongenes,par
ent2.connectiongenes(:,final_matrix_lineup(3,index_lineup))];
                end
                if rand>crossover.probability_multipoint %
promediando pesos para la descendencia, o los pesos heredados se
dejan tal cual
```

```
new_individual.connectiongenes(4,size(new_individual.connectiongen
es,2))=(parent1.connectiongenes(4,final_matrix_lineup(2,index_line
up))+parent2.connectiongenes(4,final_matrix_lineup(3,index_lineup)
))/2;
                end
            end

parent1_flag=sum(final_matrix_lineup(2,index_lineup+1:size(final_m
atrix_lineup,2)));  % probar si existe más genes de conexión desde
index_lineup+1 hasta el final de final_matrix_lineup para parent1
( para detectar exceso )

parent2_flag=sum(final_matrix_lineup(3,index_lineup+1:size(final_m
atrix_lineup,2)));  % probar si existe más genes de conexión desde
index_lineup+1 hasta el final de final_matrix_lineup para parent2
( para detectar exceso )
            % 2 casos a verificar ( la parte de genes disjuntos se
encarga del exceso )
            if (final_matrix_lineup(2,index_lineup)>0) &
(final_matrix_lineup(3,index_lineup)==0) % parent1 disjunto
                if parent1.fitness>=parent2.fitness

new_individual.connectiongenes=[new_individual.connectiongenes,par
ent1.connectiongenes(:,final_matrix_lineup(2,index_lineup))];
                end
            end
            if (final_matrix_lineup(2,index_lineup)==0) &
(final_matrix_lineup(3,index_lineup)>0) % parent2 disjunto
                if parent2.fitness>=parent1.fitness

new_individual.connectiongenes=[new_individual.connectiongenes,par
ent2.connectiongenes(:,final_matrix_lineup(3,index_lineup))];
                end
            end
        end
        new_individual.fitness=0; % no tiene impacto en el
algoritmo, sólo es necesario para asignación a la nueva población
        new_individual.species=parent1.species; % será la pista
para la especie en la especiación
    else % sin cruce, se copia a un individuo de la especie y se
lo muta

new_individual=population(NewChrIx(number_crossover+count_individu
als_species));
    end

    % Eliminacion de nodos ocultos ( remover cualuier nodo
oculto sin gen de conexión correspondiente )
    connected_nodes=[];
    for index_node_culling=1:size(new_individual.nodegenes,2)
```

```
node_connected_flag=sum(new_individual.connectiongenes(2,:)==new_i
ndividual.nodegenes(1,index_node_culling))+sum(new_individual.conn
ectiongenes(3,:)==new_individual.nodegenes(1,index_node_culling));
          if (node_connected_flag>0) |
(new_individual.nodegenes(2,index_node_culling)~=3);

connected_nodes=[connected_nodes,new_individual.nodegenes(:,index_
node_culling)];
          end
      end
      new_individual.nodegenes=connected_nodes;

      % Mutación de genes deshabilitados
      % buscar genes de conexión deshabilitados y habilitarlos con
una probabilidad de crossover.probability_gene_reenabled
      for
index_connection_gene=1:size(new_individual.connectiongenes,2)
          if
(new_individual.connectiongenes(5,index_connection_gene)==0)&
(rand<mutation.probability_gene_reenabled)

new_individual.connectiongenes(5,index_connection_gene)=1;
          end
      end

      % Mutación de Peso
      % iterar sobre los genes del nuevo individuo y decidir si
mutar o no
      for
index_connection_gene=1:size(new_individual.connectiongenes,2)
          if rand<mutation.probability_mutate_weight % linealmente
inclinado para mayores probabilidades de mutación hacia el final
de los genes de conexión

new_individual.connectiongenes(4,index_connection_gene)=new_indivi
dual.connectiongenes(4,index_connection_gene)+mutation.weight_rang
e*(rand-0.5);
          end
          % tope de peso

new_individual.connectiongenes(4,index_connection_gene)=new_indivi
dual.connectiongenes(4,index_connection_gene)*(abs(new_individual.
connectiongenes(4,index_connection_gene))<=mutation.weight_cap)+(s
ign(new_individual.connectiongenes(4,index_connection_gene))*mutat
ion.weight_cap)*(abs(new_individual.connectiongenes(4,index_connec
tion_gene))>mutation.weight_cap);
      end

      % IMPORTANTE: La búsqueda de innovaciones duplicadas en los
siguientes dos tipos de mutación sólo puede verificar la
generación actual
```

```
      % Mutación Agregar Conexión

flag_recurrency_enabled=rand<mutation.probability_recurrency;

vector_possible_connect_from_nodes=new_individual.nodegenes(1,:);
% conexiones pueden salir de cualquier nodo

vector_possible_connect_to_nodes=new_individual.nodegenes(1,find((
new_individual.nodegenes(2,:)==2)+(new_individual.nodegenes(2,:)==
3))); % conexiones pueden llegar sólo a nodos ocultos y de salida

number_possible_connection=length(vector_possible_connect_from_nod
es)*length(vector_possible_connect_to_nodes)-
size(new_individual.connectiongenes,2);

      flag1=(rand<mutation.probability_add_node);

      if (rand<mutation.probability_add_connection) &
(number_possible_connection>0) & (flag1==0) % ver si se pueden
añadir nuevas conexiones a los genes ( si existe alguna conexión
posible que todavía no exista en los genes del nuevo individuo )
         % Primero se construye una matriz con todas las posibles
conexiones nuevas para nodegene del nuevo individuo

         new_connection_matrix=[];
         for
index_connect_from=1:length(vector_possible_connect_from_nodes)
            for
index_connect_to=1:length(vector_possible_connect_to_nodes)

possible_connection=[vector_possible_connect_from_nodes(index_conn
ect_from);vector_possible_connect_to_nodes(index_connect_to)];
               if
sum((new_individual.connectiongenes(2,:)==possible_connection(1)).
*(new_individual.connectiongenes(3,:)==possible_connection(2)))==0
% Verificar si la conexión propuesta no está contenida en el gen

new_connection_matrix=[new_connection_matrix,possible_connection];
               end
            end
         end
         % Mezclar las posibles nuevas conexiones aleatoriamente

[discard,shuffle]=sort(rand(1,size(new_connection_matrix,2)));
         new_connection_matrix=new_connection_matrix(:,shuffle);

         index_new_connection=0;
         flag_connection_ok=0;
         % verificar que la conexión esté o.k. ( recurrencia ) si
no se encuentra una conexión adecuada no se agregará ninguna
         while (flag_connection_ok==0) &
(index_new_connection<size(new_connection_matrix,2))
```

```
            index_new_connection=index_new_connection+1;

new_connection=new_connection_matrix(:,index_new_connection);

            % Probar si la conexión es recurrente
            flag_recurrent=0;
            if new_connection(1)==new_connection(2)
               flag_recurrent=1;
            end
            nodes_current_level=new_connection(2);
            depth=0;
            while flag_recurrent==0 &
depth<size(new_individual.connectiongenes,2) &
~isempty(nodes_current_level)
               depth=depth+1;
               nodes_next_level=[];
               for index_check=1:size(nodes_current_level);

nodes_next_level=[nodes_next_level,new_individual.connectiongenes(
3,find(new_individual.connectiongenes(2,:)==nodes_current_level(in
dex_check)))];
               end
               if sum(nodes_next_level(:)==new_connection(1))>0
                  flag_recurrent=1;
               end
               nodes_current_level=nodes_next_level;
            end
            if flag_recurrent==0
               flag_connection_ok=1;
            elseif flag_recurrency_enabled
               flag_connection_ok=1;
            end
         end

         % Ahora se prueba si ésta es una verdadera innovación (
no se ha dado en esta generación )
         if flag_connection_ok

index_already_happened=find((innovation_record(5,:)==generation).*
(innovation_record(2,:)==new_connection(1)).*(innovation_record(3,
:)==new_connection(2))); % Setear flag de innovación real
            new_innovation=not(sum(index_already_happened));
            if new_innovation==1 % O.K. nueva innovación

new_connection=[max(innovation_record(1,:))+1;new_connection]; %
Actualizar la nueva conexión con número de innovación
               % Actualizar connection_genes

new_individual.connectiongenes=[new_individual.connectiongenes,[ne
w_connection;rand*2-1;1]];
               % Actualizar innovation_record
```

```
innovation_record=[innovation_record,[new_connection;0;generation]
];
            else % la conexión ya existe en innovation_record de
esta generación
                % Actualizar connection_genes

new_individual.connectiongenes=[new_individual.connectiongenes,[in
novation_record(1:3,index_already_happened);rand*2-1;1]];
            end
          end
       end

       % Mutación Agregar Nodo
       new_innovation=0;
       if flag1==1

max_old_innovation_number=max((innovation_record(5,:)<generation).
*innovation_record(1,:)); % la innovación más alta de la
generación antigua

vector_possible_connections=[new_individual.connectiongenes(2:3,fi
nd((new_individual.connectiongenes(5,:)==1) &
(new_individual.connectiongenes(1,:)<=max_old_innovation_number)))
;find((new_individual.connectiongenes(5,:)==1) &
(new_individual.connectiongenes(1,:)<=max_old_innovation_number))]
;  % computar vector de conexiones en las que un nuevo nodo podría
insertarse, y sus posiciones en la matriz connection_gene. Éste
vector esta compuesto de todos las conexiones no deshabilitadas de
la última generación hacia atras

          if size(vector_possible_connections,2) ~= 0

insert_node_connection=vector_possible_connections(:,round(rand*si
ze(vector_possible_connections,2)+0.5));
              new_innovation=1; % provisionalmente, se verificará

exist_innovation=find((innovation_record(5,:)==generation).*(innov
ation_record(4,:)>0).*(innovation_record(2,:)==insert_node_connect
ion(1))); % Empezando a probar innovation_record para verificar la
autenticidad de la innovación. exist_innovation contiene un vector
de índices de elementos en innovation_record que cumplen con:
generación actual, mutación agregar nodo, y la misma "conexión de"
que la innovación actual
              if sum(exist_innovation)>0 % si esto se cumple,
tenemos que probar al nodo connect_to para ver si la innovación es
realmente la misma
                  for index_check=1:length(exist_innovation)
                     if
innovation_record(3,exist_innovation(index_check)+1)==insert_node_
connection(2)
                        new_innovation=0;
```

```
index_already_existent_this_generation=exist_innovation(index_chec
k);
                    end
                end
            end
            if new_innovation==1 %O.K. innovación real
                % Actualizar node_genes
                new_node_number=max(innovation_record(4,:))+1;

new_individual.nodegenes=[new_individual.nodegenes,[new_node_numbe
r;3;0;0]];
                % Actualizar connection_genes

new_individual.connectiongenes(5,insert_node_connection(3))=0; %
deshabilitar gen de conexión antiguo

new_connections=[[max(innovation_record(1,:))+1;insert_node_connec
tion(1);new_node_number;1;1],[max(innovation_record(1,:))+2;new_no
de_number;insert_node_connection(2);new_individual.connectiongenes
(4,insert_node_connection(3));1]];

new_individual.connectiongenes=[new_individual.connectiongenes,new
_connections]; % extender connection_genes por dos conexiones
nuevas
                % Actualizar innovation_record

innovation_record=[innovation_record,[new_connections(1:3,:);new_n
ode_number,0;generation,generation]];
            else % no es innovación real
                % Actualizar node_genes

node_number=innovation_record(4,index_already_existent_this_genera
tion);

new_individual.nodegenes=[new_individual.nodegenes,[node_number;3;
0;0]];
                % Actualizar connection_genes

new_individual.connectiongenes(5,insert_node_connection(3))=0; %
deshabilitar gen de conexión antiguo

new_connections=[innovation_record(1:3,index_already_existent_this
_generation:index_already_existent_this_generation+1);1,new_indivi
dual.connectiongenes(4,insert_node_connection(3));1,1];

length_con_gen=size(new_individual.connectiongenes,2);
                if
new_individual.connectiongenes(1,length_con_gen)>new_connections(1
,2) % ver si hubo una mutación de agregar conexión al individuo
actual que tenga una innovación más alta que la actual mutación de
agregar nodo
```

```
new_individual.connectiongenes=[new_individual.connectiongenes(:,1
:length_con_gen-
1),new_connections,new_individual.connectiongenes(:,length_con_gen
)];


                else

new_individual.connectiongenes=[new_individual.connectiongenes,new
_connections];
                end
            end
        end
    end

    %% Especiación
    % Iterar sobre el vector de comparación

        species_assigned=0;
      % sacar reference_individual de poblation_ref
        index_population_ref = find( [ population_ref(:).species
] == new_individual.species );

reference_individual=population_ref(index_population_ref);
        % iterar sobre ambos genes de conexión, computar genes
disjuntos, exceso, y diferencia de peso promedio

max_num_innovation=max([new_individual.connectiongenes(1,:),refere
nce_individual.connectiongenes(1,:)]);

vector_innovation_new=[zeros(1,max(new_individual.connectiongenes(
1,:))),ones(1,max_num_innovation-
max(new_individual.connectiongenes(1,:)))];

vector_innovation_new(new_individual.connectiongenes(1,:))=2;
        vector_weight_new=zeros(1,max_num_innovation);

vector_weight_new(new_individual.connectiongenes(1,:))=new_individ
ual.connectiongenes(4,:);

vector_innovation_ref=[4*ones(1,max(reference_individual.connectio
ngenes(1,:))),8*ones(1,max_num_innovation-
max(reference_individual.connectiongenes(1,:)))];

vector_innovation_ref(reference_individual.connectiongenes(1,:))=1
6;
        vector_weight_ref=zeros(1,max_num_innovation);

vector_weight_ref(reference_individual.connectiongenes(1,:))=refer
ence_individual.connectiongenes(4,:);
```

```
vector_lineup=vector_innovation_new+vector_innovation_ref;
         excess=sum(vector_lineup==10)+sum(vector_lineup==17);
         disjoint=sum(vector_lineup==6)+sum(vector_lineup==16);
         vector_matching=find(vector_lineup==18);

average_weight_difference=sum(abs(vector_weight_new(vector_matchin
g)-vector_weight_ref(vector_matching)))/length(vector_matching);
         max_num_genes=1;

distance=speciation.c1*excess/max_num_genes+speciation.c2*disjoint
/max_num_genes+speciation.c3*average_weight_difference;
         if distance<speciation.threshold
            % asignar individuo a la misma especie

            new_individual.species=reference_individual.species;
            species_assigned=1; % indicar que el nuevo individuo
ha sido asignado a una especie
         end
         index_population_ref=0;

      while species_assigned==0 &
index_population_ref<size(population_ref,2)
         % sacar reference_individual de population_ref
         index_population_ref=index_population_ref+1;

reference_individual=population_ref(index_population_ref);
         if new_individual.species ~= reference_individual.species
            % iterar sobre ambos genes de conexión, computar
genes disjuntos, exceso, y diferencia de peso promedio

max_num_innovation=max([new_individual.connectiongenes(1,:),refere
nce_individual.connectiongenes(1,:)]);

vector_innovation_new=[zeros(1,max(new_individual.connectiongenes(
1,:))),ones(1,max_num_innovation-
max(new_individual.connectiongenes(1,:)))];

vector_innovation_new(new_individual.connectiongenes(1,:))=2;
            vector_weight_new=zeros(1,max_num_innovation);

vector_weight_new(new_individual.connectiongenes(1,:))=new_individ
ual.connectiongenes(4,:);

vector_innovation_ref=[4*ones(1,max(reference_individual.connectio
ngenes(1,:))),8*ones(1,max_num_innovation-
max(reference_individual.connectiongenes(1,:)))];

vector_innovation_ref(reference_individual.connectiongenes(1,:))=1
6;
            vector_weight_ref=zeros(1,max_num_innovation);
```

```
vector_weight_ref(reference_individual.connectiongenes(1,:))=refer
ence_individual.connectiongenes(4,:);

vector_lineup=vector_innovation_new+vector_innovation_ref;
            excess=sum(vector_lineup==10)+sum(vector_lineup==17);

disjoint=sum(vector_lineup==6)+sum(vector_lineup==16);
            vector_matching=find(vector_lineup==18);

average_weight_difference=sum(abs(vector_weight_new(vector_matchin
g)-vector_weight_ref(vector_matching)))/length(vector_matching);
            max_num_genes=1;

distance=speciation.c1*excess/max_num_genes+speciation.c2*disjoint
/max_num_genes+speciation.c3*average_weight_difference;
            if distance<speciation.threshold
               % asignar individuo a la misma especie


new_individual.species=reference_individual.species;
               species_assigned=1; % indicar que el nuevo
individuo ha sido asignado a una especie
            end
         end
      end

      % si no es compatible con ninguna especie se crea una nueva
      if species_assigned==0
         new_species_ID=size(species_record,2)+1;
           % asignar individuo a nueva especie
         new_individual.species=new_species_ID;
         % actualizar species_record
         species_record(new_species_ID).ID=new_species_ID;
         species_record(new_species_ID).number_individuals=1;
         species_record(new_species_ID).generation_record=[];
         % actualizar population reference
         population_ref(size(population_ref,2)+1)=new_individual;
      end

      % agregar nuevo inividuo a nueva población
      new_population(index_individual)=new_individual;

      % incrementar especie

matrix_existing_and_propagating_species(3,index_species)=matrix_ex
isting_and_propagating_species(3,index_species)+1;
   end
end

% actualización final de species_record
for index_species=1:size(species_record,2)
```

```
species_record(index_species).number_individuals=sum([new_populati
on(:).species]==index_species);
end
% asignar species_record actualizado a la salida
updated_species_record=species_record;
% asignar innovation_record actualizado a la salida
updated_innovation_record=innovation_record;
```

```
%%%
%%%
%%%  Controlador Neuronal
%%%
%%%

function F = Neural_Controller( Theta_Error, Best_Net)

%   Detailed explanation goes here

% umbral para determinar si el estado de un nodo ha cambiado
significativamente desde la última iteración
no_change_threshold=1e-3;

Theta_Error = ReScale_PID( Theta_Error , -2*pi , 2*pi , -1 , 1 );

sf = 1.0;

%for index_individual=1:number_individuals
   number_nodes=size(Best_Net.nodegenes,2);
   number_connections=size(Best_Net.connectiongenes,2);
   Y=0;
   Z=[];
   index_pattern=1;
      % setear el valor de entrada de los nodos para la primera
iteracion
      Best_Net.nodegenes(3, find( ( Best_Net.nodegenes(2,:) == 2 )
| ( Best_Net.nodegenes(2,:) == 3 ) ) ) = 0; % hacer cero las
salidas de todos los nodos
      Best_Net.nodegenes(3, find( Best_Net.nodegenes(2,:) == 4 ) )
= 1; % hacer la entrada de nodo bias 1
      Best_Net.nodegenes(3, find( Best_Net.nodegenes(2,:) == 1 ) )
= Theta_Error(index_pattern,:); % el nodo entrada tiene el valor
de Theta_Error

      % setear los valores de salida dependiendo de las entradas
      Best_Net.nodegenes(4, find( ( Best_Net.nodegenes(2,:) == 1 )
| ( Best_Net.nodegenes(2,:) == 4 ) ) ) = Best_Net.nodegenes(3,
find( ( Best_Net.nodegenes(2,:) == 1 ) | ( Best_Net.nodegenes(2,:)
== 4 ) ) );
      Best_Net.nodegenes(4, find( ( Best_Net.nodegenes(2,:) == 2 )
| ( Best_Net.nodegenes(2,:) == 3 ) ) ) = (
exp(sf*Best_Net.nodegenes(3, find( ( Best_Net.nodegenes(2,:) == 2
) | ( Best_Net.nodegenes(2,:) == 3 ) ) )) - exp(-
sf*Best_Net.nodegenes(3, find( ( Best_Net.nodegenes(2,:) == 2 ) |
( Best_Net.nodegenes(2,:) == 3 ) ) )) )./(
exp(sf*Best_Net.nodegenes(3, find( ( Best_Net.nodegenes(2,:) == 2
) | ( Best_Net.nodegenes(2,:) == 3 ) ) )) + exp(-
sf*Best_Net.nodegenes(3, find( ( Best_Net.nodegenes(2,:) == 2 ) |
( Best_Net.nodegenes(2,:) == 3 ) ) )) );
      no_change_count=0;
      index_loop=0;
```

```
      while (no_change_count<number_nodes) &
index_loop<3*number_connections
         index_loop=index_loop+1;
         vector_node_state=Best_Net.nodegenes(4,:);
         for index_connections=1:number_connections
            % leer contenidos relevantes de connection_gene ( ID
del Nodo donde la conexión empieza, ID del Nodo donde termina, y
el peso de conexión )

ID_connection_from_node=Best_Net.connectiongenes(2,index_connectio
ns);

ID_connection_to_node=Best_Net.connectiongenes(3,index_connections
);

connection_weight=Best_Net.connectiongenes(4,index_connections);
            % mapear el ID del nodo al índice del nodo
correspondiente en la matriz node_genes

index_connection_from_node=find((Best_Net.nodegenes(1,:)==ID_conne
ction_from_node));

index_connection_to_node=find((Best_Net.nodegenes(1,:)==ID_connect
ion_to_node));

            if Best_Net.connectiongenes(5,index_connections)==1 %
Verificar que la conexión está habilitada

Best_Net.nodegenes(3,index_connection_to_node)=Best_Net.nodegenes(
3,index_connection_to_node)+Best_Net.nodegenes(4,index_connection_
from_node)*connection_weight
            end
         end
         % pasar los valores de entrada a las salidas para la
siguiente iteración
         Best_Net.nodegenes(4, find( ( Best_Net.nodegenes(2,:) ==
2 ) | ( Best_Net.nodegenes(2,:) == 3 ) ) ) = (
exp(sf*Best_Net.nodegenes(3, find( ( Best_Net.nodegenes(2,:) == 2
) | ( Best_Net.nodegenes(2,:) == 3 ) ) )) - exp(-
sf*Best_Net.nodegenes(3, find( ( Best_Net.nodegenes(2,:) == 2 ) |
( Best_Net.nodegenes(2,:) == 3 ) ) )) )./(
exp(sf*Best_Net.nodegenes(3, find( ( Best_Net.nodegenes(2,:) == 2
) | ( Best_Net.nodegenes(2,:) == 3 ) ) )) + exp(-
sf*Best_Net.nodegenes(3, find( ( Best_Net.nodegenes(2,:) == 2 ) |
( Best_Net.nodegenes(2,:) == 3 ) ) )) );
         % Reinicializar las entradas para la siguiente iteración
         Best_Net.nodegenes(3, find( ( Best_Net.nodegenes(2,:) ==
2 ) | ( Best_Net.nodegenes(2,:) == 3 ) ) ) = 0; % hacer todas las
entradas de nodos ocultos y de salida cero
         no_change_count=sum(abs(Best_Net.nodegenes(4,:)-
vector_node_state)<no_change_threshold); % buscar a todos los
```

```
nodos cuya salida no haya cambiado más del umbral desde la última
iteración, a lo largo de conncection_genes
      end

      Z=[Z;Best_Net.nodegenes(4, find( Best_Net.nodegenes(2,:) ==
2 ) )];

%end
Y = ReScale_PID( Z(:,1) , -1 , 1 , -9000.0 , 9000.0 );

F = Y;

end
```