Diseño de una Neurona Asíncrona con aprendizaje evolutivo para Redes Neuronales de Espigas

Edgar D. Acosta-Pérez[1a], Julio C. Martínez Romo[2a], Francisco J. Luna-Rosas[3a],Carlos Paredes[4b]
[a] Tecnológico Nacional de México/Instituto Tecnológico de Aguascalientes, División de Estudios de Posgrado e Investigación. Aguascalientes, México
[b] Centro de Investigaciones en Óptica A.C. Aguascalientes, México
Email: [1]edgar.ap@aguascalientes.tecnm.mx, [2]julio.mr@aguascalientes.tecnm.mx, [3]fcoluna2000@yahoo.com.mx, [4]cparedes@cio.mx

Abstract: Las redes neuronales de espigas (SNN, por sus siglas en inglés) es la tercera generación de redes neuronales; se separa de sus predecesoras en que, en lugar de implementaciones algorítmicas en procesadores secuenciales, se persigue implementar redes neuronales directamente en arquitecturas de procesamiento en dispositivos de hardware tales como matriz de puertas programables en campo (FPGA, por sus siglas en inglés), circuitos integrados de aplicación específica o chips neuromórficos, buscando así una "ejecución" en paralelo y concurrente de la red neuronal y emulando a una neurona animal a través de modelos preexistentes. Este enfoque cuenta con ventajas y desventajas; entre las ventajas se encuentra que la operación de la SNN suele ser más rápida al estar a expensas mayormente de tiempos de propagación más que a ciclos de programación, al ser hardware dedicado, no se somete a retardos inducidos por el sistema operativo de un host que lo albergue y la eficacia en el uso energético. Entre las desventajas se encuentra la dificultad para desarrollo de algoritmos de entrenamiento dada su inherente naturaleza discreta, así como la diversidad de posibilidades de implementación. No obstante, el desarrollo de las SNN es un campo de investigación activo. En esta investigación se desarrolló una neurona digital para su utilización en una red neuronal de espigas, con implementación en un FPGA; su entrenamiento es evolutivo y le permite clasificar patrones de pulsos de entrada y generar los pulsos de salida correspondientes. La neurona se puede controlar para formar capas y, finalmente, redes neuronales. Se demostró que una única neurona es capaz de reproducir una salida deseada aprendiendo de varios canales de patrones aleatorios de espigas, con cientos de espigas por canal.

Palabras clave: *Redes neuronales de espiga; aprendizaje supervisado; Neurona; clasificador de picos, algoritmos genéticos.*

I. Introducción

Una neurona biológica es una célula especializada en la transmisión de información a otras neuronas mediante señales electroquímicas. Las señales eléctricas se manifiestan como diferencias de voltaje que atraviesan la membrana celular. Las neuronas poseen numerosas ramificaciones, llamadas dendritas, que les permiten conectarse con otras neuronas y recibir información [1]. Los axones, en cambio, actúan como salidas, transmitiendo dicha información. La conexión entre dos neuronas se denomina sinapsis, y ocurre entre un axón de una neurona y una dendrita de otra. La terminal del axón puede ser excitada o inhibida según el umbral, lo que permite a la sinapsis regular el pulso entrante y su efecto en la integración de la neurona. Este aumento o disminución en la actividad es la tasa de disparo.

Para distintas capas entre neuronas, la comunicación se realiza mediante un voltaje estático, mientras que, para distancias más largas, las señales se envían a través de pulsos de voltaje a lo largo de dendritas y axones [2]. Esta información se codifica en una tasa de pulso o de disparo, lo que asegura una comunicación eficiente en el sistema nervioso. Cada pulso recibido por una neurona se integra en el cuerpo celular hasta que se alcanza un umbral específico. Una vez alcanzado este umbral, la neurona genera su propio pulso, y el proceso de integración se reinicia [3].

Las redes neuronales artificiales (ANN, por sus siglas en inglés) son abstracciones y simulaciones de la estructura y función del sistema nervioso biológico, desempeñan funciones importantes en el procesamiento de información y reconocimiento de patrones [4]; están constituidas por neuronas artificiales, las cuales son unidades computacionales básicas que intercambian información entre ellas a través de conexiones sinápticas.

Según sus unidades computacionales, los modelos ANN se pueden dividir en tres generaciones diferentes [4]. En la primera generación utilizaron como base fundamental las neuronas de McCulloch-Pitts como unidad computacional para muchos modelos de redes neuronales como los perceptrones multicapa entre otros [2]. En la segunda generación, se ampliaron las capacidades de las ANN al introducir diversas funciones de activación continuas, como la función sigmoide. Esto permitió expandir su campo de aplicación, facilitando el desarrollo de arquitecturas como las redes neuronales de propagación hacia adelante y las redes de Kohonen [5], entre otras. En la tercera generación, surgieron las SNN como una evolución de las ANN. Estas redes incorporan el concepto del tiempo, lo que les permite ser ampliamente utilizadas en diversas aplicaciones, como el reconocimiento de patrones temporales [6].

Las SNN están inspiradas en la neurona animal por tal motivo tratan de imitar a las neuronas humanas utilizando picos para transmitir y aprender de los datos. Su aprendizaje y activación espacio-temporales dependen de los estímulos recibidos de otras neuronas. A diferencia de las redes neuronales tradicionales, las SNN se distinguen tanto por el modelo de neurona utilizado como por su función de activación, lo que les permite procesar la información de manera más cercana al funcionamiento del cerebro biológico. La neurociencia sugiere que los sistemas nerviosos biológicos codifican la información mediante el momento preciso en que ocurren las

espigas, en lugar de la tasa de disparo neuronal. Esto les otorga una mayor capacidad de cómputo, permitiendo simular una amplia variedad de señales neuronales. Las SNN son particularmente adecuadas para aproximar cualquier función continua, demostrando ser muy efectivas en el procesamiento de información espacio-temporal [7].

En la literatura se han reportado diversos algoritmos de clasificación utilizando SNN. Por ejemplo, Shirin en [8] presentó un nuevo algoritmo de aprendizaje para una SNN de tres capas, específicamente orientado a abordar problemas de clasificación de patrones. Durante su experimentación logró maximizar el margen entre clases y evaluó su rendimiento utilizando diez conjuntos de datos de referencia del repositorio de aprendizaje automático de UCI. Al comparar su algoritmo con otros métodos de aprendizaje existentes para SNN, logró obtener un mejor rendimiento de generalización en comparación con los demás. Por otro lado, Domínguez en [9] desarrolló un modelo de red neuronal de espigas para la clasificación de señales variantes en el tiempo, con un enfoque dirigido al reconocimiento de voz en tiempo real. Durante el entrenamiento, utilizó un extenso conjunto de datos que incluía los comandos de voz "izquierdo" y "derecho", alcanzando una precisión del 89.90%. Por su parte Shruti et al. en [10] desarrollaron una nueva SNN para la clasificación automática de dígitos escritos a mano en tiempo real, implementada en una plataforma GP-GPU. El procesamiento de la información en la red, desde la extracción de características hasta la clasificación, imitó los principios básicos de la iniciación y propagación de picos neuronales en el cerebro. Lograron una precisión del 98.17% en el conjunto de datos MNIST utilizando el algoritmo de aprendizaje NormAD. Por su parte Alireza et al. en [11] afirmaron que los clasificadores entrenados en ANN mediante métodos convencionales de minimización empírica del riesgo sufren degradaciones drásticas del rendimiento cuando se prueban sobre ejemplos seleccionados de forma intencional en función del conocimiento de la regla de decisión del clasificador. Debido a esto desarrolló un clasificador mediante SNN en la que puso a prueba la codificación, velocidad y tempo, dando como resultado una mejora significativa en su clasificación con respecto a las anteriores. Han et al. en [12] implementaron una SNN con un algoritmo de aprendizaje híbrido en un FPGA para realizar una tarea de clasificación en el conjunto de datos MNIST. Sus resultados demostraron una precisión del 97.06% y una velocidad de procesamiento de 161 fotogramas por segundo. De manera similar, Johnson et al. en [13] implementaron un método de aprendizaje tolerante a fallos junto con un sistema neuromórfico en un FPGA. En sus resultados, destacan que el sistema es capaz de mantener su rendimiento de manera consistente, independientemente de las fallas que puedan ocurrir. En la mayoría de los casos señalados, se pone atención a la funcionalidad de la red neuronal, pero la implementación queda en segundo término, con lo cual no se busca la optimización de hardware, ya sea en FPGAs, ASICs y otras implementaciones, como en los chips neuromórficos.

En esta investigación se desarrolló una neurona digital para su utilización en una red neuronal de espigas, con implementación en un FPGA; su entrenamiento es evolutivo y le permite clasificar patrones de pulsos (“espigas de voltaje”) de entrada y generar los pulsos de salida correspondientes, de acuerdo con los ejemplos de entrenamiento. La neurona se puede controlar para formar capas y, finalmente, redes neuronales. En cuanto a implementación, se encuentra que esta neurona digital utiliza muy pocos recursos de hardware.

El presente artículo se divide de la siguiente forma: la sección II, describe los fundamentos teóricos; en la sección III, se presenta el diseño de la neurona y la adaptación de un algoritmo genético para su entrenamiento. Los resultados y discusión se reportan en la sección IV y, por último, las conclusiones se dan en la sección V.

II. FUNDAMENTOS TEÓRICOS

II.1 Redes neuronales de espigas y aprendizaje supervisado

Existen diversos algoritmos de aprendizaje implementados en las SNN, siendo el aprendizaje supervisado el más utilizado por los investigadores. Este enfoque ha demostrado ser altamente beneficioso en diferentes áreas de investigación. Los algoritmos de aprendizaje se clasifican en varias categorías, que incluyen la arquitectura de red, el modo de desplazamiento, la codificación de información, la dinámica estructural y la representación del conocimiento. [14][15].

El aprendizaje supervisado en las SNN se implementa a través de la adquisición de patrones espacio-temporales de trenes de picos. Un tren de espigas $s = \{t^f \epsilon\, \Gamma : f = 1, \dots, F\}$ es la secuencia ordenada de picos en los que una neurona en pico dispara en el intervalo de tiempo Γ = [0, T], y puede expresarse formalmente como [15]:

$$s(t) \ = \Sigma_{f=1}^{F} \delta(t - t^f) \qquad (1)$$

Donde t^f es el f-ésimo tiempo de pico en s(t), F es el número de picos y δ(·) representa la función delta de Dirac: δ(x) = 1 si x = 0 y δ(x) = 0 en caso contrario. Aunque los diversos algoritmos de aprendizaje supervisado para SNN son diferentes, el objetivo general de ellos es consistente: para un conjunto dado de trenes de picos de entrada S_i y trenes de picos de salida deseados S_d encontrar la matriz de ponderación sináptica adecuada para los pesos (W) de las SNN para hacer que los trenes de picos de salida reales S_a sean lo más similares posible a los trenes de picos de salida deseados correspondientesS_d; es decir, el valor de la función de evaluación de errores E (S_a, S_d) entre ellos sea un mínimo. Algunos algoritmos de aprendizaje que se mencionan en la literatura son: Hodgkin-Huxley [16], integración y activación de neuronas [17] y modelo de respuestas de picos (SRM, por sus siglas en inglés) [18].

II.2 Implementación del algoritmo de aprendizaje

En la actualidad los FPGA son ampliamente utilizados para implementar funciones lógicas específicas. Una de sus principales ventajas frente a la lógica fija es que requieren menos espacio en la tarjeta de circuito impreso para una cantidad equivalente de lógica. Además, permiten modificar los diseños de manera sencilla, sin necesidad de recablear o reemplazar componentes. Asimismo, el uso de circuitos lógicos programables permite implementar un diseño lógico

de manera más rápida y a menor costo en comparación con los circuitos integrados de función fija [19].

Los FPGA están compuestos por tres elementos clave: los bloques lógicos configurables (CLB, por sus siglas en inglés), las interconexiones programables y los bloques de entrada/salida (E/S). Estos componentes permiten el desarrollo de circuitos extremadamente complejos con una funcionalidad sobresaliente. Para su programación, se emplean técnicas como VHDL, Verilog y la captura esquemática. Esta última consiste en diseñar circuitos mediante diagramas, que posteriormente se traducen a VHDL para su implementación en los FPGA[20].

II.3 Algoritmos Genéticos

Los algoritmos genéticos son métodos adaptativos utilizados para resolver problemas de búsqueda y optimización, inspirados en los procesos genéticos de los organismos vivos [21]. Estos algoritmos simulan la evolución de poblaciones a lo largo de múltiples generaciones, siguiendo los principios de la selección natural y la supervivencia del más apto, tal como lo postuló Darwin. A través de la reproducción, mutación y selección, los individuos de una población mejoran progresivamente para encontrar soluciones óptimas o cercanas al óptimo.

Un algoritmo genético se constituye con los llamados operadores genéticos [22], mismos que se describen a continuación; primero, se genera una población inicial (*Población*) y se seleccionan los individuos más aptos para formar lo que se conoce como piscina de apareamiento [23]. Durante la ejecución del algoritmo, los padres deben ser seleccionados para la reproducción (*Selección*), a continuación (*Cruza*), dichos padres seleccionados se cruzarán generando dos hijos, sobre los cuales se aplica un operador de mutación (*Mutación*). El resultado de estas operaciones es un conjunto de individuos, llamados descendientes, que representaran posibles soluciones al problema. En la evolución del algoritmo genético, estos descendientes se integrarán en la siguiente población, conocida como nueva población [23].

En un algoritmo genético se requiere una medida cuantitativa para evaluar qué tan bien adaptado es un individuo de la población para resolver el problema de optimización planteado; esta medida se obtiene de la función de costo *J* (o función de pérdida) [24]. Dicha función de costo es específica para cada problema.

II.4 Comportamiento eléctrico de una neurona.

El comportamiento eléctrico simplificado de una neurona animal se muestra en la Figura 1. El potencial de membrana comienza en su estado inicial, conocido como potencial de reposo. Al recibir estímulos, el potencial empieza a incrementarse; sin embargo, si dichos estímulos no son suficientes para que el potencial sobrepase el umbral de estimulación, este volverá a decaer hasta su valor inicial. En caso de que, en otro momento, los estímulos sean suficientes para superar el umbral, el potencial aumentará drásticamente, generando un pico de potencial de acción. Este pico da lugar a una espiga que se transmitirá a otra neurona. Posteriormente, el potencial comenzará a disminuir (repolarización), incluso si persisten los estímulos, hasta alcanzar un periodo refractario (hiperpolarización), posteriormente restableciéndose al valor inicial [25].

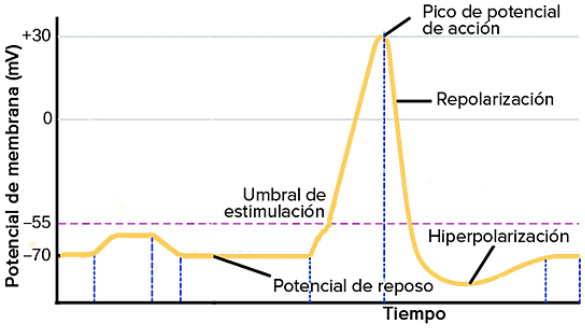


Figura 1. Comportamiento eléctrico de una Neurona animal: el potencial de membrana (Elaboración propia).

III. DESARROLLO

III.1 Descripción de la neurona.

El objetivo de esta investigación fue desarrollar una neurona digital cuyo comportamiento emule el comportamiento eléctrico de una neurona animal, como se describió en la sección II.4; con el fin de replicar el comportamiento eléctrico de la neurona animal, se efectuó un análisis detallado del potencial de membrana, tal como se muestra en la Figura 1.

La neurona digital desarrollada se muestra en la Figura 2 y está compuesta por bloques fundamentales que, en conjunto, logran objetivo planteado. La idea fundamental es considerar un elemento contador como el acumulador de voltaje (equivalente al potencial de membrana) y a partir de éste ir generando los bloques de control que regulen el comportamiento de dicho contador como el cuerpo o SOMA de una neurona. A continuación, se describen sus partes y funciones.

Cuando se le aplica una entrada de reloj a través de la terminal clock al Generador reloj secundario (Flip Flop tipo T) este a su salida entrega una frecuencia baja con respecto a la entrada, ambas frecuencias son utilizadas en conjunto con la entrada Spike_In las cuales son implementadas en el control de pendiente (multiplexor 2 a 1) del potencial eléctrico de la neurona. Al ingresar un estado en alto (uno lógico) en la entrada Spike_In este activará el habilitador de neurona que permitirá hacer el conteo ascendente en el SOMA y a su vez llegará a la primera entrada de la compuerta AND2. Si el potencial eléctrico de la neurona no está en periodo refractario en ese mismo instante la segunda entrada de la compuerta recibe un estado en alto del detector de refractario, la salida de la compuerta habilitara al control dependiente para que envíe la frecuencia alta a la entrada de SOMA (contador up) el cual interpretara esta frecuencia como potenciales de acción a sumar en el cuerpo de la neurona y por consiguiente generara una señal de salida en la terminal Spike_out cuando supere un determinado valor de umbral.

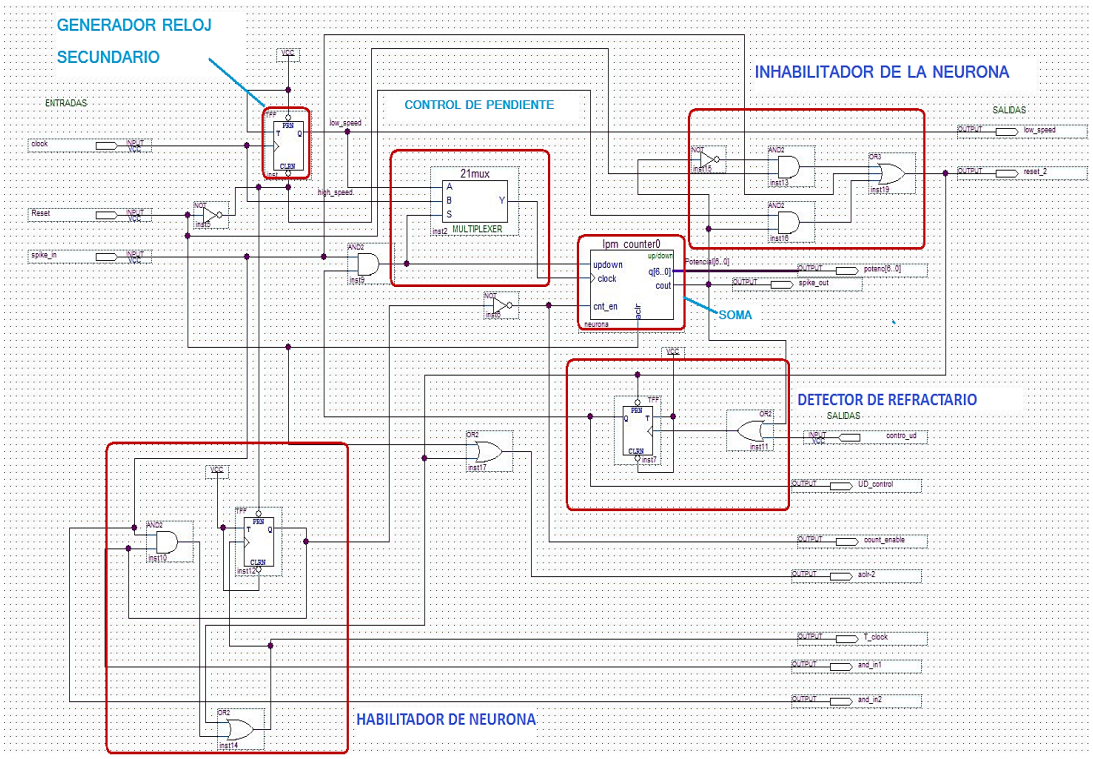


Figura 2. La neurona digital.

En caso contrario, si ya se generó la señal de salida o si el potencial eléctrico de la neurona está en periodo refractario, se activará el inhabilitador de la neurona y el detector de refractario enviará un cero lógico a la compuerta AND2 mismo que se transmitirá al control dependiente y este enviará una frecuencia baja al SOMA activando un contador descendente el cual será interpretado como etapa en que la neurona se vuelve a polarizar hasta llegar a un periodo refractario.

III.2 Configuración del algoritmo genético.

La relación entre el problema de optimización a resolver y el algoritmo genético se establece mediante la definición del cromosoma y la función de costo (*J*); el cromosoma se compone de las variables del problema a resolver, mientras que la función de costo se refiere a la relación matemática entre los parámetros y las variables de problema a resolver y se utiliza para determinar el grado de adaptación de cada cromosoma para la solución del problema.

Codificación del cromosoma.

Aquí, el cromosoma se codificó como una cadena de bits, con campos que representan las variables y parámetros de la neurona digital, los cuales se expresan en la Tabla 1. El tamaño del cromosoma es 29 bits más un numero E de bits, en donde E es el número de entradas de la neurona.

Función de costo.

La función de costo está dada por la expresión siguiente:

Tabla 1. Codificación del cromosoma.

Campo	Número de bits	Descripción del campo
1	1	Reset del potencial de membrana [0,1]
2	7	Potencial de membrana (mV)
3	3	Incremento del potencial de membrana cuando va ascendiendo (mV)
4	3	Decremento del potencial de membrana cuando va en descenso (mV) en periodo no refractario
5	3	Decremento del potencial de membrana (mV) en periodo inhibitorio
6	3	Decremento del potencial de membrana cuando va en descenso (mV) en periodo refractario
7	7	Potencial inicial de membrana (mV)
8	2	Estado de la neurona
9	E	Pesos sinápticos (E bits). E es el número de entradas a la neurona.

$$J = |(K - N)| \sum_{k=1}^{K} \min (t_k - t_n) \qquad (2)$$

en donde:

J, es la función de costo.
K, es el número de espigas que emite la neurona con los parámetros actuales del cromosoma.
N, es el número de espigas objetivo
t_k, es el tiempo en que se emite la *k*-ésima espiga por la neurona con los parámetros actuales del cromosoma.
t_n, es el tiempo en que existe la *n*-ésima espiga en el patrón de salida a aprender por la neurona.

Los parámetros del algoritmo genético fueron los siguientes:

Población: 50 individuos
Selección: uniforme.
Reproducción: elitismo de 5% de la población.
Cruce: de un punto.
Mutación: uniforme de 1%
Criterio de paro: 50 iteraciones.

La Figura 3 es una representación gráfica de los elementos de la función de costo, con K=2 (2 espigas emitidas por la neurona) y N=4 (4 espigas *objetivo* a ser aprendidas por la neurona). Las 4 espigas gruesas son el patrón objetivo de la neurona y las 2 espigas que las flanquean son las espigas que emite la neurona con el cromosoma actual durante el entrenamiento. Las líneas horizontales señalan la diferencia de tiempo entre el momento de emisión de las espigas y el momento en que existen las espigas objetivo: las líneas negras son la diferencia temporal entre la espiga 1 de la neurona y las cuatro espigas objetivo, mientras que las líneas moradas son la diferencia temporal entre la espiga 2 de la neurona y las cuatro espigas objetivo. Las diferencias temporales más cortas son las que se suman en J, con lo que se seleccionan para reproducción los cromosomas con más cercanía a las espigas objetivo.

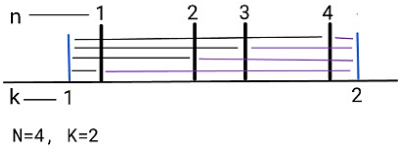


Figura 3. Relación temporal entre las espigas de salida y las espigas objetivo, en el contexto de la función de costo *J*.

Por otro lado, el término $|(K-N)|$ pondera la suma obtenida con la diferencia entre la cantidad de espigas emitidas y las espigas objetivo, de tal forma que entre menor sea esta diferencia, más apto es el cromosoma.

Dada la codificación del cromosoma, el problema de optimización a resolver es minimizar(*J*).

IV. Resultados

Para la validación funcional de la operación de la neurona, se programaron en lenguaje *m de Matlab* tanto el comportamiento fenomenológico de la neurona descrito en la sección III.1 como la función de costo de la expresión (2).

La prueba de la neurona digital propuesta se realizó en dos fases:

1. Entrenamiento evolutivo y simulación de comportamiento en Matlab®
2. Simulación de la neurona en el software Quartus II.

A continuación, se describen los resultados de cada caso.

IV.1 Aprendizaje evolutivo y simulación de software.

Como primera aproximación para verificar el funcionamiento de la neurona, se aplicaron patrones de entrada/salida consistiendo en siete entradas y una salida. No se realizó entrenamiento y en su lugar se realizaron pruebas activando cinco pesos sinápticos igual a 1, con un umbral de disparo de 50mV; se graficaron las espigas generadas por la neurona a su salida. Los patrones de entrada y las salidas se muestran en la Figura 4, en la que se puede apreciar que la simple estimulación de la neurona (4a) propicia potenciales de membrana similares (parte superior 4b) a los de la Figura 1, así como el disparo de la neurona (parte inferior 4b) cuando se alcanza el potencial de umbral (50mV).

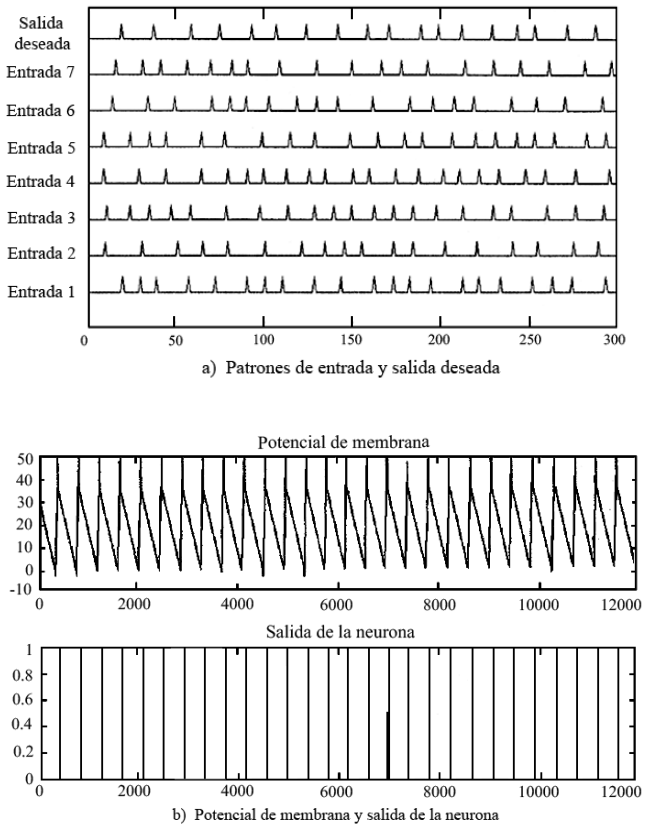


Figura 4. Relación temporal entre las espigas de salida y las espigas objetivo o salida deseada, en el contexto de la función de costo *J*.

Para implementar el algoritmo genético se utilizó el *ToolBox* de Optimización de Matlab.

Para realizar el entrenamiento supervisado se generaron nuevos conjuntos de datos, de los cuales sólo tres representativos se muestran en la Tabla 2. En la Tabla 2, se anota la longitud del vector que contiene los patrones de espigas, el número de entradas (equivalente al número de sinapsis) y el número de espigas de salida (o salida objetivo).

Tabla 2. Experimentos de validación del entrenamiento evolutivo.

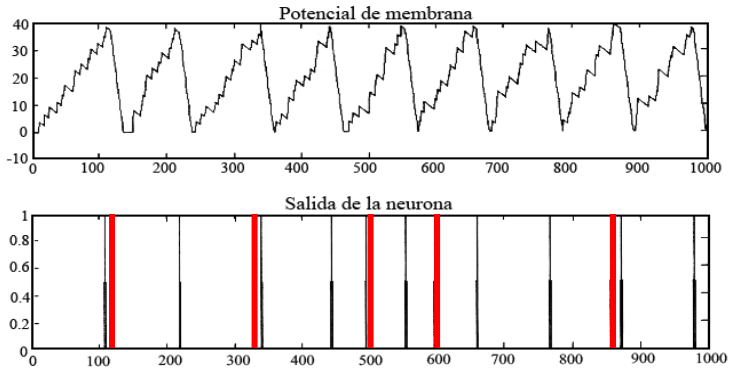
Experi-mento	Longitud entradas	Canales de entrada	Espigas objetivo
1	800	5	5
2	1000	5	5
3	10000	8	5

En la Tabla 2, se reportan tres de los varios experimentos realizados, teniendo longitudes del vector de entradas de 800, 1,000 y 10,000; la cantidad de entradas es de cinco u ocho y las salidas deseadas en todos los casos es de cinco. En la Figura 5 se reportan los resultados del experimento 2.

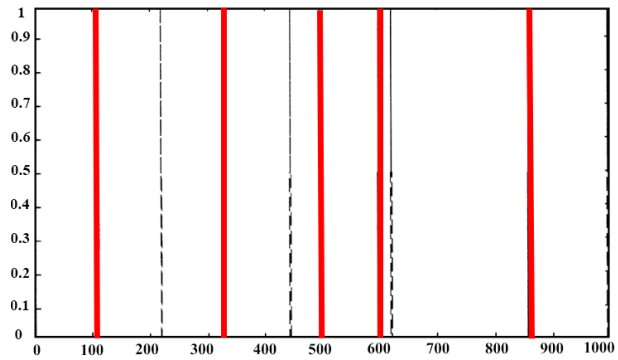
En el segundo experimento de la Tabla 2, se utilizaron como entradas 8 vectores de longitud 1000, con una cantidad aleatoria de pulsos o espigas y cinco espigas objetivo, que fueron la salida deseada de la neurona durante el entrenamiento. En la Figura 5a se muestran el potencial de membrana y la salida de la neurona (negro) comparada con la salida deseada (grueso), antes del entrenamiento evolutivo, en la que se observan las salidas generadas por la neurona debido a que alcanzó el potencial de umbral de disparo y siendo el resto las espigas de salida deseada durante el entrenamiento. De esta figura es claro que el número de espigas y su posición no se asemejan a la salida deseada. Después de entrenarse evolutivamente se observa el comportamiento de la Figura 5b, en donde la neurona pudo sintetizar la misma cantidad de espigas que la salida deseada y en zonas próximas a las espigas objetivo, incluso una espiga cercana a 900 se situó exactamente sobre la espiga objetivo en ese sitio.

Todo el proceso del párrafo anterior se dio con una población de cromosomas de 50 individuos. Para averiguar el impacto de aumentar la población -mayor área de búsqueda en la superficie de solución discreta de este problema-, se cambiaron ligeramente las ocurrencias de las cinco espigas de salida deseada, ver Figura 5c. Véase también en la Figura 5c que la neurona sin entrenamiento sintetiza sólo dos espigas. En la Figura 5d, es evidente que el mayor espacio de búsqueda propició una mejor solución, ya que ahora las cinco espigas disparadas por la neurona se aproximan más a las espigas objetivo que en el caso de búsqueda con 50 individuos.

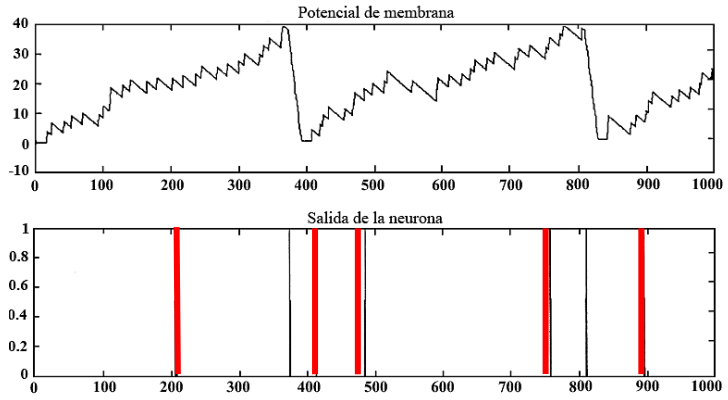
El hecho de que las salidas de una única neurona reproduzcan aproximadamente las espigas objetivo es normal, considerando que una red neuronal artificial jamás reproduce sus patrones al 100%, porque indicaría sobre ajuste. También, nótese la potencia de una única neurona, ya que está aprendiendo una secuencia temporal simple a partir de 8000 datos de entrada.



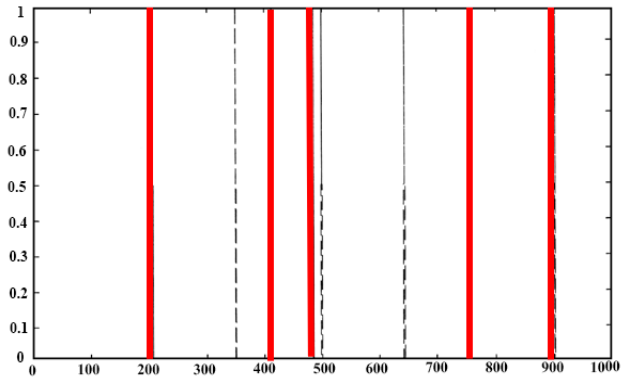
a) Potencial de membrana, salida de la neurona sin entrenamiento y salida de la deseada. 50 individuos.



b) Salida de la neurona entrenada, comparada con salida de la deseada. 50 individuos.



c)Potencial de membrana, salida de la neurona sin entrenamiento y salida de la deseada. 100 individuos.



d) Salida de la neurona entrenada, comparada con salida de la deseada. 100 individuos.

Figura 5. Entrenamiento de la neurona digital. (a) y (b), con una población de 50 individuos; (c) y (d), entrenamiento con 100 individuos.

IV.3 Implementación en FPGA.

A efectos de sintetizar la neurona digital en un FPGA, se eligió una tarjeta con el FPGA Cyclone II de Altera®, el EPM1270F256C5. El diseño se realizó en el software Quartus II de Altera®, mediante la herramienta de captura esquemática. La Figura 2 es, de hecho, el diagrama capturado en Quartus II, en el cual se pueden ver las diversas etiquetas de monitoreo y control de la neurona.

Para llevar a cabo la simulación de la neurona, se le proporcionaron pulsos (espigas) de entrada y señales de control para determinar si podría generar el disparo, así como observar el cumplimiento del comportamiento similar al de un ciclo de disparo de una neurona animal. En la Figura 6 se muestra el resultado de esta simulación; en el área media resalta el gráfico del potencial de membrana calculado con las salidas del contador o soma (cuerpo) de la neurona y es notable la similitud con el comportamiento eléctrico de la neurona animal mostrado en la Figura 1.

Adicionalmente, en la Figura 1 se marcaron algunas señales de interés, pero la importante es la espiga de salida, con una breve duración temporal. Así, se puede concluir que además de tener la capacidad de aprender patrones complejos, es congruente con el comportamiento de la neurona animal la que, por principio, es la fuente de inspiración para las SNN.

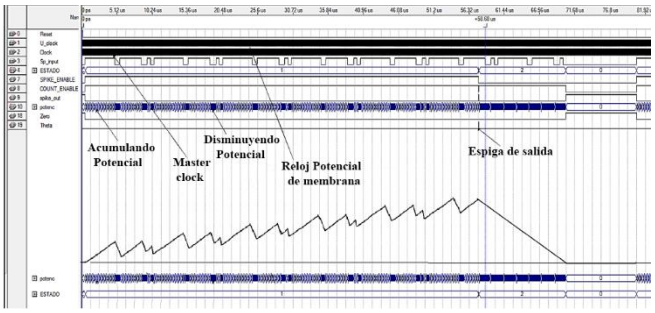


Figura 6. Simulación de la neurona digital.

V. Conclusiones

Las redes neuronales de espiga son las que mejor se asemejan a las redes neuronales biológicas ya que trabajan en base a pulsos o picos de entrada; el funcionamiento de este tipo de redes es mediante el uso de un potencial de membrana, el cual crese o aumenta de manera significativa siempre que haya pulsos o picos en las entradas, dichas entradas son combinadas con ciertos pesos sinápticos los cuales son ajustados por el algoritmo genético, y la forma de reacción de la neurona es activando la salida una vez que se ha alcanzado un cierto umbral de membrana que se puede decir es el valor de disparo con el cual actúa la neurona.

Con respecto a este proyecto se planteó como objetivo principal desarrollar una red neuronal artificial evolutiva en hardware, basada en un nuevo modelo de neurona digital capaz de responder a alta velocidad y fácilmente escalable, con un algoritmo de entrenamiento evolutivo implementado en un FPGA. Para el desarrollo de la neurona se utilizó el software de Quartus II en donde a través de simulaciones se estuvo evaluando su comportamiento. Se observó que presentó mejoras con respecto a las presentadas en la literatura de las cuales una de ellas es la capacidad de poder controlar los estímulos de entrada y el valor del umbral de disparo para mejorar su funcionamiento y como consecuencia el funcionamiento de la red.

Otra ventaja significativa fue el bajo consumo de recursos, ya que solo se utilizó menos del 1% del FPGA para implementar 29 elementos lógicos en comparación, otras aplicaciones en FPGA requirieron más recursos de hardware. Por ejemplo, Upegui en [26] implementó 53 elementos lógicos, utilizando el 2.25% de los recursos en una plataforma con un mecanismo de adaptación para una red neuronal de espiga. Por otro lado, Roggen en [25]. implementó 109 elementos lógicos para una aplicación de hardware basada en redes neuronales de espigas (SNN) con el objetivo de identificar objetos para un robot.

La mejora en el uso de recursos del FPGA de nuestra aproximación con respecto a otras aproximaciones reportadas en la literatura, estriba en un ahorro que va desde el 45% respecto a [19] hasta el 73% en [25].

El trabajo futuro incluye escalar la neurona para poder formar una SNN de dos o más capas, diseñar o elegir un patrón de codificación de datos reales a espigas.

Agradecimientos

Edgar D. Acosta-Pérez desea agradecer al CONAHCYT de México por haberme otorgado una beca para el desarrollo del Doctorado, así como al TecNM/Instituto Tecnológico de Aguascalientes por la formación recibida en nivel Doctorado.

Referencias

[1] C. Carvalhas-Almeida, J. Serra, J. Moita, C. Cavadas, and A. R. Álvaro, "Understanding neuron-glia crosstalk and biological clocks in insomnia," *Neurosci Biobehav Rev*, vol. 147, p. 105100, Apr. 2023, doi: 10.1016/J.NEUBIOREV.2023.105100.

[2] W. S. McCulloch and W. Pitts, "A logical calculus of the ideas immanent in nervous activity," *Bull Math Biophys*, vol. 5, no. 4, pp. 115–133, Dec. 1943, doi: 10.1007/BF02478259/METRICS.

[3] D. Chou and P. Y. Chen, "A perceptron-based learning method for solving the inverse problem of the brain model via poroelastodynamics," *Chaos Solitons Fractals*, vol. 172, p. 113611, Jul. 2023, doi: 10.1016/J.CHAOS.2023.113611.

[4] W. Maass, "Networks of spiking neurons: The third generation of neural network models," *Neural Networks*, vol. 10, no. 9, pp. 1659–1671, Dec. 1997, doi: 10.1016/S0893-6080(97)00011-7.

[5] "Backpropagation," *Backpropagation*, Feb. 2013, doi: 10.4324/9780203763247/BACKPROPAGATION-YVES-CHAUVIN-DAVID-RUMELHART.

[6] R. Brette *et al.*, "Simulation of networks of spiking neurons: A review of tools and strategies," *J*

Comput Neurosci, vol. 23, no. 3, pp. 349–398, Jul. 2007, doi: 10.1007/S10827-007-0038-6/METRICS.

[7] A. Borst and F. E. Theunissen, "Information theory and neural coding," *Nature Neuroscience 1999 2:11*, vol. 2, no. 11, pp. 947–957, Nov. 1999, doi: 10.1038/14731.

[8] S. Dora, S. Sundaram, and N. Sundararajan, "An Interclass Margin Maximization Learning Algorithm for Evolving Spiking Neural Network," *IEEE Trans Cybern*, vol. 49, no. 3, pp. 989–999, Mar. 2019, doi: 10.1109/TCYB.2018.2791282.

[9] J. P. Dominguez-Morales *et al.*, "Deep Spiking Neural Network model for time-variant signals classification: A real-time speech recognition approach," *Proceedings of the International Joint Conference on Neural Networks*, vol. 2018-July, Oct. 2018, doi: 10.1109/IJCNN.2018.8489381.

[10] S. R. Kulkarni, J. M. Alexiades, and B. Rajendran, "Learning and real-time classification of hand-written digits with spiking neural networks," *ICECS 2017 - 24th IEEE International Conference on Electronics, Circuits and Systems*, vol. 2018-January, pp. 128–131, Feb. 2018, doi: 10.1109/ICECS.2017.8292015.

[11] A. Bagheri, O. Simeone, and B. Rajendran, "Adversarial Training for Probabilistic Spiking Neural Networks," *IEEE Workshop on Signal Processing Advances in Wireless Communications, SPAWC*, vol. 2018-June, Aug. 2018, doi: 10.1109/SPAWC.2018.8446003.

[12] J. Han, Z. Li, W. Zheng, and Y. Zhang, "Hardware Implementation of Spiking Neural Networks on FPGA," 2020.

[13] A. P. Johnson *et al.*, "Fault-Tolerant learning in spiking astrocyte-neural networks on FPGAS," *Proceedings of the IEEE International Conference on VLSI Design*, vol. 2018-January, pp. 49–54, Mar. 2018, doi: 10.1109/VLSID.2018.36.

[14] N. Kasabov, "Time-space, spiking neural networks and brain-inspired artificial intelligence," 2019, Accessed: May 09, 2023. [Online]. Available: https://link.springer.com/content/pdf/10.1007/978-3-662-57715-8.pdf

[15] N. Kasabov, K. Dhoble, N. Nuntalid, and G. Indiveri, "Dynamic evolving spiking neural networks for on-line spatio- and spectro-temporal pattern recognition," *Neural Networks*, vol. 41, pp. 188–201, May 2013, doi: 10.1016/J.NEUNET.2012.11.014.

[16] A. L. Hodgkin and A. F. Huxley, "Currents carried by sodium and potassium ions through the membrane of the giant axon of Loligo," *J Physiol*, vol. 116, no. 4, pp. 449–472, Apr. 1952, doi: 10.1113/JPHYSIOL.1952.SP004717.

[17] A. N. Burkitt, "A review of the integrate-and-fire neuron model: I. Homogeneous synaptic input," *Biol Cybern*, vol. 95, no. 1, pp. 1–19, Jul. 2006, doi: 10.1007/S00422-006-0068-6/METRICS.

[18] "Spiking Neuron Models: Single Neurons, Populations, Plasticity - Wulfram Gerstner, Werner M. Kistler - Google Libros." Accessed: May 11, 2023. [Online]. Available: https://books.google.com.mx/books?hl=es&lr=&id=Rs4oc7HfxIUC&oi=fnd&pg=PR11&ots=2Sg-C_gQR8&sig=CTaoF44y8WprcNVJl-LaMHjTp60&redir_esc=y#v=onepage&q&f=false

[19] N. H. Noordin, P. S. Eu, and Z. Ibrahim, "FPGA Implementation of Metaheuristic Optimization Algorithm," *e-Prime - Advances in Electrical Engineering, Electronics and Energy*, vol. 6, p. 100377, Dec. 2023, doi: 10.1016/J.PRIME.2023.100377.

[20] A. Sahebi, M. Procaccini, and R. Giorgi, "HashGrid: An optimized architecture for accelerating graph computing on FPGAs," *Future Generation Computer Systems*, vol. 162, p. 107497, Jan. 2025, doi: 10.1016/J.FUTURE.2024.107497.

[21] M. E. Ortiz-Quisbert, M. A. Duarte-Mermoud, F. Milla, and R. Castro-Linares, "Control Adaptativo Fraccionario Optimizado por Algoritmos Genéticos, Aplicado a Reguladores Automáticos de Voltaje," *Revista Iberoamericana de Automática e Informática Industrial RIAI*, vol. 13, no. 4, pp. 403–409, Oct. 2016, doi: 10.1016/J.RIAI.2016.07.004.

[22] V. Imani, C. Sevilla-Salcedo, E. Moradi, V. Fortino, and J. Tohka, "Multi-objective genetic algorithm for multi-view feature selection," *Appl Soft Comput*, vol. 167, p. 112332, Dec. 2024, doi: 10.1016/J.ASOC.2024.112332.

[23] R. and H. S. Haupt, *Practical Genetic Algorithms*, Second Edition., vol. 1. New_Jersey: John Wiley_and_Sons.Inc., 2004.

[24] W. Gerstner, "Chapter 12 A framework for spiking neuron models: The spike response model," *Handbook of Biological Physics*, vol. 4, no. C, pp. 469–516, Jan. 2001, doi: 10.1016/S1383-8121(01)80015-4.

[25] W. Bechtel and L. Bich, "Using neurons to maintain autonomy: Learning from C. elegans," *Biosystems*, vol. 232, p. 105017, Oct. 2023, doi: 10.1016/J.BIOSYSTEMS.2023.105017.

[26] A. Upegui, C. A. Peña-Reyes, and E. Sanchez, "An FPGA platform for on-line topology exploration of spiking neural networks," *Microprocess Microsyst*, vol. 29, no. 5, pp. 211–223, Jun. 2005, doi: 10.1016/J.MICPRO.2004.08.012.