UNIVERSIDAD NACIONAL JORGE BASADRE GROHMANN – TACNA

Facultad de Ingeniería

Escuela Profesional de Ingeniería en Informática y Sistemas

APRENDIZAJE DE REDES NEURONALES ARTIFICIALES BASADO EN ALGORITMOS EVOLUTIVOS DE INSPIRACIÓN CUÁNTICA

TESIS

Presentada por:

Bach. Israel Nazareth Chaparro Cruz

Para optar el Título Profesional de:

INGENIERO EN INFORMÁTICA Y SISTEMAS

TACNA - PERÚ

2018

UNIVERSIDAD NACIONAL JORGE BASADRE GROHMANN - TACNA

FACULTAD DE INGENIERÍA

JURADO CALIFICADOR Y CALIFICACIÓN DE LA SUSTENTACIÓN DE TESIS

TESIS Nº: _____ TÍTULO PROFESIONAL DE:

Ingeniero en Informática y Sistemas

La Secretaría Académica Administrativa de la Facultad de Ingeniería, por resolución de Facultad Nº 04633-2017-FAIN/UNJBG, designó jurado para la sustentación oral de la Tesis titulada: APRENDIZAJE DE REDES NEURONALES ARTIFICIALES BASADO EN ALGORITMOS EVOLUTIVOS DE INSPIRACIÓN CUÁNTICA.

El mismo que está conformado por:
Presidente: Dr. Edwin Antonio Hinojosa Ramos
Secretario: MSc. Edgar Aurelio Taya Acosta
Vocal: Mgtr. Gianfranco Alexey Málaga Tejada
Para calificar la sustentación de la tesis en acto público el día 22 de diciembre del 2017, presentado por el Bachiller Israel Nazareth Chaparro Cruz, de la Escuela Profesional de Ingeniería en Informática y Sistemas.

El jurado Calificador en forma secreta e individual emitió su opinión sobre el tema de la tesis expuesta y procedió a obtener el promedio que arrojó el calificativo de aprobado con la nota de Diecisiete (17).

Para ratificar lo detallado firman:

Dr. Edwin Antonio Hinojosa Ramos
Presidente

MSc. Edgar Aurelio Taya Acosta
Secretario

Mgtr. Gianfranco Alexey Málaga Tejada
Vocal

UNIVERSIDAD NACIONAL JORGE BASADRE GROHMANN

Facultad de Ingeniería

ESCUELA PROFESIONAL DE INGENIERÍA EN INFORMÁTICA Y SISTEMAS

APRENDIZAJE DE REDES NEURONALES ARTIFICIALES BASADO EN ALGORITMOS EVOLUTIVOS DE INSPIRACIÓN CUÁNTICA

TESIS SUSTENTADA Y APROBADA EL 22 DE DICIEMBRE DEL 2017
ESTANDO EL JURADO CALIFICADOR INTEGRADO POR:

Presidente : _____

Dr. Edwin Antonio Hinojosa Ramos

Secretario : _____

MSc. Edgar Aurelio Taya Acosta

Vocal : _____

Mgtr. Gianfranco Alexey Málaga Tejada

Asesor : _____

Dr. Erbert Francisco Osco Mamani

DEDICATORIA

A:. L:. G:. D:. G:. A:. D:. U:.

AGRADECIMIENTOS

A mi padre Américo, quien es y siempre será un ejemplo a seguir.

A mi madre Clara, quien me dio la vida y su amor incondicional.

A mi hermano Jason, por cuidarme, protegerme y enseñarme.

CONTENIDO

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

ÍNDICE DE ANEXOS

RESUMEN

El propósito de la tesis denominada “Aprendizaje de redes neuronales artificiales basado en algoritmos evolutivos de inspiración cuántica” se centró en implementar un algoritmo evolutivo para solucionar el problema de búsqueda local y convergencia lenta propios de los métodos clásicos de aprendizaje supervisado de redes neuronales artificiales.

El diseño de investigación es no experimental descriptivo. Se realiza una implementación del algoritmo para luego ajustarlo al problema y finalmente compararlo con el algoritmo back-propagation. La población para el entrenamiento de la red neuronal está compuesta por el conjunto de datos “Human activity recognition using smartphones dataset” del repositorio de machine learning de la University of California, Irvine (UCI).

Los resultados de la investigación apoyan el paradigma de las metaheurísticas y su gran uso en problemas de optimización frente a otros métodos clásicos y fueron los siguientes: se logró implementar con éxito un algoritmo evolutivo de inspiración cuántica que minimice una función objetivo, se logró

ajustar el algoritmo evolutivo de inspiración cuántica al proceso de aprendizaje de la red neuronal artificial y de la comparación con el algoritmo back-propagation se obtuvo una mejora en tiempo del 46% y en convergencia del 55%.

INTRODUCCIÓN

El presente trabajo de investigación se ha estructurado de la siguiente manera: en el Capítulo I se describe la problemática, objetivos e hipótesis; en el Capítulo II se integran las bases teóricas y la definición de términos que se toman en cuenta para el desarrollo de la investigación; en el Capítulo III se desarrolla el marco metodológico de la investigación; en el Capítulo IV se muestran los resultados obtenidos a partir de la validación de las hipótesis, se discuten los resultados obtenidos teniendo como soporte y análisis los antecedentes del presente estudio. Finalmente damos a conocer las conclusiones a las que se arribó en la presente investigación, luego se presentan las recomendaciones y la bibliografía que se consideró para el presente trabajo.

CAPÍTULO I

PLANTEAMIENTO DEL PROBLEMA

1.1 Descripción del problema

1.1.1 Antecedentes del problema

Existen distintas investigaciones que se avocan a resolver el problema del aprendizaje de redes neuronales mediante el algoritmo back-propagation; a continuación, citamos las que sirvieron a plantear la problemática de investigación:

A nivel internacional, Soudry & Carmon (2016) en su investigación denominada "No bad local minima: Data independent training error guarantees" introduce ruido estocástico en el proceso de aprendizaje supervisado de la red neuronal artificial para escapar de mínimos locales en el algoritmo back-propagation.

Mohd & Muhammad (2017) en su investigación denominada "Optimal parameter selection using three-term back-propagation algorithm for data classification", propone el algoritmo BPGD-A3T en el que se plantean parámetros adaptativos para la ganancia, momento y factor de aprendizaje en el algoritmo back-

propagation gradient descendant.

Ambas investigaciones nos permiten entender que los algoritmos de aprendizaje para redes neuronales son aún un tema vigente y de estudio en el que se busca optimizar el aprendizaje de estos.

1.1.2 Problemática de la investigación

Las redes neuronales artificiales consisten en un conjunto de procesadores (neuronas) asociados entre sí (topología) que interactúan en inhibición o excitación (pesos) para relacionar de forma compleja sus entradas y salidas, logrando con esto muy buenos resultados diversos ámbitos como la clasificación y predicción.

De las diversas decisiones a tomarse para la implementación de una red neuronal artificial (tipo de red neuronal, funciones de activación, topología), el paso final es llevar a cabo el aprendizaje supervisado que es el proceso en el cual se ajustan los pesos (normalmente aleatorios al inicio), el algoritmo más conocido para lograr este objetivo es el algoritmo back-propagation que hace uso del descenso por gradiente a través una función error basada en los pesos.

Sin embargo, los algoritmos de aprendizaje que hacen uso del descenso por gradiente (como back-propagation) presentan problemas propios del método matemático que aplican:

- Suelen quedarse atrapados en mínimos locales.
- Suelen ser computacionalmente costosos.

Ambas situaciones, son producto de que estos algoritmos son muy dependientes de los pesos iniciales de la red neuronal y que el descenso por gradiente es un método de búsqueda de soluciones óptimas locales.

En este contexto, el aprendizaje de las redes neuronales artificiales puede ser formulado como un problema de optimización.

Estando establecida la topología y funciones de activación de una rede neuronal artificial, el aprendizaje se basa en encontrar los pesos de la red que minimicen su error a través de constantes iteraciones de un conjunto de datos de entrenamiento, planteándonos optimizar la función de error que tiene como parámetros todos los pesos de la red neuronal artificial.

Por otro lado, los algoritmos evolutivos son metaheurísticas que se caracterizan por realizar una búsqueda de soluciones de optimización global de forma simultánea, basados en mecanismos de los postulados de la evolución biológica; estos algoritmos tienen inspiración biológica como, por ejemplo: algoritmo genético (GA), algoritmo del murciélago (BA) o algoritmos evolutivos de inspiración-cuántica (QIEA).

En la presente tesis se planteó hacer uso de un algoritmo evolutivo de inspiración cuántica (QIEA) para llevar a cabo el proceso de aprendizaje de una red neuronal artificial, evitando así la convergencia en óptimos locales de los métodos basados en descenso por gradiente.

En tal sentido, se da origen a la formulación de las siguientes interrogantes:

1.2 Formulación del problema

1.2.1 Problema general

¿Es posible el aprendizaje de redes neuronales artificiales basado en algoritmos evolutivos de inspiración cuántica?

1.2.2 Problemas específicos

a. ¿Es posible implementar un algoritmo evolutivo de inspiración cuántica que minimice una función objetivo?
b. ¿Es posible ajustar el algoritmo evolutivo de inspiración cuántica al proceso de aprendizaje de la red neuronal artificial?
c. ¿Es posible comparar el aprendizaje de redes neuronales artificiales basado en el algoritmo back-propagation y el aprendizaje basado en el algoritmo evolutivo de inspiración cuántica?

1.3 Justificación e importancia

La búsqueda de un computador cuántico es una lucha que se batalla en estos instantes en los laboratorios de investigación de diferentes laboratorios I+D encargados de la producción de hardware.

Si bien es cierto, la existencia de un computador cuántico supone la caída de todos los sistemas de criptografía gracias al aprovechamiento del fenómeno cuántico de la superposición que permite calcular un número infinito de operaciones en un solo paso (observación), este computador cuántico también puede ser aprovechado para la optimización.

Tanto en criptografía como en optimización existen sub-ramas que se dedican al estudio de algoritmos cuánticos mediante algoritmos de inspiración cuántica, los cuales ya han sido diseñados en parte para el aprovechamiento de esta tecnología aún no existente.

La aplicación de algoritmos de inspiración cuántica en problemas de optimización, disminuye la complejidad del problema y permite obtener en una cantidad menor de iteraciones/épocas resultados óptimos, permitiendo de esta forma disminuir el gran esfuerzo computacional que se realiza con otros algoritmos.

La presente investigación se justifica en la necesidad de superar uno de los tres problemas subyacentes a las redes neuronales artificiales, que es la búsqueda de un algoritmo que escape de óptimos locales y encuentre un óptimo global respecto a la función de error; al respecto, los algoritmos evolutivos de inspiración cuántica otorgan buenos resultados en procesos de optimización al llevar a cabo un proceso evolutivo de búsqueda.

La presente tesis ayudará a los investigadores de ciencia de la computación como referencia y fuente verificable de la aplicación de un algoritmo evolutivo de inspiración cuántica como método de aprendizaje en redes neuronales artificiales.

1.4 Alcances y limitaciones

En la investigación se pretende implementar un QIEA configurado para la minimización, para luego realizar un proceso de ajuste de dicho algoritmo al proceso de aprendizaje de una red neuronal artificial de múltiples capas; finalmente, se comparará los resultados obtenidos con dicho algoritmo con los obtenidos por un algoritmo de aprendizaje clásico como back-propagation.

Cabe recalcar que solo se abordará un problema de los tres problemas relacionados a las redes neuronales artificiales, el cual es la búsqueda de óptimos globales.

1.5 Objetivos

1.5.1 Objetivo general

Llevar a cabo el proceso de aprendizaje de neuronales artificiales basado en algoritmos evolutivos de inspiración cuántica.

1.5.2 Objetivos específicos

a. Implementar un algoritmo evolutivo de inspiración cuántica que minimice una función objetivo.
b. Ajustar el algoritmo evolutivo de inspiración cuántica al proceso de aprendizaje de la red neuronal artificial.

c. Comparar el aprendizaje de redes neuronales basado en el algoritmo back-propagation y el aprendizaje basado en el algoritmo evolutivo de inspiración cuántica.

1.6 Hipótesis

Hipótesis general

H_0: No es posible llevar a cabo el proceso de aprendizaje de neuronales artificiales basado en algoritmos evolutivos de inspiración cuántica.

H_1: Si es posible llevar a cabo el proceso de aprendizaje de neuronales artificiales basado en algoritmos evolutivos de inspiración cuántica.

Hipótesis específica 01

H_0: No posible implementar un algoritmo evolutivo de inspiración cuántica que minimice una función objetivo.

H_1: Es posible implementar un algoritmo evolutivo de inspiración cuántica que minimice una función objetivo.

Hipótesis específica 02

H_0: No posible ajustar el algoritmo evolutivo de inspiración cuántica al proceso de aprendizaje de la red neuronal artificial.

H_1: Es posible ajustar el algoritmo evolutivo de inspiración cuántica al proceso de aprendizaje de la red neuronal artificial.

Hipótesis específica 03

H_0: El aprendizaje de redes neuronales artificiales con el algoritmo evolutivo de inspiración cuántica es igual o peor que el aprendizaje con el algoritmo back-propagation.

H_1: El aprendizaje de redes neuronales artificiales con el algoritmo evolutivo de inspiración cuántica es mejor que el aprendizaje con el algoritmo back-propagation.

CAPÍTULO II

MARCO TEÓRICO

2.1 Antecedentes del estudio

A nivel internacional, existen distintas investigaciones que se avocan al estudio de algoritmos evolutivos en redes neuronales artificiales, a continuación, se presentan los consultados como base para la elaboración del estado del arte de la presente investigación.

La investigación de Cardoso (2015): "Quantum-inspired features and parameter optimization of spiking neural networks for a case study from atmospheric", utiliza un modelo híbrido de red neuronal artificial con representación binario-real para el estudio de descargas atmosféricas, arribando a que su modelo presenta una exactitud superior a las técnicas tradicionales.

La investigación de Chandwani (2014): "Modeling slump of ready mix concrete using genetic algorithms assisted training of artificial neural networks", explora la hibridización de dos técnicas de inteligencia computacional (redes neuronales artificiales y algoritmos genéticos) para el modelamiento de la

descomposición del concreto liso, arribando que su modelo mejora la velocidad de convergencia de la red neuronal artificial.

La investigación de Takahashi (2014): "Multi-layer quantum neural network controller trained by real-coded genetic algorithm", presenta la aplicación de redes neuronales cuánticas en sistemas de control en remplazo del algoritmo back-propagation, presentando una función de costo del orden de 10^-2 a diferencia del algorimto back-propagation que solo decreció hasta 1.

La investigación de Yu & Xu (2014): "A short-term load forecasting model of natural gas based on optimized genetic algorithm and improved BP neural network", propone una combinación entre un algoritmo genético de codificación real y un mejorado back-propagation para el aprendizaje de una red neuronal artificial que pueda predecir la demanda de gas natural, como resultado añade a la red neuronal artificial un factor momentum adicional que permite que su solución sea ideal a comparación de otros algoritmos diferenciales.

La investigación de Lui (2013): "Single-hidden-layer feed-forward quantum neural network based on Grover learning", presenta una red neuronal artificial de una capa de inspiración cuántica aplicada al feed-forward de la red neuronal

artificial, sus simulaciones presentan una mejora al algoritmo tradicional.

2.2 Bases teóricas

2.2.1 Inteligencia artificial

Según Rich & Knight (1994): "El estudio de cómo lograr que las computadoras realicen tareas que, por el momento, los humanos hacen mejor."

La inteligencia artificial (IA) como ciencia y tecnología se han ido acumulando conocimientos sobre como emular diversas capacidades del ser humano para exhibir comportamiento vigente y se han desarrollado sistemas cada vez más perfeccionados que producen parcialmente dichas capacidades.

La IA trata de abarcar y estudiar muchas de las capacidades del hombre para poder ofrecerlas a la máquina y también al propio hombre en el entendimiento de los principios de su inteligencia. Agentes, lógica, redes neuronales, sistemas expertos, vida artificial, tecnología del habla y síntesis, visión artificial, robótica entre otros.

2.2.2 Redes neuronales

Es una de las ramas destacadas del campo científico de la inteligencia artificial es las redes neuronales artificiales (RNAs) entendiendo como tales aquellas redes en las que existen elementos procesadores de información cuyas interacciones locales depende del comportamiento del conjunto de sistema.

Tablada & Torres (2009) define las redes neuronales como "Modelos matemáticos cuya inspiración surgió en las neuronas biológicas y la estructura paralela masiva del cerebro, y poseen la capacidad de adquirir, almacenar y utilizar conocimiento a partir de los datos".

Las neuronas se distribuyen en la red formando capas de un número determinado de elementos básico. Es decir, existe una capa de entrada que recibe directamente la información proveniente de las fuentes externas de la red, capaz ocultas que son internas a la red y no tienen contacto directo con el exterior (desde cero niveles hasta un número elevado), pudiendo estar interconectadas de distintas maneras, lo que termina junto a su número, las distintas topologías y una capa de salida que transfiere la formación de la red hacia el exterior.

La topología de las redes neuronales es la forma de organización de las neuronas en la red formando capas o agrupaciones de neuronas más o nos alejadas de la entrada y la salida de la red. Por lo tanto, los parámetros fundamentales de la red serán: el número de capas, el número neuronas por capa, el grado de conectividad y el o de conexiones entre neuronas.

2.2.3 Perceptrón multicapa

Rosenblatt (1958) define un perceptrón o perceptron como una máquina que aprende, utilizando ejemplos, asigna los vectores de entrada (muestras) a diferentes clases, utilizando una función lineal de las entradas.

Minsky & Papert (1969) describen el perceptrón como un gradiente de descenso estocástico algoritmo que intenta separar linealmente un conjunto de datos de entrenamiento n-dimensionales.

Se utiliza la palabra perceptrón en el primer sentido como una máquina, siguiendo Rosenblatt, y se refiere explícitamente al perceptrón como un algoritmo de aprendizaje.

Arbib (2003) describe el funcionamiento de un perceptrón: utiliza una regla de corrección de errores para cambiar los pesos de cada unidad que hace que las respuestas erróneas a estímulos que presentan a la red, libre de bucles que tiene sus unidades dispuestas en capas, con cada unidad aportaciones solo para unidades en la capa siguiente de la secuencia. La primera capa comprende unidades de entrada; se puede entonces ser varias capas de unidades ocultas entrenables que llevan una representación interna, y por último está la capa de unidades de salida, también con pesos sinápticos entrenables.

En la figura 1 se representa una red multicapa donde se definen tres tipos de capas: de entrada, oculta y de salida.

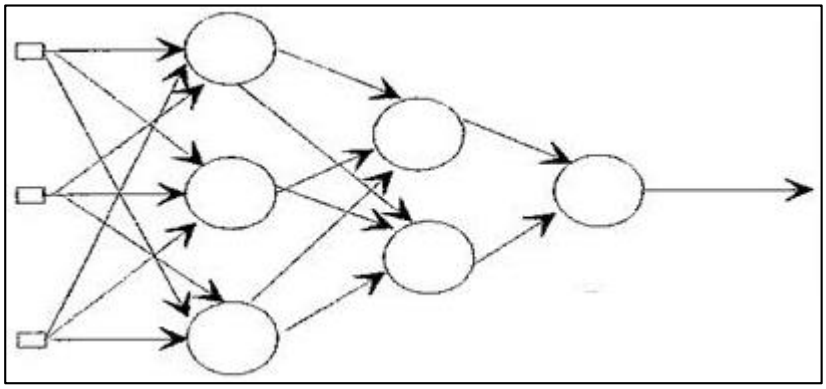

Figura 1. Red neuronal de 2 capas ocultas

Fuente: "Apuntes y transparencias de la asignatura de ingeniería neurosensorial". Intensificación de bioingeniería. ETSIT, UPM, curso 2000-2001.

Este tipo de redes se caracterizan por su facilidad de implementación. Su aprendizaje se basa en la retropropagación: se parte de unos pesos iniciales en las conexiones interneuronales.

Para un conjunto de entradas se obtiene una cierta salida, basándose en que se conoce la salida que deberíamos haber obtenido (patrón catalogado – aprendizaje supervisado), calculamos el error. A partir de este error se modifican los pesos siguiendo el sentido inverso al de evolución de la Red (se parte de la salida hasta llegar a la entrada).

De la misma manera se opera con el resto de entradas de entrenamiento. Se puede observar que el error irá disminuyendo a medida que se aplique el algoritmo.

2.2.4 Back-propagation

El algoritmo de back-propagation está basado en la regla delta generalizada, es seleccionado para entrenar la red multicapa de alimentación directa. La entrada se propaga hacia delante, desde la capa de entrada a través la capa media a la capa de salida, y luego la salida resultante se propaga hacia atrás, desde la capa de salida a través de la zona central hacia la capa de entrada, para calcular el error.

Posteriormente, los derivados de error de peso y los cambios de peso se calculan.

El entrenamiento de la red consiste en presentar repetidamente los patrones en la formación establecidas en la red hasta que el error generado en la inicial aguda de la pendiente de la curva y su sedimentación lenta indica que la red tenía ninguna dificultad en el aprendizaje de los patrones de entrada y en la formación. Se completan el número de iteraciones.

La red neuronal multicapa es la red neuronal que más se aplica, que se ha utilizado en la mayoría de las investigaciones hasta el momento. Un algoritmo de back-propagation se puede utilizar para entrenar a estas redes de alimentación directa multicapa con un diferenciable al transferir funciones para llevar a cabo la función de aproximación, la asociación de patrón y clasificación de patrones. El término back-propagation se refiere al proceso de por el cual los derivados de error en la red, con respecto a los pesos de la red y sesgos, se pueden calcular. La formación de las redes neuronales artificiales por propagación hacia atrás implica tres etapas, según Fausett (1994) son: Primera etapa, la alimentación hacia adelante del patrón de entrenamiento de entrada; segunda etapa, el cálculo y la propagación hacia atrás del error asociado; tercera etapa, el ajuste de los pesos.

2.2.5 Optimización

Pauling (1960) señala que la naturaleza busca siempre un estado óptimo. Esto es observable por ejemplo cuando los átomos intentan formar enlaces de mínima energía de sus electrones.

Por otro lado, Spencer (1960) menciona el principio biológico de sobrevivencia de los más aptos, plantea que junto a la evolución biológica, conllevan a la mejor adaptación de las especies a su ambiente. En este caso, un óptimo local es una especie que domina todas las otras a su alrededor. El homo sapiens ha alcanzado este nivel al dominar hormigas, bacterias, moscas, cucarachas y toda clase de criaturas existentes en nuestro planeta convirtiéndose en un óptimo global; mientras que, por ejemplo, el león se ha convertido en dominante entre las especies animales de la selva, convirtiéndose en un óptimo local.

Detrás de estos fenómenos existe un formalismo matemático que estudia toda esta área: optimización global, que es una rama de la matemática aplicada y análisis numérico cuyo foco es la optimización.

El objetivo de una optimización global es encontrar los mejores elementos posibles x^* de un conjunto X que siga un criterio $F = \{ f_1, f_2, \ldots, f_n \}$.

Cabe destacar que las funciones objetivo no se limitan a ser expresiones matemáticas, sino que pueden estar compuestas por ecuaciones o algoritmos complejos que para calcularse necesiten múltiples simulaciones. Así, la optimización global debe comprender todas las técnicas, analíticas y no analíticas que puedan usarse para encontrar los mejores elementos, $x \in X$ de acuerdo a la función $f \in F$. En resumidas cuentas, la optimización consiste en buscar y encontrar la solución o soluciones óptimas a un determinado problema, tomando en cuenta determinadas restricciones.

Como ejemplos podemos citar:

- Aumentar la rentabilidad de un negocio.
- Reducir gastos, multas o pérdidas.
- Aumentar la eficacia de un determinado proceso

Para poder encajar un problema cualquiera de la vida real como un problema de optimización, es necesario identificar las siguientes entidades:

1. El problema: sus características, restricciones y variables,
2. Las variables de decisión $x = x_1, \ldots, x_n$ del problema, cuyos valores influyen en la solución,

3. La función f (x) que calcula la calidad de la solución – función objetivo,

4. Un método, algoritmo o heurística de búsqueda de soluciones,

5. El espacio de soluciones válidas X discreto o continuo del problema,

6. Los recursos computacionales necesarios para: implementar el proceso, la función de evaluación (o simulación), tratamiento de características del problema y selección de la mejor solución.

2.2.6 Funciones de único objetivo

Cuando se optimiza un único criterio f(x), un óptimo es un máximo o un mínimo dependiendo de si se está maximizando o minimizando. Por ejemplo, en problemas de mundo real (de manufactura), se busca por lo general minimizar tiempos, costos o pérdidas para un determinado proceso. Por otro lado, tratándose de negocios, se busca maximizar la rentabilidad o el valor económico.

La figura 2 muestra una función f definida en un espacio 2-dimensional X con elementos $x = (x_1, x_2) \in X$. Se hace destaque en el gráfico a los óptimos locales y óptimos globales. Un óptimo global es el óptimo en todo el conjunto mientras que un óptimo local es óptimo sólo en un subespacio de X.

Máximo local: Sea una función de un objetivo f : X → R, se define un máximo local $\hat{x}_1 \in X$ como un valor de entrada que cumple f $(\hat{x}_1) \geq$ f (x) para todo x dentro de una vecindad de $\hat{x}_1$.

Mínimo local: Sea una función de un objetivo f : X → R, se define un mínimo local $\bar{x}_1 \in X$ como un valor de entrada que cumple

Óptimo local: Sea una función de un objetivo f : X → R, se define un óptimo local $\bar{x}_1^* \in X$

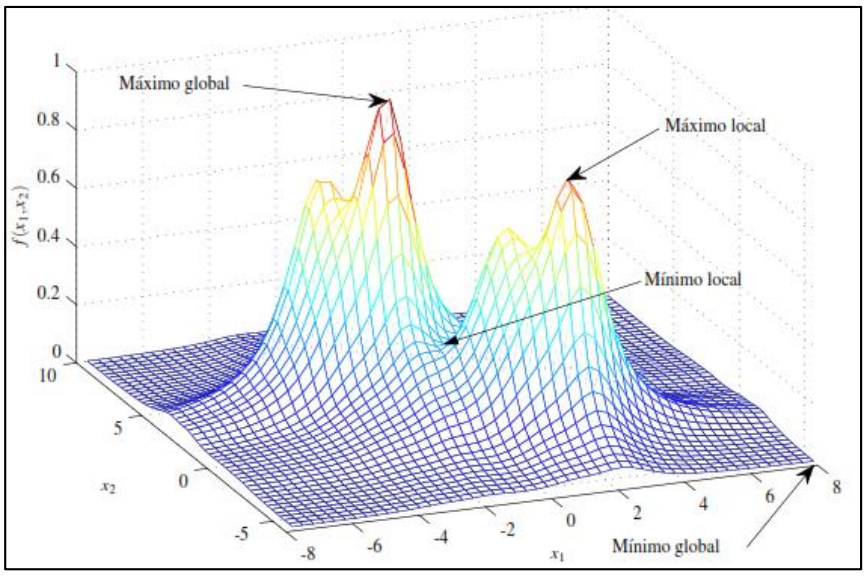

Figura 2. Ejemplo de función $f(x1, x2)$ con óptimo global y local.

Fuente: Inteligencia artificial 1a ed. - Iniciativa latinoamericana de libros de texto abiertos (LATIn), 2014

Máximo global: Sea una función de un objetivo $f: X \rightarrow R$, se define un máximo global $\hat{x} \in X$ como un valor de entrada que cumple $f(\hat{x}) \geq f(x), \forall x \in X$.

Mínimo global: Sea una función de un objetivo $f: X \rightarrow R$, se define un mínimo global $\bar{x} \in X$ como un valor de entrada que cumple $f(\bar{x}) \leq f(x), \forall x \in X$.

Óptimo global: Sea una función de un objetivo $f: X \rightarrow R$, se define un óptimo global $x* \in X$ como un valor de entrada que cumple con ser un máximo o un mínimo global.

Es común encontrar más de un máximo o mínimo globales, inclusive dentro del dominio X de una función unidimensional $f: X \subseteq R^n \rightarrow R$.

Un ejemplo de esta situación es la función coseno $\cos(x)$ donde $x = \{x\}$, que tiene valores máximos globales en $\hat{x}$ cuando $\hat{x} = 2i\pi$ y valores mínimos globales $\bar{x}$ cuando $\bar{x} = (2\pi + 1)\pi$. La solución correcta sería un conjunto $X*$ de entradas óptimas y no un mínimo o máximo aislado. Además, el significado de óptimo depende del problema:

Entonces podemos afirmar que un algoritmo de optimización tiene como tarea: Encontrar soluciones que sean lo mejor posibles para el problema, y, que, por su vez, éstas sean lo más diferentes entre sí.

2.2.7 Técnicas heurísticas de optimización

En optimización clásica (programación lineal, programación dinámica, programación no lineal, tercera derivada, etc.), se asume que los problemas a tratarse deben cumplir exigencias que garanticen la convergencia hacia el óptimo global.

Sin embargo, la mayoría de los problemas del mundo real no satisfacen estas exigencias y tampoco son problemas que pueden solucionarse en tiempo polinomial. De hecho, en muchas aplicaciones prácticas, ni siquiera es posible afirmar que existe una solución eficiente.

Cuando enfrentamos espacios de búsqueda tan grandes como el problema de la mochila (knapsack problem), y que además los algoritmos más eficientes que existen para resolver el problema requieren tiempo exponencial, resulta obvio que las técnicas clásicas de búsqueda y optimización clásicas son insuficientes. Es entonces cuando recurrimos a las heurísticas.

Heurística: La palabra heurística se deriva del griego heuriskein, que significa encontrar o descubrir.

Reeves (1993) define heurística de la siguiente forma: "Una heurística es una técnica que busca soluciones buenas (es decir, casi óptimas) a un costo computacional razonable, aunque sin garantizar factibilidad u optimalidad de las mismas. En algunos casos, ni siquiera puede determinar qué tan cerca del óptimo se encuentra una solución factible en particular."

Meta heurística: Una meta heurística es un método que se aplica para resolver problemas genéricos. Combina funciones objetivo o heurísticas de una forma eficiente y abstracta que usualmente no dependen de la estructura del problema.

2.2.8 Algoritmos evolutivos

Algoritmos evolutivos (EA) son algoritmos metaheurísticos basados en una población de individuos que emplean mecanismos biológicamente inspirados como la mutación, recombinación, selección natural y supervivencia de los más aptos para ir iterativamente ajustando o refinando un conjunto de soluciones.

Una ventaja de los algoritmos evolutivos con respecto a otros métodos de optimización es su característica de black-box, es decir tienen poco conocimiento y pocas suposiciones sobre la función objetivo a optimizar.

Inclusive, la definición de la función objetivo exige menos conocimiento de la estructura del espacio de problema que una heurística factible para el problema; adicionalmente, los algoritmos evolutivos tienen una calidad de respuesta consistente para muchas clases de problemas.

Así, cuando se trata de computación evolutiva o algoritmos evolutivos, debemos entender que existe un conjunto de técnicas y metodologías que la componen, todas ellas con inspiración biológica en la evolución neo-darwiniana.

2.2.9 Ciclo básico de un algoritmo evolutivo

Un proceso evolutivo en general se puede simular computacionalmente usando los siguientes mecanismos:

1. Una forma de codificar las soluciones x en estructuras g que se reproducirán conformando una población G0 inicial generada aleatoriamente.

2. Una función de asignación de aptitud h( ) que depende de los individuos x y su evaluación f (x).

3. Un mecanismo de selección basado en la aptitud.

4. Operaciones que actúen sobre los individuos codificados gi ∈ G para reproducirlos.

Estos mecanismos siguen un orden de ejecución como muestra la figura 3.

Aunque no es muy fácil distinguir las diferencias entre los distintos tipos de algoritmos evolutivos en la actualidad, se puede diferenciar tres principales paradigmas de la computación evolutiva.

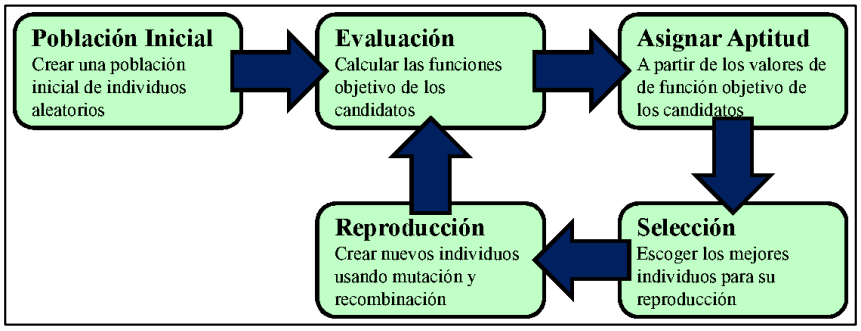

Figura 3. El ciclo básico de un algoritmo evolutivo.

Fuente: inteligencia artificial 1a ed. - Iniciativa latinoamericana de libros de texto abiertos (LATIn), 2014

2.2.10 Computación cuántica

Un computador cuántico es un dispositivo que hace uso de ciertos fenómenos de mecánica cuántica para realizar operaciones con datos. El término computación cuántica es usado para describir procesos computacionales que se basen en estos fenómenos, son capaces de disminuir el esfuerzo y la complejidad computacional para resolver determinados problemas. Según Spector (2006): "Lo principal de esta área es el poder de procesamiento y la afirmación las posibilidades cuentan, aunque no ocurran".

El bit es la menor unidad de información en un computador clásico, cuyos valores pueden ser '0' o '1'. En un computador cuántico, la unidad de información básica ser llama q-bit. Puede tener los estados '0', '1' o una sobreposición de los dos y puede ser representada:

$$|\varphi\rangle = \alpha|0\rangle + \beta|1\rangle \quad (1)$$

Donde $|\varphi\rangle$ es el estado del q-bit y α, β son números complejos que determinan las amplitudes de probabilidad de los estados correspondientes.

Y $|\alpha|^2$ indica la probabilidad del q-bit de ser encontrado en el estado "0", $|\beta|^2$ indica la probabilidad del qubit de ser encontrado en el estado "1". La normalización garantiza:

$$|\alpha|^2 + |\beta|^2 = 1 \quad (2)$$

Si hay un sistema de m q-bits, el sistema puede representar 2^m estados al mismo tiempo. Sin embargo, en la observación de un estado cuántico se limita a un sólo estado.

A pesar de la ventaja existente al usar computadores cuánticos existen dos limitaciones: la dificultad de implementar un computador cuántico y crear algoritmos que aprovechen la capacidad de estos computadores. Por otro lado, Moore plantea aprovechar los conceptos de la física cuántica en lugar de usar computadores cuánticos, para mejorar el desempeño de los algoritmos clásicos y este es el origen del término inspiración cuántica.

Moore presenta la siguiente metodología para la formulación de un algoritmo de inspiración cuántica:

1. El problema debe tener una representación numérica.
2. La configuración inicial debe ser definida (parámetros).
3. Definir una condición de parada.
4. El problema debe ser divisible en sub-problemas menores.

5. El número de universos (dominio por cada variable) debe ser identificado.

6. Cada sub-problema debe estar asociado a uno de los universos.

7. Los cálculos de los diversos universos deben ser independientes.

8. Alguna forma de interacción entre los universos debe existir.

2.2.11 QIEA-B

Han & Kim (2000) presentan el primer algoritmo evolutivo de inspiración cuántica que fue bien definido, que en la literatura original se denomina QEA (Han & Kim, 2000) pero que denominaremos QIEA-B de ahora en adelante. QIEA-B es inspirado en el concepto de computación cuántica, el modelo está diseñado con una representación q-bit, un operador de actualización Q-gate y observación. Se explica la representación y el algoritmo a continuación:

Representación

Diferentes representaciones pueden ser usadas para codificar las soluciones de los individuos de computación evolutiva. Las representaciones pueden ser clasificadas como: binaria, numérica y simbólica (Hinterding, 1999). El modelo usa una representación usando q-bits para la representación probabilística basada en el concepto de q-bit.

Un Q-bit es la unidad de información más pequeña en QIEA-B, y es definido como un par de parámetros (α, β):

$$\begin{bmatrix} \alpha \\ \beta \end{bmatrix}$$

donde se cumple $|\alpha|^2 + |\beta|^2 = 1$. $|\alpha^2|$ indica la probabilidad del q-bit de encontrarse en el estado "0", $|\beta^2|$ indica la probabilidad del q-bit de encontrase en el estado "1". Un Q-bit puede estar en el estado "0", "1", o una superposición lineal de los dos.

Individuo cuántico:

Un individuo cuántico es una cadena de m qbits, y es definido como:

$$\begin{bmatrix} \alpha_1 & \alpha_2 & \dots & \alpha_m \\ \beta_1 & \beta_2 & \dots & \beta_m \end{bmatrix}$$

donde se cumple

$$|\alpha_i|^2 + |\beta_i|^2 = 1, \forall \mathrm{i} = 1,2, \dots, \mathrm{m} \qquad (3)$$

Con esta definición, se logra que cada individuo quántico represente una superposición de individuos clásicos formados por m Q-bits(genes).

Operador Q-gate

La actualización de la población es realizada por el operador Q-gate, es

definida como una matriz de rotación.

$$U(\Delta\theta_i) = \begin{bmatrix} cos(\Delta\theta_i) & -\sin(\Delta\theta_i) \\ sin(\Delta\theta_i) & \cos(\Delta\theta_i) \end{bmatrix} \quad (4)$$

$U(\Delta\theta_i)$ multiplica cada columna del individuo, cada par de valores α y β, son tratadas como un vector de 2 dimensiones y serán operados usando $U(\Delta\theta_i)$, como se muestra en la figura siguiente.

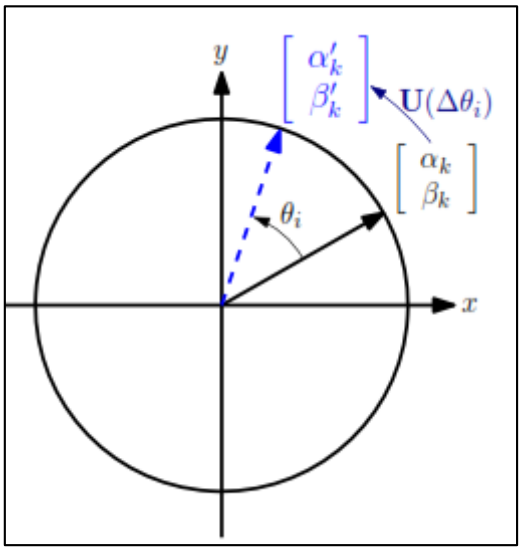

Figura 4. Rotación de un q-bit usando Q-gate

Fuente: da Cruz A. V. (2007)

La actualización de este individuo es realizada por cada gen, de acuerdo con la ecuación:

$$\begin{bmatrix} \alpha'_i \\ \beta'_i \end{bmatrix} = U(\Delta\theta_i) \begin{bmatrix} \alpha_i \\ \beta_i \end{bmatrix} \quad (5)$$

$\Delta\theta_i$ es definido de acuerdo al problema que se desee resolver, modificando α y β para dirigir a los individuos a soluciones más óptimas (Han & Kim, 2000)

El procedimiento del QIEA-B propuesto por (Han & Kim, 2000) es descrito a continuación:

Tabla 1

Algoritmo evolutivo de inspiración cuántica de codificación binaria

```
1: t ← 0
2: Inicializar Q(t)
3: Generar Pt observando los estados de Qt
4: Evaluar Pt
5: Guardar las mejores soluciones de Pt en bt
6: while condición de parada no sea alcanzada do
        7: t ← t + 1
        8: Generar Pt observando los estados de Qt-1
        9: Evaluar Pt
        10: Guardar la mejor solución de Pt en bt
        11: bt ← mejor entre bt y bt-1
        12: Actualizar Qt usando Q-gates
13: end while
```

Fuente: (Han & Kim, 2000)

Dónde: Q_t es la población, P_t es una población de individuos clásicos generada en cada iteración. En la inicialización de Q_t, sus individuos son inicializados con $\alpha_i y\ \beta_i\ con\ \frac{1}{\sqrt{2}}\ \forall i = 1, 2, \ldots, m$, para tener la misma probabilidad de generar los estados "0" o "1" y b_t es usado para guardar los mejores

en cada iteración.

El proceso del modelo es el siguiente:

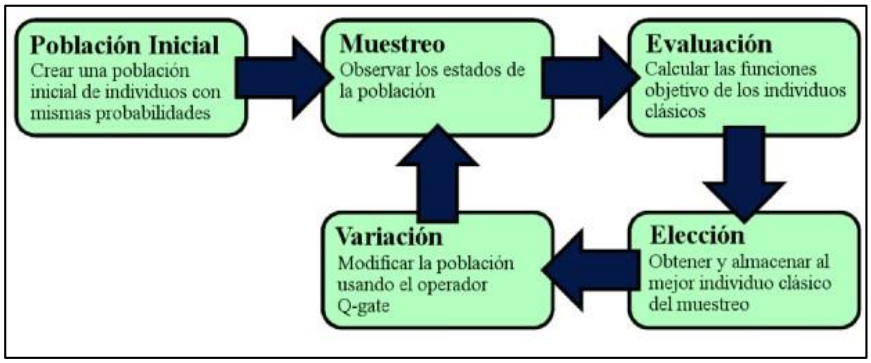

Figura 5. Modelo QIEA-B

Fuente: da Cruz (2007)

2.2.12 QIEA-R

El modelo QIEA-R que utiliza codificación real, fue propuesto por da Cruz (2007) y fue la primera en utilizar representación real en lugar de representación binaria. Tiene los siguientes pasos: generación de población cuántica, generación de población clásica al observar la población cuántica y actualización de la población cuántica.

Tabla 2

Algoritmo evolutivo de inspiración cuántica de codificación real

```
1: t ← 1
2: Generar población cuántica Q(t) con N individuos con G genes.
3: while t ≤ T do
        4: E(t) ← generar individuos clásicos observando individuos cuánticos
        5: if t = 1 then
                6: C(t) ← E(t)
        7: else
                8: E(t) ← recombinacion entre E(t) y C(t)
                9: evaluar E(t)
                10: C(t) ← K mejores individuos de [E(t) U C(t)]
        11: end if
        12: Q(t+1) ← actualiza Q(t) usando N mejores individuos de C(t)
        13: t ← t + 1
14: end while
```

Fuente: (da Cruz A. V., 2007)

Representación cuántica

El individuo cuántico representa la superposición de los posibles estados. En este modelo, el conjunto de estados observables es continuo.

Sean $\{\mathbf{q}_1, ..., \mathbf{q}_m\}$ los individuos cuánticos de la población $\mathbf{Q}_t$ en la generación t, cada individuo cuántico $\mathbf{q}_i$ está compuesto por n genes, $\boldsymbol{q}_{ij} = \{q_{i1}, \ldots, q_{in}\}$.

Cada gen q_{ij} está definido por una función de distribución de probabilidad $p_{ij}(x)$. Esta función de distribución de probabilidad es inicializada cubriendo todo el dominio de q_{ij}

Basado en esta definición, un individuo cuántico puede ser representado como:

$$\boldsymbol{q}_i = \{p_{i1}(x), p_{i2}(x), \ldots, p_{in}(x)\} \quad (6)$$

La función de distribución de probabilidad utilizada en por da Cruz (2010) es la función de densidad de probabilidad uniforme. Esta función se define por la ecuación:

$$p_{ij}(x) = \begin{cases} \dfrac{1}{U_{ij} - L_{ij}} & si \;\; L_{ij} \leq x \leq U_{ij} \\ 0 & otro \end{cases} \quad (7)$$

Donde L_{ij} y U_{ij} son los limites inferior y superior respectivamente del intervalo que el i-ésimo individuo cuántico puede asumir en la dimensión j cuando es observado.

Un ejemplo de un gen cuántico formado por un pulso cuadrado es mostrado a continuación:

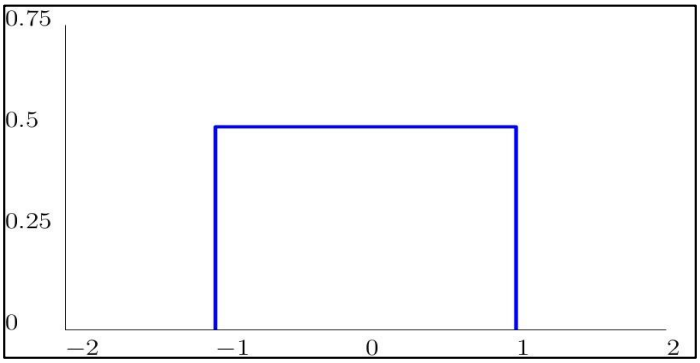

Figura 6. Ejemplo de un gen cuántico en QIEA-R, donde el eje x es el dominio de la variable y el eje y es la probabilidad

Fuente: da Cruz (2007)

En este caso los limites están definidos por -1 y 1.

da Cruz (2007) propone dos formas de representación:

- Crear pulsos cuadrados con un ancho $\frac{U_{ij}-L_{ij}}{N}$, con sus centros distribuidos uniformemente a lo largo de todo el dominio, donde N es el número de individuos cuánticos.
- Crear los pulsos con límite superior e inferior iguales a los límites del dominio.

En la siguiente figura se pueden apreciar ejemplos de ambas opciones, considerando una población de tamaño 5, 2 dimensiones y un dominio de -100 a

100 por cada dimensión:

$$Q(0) = \begin{bmatrix} q_1 = [(\mu = -80, \sigma = 40); (\mu = -80, \sigma = 40)] \\ q_2 = [(\mu = -40, \sigma = 40); (\mu = -40, \sigma = 40)] \\ q_3 = [(\mu = 0, \sigma = 40); (\mu = 0, \sigma = 40)] \\ q_4 = [(\mu = 40, \sigma = 40); (\mu = 40, \sigma = 40)] \\ q_5 = [(\mu = 80, \sigma = 40); (\mu = 80, \sigma = 40)] \end{bmatrix} \quad Q(0) = \begin{bmatrix} q_1 = [(\mu = 0, \sigma = 200); (\mu = 0, \sigma = 200)] \\ q_2 = [(\mu = 0, \sigma = 200); (\mu = 0, \sigma = 200)] \\ q_3 = [(\mu = 0, \sigma = 200); (\mu = 0, \sigma = 200)] \\ q_4 = [(\mu = 0, \sigma = 200); (\mu = 0, \sigma = 200)] \\ q_5 = [(\mu = 0, \sigma = 200); (\mu = 0, \sigma = 200)] \end{bmatrix}$$

Figura 7. Propuestas de inicialización

Fuente: da Cruz (2007)

Una vez inicializada la población cuántica Q_0 (población inicial cuántica), el ciclo principal del proceso evolutivo empieza y continúa de la siguiente forma:

Observación

La generación de individuos clásicos observando individuos cuánticos usando la función de distribución de probabilidad $p_{ij}(x)$, probabilidades acumuladas P_{ij} y un generador de números aleatorios **U**(0,1) en el dominio de 0 a 1 siguiendo el procedimiento presentado a continuación:

1: **Generar** r ~ **U**(0,1)

2: **Encontrar** x tal que

$$p_{ij}(x) = \int_{-\infty}^{\infty} p_{ij}(r)\, dr, x = p_{ij}^{-1}(r) \tag{8}$$

3: **Asignar** $x_{ij}(t) \in \mathbf{x}_i \leftarrow x$

Si se usan pulsos (μ_{ij} , σ_{ij}) y una realización $r_{ij} \sim U(0,1)$, el valor del gen clásico se obtiene por:

$$x_{ij} = r_{ij} * \sigma_{ij} + (\mu_{ij} - \frac{\sigma_{ij}}{2}) \quad (9)$$

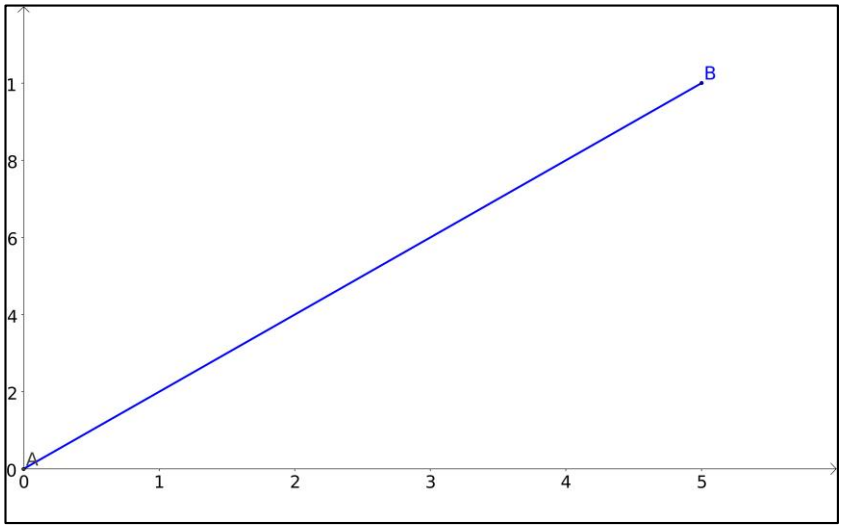

Figura 8. Ejemplo de función de distribución acumulada con μ = 2,5 y σ = 2,5 donde el eje x es el dominio de la variable y el eje y es la probabilidad de la función acumulada

Fuente: da Cruz (2007)

Actualización

La actualización tiene dos etapas: ajuste del ancho del pulso y movimiento del centro del pulso.

Ajuste ancho del pulso

QIEA-R puede reducir o incrementar el espacio de búsqueda de cualquier individuo cuántico de acuerdo a la aptitud de la población clásica durante el proceso de observación y usando la regla de 1/5 de Rechenberg (1973). Si menos del 20% de la población clásica de la actual generación tiene una aptitud mejor que la generación anterior, el ancho del gen es reducido, si es mayor que 20%, el ancho es incrementado y si es igual, el ancho no es modificado, la ecuación, muestra la regla de 1/5.

$$\sigma_{ij} = \begin{cases} \sigma_{ij} * \delta & \varphi < 1/5 \\ \sigma_{ij}/\delta & \varphi > 1/5 \\ \sigma_{ij} & \varphi = 1/5 \end{cases} \quad (10)$$

donde φ representa el porcentaje de mejores individuos de la población en comparación a la población anterior.

Movimiento del centro del pulso

Escoger los mejores individuos clásicos para actualizar la población cuántica. Por ejemplo, suponer que el centro del gen cuántico en la generación t es $\mu_{ij}(t)$ y el valor del mejor gen clásico esx_{ij} entonces la nueva posición del gen cuántico es calculada para la generación t + 1 por la ecuación:

$$\mu_{ij}(t + 1) = \mu_{ij}(t) + \lambda(x_{ij} - \mu_{ij}(t)) \quad (11)$$

donde λ ∈ [0,1], es el porcentaje de movimiento en dirección al gen clásico.

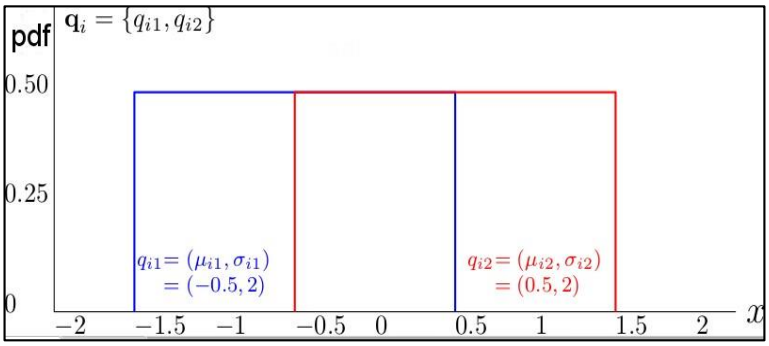

Figura 9. Ejemplo de movimiento de gen cuántico en una dimensión donde el eje x es el dominio de la variable y el eje y es la probabilidad.

Fuente: da Cruz (2007)

2.3 Definición conceptual de términos

Aprendizaje:

Proceso por el cual los parámetros libres de una red neuronal artificial son ajustados a través de un proceso continuo de estimulación por parte del entorno donde se sitúa el sistema.

Back-forward:

Proceso en el que se propagan los errores desde la capa de salida de una red neuronal hasta los pesos conectados a las entradas para el cálculo de derivadas parciales o "contribución al error".

Bias

Una intersección o desplazamiento desde un origen. El sesgo (también conocido como el término de sesgo) se denomina b o w0 en los modelos de aprendizaje automático.

Convergencia

Informalmente, a menudo se refiere a un estado alcanzado durante el entrenamiento en el que la pérdida de entrenamiento y la pérdida de validación cambian muy poco o nada con cada iteración después de un cierto número de iteraciones. En otras palabras, un modelo alcanza la convergencia cuando la capacitación adicional sobre los datos actuales no mejorará el modelo.

Época

Un pase de entrenamiento completo en todo el conjunto de datos de modo que cada ejemplo se haya visto una vez. Por lo tanto, una época representa N/iteraciones de entrenamiento donde N es el número total de instancias.

Feed-forward:

Pre alimentación, describe un tipo de sistema que reacciona a los cambios

en su entorno, normalmente para mantener algún estado concreto del sistema.

Función de activación:

Una función que toma la suma ponderada de todas las entradas de la capa anterior y luego genera y pasa un valor de salida (normalmente no lineal) a la siguiente capa.

Descenso de gradiente

Una técnica para minimizar las pérdidas mediante el cálculo de los gradientes de pérdida con respecto a los parámetros del modelo, condicionado a los datos de entrenamiento.

Modelo de clasificación:

Un tipo de modelo de aprendizaje automático para distinguir entre dos o más clases discretas. Por ejemplo, un modelo de clasificación de procesamiento de lenguaje natural podría determinar si una oración de entrada estaba en francés, español o italiano.

Normalización

El proceso de convertir un rango real de valores en un rango estándar de valores, típicamente -1 a +1 o 0 a 1. Por ejemplo, supongamos que el rango natural de una determinada función es de 800 a 6.000. Mediante la resta y la división, puede normalizar esos valores en el rango de -1 a +1.

Overfitting

Crear un modelo que coincida con los datos de entrenamiento tan de cerca que el modelo no pueda realizar las predicciones correctas en los datos nuevos.

CAPÍTULO III

MARCO METODOLÓGICO

3.1 Tipo y diseño de la investigación

El tipo de investigación es básica (también llamada investigación pura o fundamental) puesto que se tiene como finalidad la obtención y recopilación de información para ir construyendo una base de conocimiento que se va agregando a la información previa existente.

El diseño de investigación es no experimental porque no se manipulará la variable independiente y solo se observarán los cambios ocurridos.

El nivel del diseño de investigación es descriptivo porque se pretende realizar una evaluación para determinar la eficacia de la variable independiente en la dependiente.

3.2 Población y muestra

La población está compuesta por el conjunto de datos “Human activity

recognition using smartphones dataset" de los autores Anguita, Ghio, Oneto, Parra, & Reyes-Ortiz (2013).

Los experimentos se llevaron a cabo con un grupo de 30 voluntarios dentro de un intervalo de edad de 19 – 48 años.

Donde cada persona con un smartphone Samsung Galaxy S II en la cintura realiza 6 actividades: caminar (walking), subir escaleras (walcking upstairs), bajar escaleras (walcking downstairs), sentarse (sitting), pararse (standing) y echarse (laying).

Los autores del data-set usaron el acelerómetro y giroscopio que está incorporado en el smartphone, el cual capturó la aceleración lineal de 3 ejes y velocidad angular de 3 ejes a una velocidad constante de 50 Hz.

Los experimentos han sido grabados en vídeo para etiquetar los datos manualmente. Se obtiene un conjunto de datos ha sido al azar donde lo divide en dos conjuntos, el 70% de los voluntarios fue seleccionado para la generación de los datos de entrenamiento y el 30% de los datos de prueba.

Para cada instancia se proporciona:

- Triaxial aceleración del acelerómetro (total de la aceleración) y la estimación de la aceleración del cuerpo.
- Triaxial de la velocidad angular del giróscopo.
- Información de dominio de tiempo y frecuencia de variables.
- Etiqueta de Actividad.

El conjunto de datos incluye los siguientes archivos:

- 'train/X_train.txt': conjunto de entrenamiento.
- 'train/y_train.txt': la formación de las etiquetas.
- 'test/X_test.txt': conjunto de pruebas.
- 'test/y_test.txt': las etiquetas de prueba.

La data se encuentra normalizada y acotada en [-1,1].

Un vídeo del experimento como un ejemplo de las 6 actividades registradas con uno de los participantes puede ser visto en el siguiente enlace:

http://www.youtube.com/watch?v=XOEN9W05_4A.

3.3 Operacionalización de las variables

Algoritmo evolutivo de inspiración cuántica

Definición conceptual: El algoritmo evolutivo de inspiración cuántica es un nuevo algoritmo evolutivo para una computadora clásica centrado en la optimización de diversas funciones con cualidad de búsqueda estocástica.

Definición operacional: Es el algoritmo a utilizarse en el proceso de aprendizaje de la red neuronal artificial.

Aprendizaje de redes neuronales artificiales

Definición conceptual: El aprendizaje es el proceso por el cual una red neuronal artificial modifica sus pesos en respuesta a una información de entrada. Los cambios que se producen durante el proceso de aprendizaje son la destrucción, modificación y creación de conexiones entre las neuronas.

Definición operacional: Es el proceso de optimización de los pesos de la red neuronal artificial.

Dimensiones:

1. Error.

2. Convergencia (iteraciones/épocas).

3. Tiempo

Indicadores:

1. Error.

    1.1 Error medio cuadrático en validación.

    1.2 Error máximo en validación.

    1.3 Error mínimo en validación.

    1.4 Error promedio en validación.

2. Convergencia (iteraciones/épocas).

    2.1 Número de iteraciones/épocas.

    2.1 Número de iteraciones/épocas.

3. Tiempo

    3.1 Tiempo de ejecución en entrenamiento

3.4 Técnicas e instrumentos para la recolección de datos

Los datos que constituyen la población y muestra de la investigación fueros obtenidos del repositorio de Machine Learning de la UC Irvine.

Las técnicas empleadas en la presente investigación constituyen la implementación de algoritmos y pruebas hechas en computador, siendo este último el instrumento primordial para la recolección de datos durante la investigación.

3.5 Procesamiento y análisis de datos

El procesamiento de datos se hará mediante el computador a través de los algoritmos planteados, el análisis de datos se hará mediante la presentación de gráficos debido al gran volumen de datos que muestran las simulaciones.

CAPÍTULO IV

RESULTADOS Y DISCUSIÓN

4.1 Resultados

4.1.1 Implementación del algoritmo evolutivo de inspiración cuántica

De los QIEA presentados en el marco teórico, se elige el QIEA de codificación real (QIEA-R) puesto que la estructura de dato necesaria para los individuos de la población es más pequeña que la de codificación real.

Sin embargo, del planteamiento de da Cruz (2007) se desprenden algunos problemas:

1. La actualización del ancho de pulso puede llegar a exceder el dominio de determinada variable del problema, el valor aleatorio asociado puede afectar significativamente el espacio de búsqueda.
2. La actualización del centro de pulso solo se realiza en base al mejor individuo clásico, eliminándose la variedad de población.
3. Se presentan 2 estrategias explorativas (recombinación y modificación de ancho y centro de pulso) y ninguna explotativa.

Para lo cual, se plantea modificar ajuste del ancho y centro de pulso de la siguiente forma:

Modificación del ancho de pulso de forma explotativa

Reducir o incrementar el espacio de búsqueda de cualquier individuo cuántico de acuerdo a la aptitud de la población clásica durante el proceso de observación y usando una modificación de la regla de 1/5 (Rechenberg, 1973). Si menos del 20% de la población clásica de la actual generación tiene una aptitud mejor que la generación anterior, el ancho del gen es aumentado, si es mayor que 20%, el ancho es reducido y si es igual, el ancho no es modificado, la ecuación, muestra la regla de 1/5.

$$\sigma_{ij} = \begin{cases} \sigma_{ij} * \delta & \varphi > 1/5 \\ \sigma_{ij}/\delta & \varphi < 1/5 \\ \sigma_{ij} & \varphi = 1/5 \end{cases} \tag{12}$$

donde φ es un parámetro cuyo valor recomendado es $\delta = 0{,}82$, concluida esta modificación se verifica si el ancho de pulso es mayor al dominio, y de ser así este se verá reducido al dominio.

da Cruz (2007) no menciona específicamente como considerar que la aptitud de la población es mejor/igual/peor que la anterior, por lo que para nuestra

implementación, ordenamos los vectores y los comparamos mediante un contador por posición verificando la cantidad de individuos observados que son mejores que los de la anterior generación.

Modificación del centro de pulso de forma explotativa

Escoger los mejores individuos clásicos para actualizar la población cuántica. Por ejemplo, suponer que el centro del gen cuántico en la generación t es $\mu_{ij}(t)$ y el valor del gen clásico de la generación anterior es x_{ij} entonces la nueva posición del gen cuántico es calculada para la generación t + 1 por la ecuación:

$$\mu_{ij}(t+1) = \mu_{ij}(t) + \lambda(x_{ij} - \mu_{ij}(t)) \quad (13)$$

donde λ es una variable del algoritmo cuyo valor recomendado es $\lambda = 0{,}5$.

Ante lo cual, se presenta a continuación el algoritmo modificado.

Tabla 3

Algoritmo evolutivo de inspiración cuántica de codificación real modificado

```
1: t ← 1
2: Generar población cuántica Q(t) con N individuos con G genes.
3: while criterio_parada do
        4: E(t) ← generar individuos clásicos observando individuos cuánticos
        5: if t = 1 then
                6: C(t) ← E(t)
        7: else
                8: E(t) ← recombinación entre E(t) y C(t)
                9: Ordenar y evaluar E(t)
                10: C(t) ← K mejores individuos de [E(t) U C(t)]
        11: end if
        12: Q(t+1) ← actualiza Q(t) usando N mejores individuos de C(t)
                13: Modificar el ancho de pulso según λ
                14: Modificar el centro de pulso según δ
        15: t ← t + 1
16: end while
```

Fuente: Elaboración propia

Para verificar las mejoras realizadas al algoritmo, se realizaron pruebas con la función de benchmark F6 de Schaffer:

$$f(x) = 0{,}5 + \frac{(sin\sqrt{\sum_{i=0}^{d} x_i^2})^2 - 0{,}5}{[1 + 0{,}001(\sum_{i=0}^{d} x_i^2)]^2} \tag{14}$$

Donde *d* representa la dimensionalidad del problema y el óptimo global se encuentra en x* = {0,0,...,0,0}

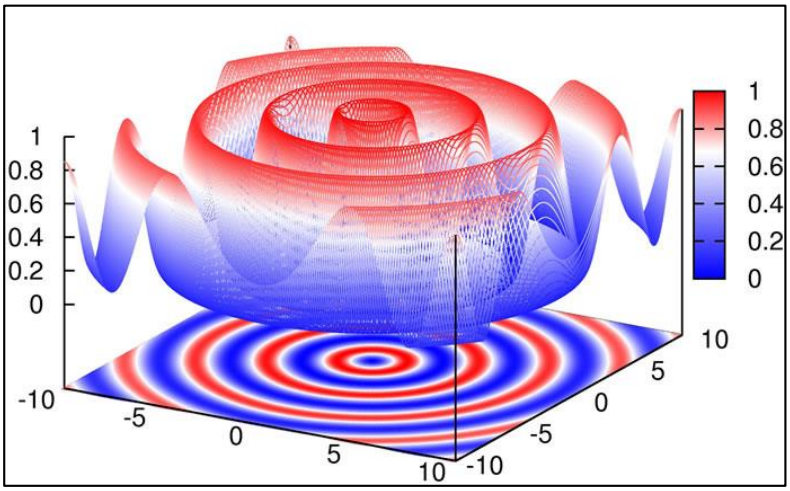

Figura 10. Función de benchmark de Schaffer's F6 en 3D

Fuente: Elaboración propia.

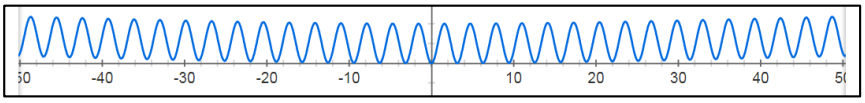

Figura 11. Función de benchmark de Schaffer's F6 en 2D

Fuente: Elaboración propia.

De donde se obtuvieron los siguientes resultados:

Tabla 4

Comparación de QIEAR vs QIEAR-modificado en F6 Schaffer

	QIEA-R	**QIEAR-R modificado**
Óptimo Global	0,00915454	1,66533e-16
Ancho de Pulso	0,01548537	3,45423e-12
Centro de Pulso	0,00891121	8,52469e-13

Fuente: Elaboración propia.

La implementación del algoritmo se adjunta en el Anexo Nº 1.

4.1.2 Ajustar del algoritmo evolutivo de inspiración cuántica al proceso de aprendizaje de la red neuronal artificial.

Modelo de aprendizaje de la red neuronal artificial.

Primeramente, se presenta al Perceptrón junto a sus ecuaciones.

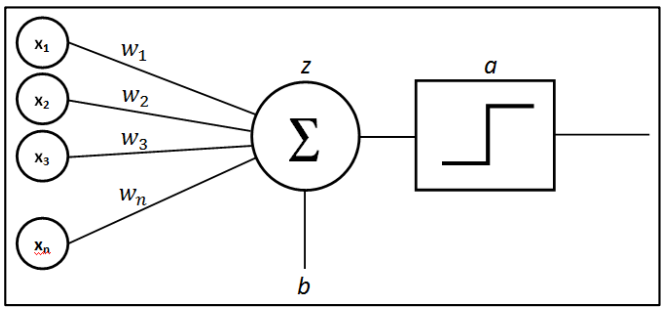

Figura 12. Perceptrón

Fuente: Elaboración propia

Donde:

$$z = \sum_{i=0}^{n}(w_i * e_i) + b, a = act(z) \quad (15)$$

Estas unidades (neuronas) organizadas por capas son llamadas Perceptrón Multicapa:

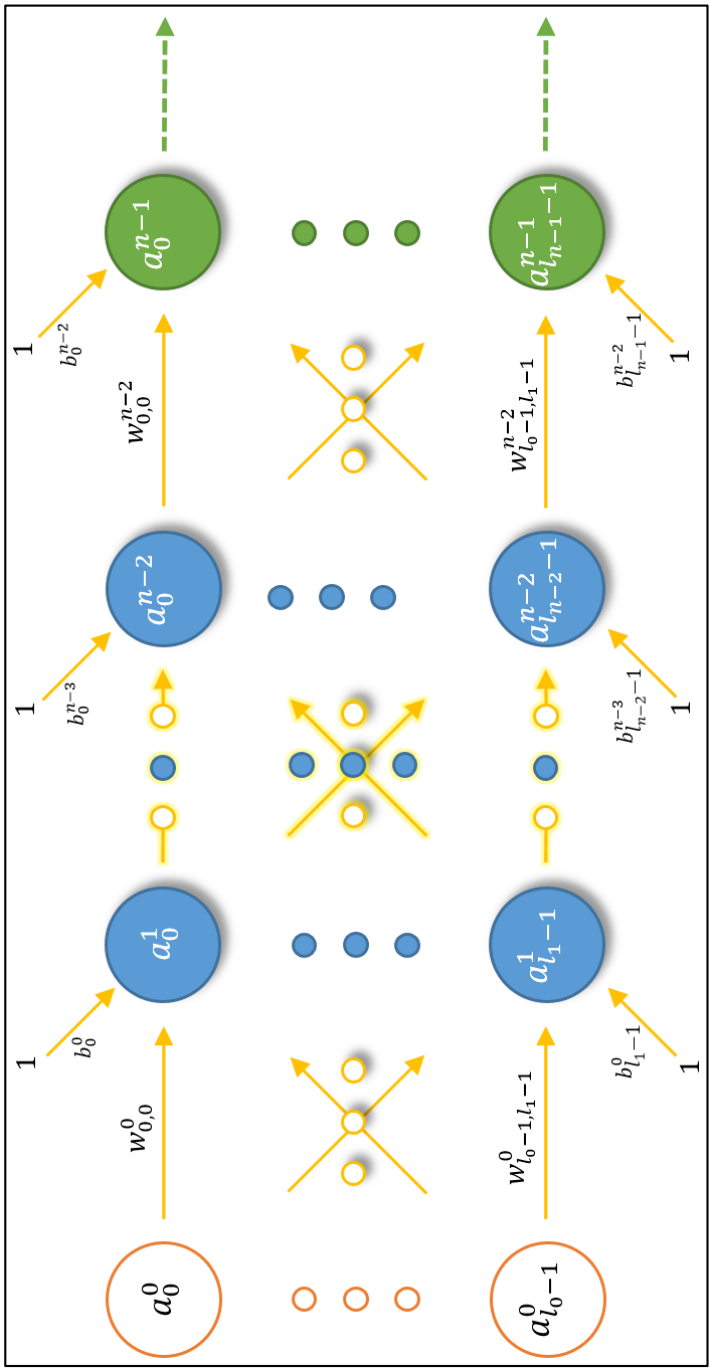

Figura 13. Modelo QIEA-B

Fuente: Elaboración propia

Donde:

⋮ Abreviación de un número de neuronas.

… Abreviación de un número de conexiones

l_i Número de neuronas en la capa i

Función de activación

El Perceptrón multicapa está compuesto por una función de activación, que para nuestro modelo es global y se define como:

$$act(z) = \frac{1}{1 + e^{-z}} \tag{16}$$

Cálculo del error

El cálculo del error se da en base a la ecuación del error.

$$Error = \frac{1}{2}\sum_{i=0}^{ns}(t_i - a_i^2)^2 \tag{17}$$

Feed-forward (propagación hacia adelante)

Teniendo la red neuronal con los pesos adecuados, el cálculo de las salidas (y por consiguiente de cada neurona intermedia) se hace mediante la siguiente

ecuación:

$$a_{neurona}^{capa+1} = act\left(b_{neurona}^{capa} + \sum_{i=0}^{l_{capa}} w_{i,neurona}^{capa} \cdot a_i^{capa}\right) \quad (18)$$

para valores de capa de $0 \rightarrow n-2$

Back-propagation

La retropropagación se basa en una regla de ajuste, que trata de determinar las conexiones sinápticas de manera que las salidas de red coincidan con las salidas deseadas o por lo menos, sean lo más próximas posibles. Utiliza el método de descenso de gradiente para ajustar de la red. La diferencia entre el perceptrón simple es que el descenso de gradiente de la multicapa varía entre las R dimensiones que tiene la red.

Habiendo inicializado los pesos de la red neuronal de forma aleatoria, la modificación de dichos pesos se hace mediante la siguiente fórmula:

$$w_{neurona}^{capa} = w_{neurona}^{capa} - \propto \frac{\partial \varepsilon}{\partial w} \quad (19)$$

Donde el proceso de cálculo de las derivadas parciales del error respecto a los *bias* nos facilita el proceso, tal que:

$$\frac{\partial \varepsilon}{\partial b_{neurona}^{n-2}} = \sum_{i=0}^{l_{n-1}} \frac{\partial \varepsilon}{\partial a_i^{n-1}} \cdot \frac{\partial a_i^{n-1}}{\partial z} \tag{20}$$

$$\frac{\partial \varepsilon}{\partial b_{neurona}^{capa}} = \left(\sum_{i=0}^{l_{capa+2}} w_{neurona,i}^{capa+1} \cdot \frac{\partial \varepsilon}{\partial b_i^{capa+1}} \cdot \right) * \frac{\partial a_{neurona}^{capa+1}}{\partial z} \tag{21}$$

Para *capa* de $n - 3 \rightarrow 0$

Teniendo como derivadas a:

$$\frac{\partial E}{\partial a_j^2} = \left(a_j^2 - t_j\right), \frac{\partial a}{\partial z} = a(1 - a) \tag{22}$$

Y la actualización de pesos se lleva a cabo mediante las siguientes fórmulas

$$b_{neurona}^{capa} - = \propto \frac{\partial \varepsilon}{\partial b_{neurona}^{capa}} \tag{23}$$

$$w_{neurona-atras,neurona}^{capa} - = \propto \frac{\partial \varepsilon}{\partial b_{neurona}^{capa}} \cdot \ a_{neurona-atras}^{capa} \tag{24}$$

Para *capa* de $0 \rightarrow \ n - 2$

Para el data-set a utilizarse, fue necesaria la elección de una topología, esta topología es de 2 capas.

La implementación de una red neuronal de n-capas se adjunta en el Anexo Nº 2.

Las salidas de la red neuronal obedecen a al parámetro de clasificación en redes neuronales, de acuerdo a la siguiente tabla:

Tabla 5

Saldas de la red neuronal para clasificación

Salida	Salida 1	Salida 2	Salida 3	Salida 4	Salida 5	Salida6
1	**1**	0	0	0	0	0
2	0	**1**	0	0	0	0
3	0	0	**1**	0	0	0
4	0	0	0	**1**	0	0
5	0	0	0	0	**1**	0
6	0	0	0	0	0	**1**

Fuente: Elaboración propia

Donde los valores de salida representan a cada una de las clases:

1.- Caminar (walking)

2.- Subir escaleras (walcking upstairs)

3.- Bajar (walcking Downstairs).

4.- Sentarse (sitting)

5.- Pararse (standing)

6.- Echarse (laying)

El ajuste entre el QIEA-R y la red neuronal se muestra a continuación:

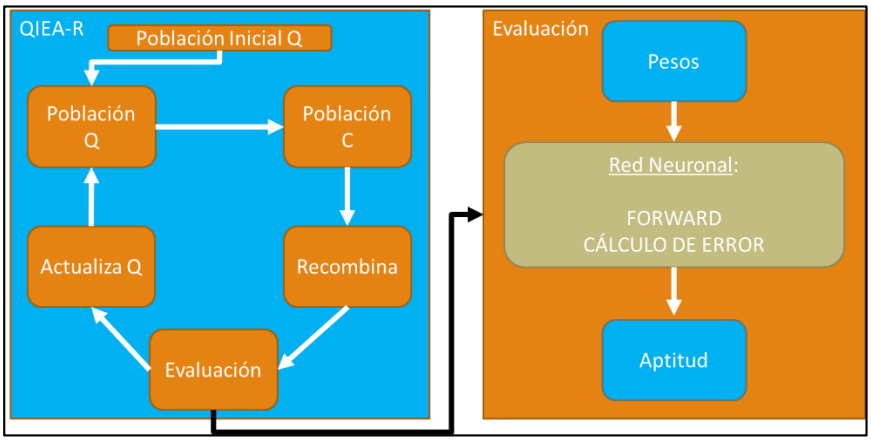

Figura 14. Modelo QIEA-B

Fuente: Elaboración propia

4.1.2 Comparar el aprendizaje de redes neuronales basados en el algoritmo evolutivo de inspiración cuántica con el algoritmo back-propagation.

La comparación entre los algoritmos QIEA-R y back-propagation (BP) en el proceso de aprendizaje de redes neuronales, fue realizado bajo los siguientes parámetros:

Tabla 6

Parámetros de algoritmos QIEA-R y BP para red neuronal

Algoritmo	α	error	T_POB	P_CRO	δ	λ
BP	0,5	**0,01**				
QIEA-R		**0,01**	50	0,8	0,82	0,5

Fuente: Elaboración propia

Donde para el QIEA-R la acotación es desde [-1,1].

Entrenamiento de la red neuronal

Se llevó a cabo el proceso de entrenamiento con topologías con 2 capas ocultas desde 60 hasta 200 neuronas por capa en un aumento de 10 en 10 hasta llegar a un error de 0,01 (1%) para ambos algoritmos (back-propagation y QIEA-R).

El número total de pesos por topología es dado por la siguiente fórmula:

$$total_{pesos} = in * capa_1 + capa_1 * capa_2 + capa_2 * out \quad (25)$$

Donde in representa el número de entradas (561), $capa_i$ representa el nro de neuronas en la capa *i* y *out* representa el número de salidas (6).

Tabla 7

Número de pesos por topología

Topología	Nº de Pesos	561:40:90:6	26716	561:60:80:6	39086
561:30:30:6	17976	561:40:100:6	27186	561:60:90:6	39756
561:30:40:6	18346	561:50:30:6	29816	561:60:100:6	40426
561:30:50:6	18716	561:50:40:6	30386	561:70:30:6	41656
561:30:60:6	19086	561:50:50:6	30956	561:70:40:6	42426
561:30:70:6	19456	561:50:60:6	31526	561:70:50:6	43196
561:30:80:6	19826	561:50:70:6	32096	561:70:60:6	43966
561:30:90:6	20196	561:50:80:6	32666	561:70:70:6	44736
561:30:100:1	20061	561:50:90:6	33236	561:70:80:6	45506
561:40:30:6	23896	561:50:100:6	33806	561:70:90:6	46276
561:40:40:6	24366	561:60:30:6	35736	561:70:100:6	47046
561:40:50:6	24836	561:60:40:6	36406	561:80:30:6	47576
561:40:60:6	25306	561:60:50:6	37076	561:80:40:6	48446
561:40:70:6	25776	561:60:60:6	37746	561:80:50:6	49316
561:40:80:1	25841	561:60:70:6	38416	561:80:60:6	50186

561:80:70:6	51056	561:90:60:6	56406	561:100:50:6	61556
561:80:80:6	51926	561:90:70:6	57376	561:100:60:6	62626
561:80:90:6	52796	561:90:80:6	58346	561:100:70:6	63696
561:80:100:6	53666	561:90:90:6	59316	561:100:80:6	64766
561:90:30:6	53496	561:90:100:6	60286	561:100:90:6	65836
561:90:40:6	54466	561:100:30:6	59416	561:100:100:6	66906
561:90:50:6	55436	561:100:40:6	60486		

Fuente: Elaboración propia

La siguiente figura muestra la cantidad de pesos por Topología:

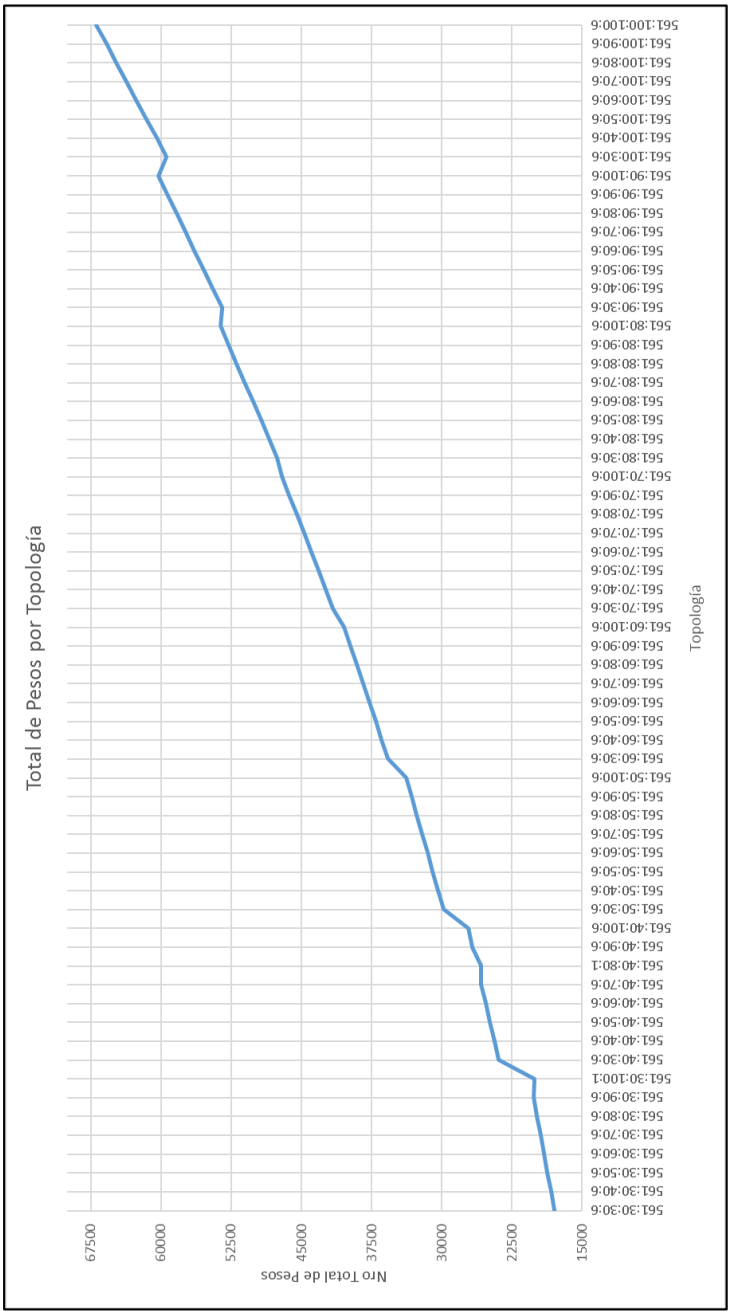

Figura 15. Número de pesos por topología

Fuente: Elaboración propia

El tiempo de ejecución de entrenamiento por algoritmo se muestra a continuación:

Tabla 8

Tiempo de ejecución por topología

Topología	**QIEA**	**BP**	**Mejora**
561:30:30:6	1,50	2,48	40%
561:30:40:6	1,56	2,60	40%
561:30:50:6	1,60	2,69	41%
561:30:60:6	1,69	2,76	39%
561:30:70:6	1,77	2,92	39%
561:30:80:6	1,82	2,94	38%
561:30:90:6	1,92	2,97	35%
561:30:100:1	1,95	3,11	37%
561:40:30:6	2,02	3,20	37%
561:40:40:6	2,08	3,27	36%
561:40:50:6	2,12	3,46	39%
561:40:60:6	2,21	3,61	39%
561:40:70:6	2,29	3,69	38%
561:40:80:1	2,30	3,78	39%
561:40:90:6	2,35	3,82	38%
561:40:100:6	2,35	3,93	40%
561:50:30:6	2,42	4,02	40%
561:50:40:6	2,46	4,10	40%
561:50:50:6	2,49	4,19	41%
561:50:60:6	2,51	4,31	42%
561:50:70:6	2,53	4,36	42%
561:50:80:6	2,55	4,39	42%
561:50:90:6	2,64	4,54	42%
561:50:100:6	2,72	4,57	40%
561:60:30:6	2,77	4,70	41%
561:60:40:6	2,81	4,80	41%
561:60:50:6	2,81	4,92	43%
561:60:60:6	2,85	4,94	42%
561:60:70:6	2,91	5,00	42%
561:60:80:6	2,93	5,14	43%
561:60:90:6	2,97	5,18	43%
561:60:100:6	3,04	5,33	43%
561:70:30:6	3,04	5,50	45%
561:70:40:6	3,07	5,63	45%
561:70:50:6	3,10	5,80	47%
561:70:60:6	3,16	5,90	46%
561:70:70:6	3,17	6,01	47%
561:70:80:6	3,24	6,16	47%
561:70:90:6	3,31	6,25	47%
561:70:100:6	3,31	6,27	47%
561:80:30:6	3,39	6,34	47%
561:80:40:6	3,43	6,40	46%
561:80:50:6	3,49	6,50	46%
561:80:60:6	3,52	6,65	47%
561:80:70:6	3,59	6,68	46%
561:80:80:6	3,69	6,77	45%
561:80:90:6	3,72	6,90	46%
561:80:100:6	3,73	6,97	46%
561:90:30:6	3,73	7,13	48%
561:90:40:6	3,74	7,26	48%
561:90:50:6	3,84	7,43	48%
561:90:60:6	3,91	7,58	48%
561:90:70:6	3,92	7,69	49%
561:90:80:6	3,99	7,78	49%
561:90:90:6	4,09	7,84	48%
561:90:100:6	4,11	7,97	48%
561:100:30:6	4,13	8,05	49%
561:100:40:6	4,17	8,21	49%
561:100:50:6	4,25	8,38	49%
561:100:60:6	4,30	8,52	50%
561:100:70:6	4,39	8,64	49%
561:100:80:6	4,40	8,73	50%
561:100:90:6	4,40	8,79	50%

561:100:100:6	4,43	8,94	50%	**Total**	**194,68**	**355,38**	**45%**

Fuente: Elaboración propia

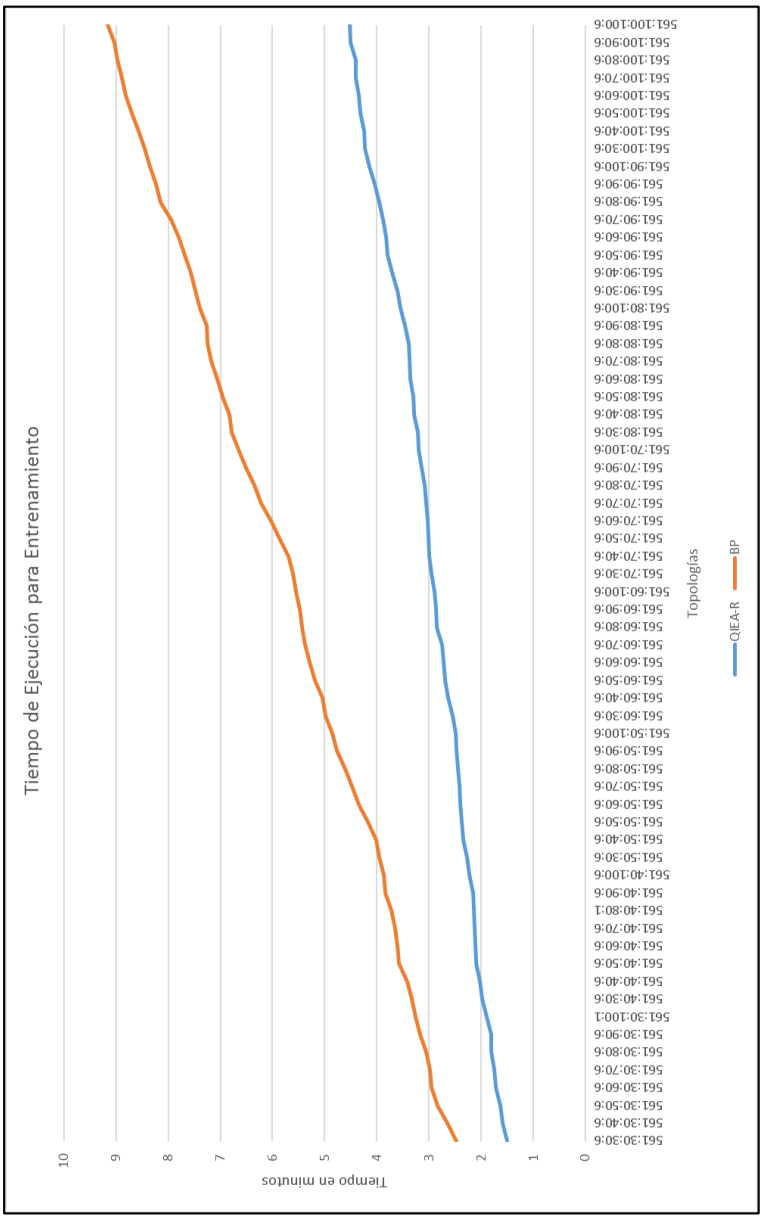

Figura 16. Tiempo de ejecución de entrenamiento

Fuente: Elaboración propia

A su vez, se consideró repetir los experimentos si es que no se lograba el objetivo (por caer en un mínimo local mayor al error mínimo), el número de experimentos por algoritmo necesarios para entrenar la red neuronal se muestra a continuación:

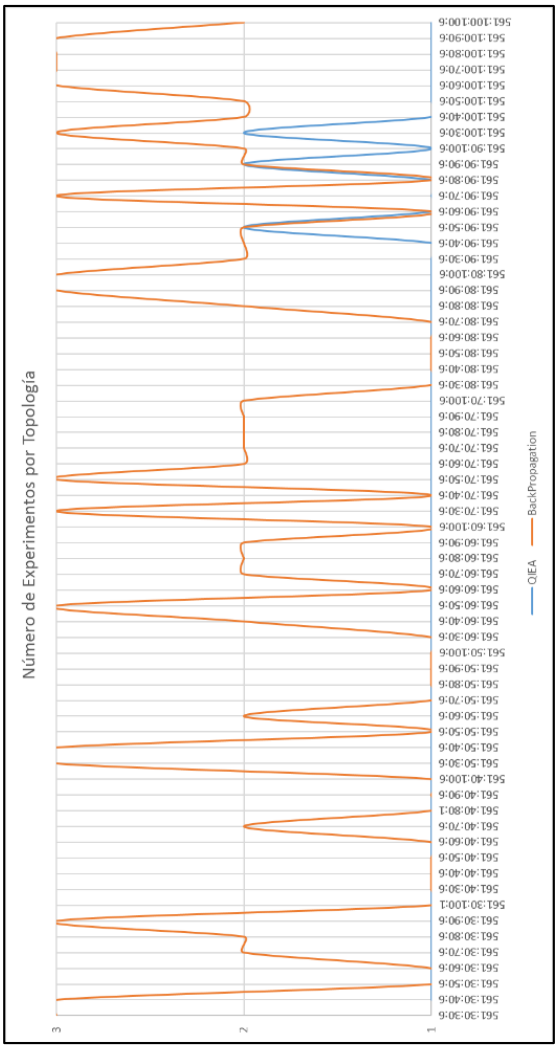

Figura 17. Número de experimentos para llegar al mínimo error

Fuente: Elaboración propia

De donde se puede denotar que el QIEA-R solo no arribó al error mínimo en 03 ocasiones, dificultades que se atribuyen a la dimensionalidad del problema; por otro lado, debido a la exigencia de un error menor al 1%, en varias ocasiones el algoritmo back-propagation tuvo que ejecutarse más de una vez, atribuimos dicho problema a que el algoritmo se quedó atrapado en un mínimo local.

El número de iteraciones que demora cada uno de los algoritmos nos indica la capacidad de convergencia de ambos, y el gráfico nos hará denotar si esta convergencia sigue un determinado patrón o corresponde más a un patrón estocástico, a continuación de muestran la tabla y gráfica de iteraciones por topología:

Tabla 9

Número de iteraciones/generaciones por topología

Topologia	BP	QIEA-R	Mejora
561:30:30:6	33988	13418	61%
561:30:40:6	33328	13542	59%
561:30:50:6	34006	13690	60%
561:30:60:6	33086	13809	58%
561:30:70:6	33892	13925	59%
561:30:80:6	34521	14073	59%
561:30:90:6	35114	14199	60%
561:30:100:1	35672	14309	60%
561:40:30:6	34984	14445	59%
561:40:40:6	35770	14549	59%
561:40:50:6	36569	14686	60%
561:40:60:6	37210	14795	60%
561:40:70:6	36637	14945	59%
561:40:80:1	35929	15083	58%
561:40:90:6	36863	15191	59%
561:40:100:6	37855	15311	60%
561:50:30:6	36973	15441	58%
561:50:40:6	37794	15566	59%
561:50:50:6	36916	15683	58%
561:50:60:6	36362	15818	56%
561:50:70:6	35403	15963	55%
561:50:80:6	34448	16106	53%
561:50:90:6	33510	16232	52%
561:50:100:6	32652	16375	50%
561:60:30:6	33271	16500	50%
561:60:40:6	32535	16603	49%
561:60:50:6	31698	16739	47%
561:60:60:6	32415	16839	48%
561:60:70:6	31664	16951	46%

561:60:80:6	30690	17098	44%	561:80:100:6	33312	19272	42%
561:60:90:6	32333	17233	47%	561:90:30:6	33958	19404	43%
561:60:100:6	35783	17340	52%	561:90:40:6	32987	19520	41%
561:70:30:6	36203	17466	52%	561:90:50:6	33694	19639	42%
561:70:40:6	38467	17615	54%	561:90:60:6	34272	19786	42%
561:70:50:6	39854	17759	55%	561:90:70:6	34975	19931	43%
561:70:60:6	37281	17861	52%	561:90:80:6	35536	20074	44%
561:70:70:6	32041	17967	44%	561:90:90:6	34538	20184	42%
561:70:80:6	30982	18096	42%	561:90:100:6	35095	20325	42%
561:70:90:6	30003	18216	39%	561:100:30:6	34158	20443	40%
561:70:100:6	29313	18334	37%	561:100:40:6	33326	20576	38%
561:80:30:6	28445	18463	35%	561:100:50:6	34128	20678	39%
561:80:40:6	29387	18581	37%	561:100:60:6	34972	20819	40%
561:80:50:6	30241	18689	38%	561:100:70:6	35575	20935	41%
561:80:60:6	30823	18789	39%	561:100:80:6	36083	21048	42%
561:80:70:6	31701	18932	40%	561:100:90:6	36903	21157	43%
561:80:80:6	32257	19059	41%	561:100:100:6	37560	21293	43%
561:80:90:6	32809	19172	42%	**Promedio**	**34230**	**17383**	**49%**

Fuente: Elaboración propia

De donde podemos ver que la mejora en iteraciones/generaciones entre ambos algoritmos es del 49% en promedio, siendo la mayor de 61% y la menor de 35%.

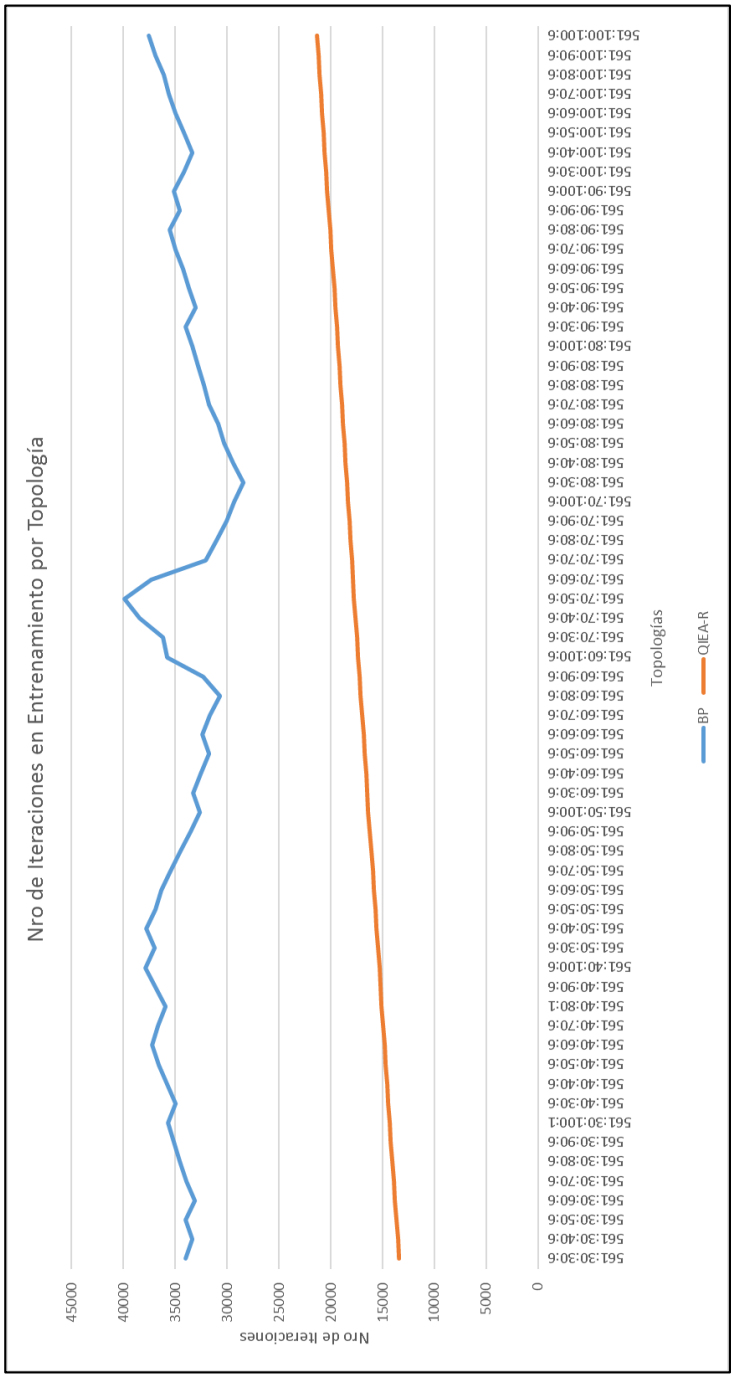

Figura 18. Iteraciones por topología para cada algoritmo

Fuente: Elaboración propia

Validación de red neuronal

La validación de la red neuronal se realiza tomando los pesos del entrenamiento, haciendo forward en la red neuronal y calculando los errores para el data-set de validación, la siguiente figura muestra las medidas del error (error total, promedio, mínimo y máximo) por cada topología.

Tabla 10

Medidas de error en validación

VALORES	SUMA_ERROR	PROM_ERROR	MIN_ERROR	MAX_ERROR
561:30:30:6	0,030759632	0,001708868	1,84274E-06	0,008086402
561:30:40:6	0,027261455	0,001514525	4,17187E-06	0,005271616
561:30:50:6	0,040901692	0,002272316	7,03764E-09	0,012958272
561:30:60:6	0,041679763	0,002315542	7,81582E-06	0,016275415
561:30:70:6	0,045083502	0,002504639	4,55201E-05	0,014753358
561:30:80:6	0,028463198	0,001581289	3,61217E-07	0,008159021
561:30:90:6	0,050476829	0,002804268	1,04422E-05	0,019022682
561:30:100:1	0,025539638	0,001418869	1,64115E-08	0,007646292
561:40:30:6	0,035714738	0,001984152	1,07275E-06	0,012326419
561:40:40:6	0,027906323	0,001550351	9,70788E-08	0,006852301
561:40:50:6	0,029609535	0,001644974	2,55118E-05	0,006868855
561:40:60:6	0,027891175	0,00154951	2,6566E-06	0,005071202
561:40:70:6	0,03732136	0,002073409	8,01076E-06	0,009937815
561:40:80:1	0,034054501	0,001891917	5,02138E-05	0,011099644
561:40:90:6	0,033443589	0,001857977	6,59499E-05	0,009160083
561:40:100:6	0,038439111	0,002135506	2,86606E-05	0,013587502
561:50:30:6	0,033111976	0,001839554	9,72878E-05	0,00987413
561:50:40:6	0,029208175	0,001622676	1,81393E-05	0,010971748
561:50:50:6	0,034521927	0,001917885	1,41526E-06	0,013550424
561:50:60:6	0,030594915	0,001699717	1,10553E-05	0,009860416
561:50:70:6	0,032248858	0,001791603	6,83645E-06	0,009014045
561:50:80:6	0,020999075	0,001166615	1,84041E-08	0,006652066
561:50:90:6	0,024845388	0,001380299	6,42288E-08	0,006188815

561:50:100:6	0,032721984	0,001817888	2,86933E-05	0,011026451
561:60:30:6	0,038725856	0,002151436	6,32025E-08	0,01148627
561:60:40:6	0,040087362	0,002227076	1,99048E-05	0,01640971
561:60:50:6	0,020403493	0,001133527	5,87716E-09	0,005099932
561:60:60:6	0,036284917	0,002015829	5,17762E-06	0,016620706
561:60:70:6	0,01950732	0,00108374	3,12737E-06	0,003619669
561:60:80:6	0,021169053	0,001176059	2,19106E-07	0,004336348
561:60:90:6	0,037565087	0,002086949	2,97313E-05	0,009217964
561:60:100:6	0,029111039	0,00161728	4,16504E-06	0,008097098
561:70:30:6	0,026647219	0,001480401	6,70504E-06	0,005121441
561:70:40:6	**0,015772821**	**0,000876268**	**1,95787E-06**	**0,003839181**
561:70:50:6	0,022367967	0,001242665	6,34939E-05	0,004073858
561:70:60:6	0,036557102	0,00203095	1,44834E-05	0,009277141
561:70:70:6	0,047329854	0,002629436	8,59076E-07	0,013388574
561:70:80:6	0,020255165	0,001125287	2,19492E-06	0,005212911
561:70:90:6	0,03001515	0,001667508	3,12593E-06	0,01096941
561:70:100:6	0,03141598	0,001745332	5,07229E-06	0,01348285
561:80:30:6	0,030735784	0,001707544	8,50306E-07	0,011522902
561:80:40:6	0,018278365	0,001015465	2,94054E-05	0,003656088
561:80:50:6	0,028197209	0,001566512	1,59783E-06	0,006012367
561:80:60:6	0,023796243	0,001322014	3,5964E-06	0,004622495
561:80:70:6	0,017908457	0,000994914	7,25764E-07	0,003438688
561:80:80:6	0,022117822	0,001228768	4,26782E-08	0,006960906
561:80:90:6	0,028082905	0,001560161	3,23157E-05	0,00832334
561:80:100:6	0,025719408	0,001428856	4,39096E-06	0,010805026
561:90:30:6	0,019361891	0,001075661	7,55547E-08	0,003937498
561:90:40:6	0,024170165	0,001342787	4,92751E-05	0,005110642
561:90:50:6	0,020381114	0,001132284	1,73712E-07	0,004316256
561:90:60:6	0,023310945	0,001295053	1,06193E-05	0,006037453
561:90:70:6	0,018773986	0,001042999	2,68632E-07	0,003810382
561:90:80:6	0,027028074	0,00150156	1,024E-08	0,007278842
561:90:90:6	0,030025571	0,001668087	2,0447E-05	0,010259273
561:90:100:6	0,037160995	0,0020645	1,33997E-05	0,012162308
561:100:30:6	0,031797179	0,00176651	1,50995E-05	0,00832686
561:100:40:6	0,022475644	0,001248647	7,68428E-06	0,003259227
561:100:50:6	0,032904949	0,001828053	4,13832E-07	0,010019377
561:100:60:6	0,032996109	0,001833117	3,2433E-06	0,009321803
561:100:70:6	0,033063607	0,001836867	5,1653E-06	0,007794761
561:100:80:6	0,028580421	0,001587801	3,2169E-07	0,006379767

561:100:90:6	0,031530868	0,001751715	2,49549E-05	0,008488033
561:100:100:6	0,02701775	0,001500986	8,06812E-05	0,008225555

Fuente: Elaboración propia

A continuación, se muestra el gráfico correspondiente en el que se ha añadido el error total de validación de la red neuronal entrenada con el algoritmo back-propagation (los pesos originados por back-propagation); gráfico del que concluimos que la topología 561:70:40:6 fue la que menor error total presentó, además, existe una diferencia entre las redes neuronales entrenadas mediante backprogation y el algoritmo evolutivo de inspiración cuántica, sin embargo, esta es pequeña y no muy significativo.

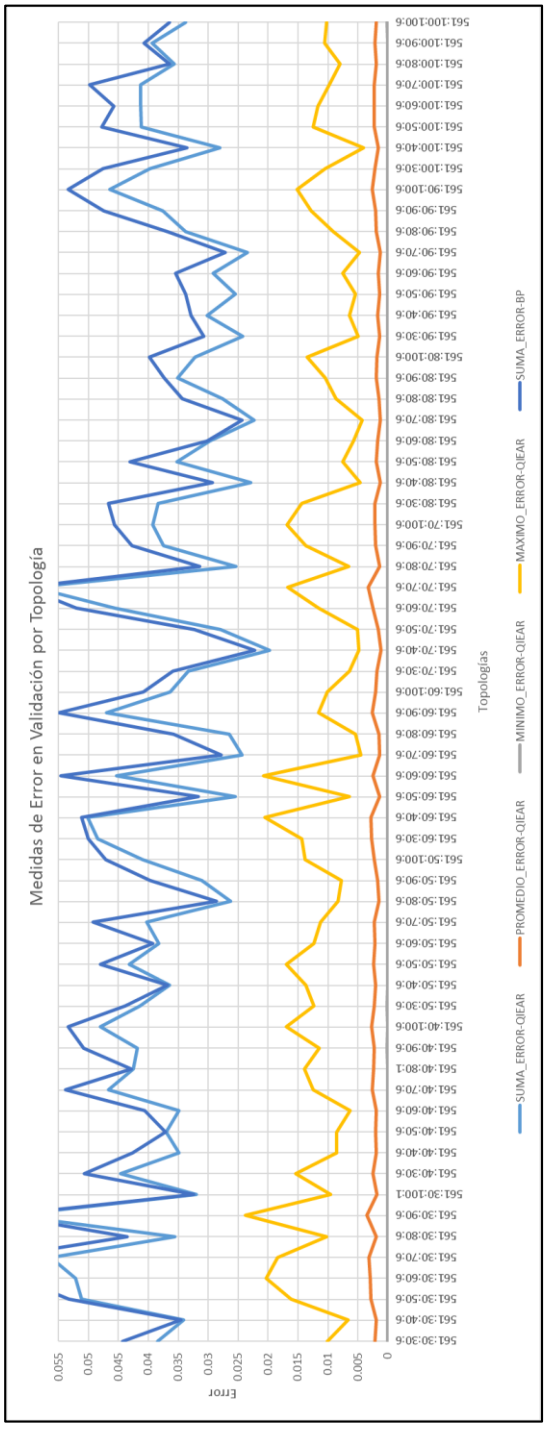

Figura 19. Medidas de error por topología con QIEA-R

Fuente: Elaboración propia

A continuación, se presentan los datos de la topología con menor error

Tabla 11

Valores en mejor topología con QIEA

Topología	**561:70:40:6**
Suma de Error	0,015772821
Promedio Error	0,000876268
Máximo Error	0,003839181
Mínimo Error	1,95787E-06

Fuente: Elaboración propia

La comparación del entrenamiento mediante backprogation y el algoritmo evolutivo de inspiración cuántica en la topología de menor error en validación se presenta a continuación:

Tabla 12

Comparación de iteraciones/generaciones en entrenamiento con topología 561:70:40

	QIEA-R	**Back-propagation**
Iteraciones	17615	38467
Error	0,015772821	0,017758379

Fuente: Elaboración propia

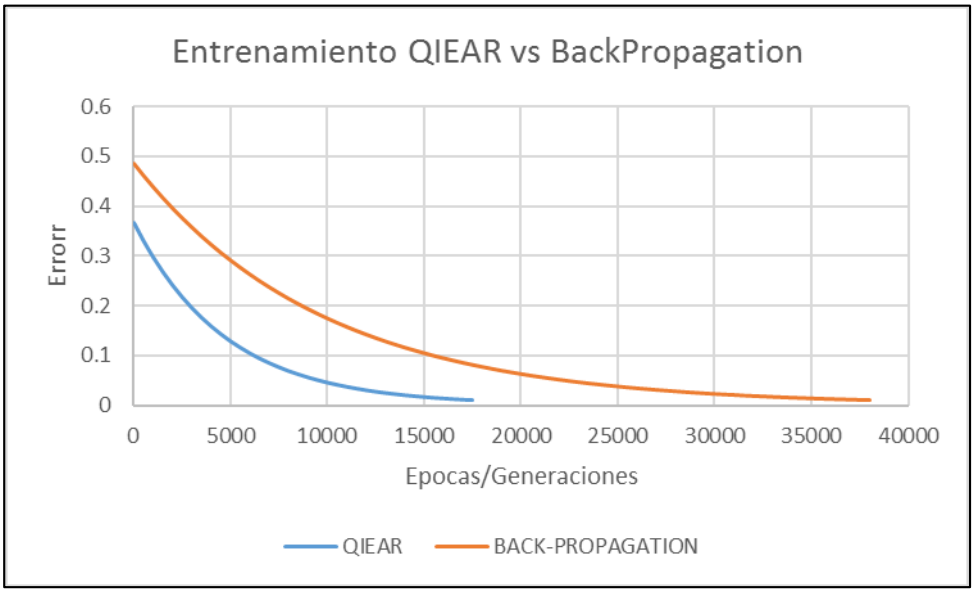

Figura 20. Entrenamiento con topología 561:70:40

Fuente: Elaboración propia

4.2 Discusión

La investigación de Cardoso (2015): "Quantum-inspired features and parameter optimization of spiking neural networks for a case study from atmospheric" utiliza un modelo híbrido de red neuronal artificial con representación binario-real para el estudio de descargas atmosféricas, arribando a que su modelo presenta una exactitud superior a las técnicas tradicionales, en la presente investigación no se obtuvo una gran diferencia en validación de resultados, sin embargo, la convergencia fue 55% más rápida y tiempos fueron 46% menores.

La investigación de Chandwani (2014): "Modeling slump of ready mix concrete using genetic algorithms assisted training of Artificial Neural Networks" de los autores (Vinay Chandwani, 2014), explora la hibridización de dos técnicas de inteligencia computacional (redes neuronales artificiales y algoritmos genéticos) para el modelamiento de la descomposición del concreto liso, arribando que su modelo mejora la velocidad de convergencia de la red neuronal artificial; en nuestra investigación arribamos a la conclusión que la velocidad de convergencia disminuye a la mitad con el algoritmo evolutivo cuántico.

La investigación de Takahashi (2014): Multi-layer quantum neural network controller trained by real-coded genetic algorithm de los autores, presenta la aplicación de redes neuronales cuánticas en sistemas de control en remplazo del algoritmo back-propagation, presentando una función de costo del orden de 10^-2 a diferencia del algoritmo back-propagation que solo decreció hasta 1; en la presente investigación se obtuvo un error de 6x10^3.

La investigación de Yu (2014) "A short-term load forecasting model of natural gas based on optimized genetic algorithm and improved BP neural network", propone una combinación entre un algoritmo genético de codificación

real y un mejorado back-propagation para el aprendizaje de una red neuronal artificial que pueda predecir la demanda de gas natural, como resultado añade a la red neuronal artificial un factor momentum adicional que permite que su solución sea ideal a comparación de otros algoritmos diferenciales; en la presente investigación fue necesaria realizar una modificación al planteamiento de da Cruz (2007) para obtener mejores resultados.

La investigación de Liu (2013): "Single-hidden-layer feed-forward quantum neural network based on Grover learning", presenta una red neuronal artificial de una capa de inspiración cuántica aplicada al feed-forward de la red neuronal artificial, sus simulaciones presentan una mejora al algoritmo tradicional; en la presente investigación se aplica el algoritmo evolutivo de inspiración cuántica para el aprendizaje de una red neuronal de 2 capas, obteniendo buenos resultados.

CONCLUSIONES

Se logró implementar con éxito un algoritmo evolutivo de inspiración cuántica que minimice una función objetivo, siendo este de codificación real para evitar problemas de dimensionalidad y coste computacional; fue necesario realizar algunas modificaciones en dicho algoritmo para lograr una mejor convergencia y no quedar atrapados en mínimos locales.

Se logró ajustar el algoritmo evolutivo de inspiración cuántica al proceso de aprendizaje de la red neuronal artificial, para lo cual fue necesario implementar una red neuronal artificial y modificarla para recibir los pesos del algoritmo evolutivo de inspiración cuántica.

Se comparó el aprendizaje de redes neuronales artificiales con el algoritmo evolutivo de inspiración cuántica y el algoritmo back-propagation, obteniendo que el primero mejora el tiempo en un 46% y la convergencia en un 55%.

RECOMENDACIONES

La modificación presentada respecto al ancho y centro de pulso mejora el rendimiento del algoritmo dentro del modelo de función presentado, sin embargo, queda abierta la posibilidad de realizar pruebas de esta modificación con otros problemas de optimización, especialmente con los problemas de múltiples objetivos.

Los parámetros del algoritmo evolutivo de inspiración cuántica fueron elegidos por pura experimentación, se deja abierta la posibilidad de optimizar los parámetros del algoritmo en base al propio algoritmo.

El algoritmo evolutivo de inspiración cuántica es altamente paralelizable, sin embargo, su paralelización escapa las limitaciones de la presente investigación, esto permitiría obtener resultados en un tiempo menor.

REFERENCIAS BIBLIOGRAFÍCAS

Anguita, D., Ghio, A., Oneto, L., Parra, J., & Reyes-Ortiz, J. L. (2013). Un Dominio Público del conjunto de datos para la Actividad Humana de Reconocimiento de Uso de los Smartphones. *21 Simposio Europeo sobre Redes Neuronales Artificiales, Inteligencia Computacional y el Aprendizaje de Máquina, ESANN de 2013.* Brujas, Bélgica. *https://archive.ics.uci.edu/ml/datasets/Human+Activity+Recognition+Using+Smartphones*

Arbib, M. (2003). *The Handbook of brain theory and neural networks.* Massachusetts: MIT Press Cambridge.

Cardoso & Silvia. (2015). Quantum-Inspired Features and Parameter Optimization of Spiking Neural Networks for a Case Study from Atmospheric. *Procedia COmputer Science*, Vol. 53, Pg. 74-81.

Chandwani & Agrawal. (2015). Modeling slump of ready mix concrete using genetic algorithms assisted training of Artificial Neural Networks. *Expert Systems with Applications*, 885-893.

Cheng-Yi Liu, C. C.-T.-M. (2013). Single-hidden-layer feed-forward quantum neural network based on Grover learning . *Neural Networks*, accepted

manuscript.

da Cruz, A. V. (2007). *Algoritmos Evolutivos com Inspiração Quântica para. Problemas com Representação Numérica.* Rio de Janeiro: Departamento de Engenharia.

da Cruz, A., Vellasco, M., & al, e. (2010). Quantum-inspired evolutionary algorithms applied to numerical optimization problems. (IEEE, Ed.) *In Evolutionary Computation (CEC) IEEE Congress*, 1-6.

Fausett, L. (1994). *Fundamentals of neuroal networks, architectures, algorithms and applications*.

Feng Yu, X. X. (2014). A short-term load forecasting model of natural gas based on optimized genetic algorithm and improved BP neural network . *Applied Energy*, 102-113.

Han, K.-H., & Kim, J.-H. (2000). Genetic quantum algorithm and its application to combinatorial optimization problem. (IEEE, Ed.) *Evolutionary Computation, 2000. Proceedings of the 2000 Congress on, 2*, 1354-1360.

Hinterding, R. (1999). Representation, constraint satisfaction and the knapsack problem. *Evolutionary Computation, 2*, 1292.

Kazuhiko Takahashi, M. K. (2014). Multi-layer quantum neural network controller trained by real-coded geneticalgorithm. *Neurocomputing*, 159-164.

Kurokawa, T. &. (2014). Multi-layer quantum neural network controller trained by real-coded genetic algorithm. *Neurocomputing*, 159-164.

Liu & Chen. (2013). Single-hidden-layer feed-forward quantum neural network based on Grover learning. *Neural Networks*, 144-150.

Marcelo C. Cardoso1, M. S. (2015). Quantum-Inspired Features and Parameter Optimization of Spiking Neural Networks for a Case Study from Atmospheric. *Procedia Computer Science*, 74-81.

Misky & Papert. (1987). *Perceptrons: An Introduction to Computational Geometry, Expanded Edition.* Massachusetts: MIT Press.

Mohd & Muhammad . (2017). Optimal Parameter Selection Using Three-term Back-propagation Algorithm for Data Classification. *International Journal on Advanced Science, Engineering and Information Technology*.

Pauling, L. (1960). *The Nature of the Chemical Bond.* New York.

Pino, R., Gómez, A., & de Abajo, N. (2001). *Introducción a la ingeniería Artificial: Sistemas Expertos, Redes Neuronales Artificiales y Computación Evolutiva.* Oviedo, España: Servicios de Publicaciones Universidad de Oviedo. Recuperado el 2017 de Julio de 18, de https://docs.google.com/file/d/0By5TADaAJqrLd2R0UmRZU05JQlE/edit?hl=es&forcehl=1

Prechelt, L. (1994). A Set of Neural Network Benchmark Problems and Benchmarking Rules. *Universit□at Karlsruhe*.

Rechenberg, I. (1973). *Evolutionsstrategie: Optimierung technischer systeme nach prinzipien der biologischen evolution.*

Reeves, C. B. (1993). *Modern Heuristic Techniques for Combinatorial Problems.*

Rich, E., & Knight, K. (1994). *Inteligencia artificial* (2 ed.). (P. A. González Calero, & F. Trescastro Bodega, Trads.) McGraw-Hill. Recuperado el 18 de Julio de 2017

Rosenblatt. (1958). The Perceptron: a probabilistic model for information storage and organization in the brain. *Phychological Review, Volumen 65*(No. 6), 24.

Soudry & Carmon. (2016). No bad local minima: Data independent training error guarantees. *arXiv Machine Learning*.

Spector, L. (2006). *Automatic Quantum Programing: A Genetic Programming Approach Genetic Programming* . Springer US.

Spencer, H. (1960). *The Principles of Biology.* London.

Tablada, C. J., & Torres, G. A. (2009). Redes Neuronales Artificiales. *Redes Neuronales Artificiales, 24*(24.3), 9. Recuperado el 18 de Julio de 2017, de http://www.famaf.unc.edu.ar/~revm/digital24-3/redes.pdf

Vinay Chandwani, V. A. (2014). Modeling slump of ready mix concrete using genetic algorithms assisted training of Artificial Neural Networks. *Expert Systems with Applications*, 885-893.

Anexo 1: Código del algoritmo evolutivo de inspiración cuántica

```
        QIEA-R*/


        //#include <iostream.h>


        #include <iostream>
        #include <random>
        #include <math.h>
        #include <stdlib.h>
        #include <time.h>
        #include <stdio.h>
        #include <vector>
        #include <algorithm>


        #define DBG 1


        using namespace std;


        ///Mersenne Twister
        typedef std::mt19937 MyRNG;  // the Mersenne Twister with a popular choice of
        parameters
        uint32_t seed_val;           // populate somehow
        MyRNG rng;                   // e.g. keep one global instance (per thread)
        void initialize()
        {
          rng.seed(seed_val);
        }
        //Mersenne Twister


        pair<double,int> get_min(vector<double>&fit_t) //get min and position
        {
                double min=fit_t[0];
                int idx=0;


                for(int i=1;i<(int)fit_t.size();i++)
                        if(fit_t[i]<min)
                        {
                                min=fit_t[i];
                                idx=i;
                        }


                pair<double,int> ans;
                ans.first = min;
                ans.second = idx;


                return ans;
        }
```

```
void Q_initialization(vector<vector<vector<double>>>&Q_t,double U_LIM,double
L_LIM)
{
	//Same Domain
	for(int i=0;i<(int)Q_t.size();i++)
		for(int j=0;j<(int)Q_t[0].size();j++)
		{
			Q_t[i][j][0]=U_LIM-L_LIM; //sigma -- pulse width
			Q_t[i][j][1]=L_LIM+(U_LIM-L_LIM)/2.0; //mu -- square
pulse
		}
}


void C_generation(vector<vector<double>>&C_t,
vector<vector<vector<double>>>&Q_t)
{
	uniform_real_distribution<double> dist(0,1);
	for(int i=0;i<(int)Q_t.size();i++)
		for(int j=0;j<(int)Q_t[0].size();j++)
			C_t[i][j]=dist(rng)*Q_t[i][j][0]+(Q_t[i][j][1]-
Q_t[i][j][0]/2.0);
}




/*
		//inicializa Qt con m individuos de n genes.
		//while t <= T do
			//generar Et observando los individuos Qt
			//if t = 1 then
			//	Ct = Et
			//else
			//recombinar(Et,Ct) -> Et
			evaluar Et
			seleccionar Ct<-k mejores individuos de Et U Ct
			end if
			actualizar Qt+1 con los m mejores individuos de Ct
			t = t + 1
		end while
*/


double evaluation(vector<double>&ind)
{
	double sum=0,0,a,b;
	for(int i=0;i<(int)ind.size();i++)
		sum+=pow(ind[i],2);
	a=pow(sin(sqrt(sum)),2)-0.5;
	b=pow(1+0,001*sum,2);
	return 0.5+(a/b);
}


void evaluation_2_fit(vector<double>&fit_t, vector<vector<double>>&C_t)
{
	for(int i=0;i<(int)C_t.size();i++)
	{
```

```
                        fit_t[i]=evaluation(C_t[i]);
                }
        }


        bool sort_evaluation(vector<double>&ind_1, vector<double>&ind_2)
        {
                return (evaluation(ind_1)<evaluation(ind_2));
        }


        void print_Q(vector<vector<vector<double>>>&Q_t)
        {
                for(int i=0;i<(int)Q_t.size();i++)
                {
                        cout<<endl<<"Q("<<i<<")";
                        for(int j=0;j<(int)Q_t[0].size();j++)
                                cout<<endl<<"Q("<<i<<")("<<j<<"): o="<<Q_t[i][j][0]<<"
        u="<<Q_t[i][j][1];
                }
        }




        void print_C(vector<vector<double>>&C_t)
        {
                for(int i=0;i<(int)C_t.size();i++)
                {
                        cout<<endl<<"C("<<i<<"):";
                        for(int j=0;j<(int)C_t[0].size();j++)
                                cout<<" "<<C_t[i][j];
                }
        }




        void print_fit(vector<double>&fit_t)
        {
                cout<<endl<<"Fit:";
                for(int i=0;i<(int)fit_t.size();i++)
                        cout<<" "<<fit_t[i];
        }


        int main(int argc,char **argv)
        {


                int T_POB=20,N_DIM=2,U_LIM=10,L_LIM=-10,N_GEN=100;
                double P_CRO=0.5,P_UD=1,DELTA=0.82;


                double best_fit_t,best_fit_t_1,best_global;
                int best_pos=0; //elitism does that the best are in the first position
        ever
```

```
        vector<vector<vector<double>>>Q_t(T_POB,vector<vector<double>>(N_DIM,ve
ctor<double>(2)));


        vector<vector<double>>C_t(T_POB,vector<double>(N_DIM));
        vector<vector<double>>E_t(T_POB,vector<double>(N_DIM));


        vector<vector<double>>E_U_C(T_POB*2,vector<double>(N_DIM));


        vector<double>fit_t(T_POB);
        vector<double>fit_t_1(T_POB);


        vector<double>best_ind(N_DIM);


        uniform_int_distribution<int>dist_T_POB(0,T_POB-1);      //To get index
of classical individuals
        uniform_real_distribution<double>dist_01(0,1);  //To get prob of
crossover


        //t=1
        int t=0;


        //Initialize Q(t) with m individuals from n genes
        Q_initialization(Q_t,U_LIM,L_LIM);


        if(DBG) {cout<<endl<<"Q inicial";} print_Q(Q_t);




        //while t<=T
        while(t<N_GEN)
        {
                if(DBG)
cout<<endl<<":::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::t="<<t;


                //generate E(t) observing the Q(t) individuals
                C_generation(E_t,Q_t);


                if(DBG) {cout<<endl<<"E::::::::::"; print_C(E_t);}


                //if t=1 then
                if(t==0)
                {
                        //C(t)=E(t)
                        C_t=E_t;


                        evaluation_2_fit(fit_t,C_t);
```

```
                        //eliminar a futuro;
                        print_fit(fit_t);


                        sort(C_t.begin(),C_t.end(),sort_evaluation);



                        evaluation_2_fit(fit_t,C_t);



                        //eliminar a futuro
                        print_C(C_t);


                        best_fit_t=fit_t[0];


                        if(DBG) {print_fit(fit_t); cout<<"best: "<<best_fit_t;}
                }
                //else
                else
                {
                        //recombine {E(t),C(t)} -> E(t) -- Using Differential
Evolution: crossover


                        int father,mother;      //parents
                        vector<double> x_c1(N_DIM);//To save crossover1
            vector<double> x_c2(N_DIM);//To save crossover2


                        for(int i=0;i<T_POB;i++)
                        {
                                double random=dist_01(rng);
                                if(random<=P_CRO)
                                {
                                        do
                                                father=dist_T_POB(rng);
                                        while(father==i); //bestpos ?


                                        do
                                                mother=dist_T_POB(rng);
                                        while(mother==i||mother==father);
//bestpos ?


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


                            double alpha=dist_01(rng);
                            x_c1[j]=alpha*E_t[father][j]+(1-
alpha)*C_t[mother][j];
```

```
                                x_c2[j]=alpha*C_t[mother][j]+(1-
alpha)*E_t[father][j];
                                }


                                E_t[father]=x_c1; //for all dimensions
:)
                        E_t[mother]=x_c2; //for all dimensions :)
                                }
                        }


                        if(DBG) {cout<<endl<<"recombinar(E,C) -> E";
print_C(E_t);}


                        //evaluate E(t)


                        fit_t_1=fit_t;


                        evaluation_2_fit(fit_t,E_t);


                        best_fit_t_1=best_fit_t;


                        best_fit_t=get_min(fit_t).first;


                        if(DBG) {print_fit(fit_t);
cout<<endl<<"best:"<<best_fit_t;}


                        //select C(t) <- k best individuals of E(t) U C(t)


                        int idx=0;
                        for(int i=0;i<(int)E_t.size();i++)
                        {
                                E_U_C[idx]=E_t[i];
                                idx++;
                        }
                        for(int i=0;i<(int)C_t.size();i++)
                        {
                                E_U_C[idx]=C_t[i];
                                idx++;
                        }


                        sort(E_U_C.begin(),E_U_C.end(),sort_evaluation);


                        for(int i=0;i<(int)C_t.size();i++)
                                C_t[i]=E_U_C[i];


                        best_global=evaluation(C_t[0]);
```

```
                        if(DBG) cout<<endl<<"BEST GLOBAL: "<<best_global;


                        if(DBG) {cout<<endl<<"C <- E U C : "; print_C(C_t);}
                }


                //update Q(t+1) with the individuals of C(t)
        double random=dist_01(rng);


        if(random<P_UD && t!=0)
        {
            //Pulse width adjustment (sigma) -- Rule 1/5th
            int counter = 0;


            for(int i=0;i<T_POB;i++) //Josimar Chire's Strategy (he have a
extitive gap generational)
                            if(fit_t[i]<best_fit_t_1)
                                    counter++;


            double phi=(double)counter/T_POB;
                        cout<<endl<<"phi"<<phi;


                        double mul_fact=1.0;


            if(phi<0.2)
                mul_fact=DELTA;
            else if(phi>0.2)
                mul_fact=1.0/DELTA;


                        cout<<endl<<"mul_fact="<<mul_fact;


            for(int i=0;i<T_POB;i++)
                for( int j=0;j<N_DIM;j++)
                    Q_t[i][j][0]*=mul_fact;


        //Moving Square pulse (mu)
            double lambda=dist_01(rng);


            for (int i=0;i<T_POB;i++)
                for (int j=0;j<N_DIM;j++)
                    Q_t[i][j][1]+=lambda*(C_t[best_pos][j]-Q_t[i][j][1]);
        }


                //t=t+1
                t++;
        }
}
```

Anexo 2: Código de la red neuronal de n-capas

```
#include<iostrea
m>
                    #include<time.h>
                    #include<math.h>
                    #include<stdlib.h>


                    using namespace std;


                    double act(double z)
                    {
                      return 1.0/(1.0+exp(-z));
                    }


                    double dact(double a)
                    {
                      return a*(1.0-a);
                    }


                    double derror(double t,double a)
                    {
                      return (a-t);
                    }


                    double mse(double *t,double *a,int n)
                    {
                      double sum=0,0;
                      for(int i=0;i<n;i++)
                        sum+=0.5*pow(t[i]-a[i],2);
                      return sum;
                    }


                    int main(void)
                    {
                      cout.precision(8);
                      int n_instances,n_layers,n_epoch;
                      double alpha,error;


                      cout<<" Artificial Neural Network with n-
                    layers\n=======================================\n\n";


                      //topology
```

```
  cout<<"Ingrese Nro de Capas";cin>>n_layers;
  int *layer = new int[n_layers];
  for(int i=0;i<n_layers;i++)
  {
    cout<<"\nIngrese el Nro de Neuronas en la capa
"<<i<<": ";
    cin>>layer[i];
  }


  cout<<"\nIngrese el Nro de Instancias:
";cin>>n_instances;


  //create matrix of inputs and outputs
  double **input = new double*[n_instances];
  double **output = new double*[n_instances];
  for(int i=0;i<n_instances;i++)
  {
    input[i] = new double[layer[0]];
    output[i] = new double[layer[n_layers-1]];
  }
  cout<<"\nIngrese la matriz de
instancias*(inputs+outputs)\n";

  //read matrix of inputs and outputs
  for(int i=0;i<n_instances;i++)
  {
    for(int j=0;j<layer[0];j++)
      cin>>input[i][j];
    for(int j=0;j<layer[n_layers-1];j++)
      cin>>output[i][j];
  }

  cout<<"\nIngrese el Nro de epocas\n";cin>>n_epoch;

  cout<<"\nIngrese el Factor de
Aprendizaje\n";cin>>alpha;

  //create structure of data for forward
  double z;
  double **a = new double*[n_layers];
  double **b = new double*[n_layers-1];
  double ***w = new double**[n_layers-1];
  for(int i=1;i<n_layers;i++)
    a[i] = new double[layer[i]];
  for(int i=0;i<n_layers-1;i++)
  {
    b[i] = new double[layer[i+1]];
    for(int j=0;j<layer[i+1];j++)
      b[i][j]=((double)rand()/(RAND_MAX));
  }
  for(int i=0;i<n_layers-1;i++)
  {
    w[i] = new double*[layer[i]];
```

```
      for(int j=0;j<layer[i];j++)
      {
        w[i][j] = new double[layer[i+1]];
        for(int k=0;k<layer[i+1];k++)
          w[i][j][k]=((double)rand()/(RAND_MAX));
      }
    }

        /*b[0][0]=0,05;
        b[0][1]=0.1;
        b[1][0]=0.15;
        b[1][1]=0.2;
        b[2][0]=0.25;
        //b[2][1]=0.3;
        w[0][0][0]=0.35;
        w[0][0][1]=0.4;
        w[0][1][0]=0.45;
        w[0][1][1]=0.5;
        w[1][0][0]=0.55;
        w[1][0][1]=0.6;
        w[1][1][0]=0.65;
        w[1][1][1]=0.7;
        w[2][0][0]=0.75;
        //w[2][0][1]=0.8;
        w[2][1][0]=0.85;
        //w[2][1][1]=0.9;*/


    //create structure of data for backward
    double **d_b = new double*[n_layers-1];
    for(int i=0;i<n_layers-1;i++)
      d_b[i] = new double[layer[i+1]];


    for(int epoch=0;epoch<n_epoch;epoch++)
    {
      error=0,0;
      for(int instance=0;instance<n_instances;instance++)
      {
        //forward propagation


        a[0]=input[instance];
        for(int i=0;i<n_layers-1;i++)
          for(int j=0;j<layer[i+1];j++)
          {
            z=b[i][j];
            for(int k=0;k<layer[i];k++)
              z+=a[i][k]*w[i][k][j];
            a[i+1][j]=act(z);
          }


        //calculate error
```

```
            error+=mse(output[instance],a[n_layers-
1],layer[n_layers-1]);

            //backward propagation
            //for the last layer
            int m=n_layers-2;
            for(int i=0;i<layer[m+1];i++)
            {

d_b[m][i]=derror(output[instance][i],a[m+1][i])*dact(a[m+
1][i]);
            }


            //for another layers
            for(int m=n_layers-3;m>-1;m--)
              for(int i=0;i<layer[m+1];i++)
              {
                z=0,0;
                for(int j=0;j<layer[m+2];j++)
                  z+=w[m+1][i][j]*d_b[m+1][j];
                d_b[m][i]=z*dact(a[m+1][i]);
              }


            //new weight and bias ///////falta verificar
            for(m=0;m<n_layers-1;m++)
               for(int j=0;j<layer[m+1];j++)
               {
                     b[m][j]-=alpha*d_b[m][j];
                     for(int i=0;i<layer[m];i++)
                     w[m][i][j]-=alpha*d_b[m][j]*a[m][i];
               }
          }
          cout<<endl<<epoch<<" : "<<error;
        }


        return 0;
}
```