Benemérita Universidad Autónoma de Puebla

Facultad de Ciencias de la Electrónica

Maestría en Ciencias de la Electrónica,
Opción en Automatización

Tesis

CONTROL NEURO-INSPIRADO DE UN SISTEMA NO LINEAL

Presenta: **Ing. Valentín García Cervantes**

Directores de Tesis: **Dra. Amparo D. Palomino Merino (FCE-BUAP)**
Dra. Ma. Aurora Diozcora Vargas Treviño (FCE-BUAP)
Director Externo: **Dr. Juan Escareno (Univ. de Limoges, FRANCIA)**

Puebla, Puebla. Julio 2024.

Agradecimientos

Agradezco infinitamente a mi familia. A mis padres, Teresa y Valentín, quienes siempre me han apoyado de manera incondicional en cada paso de mi vida. A mis hermanos, José, Felipe, César, Teresa y Jessica, los cuales me han apoyado y compartido su conocimiento. A mi novia Diana, quien me ha acompañado durante estos años.

A mis asesores de tesis, la Dra. Amparo Dora Palomino Merino, la Dra. Ma. Aurora Diozcora Vargas Treviño y el Dr. Juan Escareno, quienes me acompañaron y guiaron en el proceso de realización de este trabajo. De la misma manera, a mi jurado, el Dr. Jorge Dionisio Fierro Rojas, el Dr. Sergio Vergara Limón y el Dr. Luis Abraham Sánchez Gaspariano, los cuales gracias a sus observaciones y comentarios me permitieron desarrollar un mejor trabajo.

A mis amigos y compañeros de maestría, María, Juan, Michelle, Alfredo, Luís, Steven y Ángel por los buenos momentos que compartimos como generación. A mi amigo y futuro colega Razhiel.

Al coordinador el Dr. José Eligio Moisés Gutiérrez Arias y la planta docente del posgrado, el Dr. Reyes, la Dra. Olga, el Dr. Sergio y la Dra. Amparo, quienes me compartieron su conocimiento durante este proceso.

A la Benemérita Universidad Autónoma de Puebla y la Facultad de Ciencias de la Electrónica que me dieron la oportunidad de estudiar el posgrado de Maestría en Ciencias de la Electrónica, Opción en Automatización. Finalmente, pero no menos importante, al CONAHCYT por el apoyo económico que me permitió continuar con mis estudios.

Resumen

En el presente documento se describe el desarrollo de un algoritmo de control neuro-inspirado para el control de un sistema multiagente no lineal de primer orden. El trabajo abarca la investigación, diseño, entrenamiento y desarrollo de una red neuronal del tipo Spiking (SNN), la cual utiliza un modelo más cercano a la red biológica en comparación con las redes neuronales artificiales tradicionales (ANN). Mediante su implementación, se desarrolla un control neuro-inspirado el cual es robusto ante perturbaciones externas al sistema. Además, se realiza un análisis comparativo entre la implementación de diferentes controladores neuronales, como lo son el SNN, RBF y un ANN que utiliza la función de activación sigmoide. Finalmente, se simula el control dentro de un contexto multiagente, en un sistema compuesto por dos agentes, cuya dinámica se ve afectada por perturbaciones. El algoritmo con una red RBF muestra una respuesta rápida, aproximándose a la posición deseada en un menor tiempo mientras que el algoritmo con una red Spiking tiene una respuesta más lenta, pero demuestra ser más robusto ante perturbaciones externas.

Índice general

Introducción

Antecedentes

Los sistemas multiagente permiten solucionar problemas complejos mediante la inteligencia distribuida y el trabajo colaborativo, sin embargo, su autonomía se ve limitada por las baterías, es por ello que actualmente se buscan nuevas formas de conseguir una mayor eficiencia energética. Las redes neuronales Spiking (SNN) han demostrado un bajo consumo de energía, al emular características biológicas del cerebro, las SNN son una alternativa energéticamente eficiente a las redes neuronales artificiales tradicionales (ANN) [1, 2].

Actualmente las aplicaciones de sistemas autónomos se han multiplicado exponencialmente en diversos ámbitos desde el militar al civil. Una categoría en pleno auge son los sistemas multiagentes (SMA), los cuales permiten realizar tareas complejas basadas en la inteligencia distribuida. En este sentido, los algoritmos de control han evolucionado a fin de realizar tareas más complejas mostrando una autonomía elevada. Los sistemas de control inteligente se distinguen por la plasticidad y desempeño frente a condiciones inesperadas. En este ramo, las redes neuronales de tipo Spiking son las más próximas a sus contra partes biológicas teniendo como ventajas un mayor desempeño con un bajo costo computacional y por ende un ahorro energético.

Estado del arte

Redes neuronales Spiking (SNN)

Las redes neuronales Spiking son consideradas redes de tercera generación, tienen como objetivo cerrar la brecha entre la neurociencia y el aprendizaje automático, utilizando modelos neuronales biológicamente realistas para llevar a cabo el cálculo. A continuación; se hace referencia a artículos recientes y sus principales áreas de aplicación.

Redes neuronales Spiking y sus aplicaciones: una revisión [3]

En este artículo podemos encontrar antecedentes de las redes neuronales Spiking, iniciando por la descripción de la red biológica para posteriormente describir el modelo de las redes Spiking y sus similitudes entre sí. Además, se hace mención sobre sus aplicaciones en visión por computadora y control robótico, en donde son utilizadas para clasificación de imágenes, detección de objetos, seguimiento de objetos, generación de patrones y control de motores. A

continuación, en la tabla 1 se presenta una breve descripción de los trabajos relacionados con algunas de las aplicaciones mencionadas anteriormente.

Método y año	**Paradigma de entrenamiento**	**Descripción**	**Desempeño**
Generación de patrones			
Cuevas-Arteaga et al. (2017)	PyNN/ SpiNNaker	Spiking CPG (sCPG) para ordenar a un robot hexápodo que camine, trote o corra.	
NeuroPod (2019)	SpiNNaker	La primera implementación sCPG neuromorfico en tiempo real que se ejecuta en SpiNNaker para ordenar a un robot hexápodo que camine, trote o corra.	
Control de motores			
Dupeyroux et al. (2020)	PySNN/ Loihi	Se trata de una aplicación totalmente embebida del prototipo de chip neuromórfico Loihi en un robot volador. Utiliza un algoritmo evolutivo para optimizar los controladores SNN y un entorno de simulación abstraído para su evaluación	La arquitectura de red resultante consta de solo 35 neuronas distribuidas entre tres capas. El análisis cuantitativo entre la simulación y Loihi revela un error cuadrático medio de la consigna de empuje tan bajo como 0.005 g, junto con una coincidencia del 99.8 % de las secuencias de picos en la capa oculta y el 99.7 % en la capa de salida.
Stagsted et al. (2020)	Nengo/ Loihi	Modificando la arquitectura de la SNN y mejorando la interfaz entre el chip neuromorfico y el servidor de la computadora, permitió mejorar la latencia y la frecuencia del controlador.	El controlador PID basado en SNN se probó en un dron limitado a girar sobre un solo eje. Se obtuvieron resultados comparables en los tiempos de sobre impulso, subida y estabilización
Clasificación de imágenes			
DCSNN (2018)	STDP y RSTDP	Se entrenó un SNN convolucional mediante una combinación de STDP para las capas inferiores y STDP modulado por recompensa para las superiores, lo que permite entrenar toda la red de una manera biológicamente plausible de extremo a extremo.	Han logrado una precisión del 97.2 % en el conjunto de datos del MNIST.
LM-SNNs (2020)	STDP	Se introdujo el modelo de red neuronal spiking de mapas de celosía (LM-SNN) con una regla de aprendizaje STDP modificada y un mecanismo de toma de decisiones inspirado en la biología. El algoritmo de aprendizaje en LM-SNN manifiesta un esquema de aprendizaje no supervisado.	Conjunto de datos: dígitos escritos a mano por el MNIST y una colección de instantáneas de Atari Breakout.

Método y año	**Paradigma de entrenamiento**	**Descripción**	**Desempeño**
Clasificación de imágenes			
Zhou et al. (2020)	STDP	Primer trabajo para aplicar SNN a una tarea de clasificación de imágenes médicas. Se utilizó un SNN convolucional basado en STDP para distinguir el melanoma de los nevus melanocíticos.	Conjunto de datos: ISIC 2018 incluye 1113 imágenes de MEL y 6705 imágenes de NV. Rendimiento: Sin selección de funciones: 83.8 % Con selección de funciones: 87.7 %
Lou et al. (2020)	STDP	Tanto las características temporales como espaciales de SNN se emplean para reconocer señales de EEG y clasificar estados emocionales. Los datos neuroinformáticos tanto espaciales como temporales se codificarán con las ubicaciones de sinapsis y neuronas, así como el momento de las actividades de pico.	74 %, 78 %, 80 % y 86.27 % para el conjunto de datos DEAP, y la precisión general es 96.67 % para el conjunto de datos SEED
Detección de objetos			
Spiking YOLO (2019)	Conversión de ANN a SNN	Spiking-YOLO se presentó como el primero en realizar detección de objetos con eficiencia energética. Propusieron la normalización por canales y firmaron neuronas con un umbral desequilibrado para convertir LeakyReLU de una manera biológicamente plausible.	Han logrado mAP % 51.83 en PASCAL VOC y mAP % 25.66 en MS COCO
SCNN profundo (2020)	Back-propagation	SNN basado en Complex-YOLO se aplicó en datos de nubes de puntos 3D adquiridos de un sensor LiDAR convirtiéndolos en datos de tiempo de pico. El modelo SNN propuesto en el artículo utilizó la neurona IF en forma de tiempo pico entrenada con retropropagación ordinaria.	Obtuvieron resultados comparativos en el conjunto de datos KITTI con vista de pájaro en comparación con Complex-YOLO con menos requisitos de energía. Escasez media del 56.24 % y consumo de energía total extremadamente bajo de solo 0.247 mJ.
Seguimiento de objetos			
SiamSNN (2020)	Conversión de ANN a SNN	SiamSNN, el primer SNN para el seguimiento de objetos que logra una latencia corta y una baja pérdida de precisión del SiamFC original, se introdujo junto con un método de estimación de similitud híbrido optimizado para SNN.	OTB-2013, OTB-2015 y VOT2016. SiamSNN logra 50 FPS y consumo de energía extremadamente bajo en TrueNorth.
Dos SNN multicapa (2020)	R-STDP	Esto abordó el problema del seguimiento de objetivos en movimiento basado en SNN en un robot serpiente sin ruedas. Se utiliza un sensor de visión dinámica (DVS) para percibir el objetivo y codificarlo como picos que se introducen en el SNN para impulsar el controlador de locomoción del robot serpiente.	Los experimentos de simulación realizados en el NRP. En comparación con SNN, la dirección relativa del objetivo al robot tiene menos fluctuaciones cuando se usa el SNN multicapa.

Tabla 1: Resumen de aplicaciones recientes de SNN en visión artificial y robótica (tabla tomada de [3]).

Redes neuronales Spiking para regresión no lineal [4]

En este artículo se hace referencia a la reducción en el consumo de memoria y energía que se obtiene utilizando redes neuronales de tipo Spiking en comparación con la redes neuronales tradicionales. Se plantea utilizar una red Spiking para la regresión y modelar con precisión materiales sometidos a tensiones que van mas allá de la reversibilidad, lo que constituye un tipo de no linealidad desafiante. Además se realiza una comparación directa con sus homólogos de las redes neuronales tradicionales, logrando reducir unas 120 veces el consumo energético.

Redes neuronales de tipo función de base radial (RBF)

Las redes neuronales de tipo función base radial son redes con una arquitectura sencilla, puesto que solo cuentan con 3 capas, una de entrada, una oculta y una de salida. Esto permite un obtener un aprendizaje rápido. Además, usan funciones gaussianas como función de activación. A continuación, se hace referencia a algunos artículos recientes relacionados con aplicaciones de control.

Control PID de red neuronal difusa basado en red neuronal RBF para naves espaciales de configuración variable [5]

Los autores proponen una variante de PID adaptativo, el cual hace uso de redes neuronales RBF para realizar la estimación del modelo dinámico y ajustar las ganancias. El uso de las redes RBF permite una mejor calidad de control en tiempo de convergencia, sobreimpulso y precisión en comparación con el controlador PID tradicional

Investigación sobre estrategia de control de modo deslizante de tiempo fijo basada en red neuronal RBF [6]

Los autores realizan una implementación híbrida con un control por modo deslizante adaptativo mediante redes neuronales RBF para lograr un control de alta precisión de la trayectoria final de un robot manipulador lo cual mejora el tiempo de convergencia.

Justificación

Las redes neuronales tratan de emular el funcionamiento de la mente humana mediante neuronas y sus interconexiones. Existen diferentes modelos que permiten solucionan problemas difíciles de resolver para algoritmos de control convencionales. Las redes neuronales tienen la capacidad de aprender, por lo que si en un sistema conocemos su entrada, el valor de salida deseado y el valor de salida actual, un controlador integrado por una red neuronal es capaz de modificar sus parámetros aprendiendo la dinámica de la planta hasta conseguir un sistema confiable. Por otro lado, el consumo energético es un factor importante en los sistemas móviles, principalmente si su aplicación requiere un tamaño reducido, al formar parte de un sistema

multiagente. Las redes neuronales de tipo Spiking buscan cerrar la brecha entre las redes neuronales artificiales y la red biológica, siendo más fácil de procesar y por ende obteniendo un ahorro energético. En la Figura 1 se muestra un diagrama representativo de la propuesta de solución, en donde, se obtiene información de posición del sistema; dicha información pasa a través de una red neuronal de tipo Spiking para realizar un control neuro-inspirado.

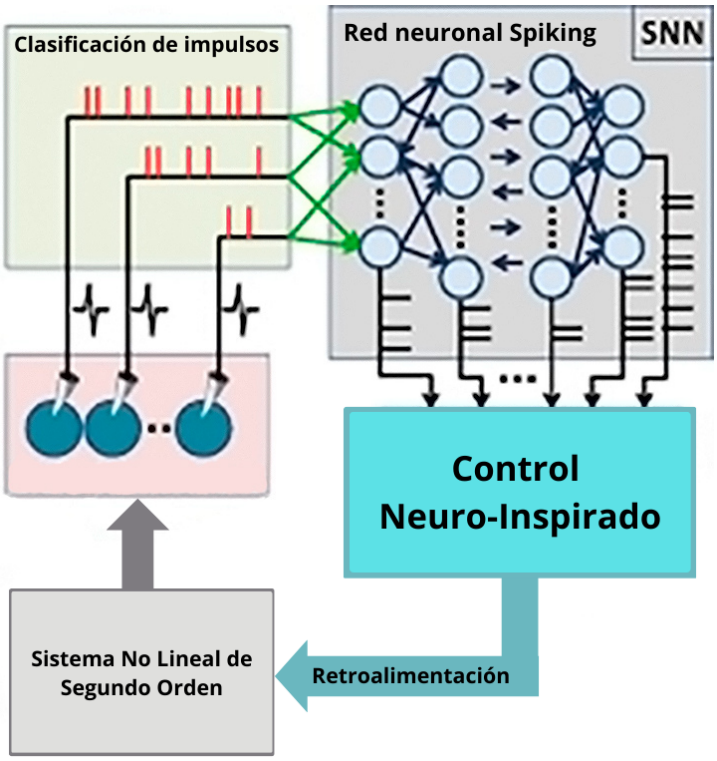

Figura 1: Diagrama representativo del trabajo de tesis.

Objetivos

Los objetivos general y específicos de este trabajo de tesis se enuncian a continuación.

Objetivo general

Diseñar y simular un controlador basado en redes neuronales de tipo Spiking para compensar incertidumbres no lineales.

Objetivos particulares

1. Realizar un estudio sobre las redes neuronales de tipo Spiking.
2. Simular redes neuronales de tipo Spiking.
3. Analizar e integrar las redes neuronales de tipo Spiking dentro de un contexto de control.
4. Simular un control neuro-inspirado aplicado sobre un sistema multiagente.

Capítulo 1

Marco teórico

Las redes neuronales artificiales (RNAs) tratan de emular el comportamiento del cerebro humano, caracterizado por el aprendizaje a través de la experiencia y la extracción de conocimiento genérico a partir de un conjunto de datos. Estos sistemas imitan esquemáticamente la estructura neuronal del cerebro, mediante un programa de ordenador (simulación), mediante su modelado a través de estructuras de procesamiento con cierta capacidad de cálculo paralelo (emulación), o bien mediante la construcción física de sistemas cuya arquitectura se aproxima a la estructura de la red neuronal biológica (implementación hardware de RNAs) [7].

En este capítulo se aborda de una manera general el fundamento teórico de las redes neuronales, iniciando por las principales características de la red biológica, para posteriormente hacer una revisión de los fundamentos de las redes neuronales artificiales y finalmente se revisa de manera particular el modelo de las redes neuronales de tipo Spiking. La investigación se realiza con el objetivo de estudiar y analizar las redes neuronales de tipo Spiking, pues se busca que esta sea la que finalmente se implemente en nuestro sistema.

1.1. Redes neuronales biológicas

El elemento fundamental de los sistemas neuronales biológicos es la neurona, una célula viva que, como tal, contiene todos los elementos que integran las células biológicas, si bien incorpora otros elementos que la diferencian. De forma genérica, una neurona consta de un cuerpo celular o soma más o menos esférico, del que parten una rama principal o axón y un denso árbol de ramificaciones más cortas (árbol dendrítico), compuesto por dendritas. A su vez, el axón puede ramificarse en su punto de arranque, y con frecuencia presenta múltiples ramas a su extremo. La forma final de que la neurona depende de la función que cumple, esto es, de la posición que ocupa en el conjunto del sistema y de los estímulos que recibe [7].

La misión de las neuronas comprende generalmente cinco funciones parciales [8]:

- Las neuronas recogen la información que llega a ellas en forma de impulsos procedentes de otras neuronas o de receptores.
- La integran en un código de activación propio de la célula.

- La transmiten codificada en forma de frecuencia de impulsos a través del axón.
- A través de sus ramificaciones el axón efectúa la distribución espacial de los mensajes.
- En sus terminales transmite los impulsos a las neuronas subsiguientes

La neurona, como toda célula, consta de una membrana exterior M, que limita y le sirve de órgano de intercambio con el medio exterior, de un citoplasma C, que es el cuerpo principal de la célula donde radica el grueso de sus funciones y de un núcleo N, que constituye el material genético de la célula. En la Figura 1.1 se muestra un esquema general de una neurona.

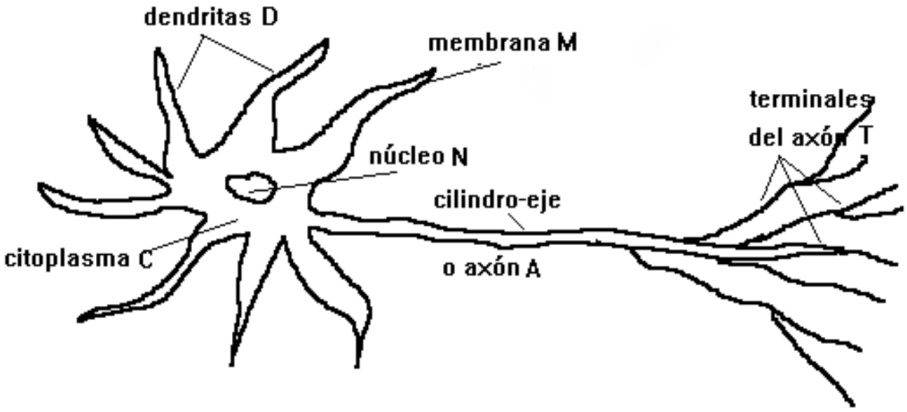

Figura 1.1: Esquema de una neurona (imagen tomada de [9]).

El citoplasma presenta unos alargamientos D, llamados dendritas, que son los órganos de recepción. En las dendritas termina un gran número de fibras F que son conductores que llevan la señal o impulso nervioso de los receptores o de otras neuronas hacia la neurona. Estas fibras terminan en un pequeño corpúsculo llamado sinapsis, que constituye un relevador bioquímico y que sirve para transferir la señal de una neurona a otra.

Existen dos clases de sinapsis; actuadoras, que favorecen el disparo de la neurona receptora e inhibidoras, que dificultan éste. Cuando se presenta un cierto desbalance entre las sinapsis actuadoras y las inhibidoras activas, la neurona dispara un impulso de salida, que constituye la respuesta de la neurona. Este impulso nervioso de salida es conducido por una propagación cilíndrica alargada (hasta de varios decímetros de largo) de la neurona, que se llama cilindro eje o axón A, que en su extremo se divide en varias fibras para comunicarse con otras neuronas órganos efectores o motores como glándulas o músculos. El citoplasma de las neuronas forma la masa gris de los centros nerviosos y el conjunto de cilindros ejes forma la masa blanca de aquellos.

En el cerebro humano la corteza cerebral contiene la mayor parte de las neuronas de este órgano y constituye, por ese hecho, la red neuronal natural más compleja. La corteza cerebral tiene un espesor promedio de 3 mm y una superficie de unos 2,000 cm^2. En ella se sitúan unos

cien mil millones de neuronas y de cien a mil billones de sinapsis. Es el centro superior de la memoria, del procesamiento de sensaciones, de la regulación motora, del lenguaje, de los afectos y de los mecanismos de inferencia. Esta situada en la periferia del cerebro, formado pliegues llamados circunvoluciones cerebrales. En la corteza hay zonas especializadas en determinado trabajo, llamadas zonas o localizaciones cerebrales. Aunque su definición aún está en estudio, se tiene identificadas claramente las zonas motoras, las zonas del lenguaje, así como las zonas sensitivas correspondientes a los diferentes sentidos.

Conjuntamente con los descubrimientos de los neurofisiólogos y neuroanatomistas, algunos psicólogos comenzaron a desarrollar modelos neuronales de aprendizaje. Uno de los que tuvo mayor reconocimiento fue el de Donal O. Hebb (1949), que propuso su ley de aprendizaje, que posteriormente fue el fundamento de los algoritmos de aprendizaje de las redes neuronales artificiales. Hebb postuló que si por alguna circunstancia dos neuronas se activan simultáneamente, se fortalecen las conexiones entre esas dos neuronas, haciendo que para una de ellas sea más fácil disparar, si la otra dispara. Esto explicaría la formación de reflejos condicionados en los animales.

Esta teoría Hebb dió lugar a nuevas teorías sobre la memoria y el aprendizaje. La parte del tejido nervioso más plástica es precisamente la sinapsis, donde la conductividad sináptica puede ser alterada por cambios en la composición química, en el tamaño y en la forma de dicha sinapsis. De esta forma puede explicarse la memoria como un conjunto de cambios en las sinapsis entre neuronas, en cuyo caso la información queda codificada por el número de interconexiones entre neuronas y el grado diferencial de conductividad de las sinapsis correspondientes [9].

1.2. Redes neuronales artificiales

El hombre se ha caracterizado siempre por su búsqueda constante de nuevas vías para mejorar sus condiciones de vida. Estos esfuerzos le han servido para reducir el trabajo en aquellas operaciones en las que la fuerza juega un papel primordial. Los progresos obtenidos han permitido dirigir estos esfuerzos a otros campos, por ejemplo, a la construcción de máquinas calculadoras que ayuden a resolver de forma automática y rápida determinadas operaciones que resultan tediosas cuando se realizan a mano.

Estas máquinas permiten implementar fácilmente algoritmos para resolver multitud de problemas que antes resultaban engorrosos de resolver. Sin embargo, se observa una limitación importante: ¿qué ocurre cuando el problema que se quiere resolver no admite un tratamiento algorítmico, como es el caso, por ejemplo, de la clasificación de objetos por rasgos comunes? Este ejemplo demuestra que la construcción de nuevas máquinas más versátiles requiere un enfoque del problema desde otro punto de vista. Los desarrollos actuales de los científicos se dirigen al estudio de las capacidades humanas como una fuente de nuevas ideas para el diseño de las nuevas máquinas. Así, la inteligencia artificial es un intento por descubrir y describir aspectos de la inteligencia humana que pueden ser simulados mediante máquinas. Esta disciplina se ha desarrollado fuertemente en los últimos años teniendo aplicación en algunos campos como visión artificial, demostración de teoremas, procesamiento de información expresada mediante

lenguajes humanos, etc.

Las redes neuronales artificiales son más que otra forma de emular ciertas características propias de los humanos, como la capacidad de memorizar y de asociar hechos. Si se examinan con atención aquellos problemas que no pueden expresarse a través de un algoritmo, se observará que todos ellos tienen una característica en común: la experiencia. El hombre es capaz de resolver estas situaciones acudiendo a la experiencia acumulada. Así, parece claro que una forma de aproximarse al problema consista en la construcción de sistemas que sean capaces de reproducir esta característica humana. En definitiva, las redes neuronales no son más que un modelo artificial y simplificado del cerebro humano, que es el ejemplo más perfecto del que disponemos para un sistema que es capaz de adquirir conocimiento a través de la experiencia. Una red neuronal es "un nuevo sistema para el tratamiento de la información, cuya unidad básica de procesamiento está inspirada en la célula fundamental del sistema nervioso humano: la neurona" [10].

En esta sección se describen las Redes Neuronales Artificiales (RNAs) y se describen algunas de sus ventajas, posteriormente se realiza un breve bosquejo histórico y finalmente, se describe la estructura y representación básica de las RNAs, así como de sus principales componentes.

1.2.1. Descripción y ventajas

Existen numerosas formas de definir a las redes neuronales artificiales; desde las definiciones cortas y genéricas hasta las que intentan explicar más detalladamente qué son las redes neuronales. Por ejemplo [10]:

1. Una nueva forma de computación, inspirada en modelos biológicos.
2. Un modelo matemático compuesto por un gran número de elementos procesales organizados en niveles.
3. Un sistema de computación compuesto por un gran número de elementos simples, elementos de procesos interconectados, los cuales procesan información por medio de su estado dinámico como respuesta a entradas externas.
4. Redes interconectadas masivamente en paralelo de elementos simples (usualmente adaptativos) y con organización jerárquica, las cuales intentan interactuar con los objetos del mundo real del mismo modo que lo hace el sistema nervioso biológico.

Debido a su constitución y a sus fundamentos, las redes neuronales artificiales presentan un gran número de características semejantes a las del cerebro. Por ejemplo, son capaces de aprender de la experiencia, de generalizar de casos anteriores a nuevos casos, de abstraer características esenciales a partir de entradas que representan información irrelevante, etc. Esto hace que ofrezcan numerosas ventajas y que este tipo de tecnología se esté aplicando en múltiples áreas. Entre las ventajas se incluyen [10]:

- Aprendizaje Adaptativo. Capacidad de aprender a realizar tareas basadas en un entrenamiento o en una experiencia inicial.

- Autoorganización. Una red neuronal puede crear su propia organización o representación de la información que recibe mediante una etapa de aprendizaje.
- Tolerancia a fallos. La destrucción parcial de una red conduce a una degradación de su estructura; sin embargo, algunas capacidades de la red se pueden retener, incluso sufriendo un gran daño.
- Operación en tiempo real. Los cómputos neuronales pueden ser realizados en paralelo; para esto se diseñan y fabrican máquinas con hardware especial para obtener esta capacidad.
- Fácil inserción dentro de la tecnología existente. Se pueden obtener chips especializados para redes neuronales que mejoran su capacidad en ciertas tareas. Ello facilitará la integración modular en los sistemas existentes.

1.2.2. Bosquejo histórico

En el año 1936, Alan Turing fue el primero en estudiar el cerebro como una forma de ver el mundo de la computación. Sin embargo, los primeros teóricos que concibieron los fundamentos de la computación neuronal fueron Warren McCulloch, un neurofisiólogo y Walter Pitts un matemático, quienes, en 1943, lanzaron una teoría acerca de la forma de trabajar de las neuronas. Ellos modelaron una red neuronal simple mediante circuitos eléctricos [10].

En la década de los 40's y principios de los 50's, varios investigadores entre los que destacan McCulloch y Pitts (1943), Householder y Landahl (1945), Kleene (1956), Von Neumann (1956) y Culbertson (1956) elaboraron modelos matemáticos de neuronas y Redes Neuronales. En la década de los 50, varios investigadores, entre ellos Farley y Clark (1954) y Rochester, Holland, Haibt y Duda (1956), combinaron los resultados obtenidos por los matemáticos, biólogos y los psicólogos y desarrollaron modelos de simulación en computadora de neuronas y Redes Neuronales, dando lugar a la forma actualmente más generalizada de trabajar con estos sistemas: su simulación mediante software, en una computadora digital común.

Pronto se obtuvieron éxitos muy promisorios. Frank Rosenblatt desarrolló el Perceptrón (Rosenblatt 1958), que fue la primera red neuronal artificial especificada con toda precisión y orientada computacionalmente. Como era una máquina que podía aprender y demostrar comportamiento adaptativo complejo, atrajo de inmediato la atención de los investigadores. Su procedimiento de convergencia de aprendizaje fue un avance definitivo sobre la teoría de Hebb. Asimismo, Rosenblatt desechó el enfoque de teóricos anteriores, que veían al cerebro como una computadora lógica. En vez de ello, lo consideró como un asociador y clasificador, cuya misión era asociar respuestas de clasificación a estímulos específicos. En 1962 Rosenblatt publicó su libro Principles of Neurodynamics (Rosenblatt 1962) en el que presentó formalmente el Perceptrón como modelo para construir Redes Neuronales artificiales.

Los perceptrones se aplicaron rápidamente a resolver problemas tales como la predicción climatológica, la interpretación de electrocardiogramas y otros. Tal parecía que se había hallado la clave para comprender el funcionamiento cerebral, emulando las Redes Neuronales naturales

mediante redes complejas de perceptrones.

Sin embargo, pronto se comprobó que las redes con una capa de perceptrones eran incapaces de resolver problemas tan simples como la simulación de una compuerta lógica de tipo OR exclusivo y, tras una investigación sobre las limitaciones de los perceptrones, Minsky y Pappert publicaron el libro Perceptrons (Minsky & Pappert 1969) donde se hacían patentes estas limitaciones. Como consecuencia, los fondos para nuevas investigaciones se congelaron y la mayoría de los investigadores reorientaron su objeto de estudio. Sólo un par de investigadores aislados como Teuvo Kohonen en Finlandia, Stephen Grossberg y James Anderson en Estados Unidos, continuaron sus esfuerzos en este campo, dando lugar lentamente a que, a través de los años, (Kohonen 1972), (Anderson 1972) y (Grossberg 1987), emergiera un nuevo cuerpo teórico alrededor de las Redes Neuronales multicapas, que superó las limitaciones encontradas por Minsky y dio nuevo impulso al desarrollo de Redes Neuronales artificiales [9].

Actualmente, se realizan y publican cada año nuevos trabajos sobre RNAs, surgen aplicaciones nuevas y se lanzan al mercado productos nuevos que implementan estos avances, tanto hardware como software.

1.2.3. Modelo básico de una neurona y una red neuronal artificial

La neurona artificial es una unidad procesadora con cuatro elementos funcionales, a continuación, se hace una breve descripción de cada elemento, además, en la Figura 1.2 se hace un bosquejo gráfico de dichos elementos.

1. El elemento **receptor**, a donde llegan una o varias señales de entrada x_i, que generalmente provienen de otras neuronas y que son atenuadas o amplificadas cada una de ellas con arreglo a un factor de peso w_i que constituye la conectividad entre la neurona fuente de donde provienen y la neurona de destino en cuestión.

2. El elemento **sumador**, que efectúa la suma algebraica ponderada de las señales de entrada, ponderándolas de acuerdo con su peso, aplicando la siguiente expresión:

$$S = \sum_i w_i x_i$$

3. El elemento de función **activadora**, que aplica una función no lineal de umbral (que frecuentemente es una función escalón o una curva logística) a la salida del sumador para decidir si la neurona se activa, disparando una salida o no.

4. El elemento de **salida** que es el que produce la señal, de acuerdo con el elemento anterior, que constituye la salida de la neurona.

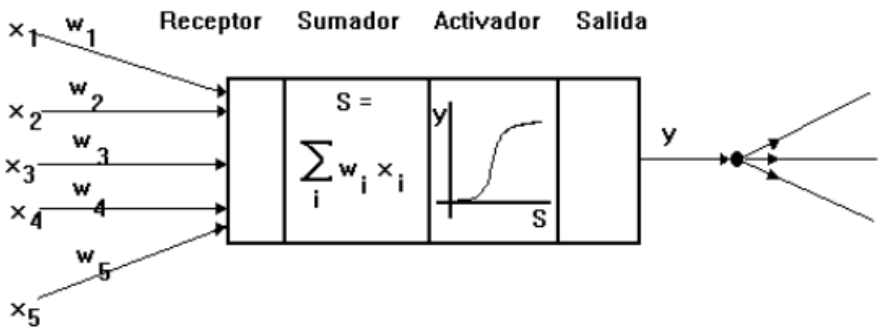

Figura 1.2: Esquema de una neurona artificial (imagen tomada de [9]).

Este modelo neuronal es el utilizado en casi todas las Redes Neuronales Artificiales, variando únicamente el tipo de función activadora [9].

Una Red Neuronal Artificial, está constituida por un conjunto de neuronas interconectadas y arregladas en tres capas (esto último puede variar). Los datos ingresan por medio de la "capa de entrada", pasan a través de la "capa oculta" y salen por la "capa de salida". Cabe mencionar que la capa oculta puede estar constituida por varias capas. A continuación, en la Figura 1.3, se presenta un esquema de una red neuronal.

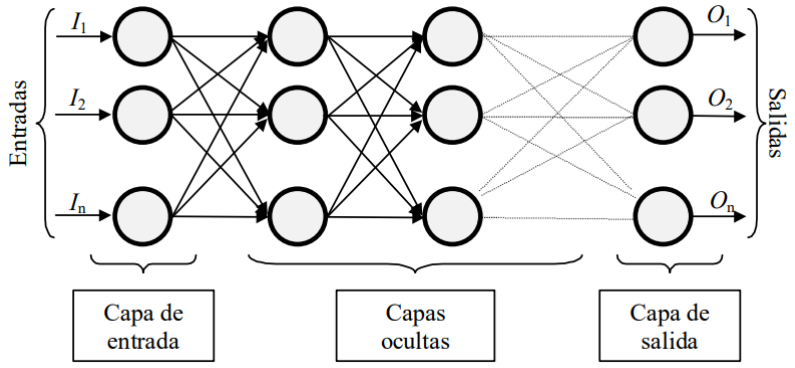

Figura 1.3: Esquema de una neurona artificial (imagen tomada de [10]).

1.3. Redes neuronales de tipo función base radial

Las redes neuronales de función base radial (RBF por sus siglas en inglés) han demostrado tener una buena capacidad de aproximación no lineal de manera rápida [6]. Las funciones de activación en una RBF son gaussianas, estas permiten modelar relaciones no lineales complejas en los datos [11]. Las RBF son potentes herramientas computacionales que se han utilizado

ampliamente en las áreas de reconocimiento de patrones, modelado e identificación de sistemas. Este tipo de redes cuentan con una velocidad de aprendizaje más rápida, lo que representa una gran ventaja.

En una red neuronal RBF, las neuronas en la capa oculta utilizan funciones radiales para medir la distancia existente entre los datos de entrada y ciertos puntos de referencia o centros. Estos centros representan puntos significativos y se utilizan para ponderar la influencia que tiene cada neurona de la capa oculta en la predicción o clasificación final. En la Figura 1.4 se presenta un esquema de las redes neuronales RBF, en general, siguen la siguiente arquitectura [11]:

1. Capa de entrada: En esta primera capa, los datos de entrada son proporcionados a la red.
2. Capa oculta: Se encuentra interconectada entre todos sus nodos con la capa de entrada y es activada a través de la función radial (gaussiana).
3. Capa de salida: Una vez que las neuronas de la capa oculta se activan, sus salidas se combinan ponderadamente para formar la salida de la red. Esta capa es activada a través de una función lineal continua.

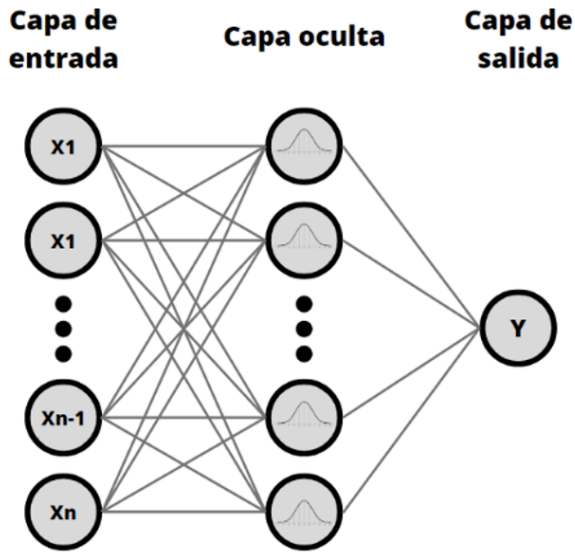

Figura 1.4: Esquema de una red neuronal de función base radial.

1.4. Redes neuronales Spiking

La última década ha sido testigo del gran éxito de las redes neuronales profundas en varios dominios. Sin embargo, las redes neuronales profundas consumen muchos recursos en términos de consumo de energía, requerimiento de datos y altos costos computacionales. Con la creciente necesidad reciente de autonomía de las maquinas en el mundo real, por ejemplo, vehículos autónomos, drones y robots colaborativos, se ha investigado activamente la explotación de redes neuronales profundas en esas aplicaciones. En estas aplicaciones, la eficiencia energética y

computacional son especialmente importantes debido a la necesidad de respuestas en tiempo real y al suministro de energía limitado. Una solución prometedora para estas aplicaciones previamente inviables ha sido proporcionada recientemente por las redes neuronales Spiking biológicamente plausibles. Las redes neuronales Spiking tienen como objetivo cerrar la brecha entre la neurociencia y el aprendizaje automático, utilizando modelos neuronales biológicamente realistas para llevar a cabo el cálculo. Debido a su similitud funcional con la red neuronal biológica, las redes neuronales Spiking pueden abarcar la escasez que se encuentra en la biología y son altamente compartibles con el código temporal [3].

1.4.1. Comparación entre ANN y SNN

Aunque históricamente las redes neuronales artificiales (ANNs) convencionales y las Redes Neuronales Profundas (DNNs) están inspiradas en el cerebro, son fundamentalmente diferentes en estructura, cálculos neuronales y reglas de aprendizaje en comparación con la red neuronal biológica. Esta observación conduce a las redes neuronales Spiking (SNNs), que a menudo se denominan la tercera generación de redes neuronales que podrían ser un gran avance de los cuellos de botella de las ANNs [3].

En comparación con las ANNs convencionales y DNNs que usan valores analógicos para representar activaciones dentro de las redes, las SNNs imitan neuronas biológicas reales y codifican la activación con la información de tiempo de los picos de neuronas. A diferencia de las ANNs, donde las operaciones principales son la multiplicación matricial de pesos y la activación de capas de red, la naturaleza de picos de las SNNs evita multiplicaciones complejas de matriz y, por lo tanto, requieren menos recursos de computación y son más eficientes energéticamente [12]. En los SNNs, como en las redes neuronales biológicas, las neuronas se comunican entre sí con señales eléctricas discretas llamadas picos y funcionan en tiempo continuo.

Debido a su similitud funcional con las redes neuronales biológicas, los SNNs pueden abarcar la escasez que se encuentra en la biología y son altamente compatibles con el código temporal [13]. Si bien los SNNs todavía están por debajo de los DNNs en términos de rendimiento, la brecha se está desvaneciendo en algunas tareas, mientras que los SNNs generalmente requieren mucha menos energía para la operación. Sin embargo, los SNNs siguen siendo difíciles de entrenar en general, principalmente debido a su compleja dinámica de neuronas y la naturaleza no diferenciable de las operaciones de picos. En la Tabla 1.1 se proporciona una comparación entre redes neuronales biológicas, ANNs y SNNs. BP (backpropagation) es retro propagación y VLSI (very large-scale integration) es integración a gran escala. Mientras que en la Figura 1.5, se presenta una comparación esquemática entre la neurona biológica, la neurona artificial y la neurona Spiking [3].

Propiedades	NNs biológicas	ANNs	SNNs
Representación de información	Spikes	Escalares	Spikes
Paradigma de aprendizaje	Plasticidad sinaptica	BP	Plasticidad/BP
Plataforma	Cerebro	VSLI	VSLI neuromórfico

Tabla 1.1: comparación de propiedades entre redes neuronales biológicas, ANN y SNN (tabla tomada de [3]).

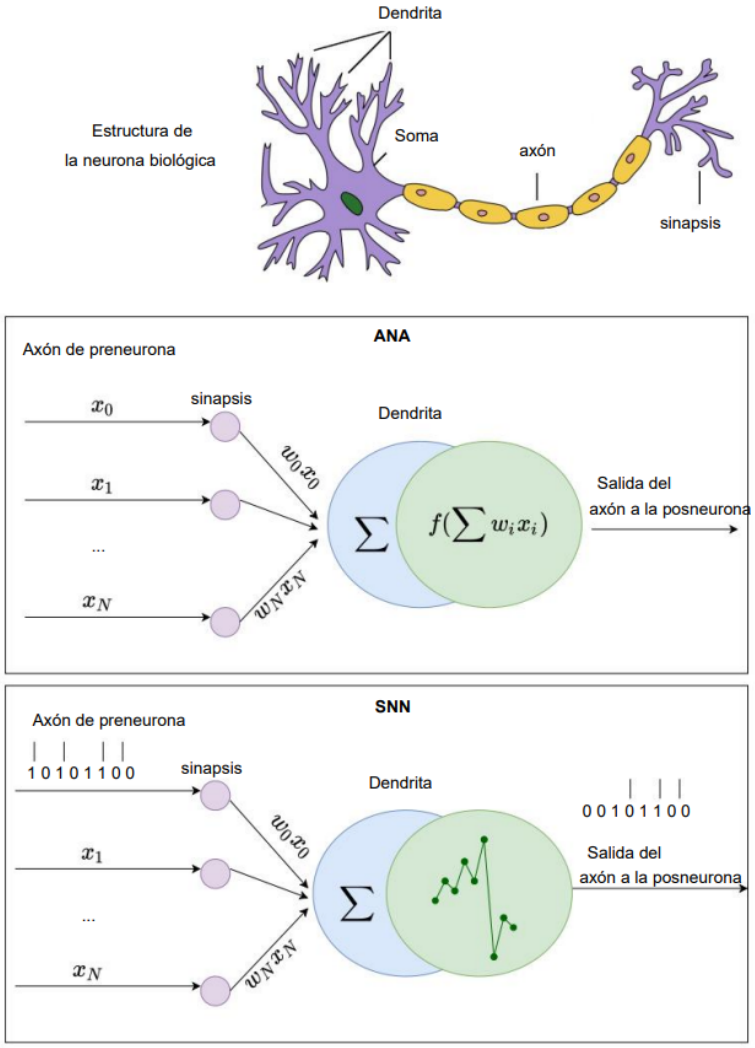

Figura 1.5: Comparación entre la neurona biológica, la neurona artificial y la neurona Spiking (imagen tomada de [3]).

1.4.2. Modelo básico de Redes Neuronales Spiking

Las SNN están construidas con neuronas que están conectadas con sinapsis ponderadas para formar capas y redes. Como componente básico de las SNN, cada neurona Spiking mantiene un valor para representar su estado actual. Las entradas a estas neuronas son picos transmitidos a través de sinapsis. Las neuronas recogen estos picos de entrada e integran los pesos correspondientes para actualizar sus estados internos. Cada vez que el estado alcanza un determinado valor de umbral, la neurona se activa y envía un pico a sus neuronas sucesoras. Después de un retraso de propagación preconfigurado, los sucesores reciben el pico y actualizan sus estados en consecuencia. Además de estos principios operativos comunes, existen diferentes modelos neuronales que presentan diferentes comportamientos de neuronas. Uno de los modelos neuronales más utilizados es el modelo "Leaky Integrate-and-Fire" (LIF), o bien, "Integración y Disparo con Fugas" si se traduce al español. Las neuronas LIF se inspiran en la fuga de voltaje de la membrana en las neuronas biológicas, y sus estados disminuyen con el tiempo si no hay picos de entrada presentes [12].

Particularmente en esta sección se analiza el modelo LIF, que simula la fuga de voltaje de la membrana de las neuronas biológicas con un proceso exponencial:

$$V(t_2) = V(t_1)e^{-(t_2-t_1)/\tau}, \tag{1.1}$$

donde $V(t)$ son los estados de la neurona en el tiempo t (el voltaje de la membrana en las neuronas biológicas) y τ es la constante de fuga. Para el algoritmo de actualización por pasos de tiempo, los valores de t son discretos y el término $t_1 - t_2$ será una constante Δt. Para el algoritmo impulsado por eventos, t_1 es la marca de tiempo del evento anterior que actualizó esta neurona y t_2 es la marca de tiempo del evento actual.

El estado de la neurona también cambia por la corriente de entrada de otras neuronas. Se asigna un peso $w_{i,j}$ a la sinapsis de la neurona i a la neurona j. Cuando la neurona i genera un pico de salida, $w_{i,j}$ se añade al estado de la neurona j después de un retraso fijo. Por lo tanto, para el algoritmo de actualización por pasos de tiempo, la ecuación de actualización de la neurona j desde el paso de tiempo t_n hasta t_{n+1} es:

$$V(t_{n+1}) = V(t_n)e^{-\Delta t/\tau} + \sum_{i=0}^{m-1} s_i(t_n)w_{i,j}, \tag{1.2}$$

donde m es la entrada de la neurona j, y $s_i(t_n)$ es el estado de pico de la neurona i en el paso de tiempo t_n. El estado de pico s_i es 1 si la neurona i emite un pico, de lo contrario es 0,

$$s_i(t_{n+1}) = \begin{cases} 1, & si \quad V(t_n) \geq V_{th} \\ 0, & si \quad V(t_n) < V_{th} \end{cases} . \tag{1.3}$$

Tras actualizar los estados de las neuronas, se utiliza un valor umbral V_{th} para decidir si una neurona se activa y emite un pico. Después de que una neurona se active, su estado se restablece a un valor V_{reset} [12].

Capítulo 2

Simulación de una red neuronal de tipo Spiking

En este capítulo se describe el proceso para la simulación de una red neuronal de tipo Spiking, basada en el modelo LIF. Utilizando el lenguaje de programación Python, se realiza la simulación de una neurona de tipo Spiking, posteriormente, se generan funciones las cuales permiten estructurar una red neuronal Spiking utilizando las neuronas previamente diseñadas. Finalmente, se realizan las primeras pruebas de la propagación hacia adelante de una SNN.

2.1. Neurona modelo LIF

En el capítulo anterior se describió de manera general el modelo LIF (*Leaky Integrate-and Fire*), el cual es un modelo simplificado de una neurona biológica. El modelo LIF se basa en la idea de que una neurona puede ser vista como un circuito eléctrico simple, donde la entrada de señales se integra a lo largo del tiempo y, cuando el voltaje de membrana alcanza un cierto umbral, se dispara un potencial de acción (spike) que transmite una señal a otras neuronas conectadas.

El modelo LIF tiene tres componentes principales:

1. Una entrada de corriente: esta entrada representa la actividad de otras neuronas que se conectan a la neurona LIF a través de sinapsis. La entrada de corriente se puede modelar como una función de tiempo que simula la actividad de otras neuronas.

2. Una capacitancia de membrana: esta capacitancia representa la capacidad de la membrana celular para almacenar cargas eléctricas. La capacitancia de membrana se carga con la entrada de corriente y se descarga a través de una resistencia de fuga.

3. Una resistencia de fuga: esta resistencia representa las corrientes de escape que fluyen a través de la membrana celular debido a las propiedades eléctricas de la membrana. Esta resistencia permite que la carga eléctrica se escape gradualmente de la membrana, lo que evita que el voltaje de membrana se vuelva demasiado grande.

Matemáticamente, el modelo LIF toma en cuenta una variable de potencial $V(t)$ que representa el potencial de la membrana. La dinámica de la membrana modelada mediante un circuito RC, puede ser representada de acuerdo a la siguiente ecuación diferencial:

$$\tau \frac{dV(t)}{dt} = -V(t) + I_{in}(t)R, \tag{2.1}$$

en donde τ es la constante de tiempo de la neurona y controla la rapidez con la que la neurona integra sus entradas. La solución de la ecuación (2.1) para una entrada de corriente constante es:

$$V(t) = I_{in}(t)R + [V_0(t) - I_{in}(t)R]e^{-\frac{t}{\tau}}. \tag{2.2}$$

En ausencia de cualquier entrada, la expresión (2.2) se simplifica, obteniendo:

$$V(t) = V_0(t)e^{-\frac{t}{\tau}}. \tag{2.3}$$

Esto significa que el potencial de la neurona decae exponencialmente al valor 0, y cuanto menor es τ, más rápido lo hace.

Ahora bien, sabemos que la entrada es ponderada, por lo que depende directamente de su peso sináptico w y al únicamente recibir Spikes de entrada (valor 1), la entrada puede ser representada únicamente por el peso. Por tanto, al presentarse una entrada, dentro del código el potencial se comportará de la siguiente manera:

$$V_{siguiente} = V_{actual} + w. \tag{2.4}$$

Si uno de los Spikes de entrada hace que el potencial actual (V_{actual}) cruce un valor de umbral (V_{umbral}), entonces la neurona disparará un pico o spike de salida y se restablecerá instantáneamente a su valor de reinicio ($V_{reinicio}$).

2.1.1. Pseudocódigo - Neurona Spiking

De manera general, para integrar las expresiones anteriores dentro de un código, se sigue el procedimiento descrito en el siguiente pseudocódigo.

	Pseudocódigo para una neurona Spiking.
1	Para cada t en el intervalo de tiempo T:
2	Actualizar el valor de V en el tiempo t
3	Si existe una entrada:
4	V = V + w
5	Si V > Umbral
6	Emitir un pulso en la salida
7	Restablecer V a su valor de reinicio

El comportamiento de la neurona se puede visualizar en la Figura 2.1, mientras que en la sección 2.4.1 se realiza un análisis de resultados.

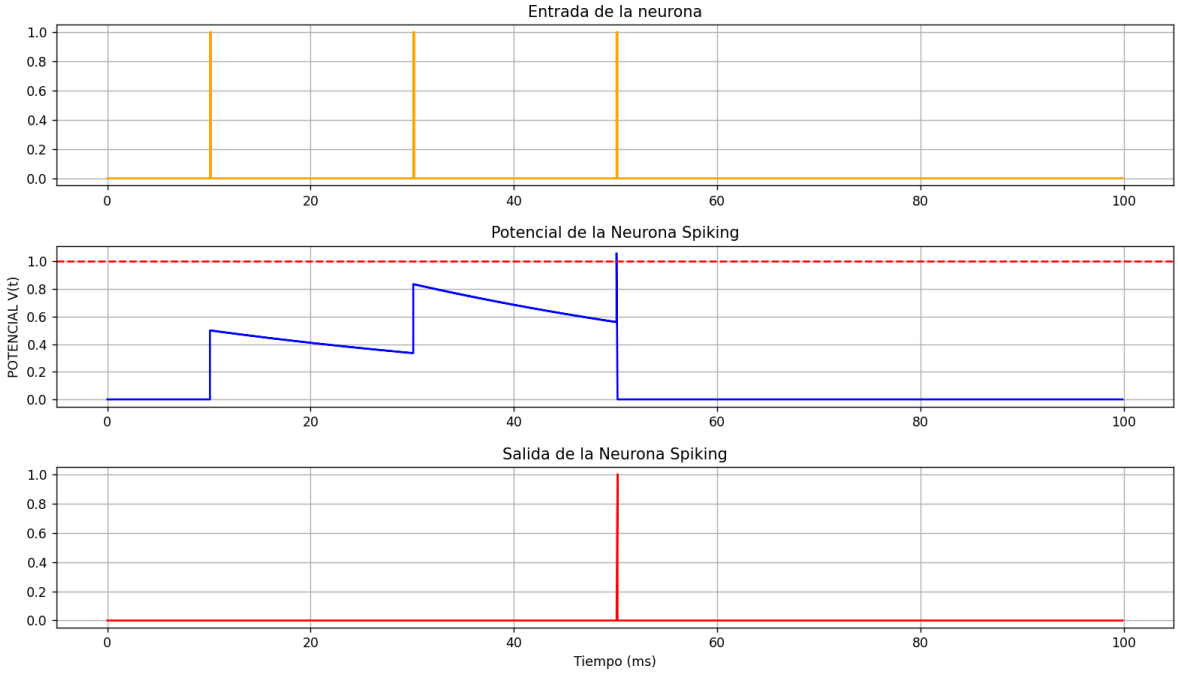

Figura 2.1: Comportamiento de la neurona Spiking.

El comportamiento de la neurona puede ser alterado si se modifica el umbral. En algunas de las aplicaciones puede implementarse incluso un umbral dinámico, en donde V_{umbral} deja de ser una constante y pasa a ser una variable que depende del tiempo, por lo que mientras el valor de umbral sea mayor, será más difícil producir un pico a la salida. La dinámica del umbral (ahora V_u) se encuentra determinada por la siguiente ecuación diferencial.

$$\tau_u \frac{dV_u}{dt} = V_{u_{min}} - V_u, \tag{2.5}$$

en donde $V_{u_{min}}$ es el valor de umbral mínimo de la neurona y τ_u es la constante de adaptación del umbral. Adicionalmente, cada que se emita un pico de salida el valor el valor de umbral crece un valor δ. Por lo tanto, al superarse el umbral mínimo, dentro del código el umbral dinámico se comportará de la siguiente manera:

$$V_{u_{siguiente}} = (V_{u_{actual}} - V_{u_{min}})e^{-\frac{t}{\tau_u}} + V_{u_{min}}. \tag{2.6}$$

A continuación, se presenta un pseudocódigo para una neurona Spiking con umbral dinámico.

2.1.2. Pseudocódigo - Neurona Spiking con umbral dinámico

De manera general, para integrar las expresiones anteriores dentro de un código, se sigue el procedimiento descrito en el siguiente pseudocódigo.

	Pseudocódigo para una neurona Spiking con umbral dinámico.
1	Para cada t en el intervalo de tiempo T:
2	Actualizar el valor de V en el tiempo t
3	Actualizar el valor de V_u en el tiempo t
4	Si existe una entrada:
5	$V = V + w$
6	Si $V > V_u$
7	Emitir un pulso en la salida
8	$V_u = V_u + \delta$
9	Restablecer V a su valor de reinicio

El comportamiento de la neurona se puede visualizar en la Figura 2.9, mientras que en la sección 2.4.1 se realiza un análisis de resultados.

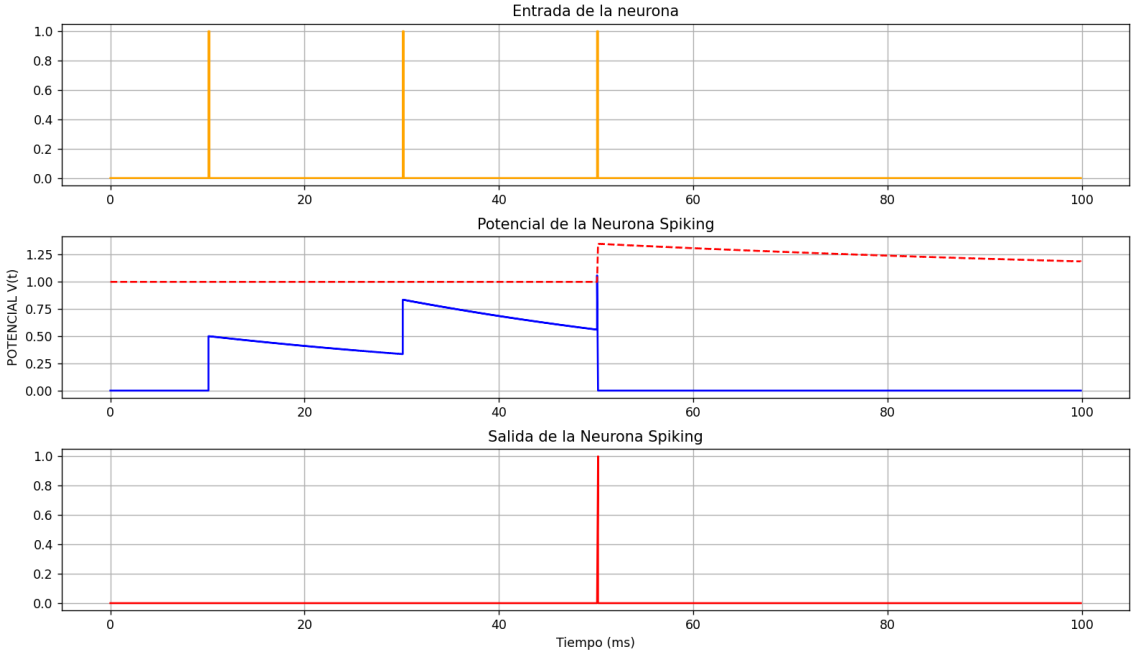

Figura 2.2: Comportamiento de la neurona Spiking con umbral dinámico.

2.2. Red Neuronal Spiking usando el modelo LIF

Las redes neuronales son algoritmos inspirados en el funcionamiento de la mente humana al procesar información. Las redes están compuestas por un conjunto de unidades llamadas neuronas, las cuales se agrupan en capas y en donde cada neurona de la capa se encuentra conectada a todas las neuronas de la capa anterior. Cada una de estas neuronas procesa la información de manera diferente y mediante un correcto entrenamiento, las redes son capaces de resolver gran número de problemas.

Las redes neuronales Spiking, basadas en el modelo LIF representan un nuevo enfoque en el campo de la inteligencia artificial. De la misma manera que en una red convencional, ésta es compuesta por múltiples neuronas interconectadas, en donde cada neurona sigue los procesos de integración y disparo descritos en la sección 2.1. La comunicación entre las neuronas se realiza únicamente a través de la transmisión de Spikes de una neurona a otra.

Una ventaja clave de las redes neuronales Spiking es su capacidad para representar y procesar información temporal de manera eficiente, lo cual impacta directamente en su consumo energético, pues al utilizar únicamente Spikes para transmitir información y realizar cálculos, las redes neuronales Spiking pueden reducir la cantidad de operaciones necesarias en comparación con los modelos tradicionales, lo que las hace atractivas para aplicaciones de baja potencia o en dispositivos con limitaciones energéticas.

En la sección anterior se realizó la implementación de una neurona de tipo Spiking utilizando el lenguaje de programación Python, por lo que, para poder estructurar la red, cada unidad será simulada en base al código generado previamente. La red seguirá la estructura mostrada en la Figura 2.3 y de manera general, funciona de la siguiente manera:

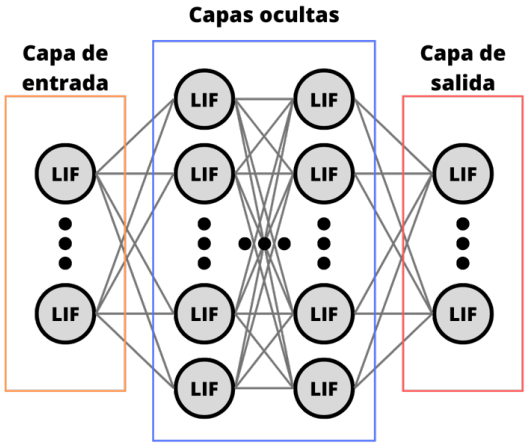

Figura 2.3: Estructura de una red neuronal Spiking.

- **Capa de entrada:** En esta capa, cada neurona recibe los datos iniciales que determinaran el comportamiento de la red, en base a la dinámica de cada neurona se transmitirá determinado número de Spikes a la capa siguiente.
- **Capas ocultas:** Este es un conjunto de una o más capas, en donde se encuentra el mayor número de neuronas. Cada neurona realiza una suma ponderada de la capa anterior y transmite sus Spikes de salida a la capa siguiente.
- **Capa de salida:** Finalmente, en esta capa se colocan las neuronas necesarias para construir la salida requerida por el sistema. Las entradas de esta capa son las salidas de la última capa oculta y de la misma manera, los Spikes de salida son determinados por la dinámica de cada neurona.

2.2.1. Pseudocódigo - Red Spiking

De manera general, para integrar las expresiones anteriores dentro de un código, se sigue el procedimiento descrito en el siguiente pseudocódigo.

	Pseudocódigo para una red neuronal Spiking.
1	Definir los parámetros de la red (neuronas, capas)
2	Definir parámetros de simulación (tau, *reset*, delta t, entradas)
3	Generar un vector de pesos aleatorios
4	Para cada capa en la red:
5	Para cada neurona en la capa:
6	Realizar la suma de todas las entradas
7	Llamar la función que simula una neurona Spiking
8	Generar un nuevo vector de salidas (entradas de la nueva capa)

El comportamiento de una red neuronal Spiking de 10 neuronas, se puede visualizar en la Figura 2.4, mientras que en la sección 2.4.2 se realiza un análisis de resultados de la red actual y de otras redes generadas con diferente número de neuronas.

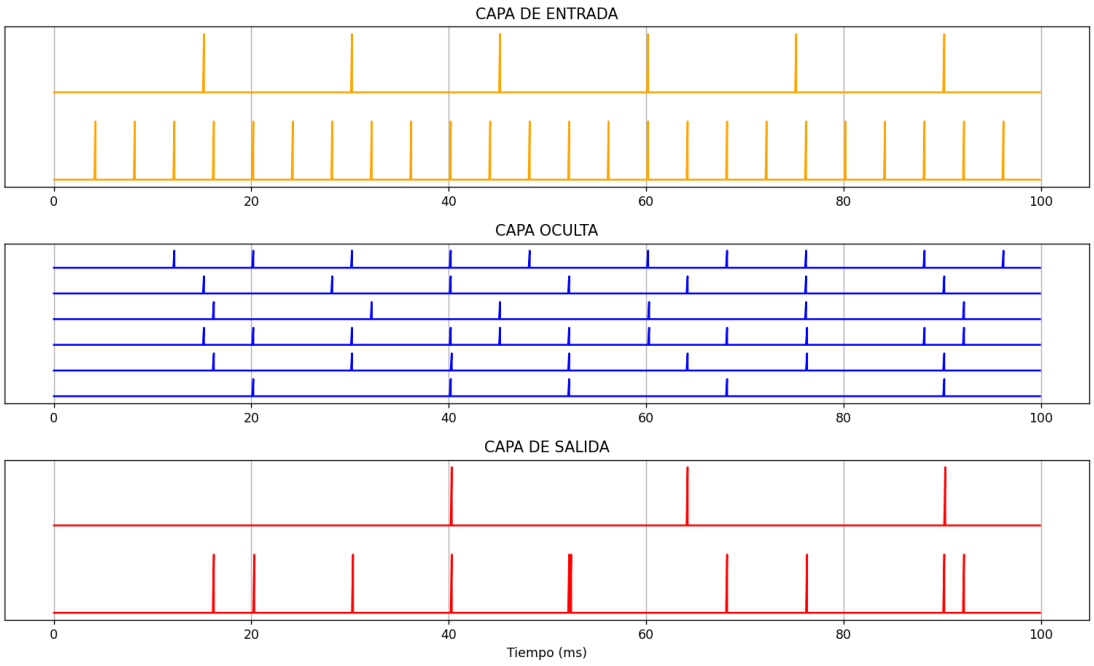

Figura 2.4: Comportamiento de una red neuronal Spiking formada de 10 neuronas.

2.3. Codificación Neuronal en una Red Spiking

Como ya se ha mencionado, las redes neuronales Spiking se inspiran en el funcionamiento de la mente humana, en donde la comunicación entre neuronas se realiza a través de pulsos conocidos como *Spikes*, estas redes son especialmente adecuadas para tareas que involucran procesamiento de señales en tiempo real, sin embargo, la interpretación de sus entradas y salidas no es tan simple y requiere de algo llamado codificación. La codificación en las SNN implica representar la información de entrada como patrones de espigas y diseñar algoritmos de decodificación que extraigan información relevante de estas señales. Las SNN se centran en el momento y la frecuencia de las espigas generadas por las neuronas para transmitir y procesar información.

En el articulo [14] se realizó un estudio comparativo sobre el impacto y rendimiento de cuatro importantes esquemas de codificación neuronal. Se consideraron varios aspectos del rendimiento de la red, incluyendo la precisión de la clasificación, la latencia del procesamiento, las operaciones sinápticas, la implementación del hardware, la eficacia de la compresión de la red, la resistencia al ruido sináptico y de entrada, y la tolerancia a los fallos sinápticos. Los cuatro esquemas de codificación se describen a continuación:

1. **Rate Coding**: Este modelo utiliza el numero de pulsos emitidos para representar la información. Es uno de los más comunes debido a su simplicidad y robustez. Sin embargo, este esquema de codificación se ve limitado por un largo periodo de procesamiento y una lenta transmisión de información.

2. **Time-to-First-Spike Coding**: Este modelo busca trasmitir la información desde el primer pulso de salida, obteniendo una respuesta muy rápida. En otras palabras, en este modelo los primeros pulsos tienen mayor valor, teniendo una función que decae de manera exponencial. De esta manera se reduce el número de Spikes y se aumenta la velocidad de respuesta.

3. **Phase Coding**: Este modelo codifica la información de manera binaria, en donde a cada spike se le asigna un peso diferente, el cual depende del rango de tiempo en el que el pulso es producido, los rangos están determinados por el numero de fases. El peso se determina como 2^{-n}, en donde n es la fase actual.

4. **Burst Coding**: La codificación en ráfaga permite una transmisión de información rápida y eficiente mediante el envío de una ráfaga de picos de una sola vez. Este modelo toma en cuenta el numero de pulsos y el intervalo entre los mismos.

En la Figura 2.5 se ilustra el funcionamiento de cada uno de estos modelos. De acuerdo a los resultados que obtuvieron en [14], el modelo "Time-to-First-Spike Coding" (TTFS) es la mejor opción para conseguir el mejor rendimiento computacional. En la sección 2.4.3, se muestran los resultados obtenidos para una de las redes desarrolladas utilizando los modelos "Rate Coding" y TTFS.

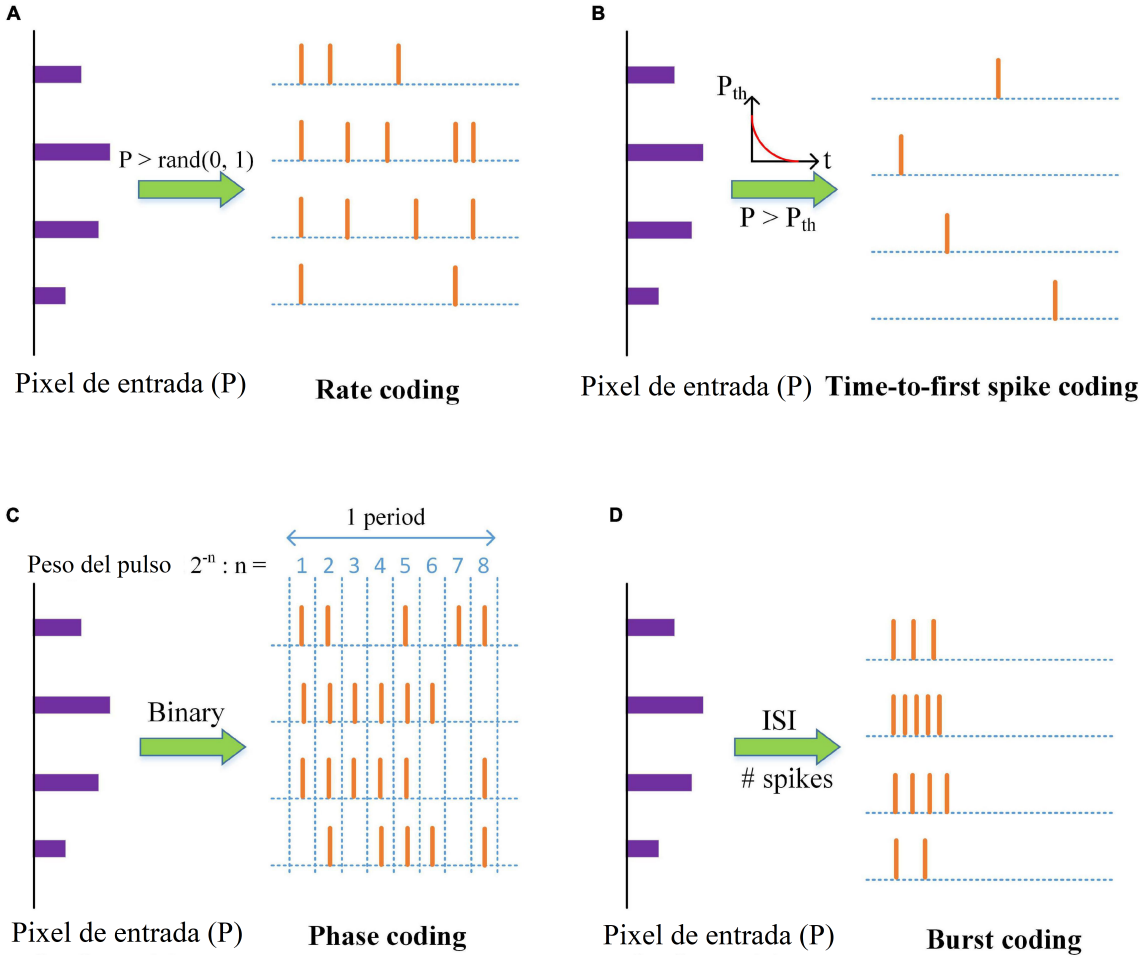

Figura 2.5: Ilustración de los esquemas neuronales de codificación (imagen tomada de [14]).

2.4. Resultados de simulación (propagación hacia adelante)

2.4.1. Neurona Spiking

Variaciones en τ

El funcionamiento de este tipo de neuronas artificiales depende especialmente del valor de la constante τ, usualmente su valor se encuentra en el orden de 1 - 100 ms. Cuando τ es grande la neurona actúa como integrador, sumando sus entradas y emitiendo spike a la salida cuando se sobrepasa el umbral establecido. En la Figura 2.6, se muestra una neurona con entradas cada 5 ms y un $\tau = 100$ ms. Se puede visualizar que al ser muy lento el efecto de descarga en la neurona, efectivamente se comporta como un integrador y emite un spike de salida por cada 3 entradas.

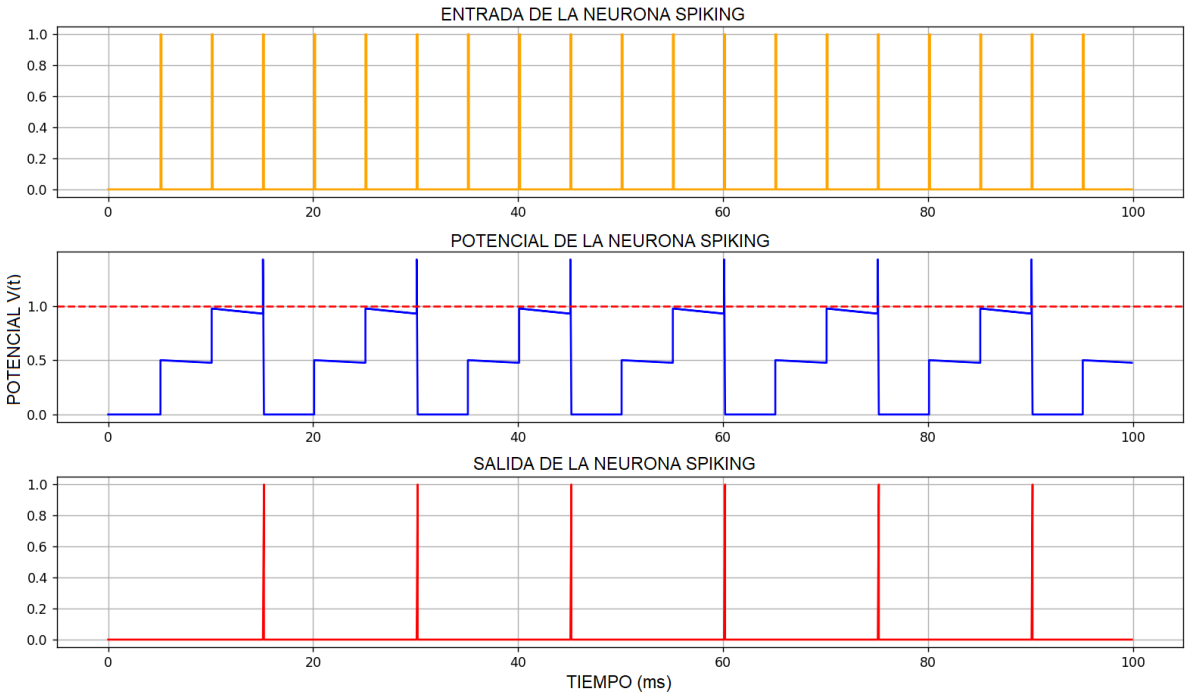

Figura 2.6: Comportamiento de una neurona Spiking, con $\tau = 100$.

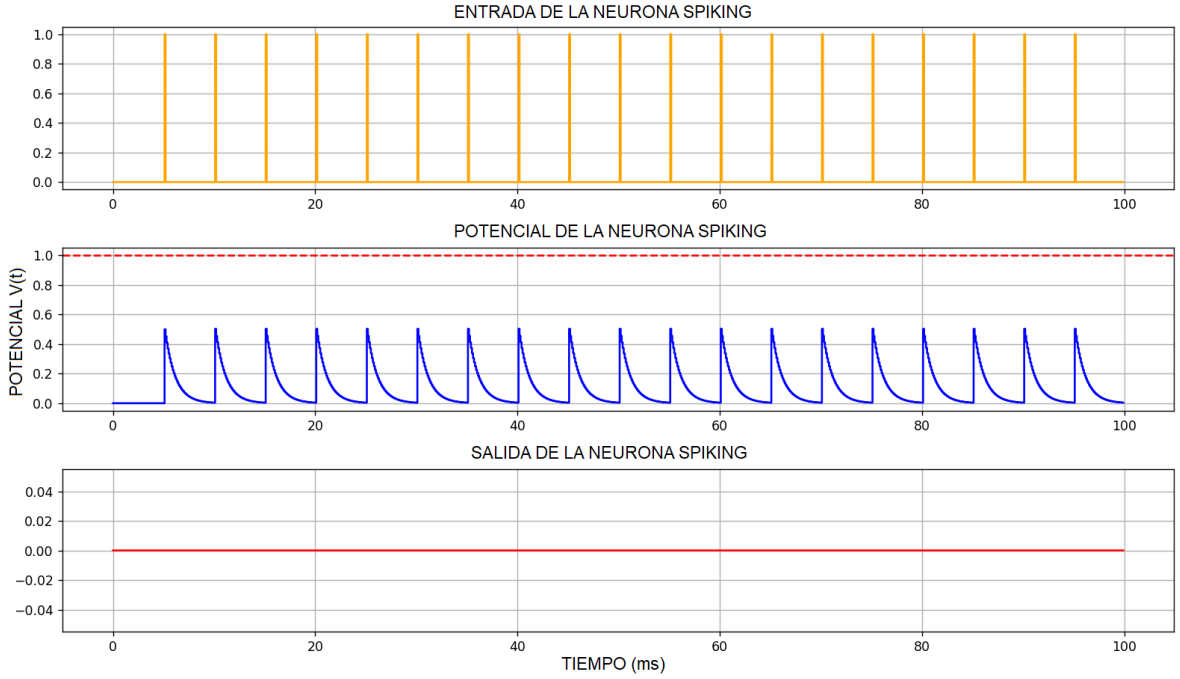

Figura 2.7: Comportamiento de una neurona Spiking, con $\tau = 1$.

Por otro lado, en la Figura 2.7 se puede visualizar el comportamiento de una neurona

igualmente con entradas cada 5 ms, pero esta vez con un $\tau = 1$ ms. Se puede visualizar que al ser muy rápido el efecto de descarga en la neurona, esta no alcanza a emitir ningún Spike de salida. Usualmente, con un τ pequeño, la neurona se puede utilizar como un detector de coincidencias, pues únicamente disparará un Spike de salida si llegan dos o más entradas simultáneamente.

Variaciones en el valor de *reset*

El valor de *reset* es otro parámetro importante que nos permitirá configurar a la neurona adecuadamente dependiendo de la aplicación. En la Figura 2.8 se presenta el comportamiento de la neurona con entradas cada 5 ms, $\tau = 50$ ms y con un valor de *reset* igual a 0.3. Se puede observar que se modifica el comportamiento de la neurona a partir del primer pico de salida emitido, pues pasa de disparar cada 3 entradas a cada 2 entradas.

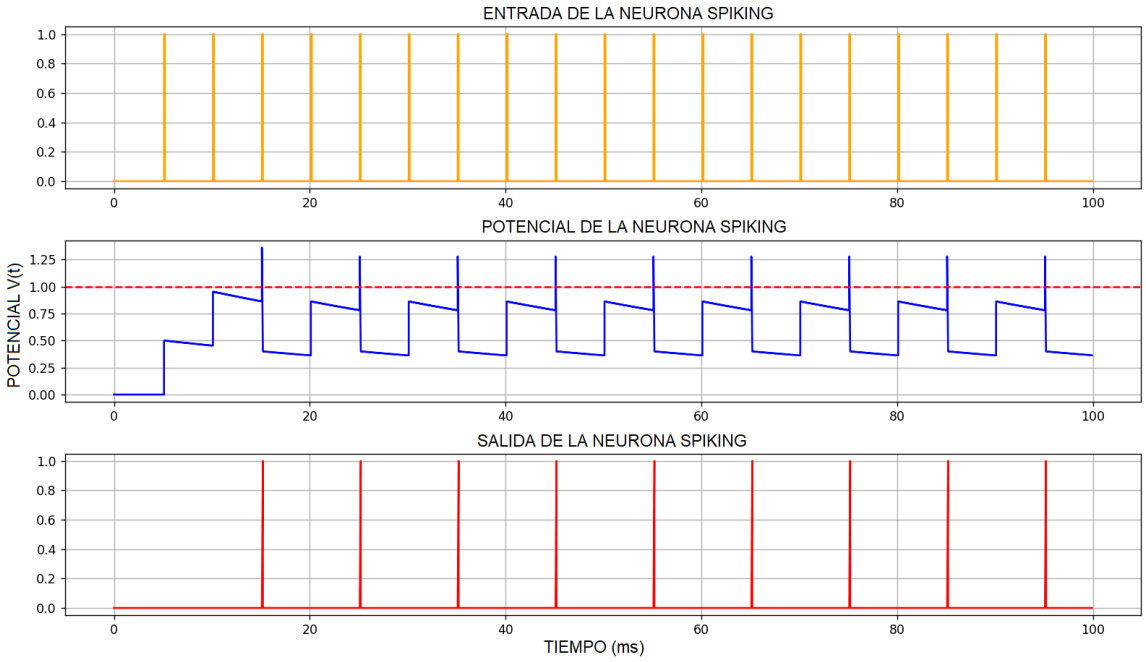

Figura 2.8: Comportamiento de una neurona Spiking, con $V_{reset} = 0.3$.

Neurona con umbral dinámico

De manera similar, se puede alterar el comportamiento de la neurona modificando el umbral. Como se explica en la sección 1.1, algunas aplicaciones pueden llegar a requerir un umbral dinámico. En la Figura 2.9 se presenta el comportamiento de una neurona Spiking con umbral dinámico, en donde se presentan entradas cada 5 ms, con $\tau = 50$ para la neurona y $\tau_u = 80$ ms para el umbral, además, se utiliza $V_{u_{min}} = 1$ y $\delta = 0.35$. Se puede visualizar un total de 5 picos de salida, en donde para los primeros dos picos de salida emitidos se requiere de tan sólo 3 picos de entrada, para los siguientes dos se requieren 4 y para el último se requiere un total de 5 picos a la entrada para poder emitir una salida, por lo que cada vez se hace más costoso emitir un pico a la salida.

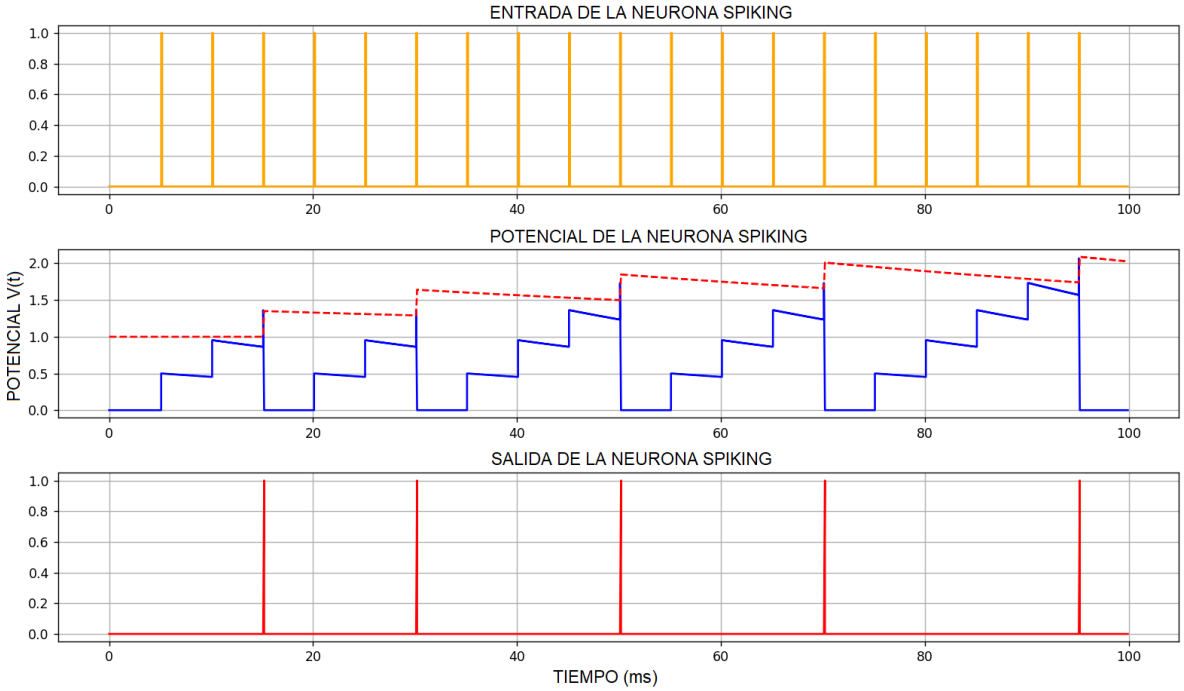

Figura 2.9: Comportamiento de la neurona Spiking con umbral dinámico.

2.4.2. Red Neuronal Spiking

Red con 10 neuronas

El funcionamiento de este tipo de redes neuronales depende principalmente de dos parámetros, el primero es el valor de los pesos y el segundo es el número de neuronas. Los valores de los pesos se modifican mediante el entrenamiento, sin embargo, resulta ser complicado, por lo que en estas pruebas el vector de pesos se asigna de manera aleatoria. Se realizó la simulación de una red con los mismos parámetros de entrada, en donde los valores de $\tau = 50$, $reset = 0$ y $umbral = 1$ son los mismos para todas las neuronas. Esta red cuenta con 10 neuronas distribuidas de la siguiente manera:

- Capa de entrada con 2 neuronas.
- Dos capas ocultas, con 3 neuronas en cada capa (6 en total).
- Capa de salida con 2 neuronas.

Una de las neuronas de entrada recibe Spikes cada 5 ms, mientras que la otra recibe cada 2 ms. En las Figuras 2.10 y 2.11, se presentan los resultados obtenidos de ejecutar la simulación 2 veces, sin embargo, a pesar de que los parámetros de simulación son los mismos, se obtienen resultados muy diferentes. La única variante es el vector de pesos, que en esta ocasión se determina de manera aleatoria, esto demuestra la importancia que tienen los pesos sobre la red neuronal.

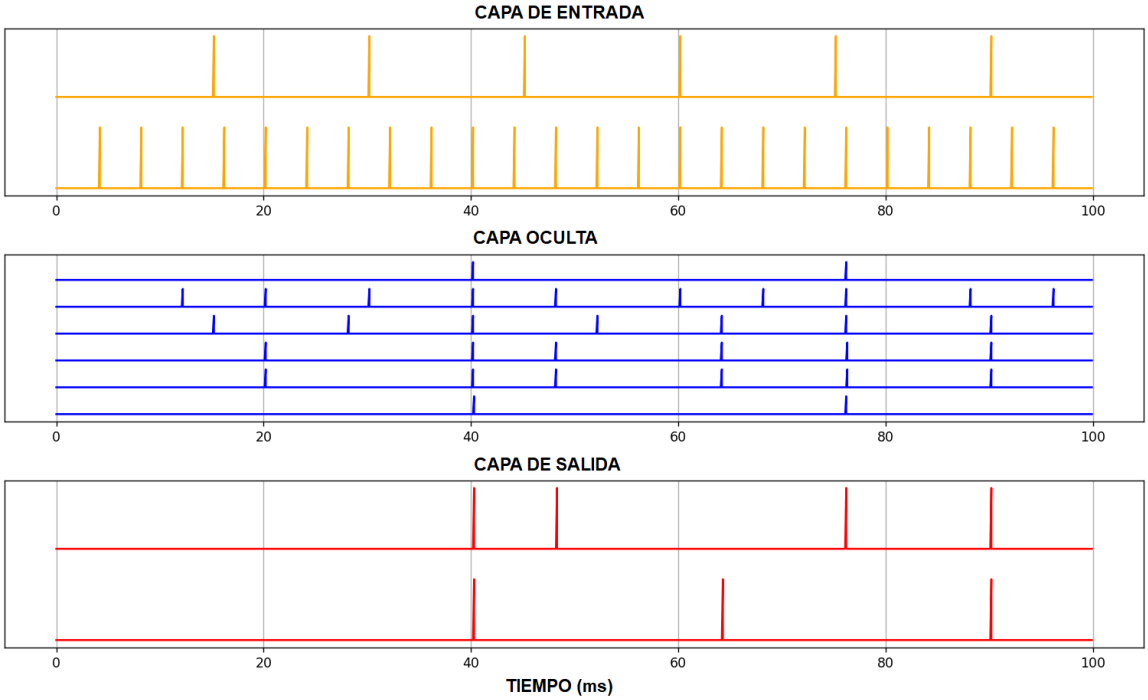

Figura 2.10: Primera simulación de una red neuronal Spiking, con 10 neuronas.

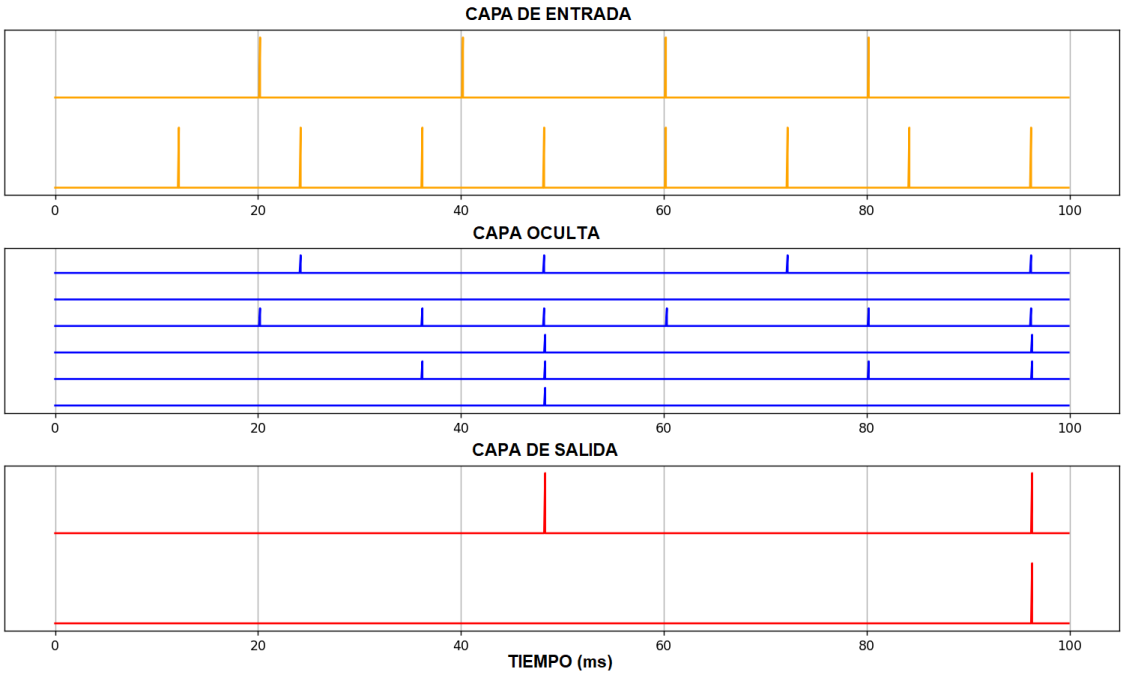

Figura 2.11: Segunda simulación de una red neuronal Spiking, con 10 neuronas.

Red con 34 neuronas

Se simuló una red de 34 neuronas con los mismos parámetros de entrada, en donde los valores de $\tau = 50$, $reset = 0$ y $umbral = 1$ son los mismos para todas las neurona. Las neuronas se encuentran distribuidas de la siguiente manera:

- Capa de entrada con 2 neuronas.
- Tres capas ocultas, con 10 neuronas en cada capa (30 en total).
- Capa de salida con 2 neuronas.

Una de las neuronas de entrada recibe Spikes cada 5 ms, mientras que la otra recibe cada 2 ms. En las Figuras 2.12 y 2.13, se presentan los resultados obtenidos de ejecutar la simulación 2 veces, de la misma manera que en la red de 10 neuronas, a pesar de que los parámetros de simulación son los mismos, se obtienen resultados muy diferentes debido a la asignación de los pesos. En una de las simulaciones la primera neurona tiene prácticamente el doble de Spikes que la segunda, mientras que en la segunda simulación ambas neuronas de salida cuentan con el mismo número de Spikes.

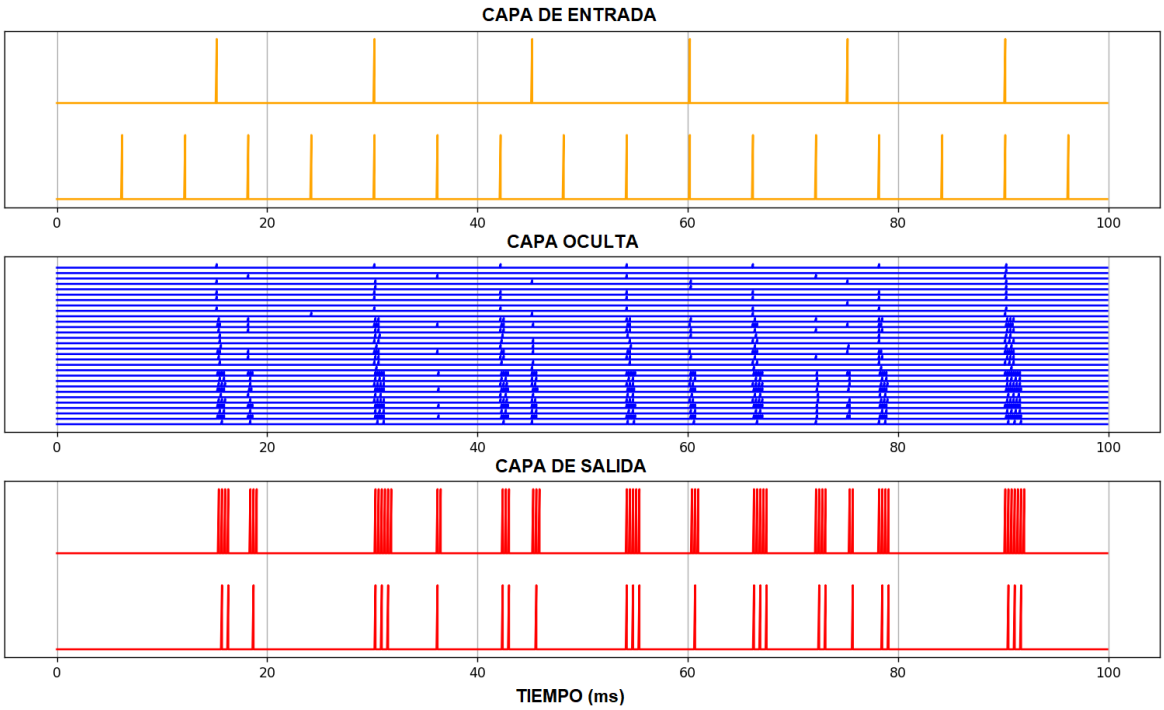

Figura 2.12: Primera simulación de una red neuronal Spiking, con 34 neuronas.

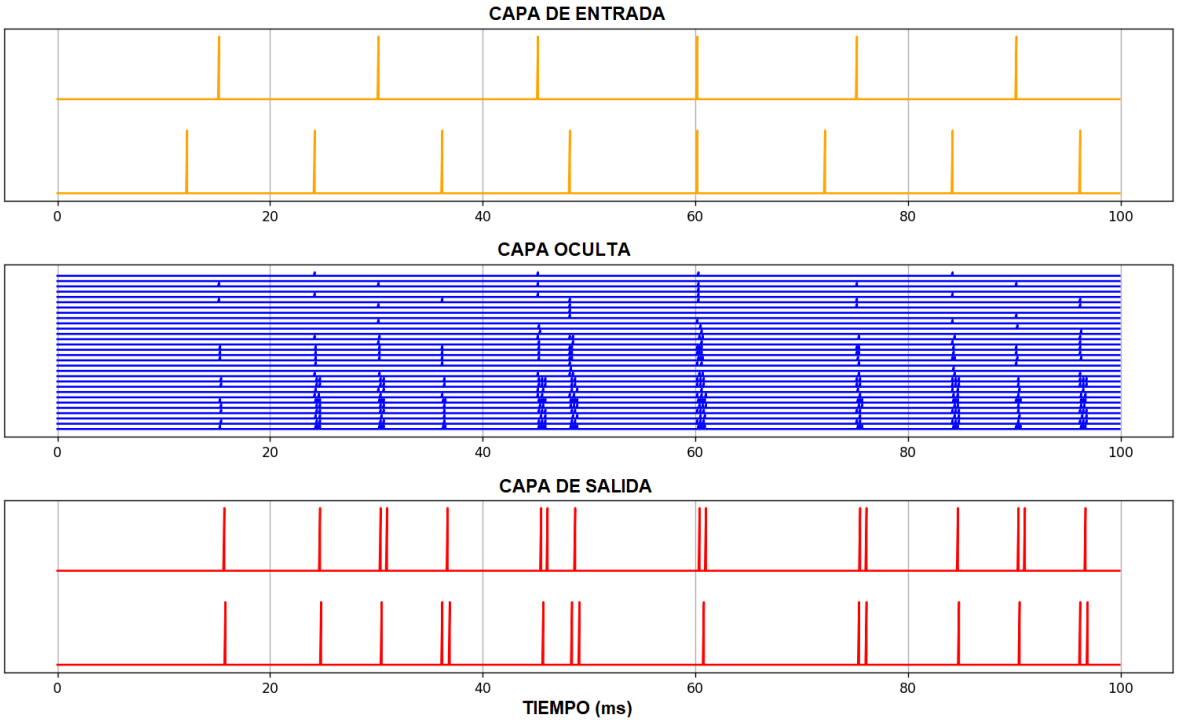

Figura 2.13: Segunda simulación de una red neuronal Spiking, con 34 neuronas.

Red con 100 neuronas

Se simuló una red de 100 neuronas, en donde los valores de $\tau = 50$, $reset = 0$ y $umbral = 1$ son los mismos para todas las neurona. Las neuronas se encuentran distribuidas de la siguiente manera:

- Capa de entrada con 5 neuronas.
- Cuatro capas ocultas, con 23 neuronas en cada capa (92 en total).
- Capa de salida con 3 neuronas.

Dos de las neuronas de entrada reciben Spikes cada $5ms$, otras dos reciben cada $2ms$ y la última neurona recibe cada $1ms$. En las Figuras 2.14 y 2.15, se presentan los resultados obtenidos de ejecutar la simulación 2 veces y de la misma manera, a pesar de que los parámetros de simulación son los mismos, se obtienen resultados diferentes debido a la asignación de los pesos. En la Figura 2.14 se obtienen resultados muy diferentes en el número de Spikes de cada neurona. Por otro lado, en la Figura 2.15, se observa un gran número de Spikes en cada neurona, además, se observa una neurona "muerta" en la capa de entrada, es decir, debido al peso asignado a pesar de recibir varias entradas, la neurona nunca alcanza a emitir una salida.

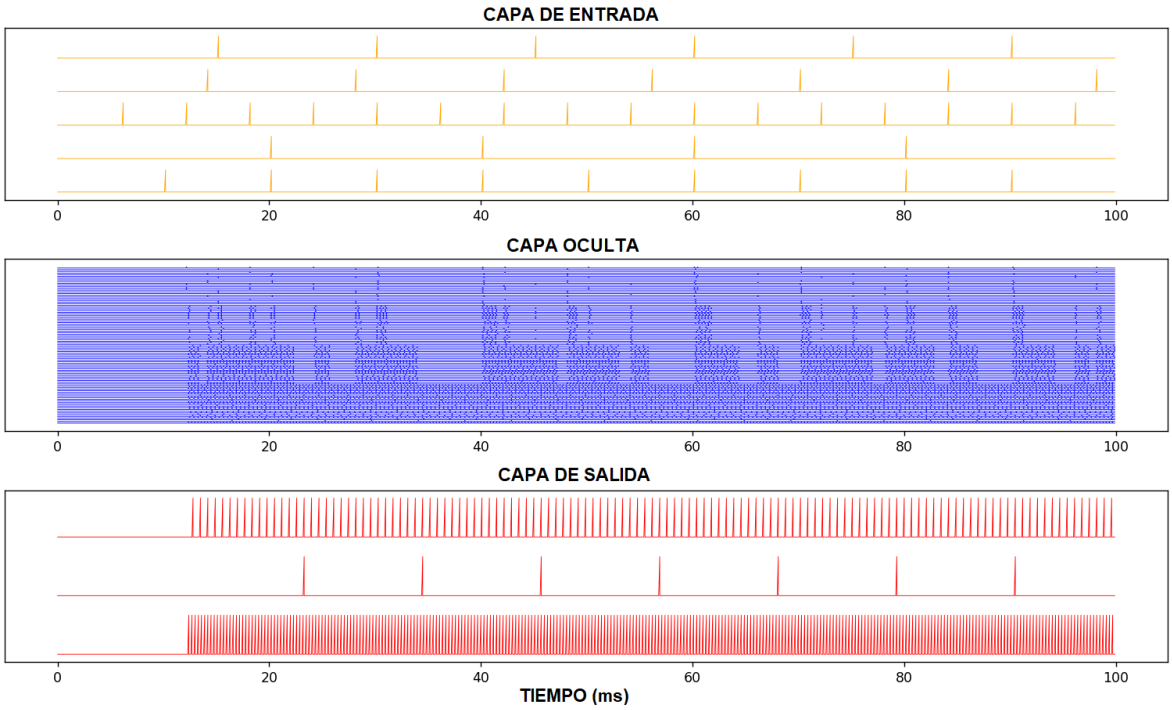

Figura 2.14: Primera simulación de una red neuronal Spiking, con 100 neuronas.

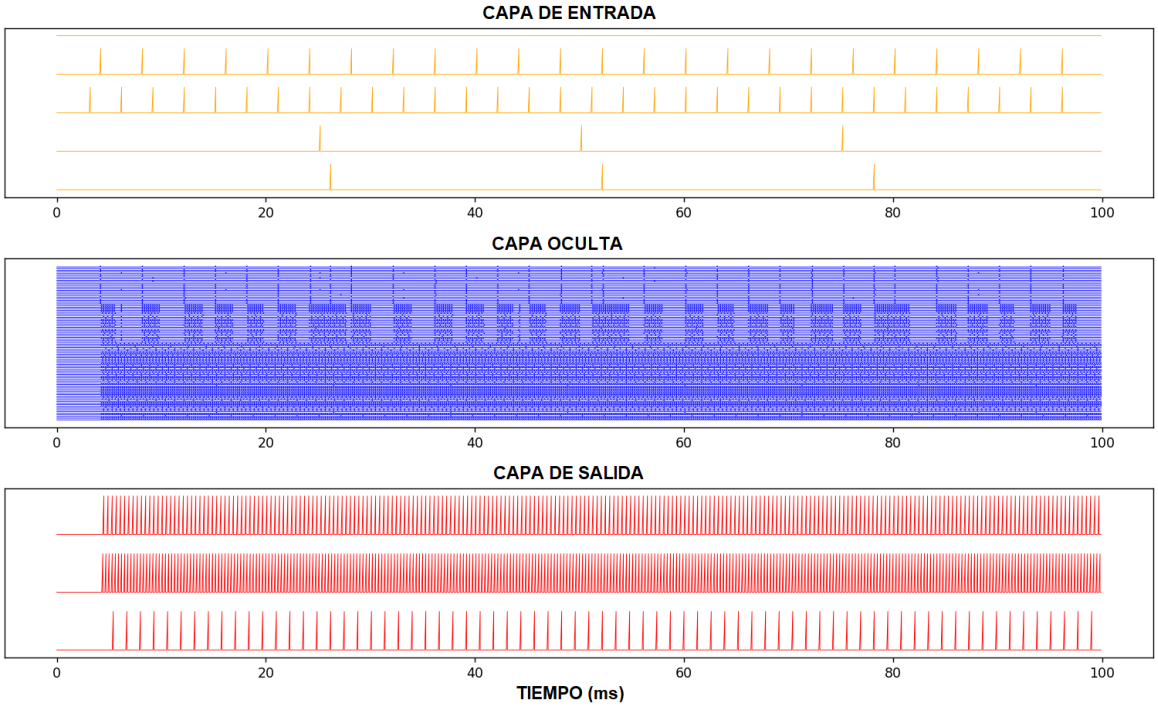

Figura 2.15: Segunda simulación de una red neuronal Spiking, con 100 neuronas.

2.5. Codificación Neuronal de una Red Spiking

Se realizó la implementación de dos diferentes esquemas de codificación sobre la red Neuronal con 100 Neuronas. Los esquemas elegidos son: "Time-to-First-Spike Coding" (TTFS) y "Rate Coding". A continuación, en la Figura 2.16 se muestra el comportamiento gráfico de la red, mientras que en la Figura 2.17 se pueden visualizar los resultados obtenidos en la terminal de los dos diferentes modelos de codificación para cada una de las neuronas de salida.

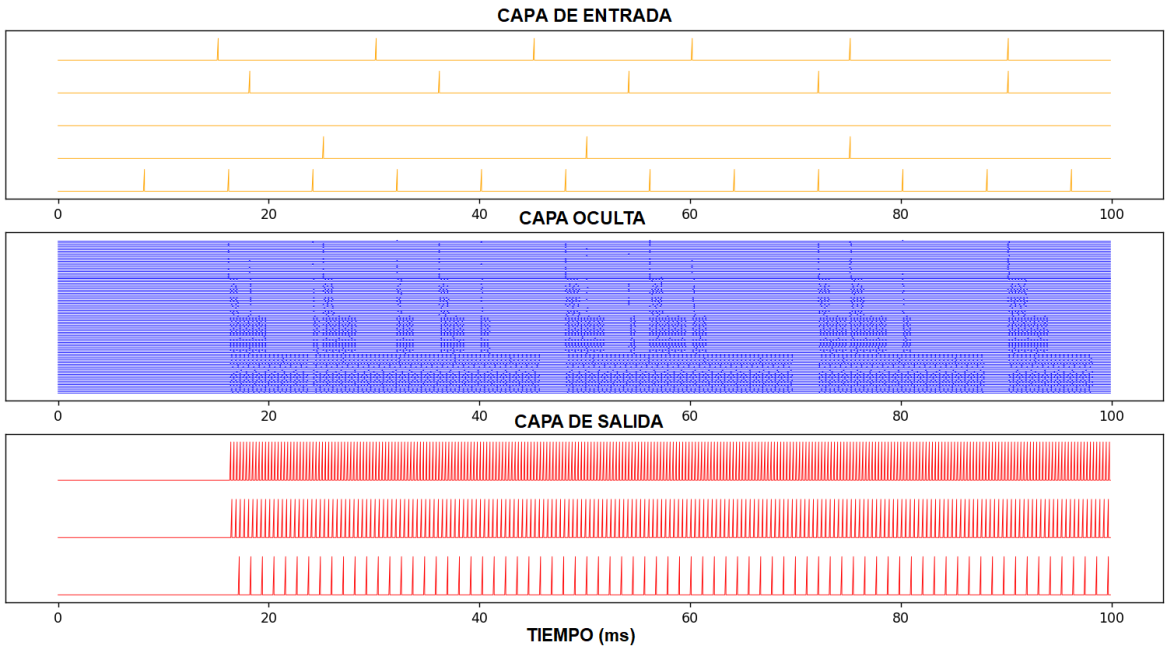

Figura 2.16: Comportamiento gráfico de una red neuronal Spiking, con 100 neuronas.

```
PS C:\Users\vale_> & C:/Users/vale_/AppData/Local/Programs/Python/Python311/python.exe "c:/Users/vale_/OneDrive -
05 - TESIS/EJEMPLO/RED_100_DECO.py"
Los valores de las neuronas en la capa de salida usando "Rate Coding" son:
N1 = 0.279
N2 = 0.209
N3 = 0.076
Los valores de las neuronas en la capa de salida usando "TTFS Coding" son:
N1 = 0.11311286837147269
N2 = 0.08471060687618637
N3 = 0.030588447683112138
```

Figura 2.17: Resultados de la decodificación usando los modelos TTFS y "Rate Coding".

En la sección 2.3 se menciona que el modelo "Time-to-First-Spike Coding" (TTFS) es la mejor opción para conseguir el mejor rendimiento computacional, debido a que los primeros Spikes de salida tienen un peso mucho mayor en comparación con los últimos, esto se encuentra directamente relacionado al valor de la constante de descarga τ elegido. Por otro lado en el modelo "Rate Coding" la salida está determinada por el número de Spikes lo que significa que

cada spike tiene exactamente el mismo valor.

En las Figuras 2.18 y 2.19, se pueden visualizar los resultados obtenidos al ejecutar una segunda vez la simulación. Como se observa en la Figura 2.19, se pueden obtener interpretaciones completamente distintas al usar modelos de codificación diferentes.

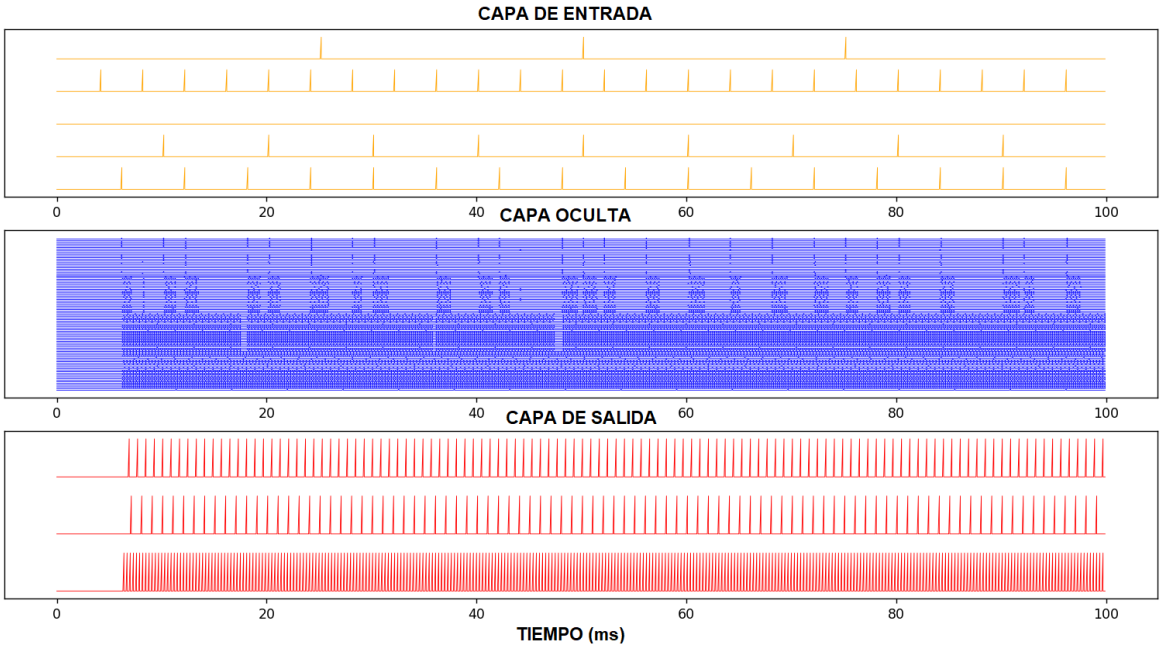

Figura 2.18: Comportamiento gráfico de una red neuronal Spiking, con 100 neuronas.

```
PS C:\Users\vale_> & C:/Users/vale_/AppData/Local/Programs/Python/Python311/python.exe "c:/Users/vale_/OneDrive -
05 - TESIS/EJEMPLO/RED_100_DECO.py"
Los valores de las neuronas en la capa de salida usando "Rate Coding" son:
N1 = 0.117
N2 = 0.093
N3 = 0.312
Los valores de las neuronas en la capa de salida usando "TTFS Coding" son:
N1 = 0.05368660564741431
N2 = 0.042769153493746215
N3 = 0.14388321459512565
```

Figura 2.19: Resultados de la decodificación usando los modelos TTFS y "Rate Coding".

Capítulo 3

Entrenamiento de una red neuronal de tipo Spiking

En este capítulo se describe el proceso para realizar el entrenamiento de una red neuronal de tipo Spiking. Se realiza una investigación y estudio de los diferentes métodos de entrenamiento existentes para las redes neuronales Spiking (SNN). Posteriormente se lleva a cabo una simplificación y optimización de la neurona Spiking, así como la librería desarrollada en el capítulo anterior, esto con la finalidad de facilitar el entrenamiento. Finalmente se describe el entrenamiento de una SNN utilizando el método *Spike-Based Backpropagation*, implementado en el lenguaje de programación Python, además se realiza un análisis comparativo con una red neuronal artificial (ANN).

3.1. Métodos de entrenamiento para una SNN

El entrenamiento de redes neuronales Spiking es un campo de estudio en constante evolución dentro de la inteligencia artificial. Como se ha mencionado anteriormente, estas redes son capaces de simular de manera más precisa el funcionamiento de las neuronas biológicas, lo que las hace especialmente útiles para aplicaciones en el campo de la neurociencia y la robótica.

A diferencia de las redes neuronales tradicionales, las redes Spiking utilizan pulsos discretos de información para comunicarse entre las neuronas. Esto presenta un desafío único a la hora de diseñar algoritmos de entrenamiento efectivos, ya que se deben considerar factores como la sincronización de los pulsos y la codificación de la información.

En este contexto, el entrenamiento de una SNN implica una cuidadosa selección de los parámetros de la red y la implementación de algoritmos de aprendizaje adecuados. El objetivo final es lograr una red que sea capaz de realizar tareas complejas de manera eficiente, utilizando pulsos de información en lugar de señales analógicas continuas.

La rica dinámica temporal de las SNN da lugar a una gran variedad de formas de interpretar el patrón de disparo de una neurona. Naturalmente, esto significa que existen varios métodos

para entrenar las SNN. En general, se pueden clasificar en las siguientes categorías [15]:

- ***Shadow training***: Se entrena una RNA tradicional y se convierte en una SNN interpretando las activaciones como una tasa de disparo o el tiempo en el que aparece un Spike.
- **Reglas de aprendizaje local**: Las actualizaciones de peso son una función de señales que son espacial y temporalmente locales al peso, en lugar de una señal global como en la retro propagación de error.
- ***Spike-Based Backpropagation***: La SNN se entrena de forma nativa utilizando retro propagación de errores, generalmente a través del tiempo como se hace en los modelos secuenciales.

A continuación, en las siguientes subsecciones, se describen algunos de los métodos más comunes para el entrenamiento de una SNN.

3.1.1. Conversión de una ANN a una SNN

La conversión de una ANN a una SNN es un método perteneciente a la categoría *shadow training* y de manera general, como su nombre lo indica consiste en tomar una ANN previamente entrenada y sustituir su función de activación por un modelo Spiking. La mayoría de los trabajos de conversión de una ANN a una SNN se han enfocado en convertir las neuronas que utilizan la función de activación tipo ReLU a LIF, sin embargo, es posible convertir redes que utilicen algún otro tipo de función de activación [3].

La hipótesis central de este método es que el alto costo computacional de las ANN existentes resulta de la transmisión continua de datos entre los nodos en la red, así como de la posterior multiplicación o convolución de matrices [16]. Como resultado, implementar la conversión de ANN a SNN puede permitir la misma función y transmisión de información, pero disminuir los costos de transmisión y cálculos de señales. Los picos de valores binarios reducen la cantidad de bits por transmisión al convertir señales de valores reales en binarias, esto permite que las señalas sean escasas en el tiempo al no transmitir información para cada conexión en cada paso de tiempo.

Este método de conversión se basa en importar los parámetros (pesos) previamente entrenados de una ANN a una SNN. Las conversiones han logrado resultados comparables a las ANN y pueden considerarse como una solución al problema de eficiencia energética de las ANN [3]. Sin embargo, esta implementación presenta varios inconvenientes, además del ineficiente proceso de entrenamiento, convertir funciones de activación en spikes usualmente requiere un gran número de pasos de tiempo en la simulación, lo que puede compensar los beneficios de potencia y latencia inicialmente buscados en una SNN. Además, el proceso de conversión es necesariamente una aproximación. Por lo tanto, es muy poco probable que una SNN entrenado de esta manera alcance el rendimiento de la red original [17].

El problema de las neuronas muertas se puede evitar por completo entrenando una ANN y convirtiéndola en una SNN. La función de activación de cada neurona se convierte en una

tasa de spikes [18] o en un código de latencia [19]. Una de las razones mas convincentes para utilizar este método es que los avances de aprendizaje profundo convencional se pueden aplicar directamente a las SNN. Por esta razón actualmente la conversión de ANN a SNN es el método más utilizado en tareas de clasificación de imágenes estáticas en conjuntos de datos complejos. Tareas donde la eficiencia de la inferencia es más importante que la eficiencia del entrenamiento, y si los datos de entrada no varían en el tiempo, entonces la conversión de una ANN a SNN podría ser el camino óptimo a seguir [17].

3.1.2. Plasticidad dependiente del tiempo del pico

El método de plasticidad dependiente del tiempo de pico es mejor conocido como STDP por su nombre en inglés *Spike-Timing-Dependent Plasticity*. Para los enfoques de entrenamiento directo o reglas de aprendizaje local se suele emplear el método STDP, que imita las reglas de aprendizaje del cerebro biológico [20]. De manera general, este método nos dice que, si una neurona provoca que otra se active, entonces se debe aumentar su fuerza sináptica, de lo contrario, si un par de neuronas parecen no estar correlacionadas, su fuerza sináptica debería disminuir.

Varios experimentos han demostrado que la sincronización relativa de los spikes entre las neuronas presinápticas y postsinápticas se pueden utilizar para definir una regla de aprendizaje y actualizar el peso sináptico. STDP, es un método de aprendizaje que ajusta el peso sináptico en función del orden temporal de los spikes pre y postsinápticos. Si t_{pre} y t_{post} representen el momento de los spikes presinápticos y postsinápticos respectivamente. La diferencia en el tiempo del spike es [17]:

$$\Delta t = t_{pre} - t_{post}. \tag{3.1}$$

Cuando el spike presináptico llega antes que el spike postsináptico, el peso aumenta, lo que se conoce como potencial a largo plazo (LTP). Si se invierte el momento de llegada del spike sináptico, el peso disminuye, lo que se conoce como depresión a largo plazo (LTD) [3]. El ajuste de curvas a mediciones experimentales toma la siguiente forma:

$$\Delta W = \begin{cases} A_+ e^{\Delta t/\tau_+}, & si \quad t_{post} > t_{pre} \\ A_- e^{-\Delta t/\tau_-}, & si \quad t_{post} < t_{pre} \end{cases}, \tag{3.2}$$

donde ΔW es el cambio de peso sináptico, A_+ y A_- representan la cantidad máxima de modulación sináptica que tiene lugar cuando la diferencia entre los tiempos de spikes se aproxima a cero, τ_+ y τ_- son las constantes de tiempo que determinan la fuerza de actualización durante un intervalo entre spikes determinado. Si la neurona presináptica aumenta antes que la neurona postsináptica, $\Delta t < 0 \Rightarrow \Delta W > 0$, y la fuerza sináptica entre las dos neuronas aumenta. Si la neurona presináptica aumenta después de la neurona postsináptica, $\Delta t > 0 \Rightarrow \Delta W < 0$, y la fuerza sináptica disminuye [15]. Este comportamiento se muestra en la Figura 3.1.

Para una conexión sináptica excitadora fuerte, un spike presináptico desencadenará un gran potencial postsináptico. A medida que el potencial de membrana se acerca al umbral de activación neuronal, un caso excitador de este tipo sugiere que un spike postsináptico probablemente seguirá a un spike presináptico. Esto conducirá a un cambio positivo en el peso sináptico, aumentando así la posibilidad de que un spike postsináptico siga a un spike presináptico en el

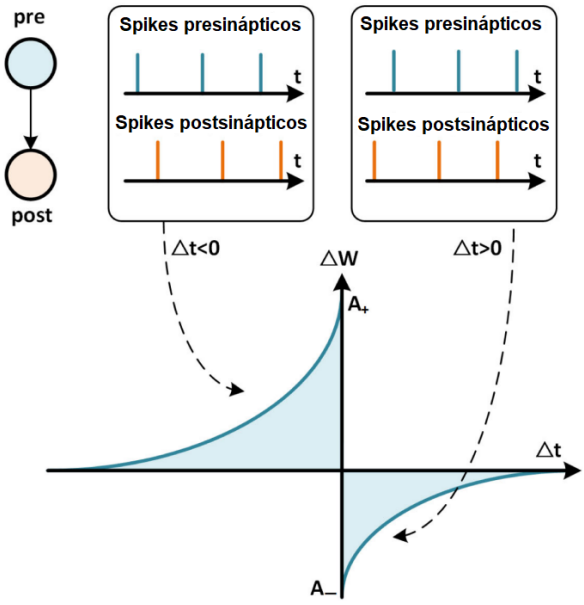

Figura 3.1: Curva estándar del modelo STDP (Imagen tomada de [17]).

futuro. Esta es una forma de spike y STDP refuerza el spike causal al continuar aumentando la fuerza de la conexión sináptica [17].

Los datos sensoriales de entrada suelen estar correlacionados tanto en el espacio como en el tiempo, por lo que la respuesta de una red a un tren de spikes correlacionados aumentara los pesos mucho más rápido que los trenes de spikes no correlacionados. Este es un resultado directo del aumento caudal. Intuitivamente, un grupo de spikes correlacionados de múltiples neuronas presinápticas llegará a una neurona posináptica en un intervalo de tiempo cercano, provocando una despolarización más fuerte del potencial de membrana de la neurona y una mayor probabilidad de que se desencadene un spike postsináptico.

Sin embargo, sin un límite superior, esto conducirá a un crecimiento inestable e indefinidamente grande del peso sináptico. En la practica se debe aplicar un límite superior para restringir la potenciación. Alternativamente, también se pueden utilizar mecanismos homeostáticos para compensar este crecimiento limitado, como un umbral adaptativo que aumenta cada vez que se activa un spike desde la neurona [17].

Muchos trabajos aprovechan el enfoque STDP, que esta inspirado en la biología para actualizar los pesos de las SNN. Sin embargo, el aprendizaje basado en STDP todavía no puede ayudar a entrenar redes a gran escala, debido a la ausencia de una regla de optimización global, lo que limita las aplicaciones prácticas de las SNN [2]. Además, para obtener mejores resultados, es principalmente implementado en hardware neuromórfico [21].

3.1.3. Retropropagación de gradiente sustituto

El método de la retropropagación de gradiente sustituto, es mejor conocido por su nombre en inglés *surrogate gradient backpropagation* y pertenece a la categoría de *Spike-Based Backpropagation*. De manera general y como su nombre lo indica, consiste en sustituir el gradiente mientras ocurre la propagación hacia atrás, es decir, se hace uso de un modelo spiking para la propagación hacia delante de la red (*forward propagation*) y se sustituye el gradiente para poder realizar el cálculo de la propagación hacia atrás (*backpropagation*). De esta manera se evita el problema de las neuronas muertas y permite entrenar de manera adecuada una SNN.

Debido al comportamiento de integración y disparo no diferenciable de las neuronas spiking del modelo LIF, en una SNN no es posible calcular el gradiente de manera directa como lo hace una ANN, esto se ilustra en la Figura 3.2.

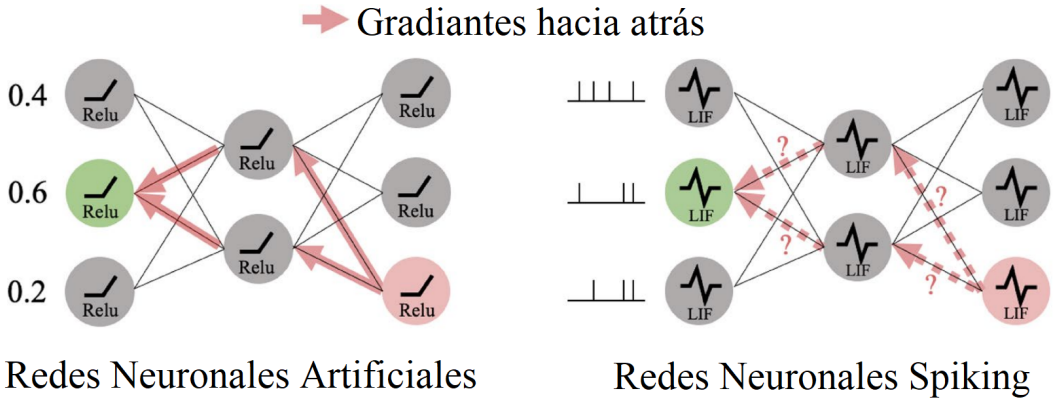

Figura 3.2: Diferencias entre una ANN y una SNN (Imagen tomada de [2]).

Durante la propagación directa, el potencial de la membrana aumenta de acuerdo con los spikes de entrada presinápticos. Si el potencial de membrana excede el umbral de activación, la neurona LIF genera el spike postsináptico con el reinicio del potencial de la membrana. Como ya se mencionó, este comportamiento de integración y disparo induce la no diferenciabilidad del potencial de la membrana. Por lo tanto, se utilizan funciones de gradiente sustitutas para implementar el gradiente hacia atrás. En la Figura 3.3 se muestra de manera gráfica el comportamiento de la propagación hacia delante de la neurona LIF y la propagación hacia atrás con el gradiente sustituto.

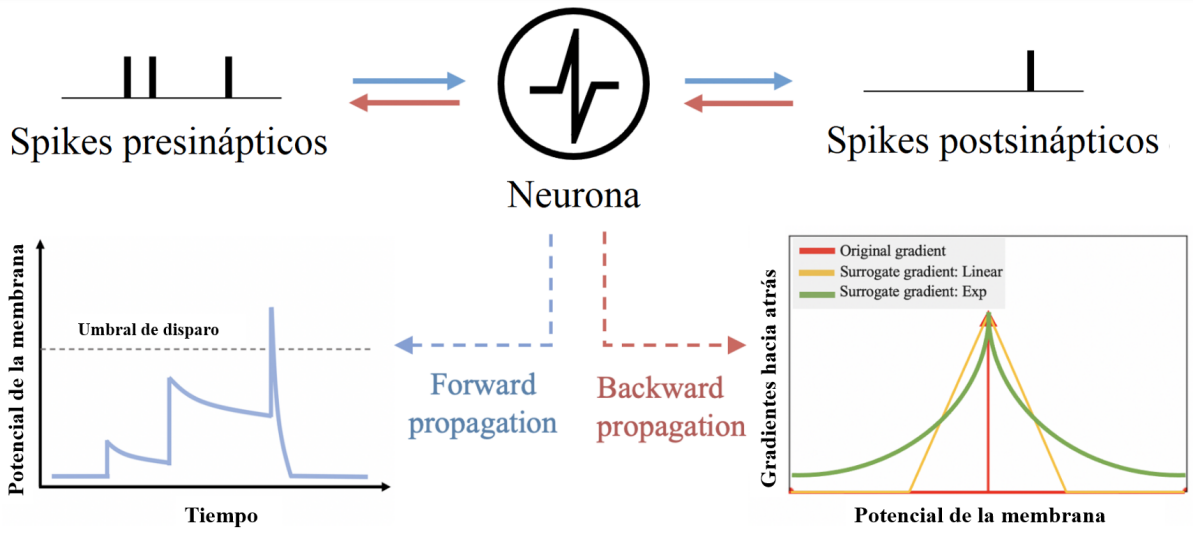

Figura 3.3: Ilustración del forward y backward propagation (Imagen tomada de [2]).

Existen muchas sugerencias para la forma del gradiente sustituto. Una sugerencia popular es aproximar la función escalón Heaviside con una función Sigmoide [1] [17] [22], haciendo que el gradiente sustituto sea la derivada de la función sigmoide, únicamente durante la propagación hacia atrás. En resumen, el gradiente sustituto permite que los errores se propaguen a capas anteriores, independientemente de los spikes. La mayoría de los métodos de entrenamiento de gradientes sustitutos se limitan a conjuntos de datos pequeños debido a que son gradientes inversos aproximados, por lo que es difícil entrenar SNN sin técnicas de optimización adicionales [2].

3.2. Entrenamiento de la red usando un gradiente sustituto

Debido a que las neuronas del modelo LIF únicamente emiten un spike cuando se rebasa el umbral y se mantiene inactiva el resto del tiempo, la función de activación se puede interpretar de diferentes formas, algunos autores lo interpretan como una función Delta de Kronecker [1], sin embargo, también se puede interpretar como una función escalón de Heaviside [17]. Una forma alternativa de representar la relación entre los spikes de salida y el potencial de la membrana es:

$$S[t] = H(V[t] - V_{th}), \tag{3.3}$$

donde $S[t]$ representa la salida de la neurona en el tiempo t, $H(\cdot)$ es la función escalón de Heaviside, $V[t]$ es el potencial de la neurona en el tiempo t y V_{th} es el umbral de la neurona. Básicamente, esta expresión nos dice que si el potencial de la neurona $V[t]$ es mayor a al umbral V_{th}, entonces la salida $S[t] = 1$ y en cualquier otro caso $S[t] = 1$.

La derivada de la función escalón de Heaviside es igual a la función Delta de Dirac y es 0 casi en todo momento, excepto en el momento de activación ($V[t] = V_{th}$), ya que en ese momento tiende a infinito. Esto provoca que en casi todo momento el gradiente también sea 0,

o se sature en caso de que se encuentre precisamente en el momento de activación. A esto se le conoce como *the dead neuron problem* o bien, el problema de la neurona muerta traducido al español. En la Figura 3.4 se ilustra este problema.

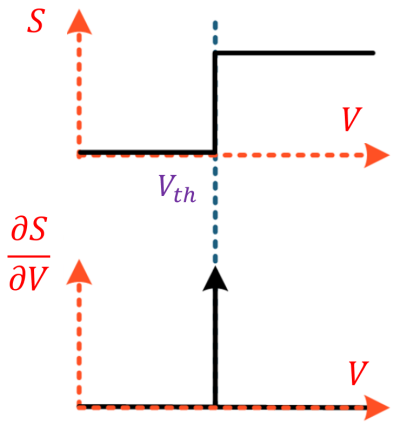

Figura 3.4: Problema de la neurona muerta.

El objetivo es entrenar la red utilizando el gradiente de pérdida con respecto a los pesos, de modo que los pesos se actualicen para minimizar la pérdida. El al algoritmo de retro propagación logra esto usando la regla de la cadena. La diferenciación de la función de pérdida se puede escribir como:

$$\frac{\partial \mathcal{L}}{\partial W} = \underbrace{\frac{\partial \mathcal{L}}{\partial S_{out}}}_{\text{I}} \underbrace{\frac{\partial S_{out}}{\partial V}}_{\text{II}} \underbrace{\frac{\partial V}{\partial W}}_{\text{III}}. \tag{3.4}$$

La derivada I es el gradiente de la función de pérdida, la salida del estimador, la cual se abordará más adelante. La derivada II es la derivada de la función de Heaviside y sería la función delta de Dirac que es cero en todas partes excepto en el origen, en donde es infinita. Esto es lo que no permite entrenar a una SNN de manera directa [22]. Por lo que se busca solucionar este problema remplazando el gradiente con una aproximación que tenga un buen comportamiento. La aproximación de la derivada, denotada como σ' y se conoce como el gradiente sustituto,

$$\frac{\partial S_{out}}{\partial V} = \frac{\partial}{\partial V} H(V - V_{th}) \approx \sigma'(V). \tag{3.5}$$

Considerando la investigación previamente realizada, la función elegida para la sustitución del gradiente será la función sigmoide atenuada, expresada como:

$$\sigma(x, \alpha) = \frac{1}{1 + e^{-\alpha x}}, \tag{3.6}$$

donde α es un parámetro que describe la suavidad de la función sigmoide. Por lo tanto, la derivada de II se expresa como:

$$\sigma'(V, \alpha) = \frac{\alpha e^{-\alpha V}}{(1 + e^{-\alpha V})^2}. \tag{3.7}$$

En la Figura 3.5 se muestran diferentes aproximaciones, obtenidas con diferentes valores de α.

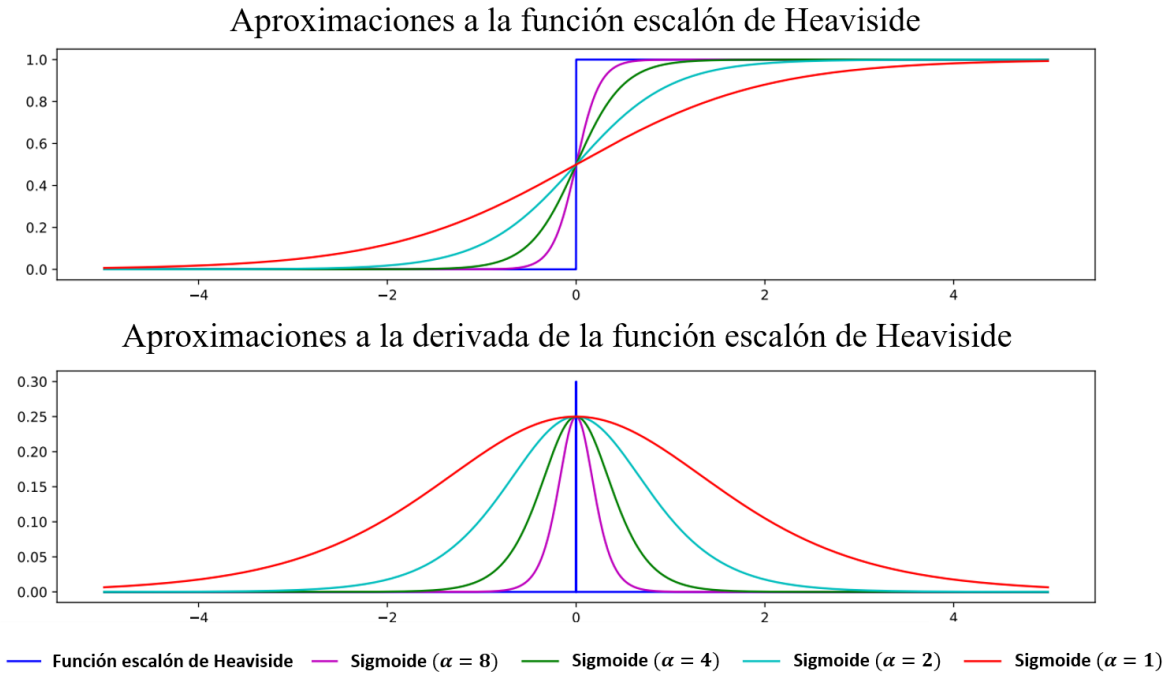

Figura 3.5: Aproximaciones de la función sigmoide con diferentes valores de α.

Para calcular la derivada III, se hace uso de una versión simplificada del modelo LIF, la cual tiene la siguiente forma:

$$V[t+1] = \underbrace{\beta V[t]}_{\text{Descarga}} + \underbrace{WX[t+1]}_{\text{Entradas}} - \underbrace{R[t]}_{\text{reset}}, \tag{3.8}$$

donde la corriente sináptica de entrada se interpreta como $I_{in}[t] = WX[t]$ y $X[t]$ puede ser una entrada arbitraria de Spikes [17]. Por tanto, la derivada III es igual al valor de las entradas, como se muestra a continuación:

$$\frac{\partial V}{\partial W} = X. \tag{3.9}$$

3.2.1. Función de pérdida

Cada vez que se realiza una propagación hacia adelante es necesario interpretar la salida de la red. En el capítulo anterior se revisaron las diferentes formas de codificación que existen para las SNN. El modelo de codificación por el que se optó para este trabajo es el *Rate coding*. Este modelo nos da un resultado entre 0 y 1 proporcional al número de spikes que se emitieron en un determinado rango de tiempo, por ejemplo, para un resultado de 0.6, las neuronas deberían disparar el 60 % de todos los pasos de tiempo.

Sea $\vec{S}[t] \in \mathbb{R}^{N_C}$ un vector variante en el tiempo que representa los spikes emitidos por cada neurona de salida a través del tiempo, donde N_C es el número de clases de salida. Sea $\vec{c} \in \mathbb{R}^{N_C}$ la cuenta de spikes de cada una de las neuronas de salida, la cual se puede obtener sumando $\vec{S}[t]$ a través de un numero de pasos de tiempo T,

$$\vec{c} = \sum_{j=0}^{T} \vec{S}[t]. \tag{3.10}$$

La función de pérdida elegida es el error cuadrático medio (MSE). Sea $y_i \in \mathbb{R}$ la cuenta de spikes objetivo en un periodo de tiempo T para la i-ésima neurona de salida. Por lo tanto, la función de pérdida utilizando el MSE es [17]:

$$\mathcal{L}_{MSE} = \sum_{i}^{N_C} (y_i - c_i)^2. \tag{3.11}$$

3.3. Implementación en Python

En el capítulo anterior se simularon neuronas del modelo LIF y con ellas se estructuraron redes neuronales Spiking con diferentes estructuras y neuronas. Estas redes fueron de gran ayuda para comprender el modelo LIF y visualizar como influyen los diferentes parámetros presentes en este modelo. Se realizaron variaciones en tau, en el valor de reset y se implementó un umbral dinámico, sin embargo, a pesar de que estos parámetros sirven para darle versatilidad a las SNN, no siempre es necesario tomarlos en cuenta. Por lo que, para facilitar la implementación y entrenamiento de las SNN, se utilizan modelos simplificados y solo se agregan los parámetros necesarios para una aplicación determinada. A continuación, se realiza una simplificación de la red y posteriormente se describe la implementación del entrenamiento sobre esta nueva red simplificada.

3.3.1. Simplificación de la SNN por el método de Euler

Del capítulo anterior se conoce que el modelo LIF consiste en modelar una neurona como un circuito RC, cuyo comportamiento puede ser representado por la siguiente ODE:

$$\tau \frac{dV(t)}{dt} = -V(t) + RI_{in}(t), \tag{3.12}$$

aplicando el método de Euler obtenemos:

$$\tau \frac{V(t + \Delta t) - V(t)}{\Delta t} = -V(t) + RI_{in}(t). \tag{3.13}$$

Para valores lo suficientemente pequeños de Δt, se obtiene una buena aproximación de integración en tiempo continuo. Despejando el potencial de la membrana en el siguiente paso de tiempo, se obtiene:

$$V(t + \Delta t) = \left(1 - \frac{\Delta t}{\tau}\right) V(t) + \frac{\Delta t}{\tau} I_{in}(t) R, \tag{3.14}$$

asumiendo que no se presenta ninguna entrada,

$$V(t + \Delta t) = \left(1 - \frac{\Delta t}{\tau}\right) V(t). \tag{3.15}$$

De la expresión 3.15, podemos obtener una nueva constante de descarga, también conocida como constante de tiempo inversa β [17],

$$\beta = \left(1 - \frac{\Delta t}{\tau}\right). \tag{3.16}$$

Para reducir el número de parámetros, asumimos $R = 1$ y obtenemos la siguiente expresión simplificada:

$$V[t + 1] = \beta V[t] + (1 - \beta) I_{in}[t + 1], \tag{3.17}$$

la corriente de entrada está ponderada por $(1 - \beta)$ y desplazada un paso en el tiempo, de modo que pueda contribuir instantáneamente al potencial de la membrana. Si bien, esta no es una suposición fisiológicamente precisa, le da al modelo neuronal una forma más parecida a una red neuronal recurrente (RNN).

Se hace una segunda suposición no fisiológica, donde el efecto de $(1 - \beta)$ es absorbido por un peso W que puede ser aprendido, es decir,

$$WX[t] = (1 - \beta) I_{in}[t + 1], \tag{3.18}$$

esto se puede interpretar como que $X[t]$ es una entrada de voltaje o spike y puede ser escalada por una conductancia sináptica W para generar una inyección de corriente a la neurona. Esto nos da como resultado la siguiente expresión:

$$V[t + 1] = \beta V[t] + WX[t + 1], \tag{3.19}$$

donde los efectos de W y β están desacoplados, favoreciendo la simplicidad sobre la precisión biológica.

Se sabe que, si el potencial excede el umbral (V_{th}), la neurona emitirá un spike, como se muestra en la siguiente expresión:

$$S(t) = \begin{cases} 1, & si \quad V(t) > V_{th} \\ 0, & si \quad V(t) < V_{th} \end{cases} . \tag{3.20}$$

Si la neurona emite un spike, el potencial de la membrana debe ser restablecido. Por lo que se establece un mecanismo de reinicio por sustracción [17], el cual se puede expresar como:

$$V[t + 1] = \beta V[t] + WX[t + 1] - S[t] V_{th}, \tag{3.21}$$

de esta manera, W es un parámetro que se aprende y V_{th} usualmente se establece como 1. Por lo tanto, se obtiene un nuevo modelo que tiene como único parámetro configurable β.

3.3.2. Actualización de la librería local

Con la finalidad de poder simular SNN con diferentes características, se actualizo la librería local, agregando nuevas funciones necesarias para el entrenamiento, como lo son el escalamiento y des escalamiento de datos, la codificación y decodificación. Además, se modificando algunas otras con el modelo LIF simplificado.

Escalamiento

Para poder introducir datos de entrada a una ANN o SNN, es necesario hacer un escalamiento, ya que se requiere trabajar con valores entre 0 y 1. El proceso de escalamiento es sencillo de implementar y se obtiene con la siguiente expresión:

$$V_{escalado} = \frac{V_{desescalado} - V_{minimo}}{V_{maximo} - V_{minimo}}. \tag{3.22}$$

De la misma manera, para poder interpretar los resultados emitidos por la red es necesario realizar un des escalamiento del dato de la siguiente forma:

$$V_{desescalado} = (V_{maximo} - V_{minimo})V_{escalado} + V_{minimo}. \tag{3.23}$$

Codificación usando *rate coding*

Para poder pasar de números reales a vectores de spikes, es necesario utilizar un método de codificación. Como ya se ha mencionado, el *rate coding* se basa en interpretar el número de spikes emitidos por una neurona en un determinado intervalo de tiempo. La entrada a esta función es un número entre 0 y 1, el cual nos indicara el porcentaje del intervalo de tiempo en el que se emitieron spikes, es decir, para una entrada de 0.35, se debe generar un vector el cual tenga un 35 % de spikes en un intervalo de tiempo T. De manera general, se sigue el procedimiento descrito en el siguiente pseudocódigo.

	Codificación usando rate coding.
1	Calcular el número de spikes N para determinado número de pasos P
2	Generar un vector de salida O de longitud P
3	Para cada spike en el rango(N)
4	Posicionar de manera aleatoria el spike en el vector O
5	Retornar el vector de salida O

Por otro lado, para poder interpretar la salida de una neurona spiking es necesario poder decodificar un vector de spikes a un valor real entre 0 y 1. Esto es más sencillo de implementar y el procedimiento se describe en el siguiente pseudocódigo.

	Codificación usando rate coding.
1	NS = Suma de todos los spikes en el vector
2	Valor de salida V = Valor entero de (NS/(Numero de pasos de tiempo))
3	Retornar V

Actualización del modelo LIF simplificado

La implementación en código de este nuevo modelo simplificado se vuelve mucho más sencillo que el desarrollado en el capítulo anterior. A continuación, se presenta el código de una función en Python que simula la neurona LIF simplificada.

Código para una neurona LIF simplificada en Python.

```
1  def LIF_S(potencial, x, beta, w=1, umbral=1):
2          spk = (potencial > umbral)
3          potencial = beta * potencial + w*x - spk*umbral
4          return potencial, spk
```

El modelo LIF simplificado nos ofrece resultados equivalentes a los obtenidos en el capítulo 2. En la Figura 3.6 se puede visualizar el comportamiento de una neurona LIF simplificada con una entrada codificada de 0.35.

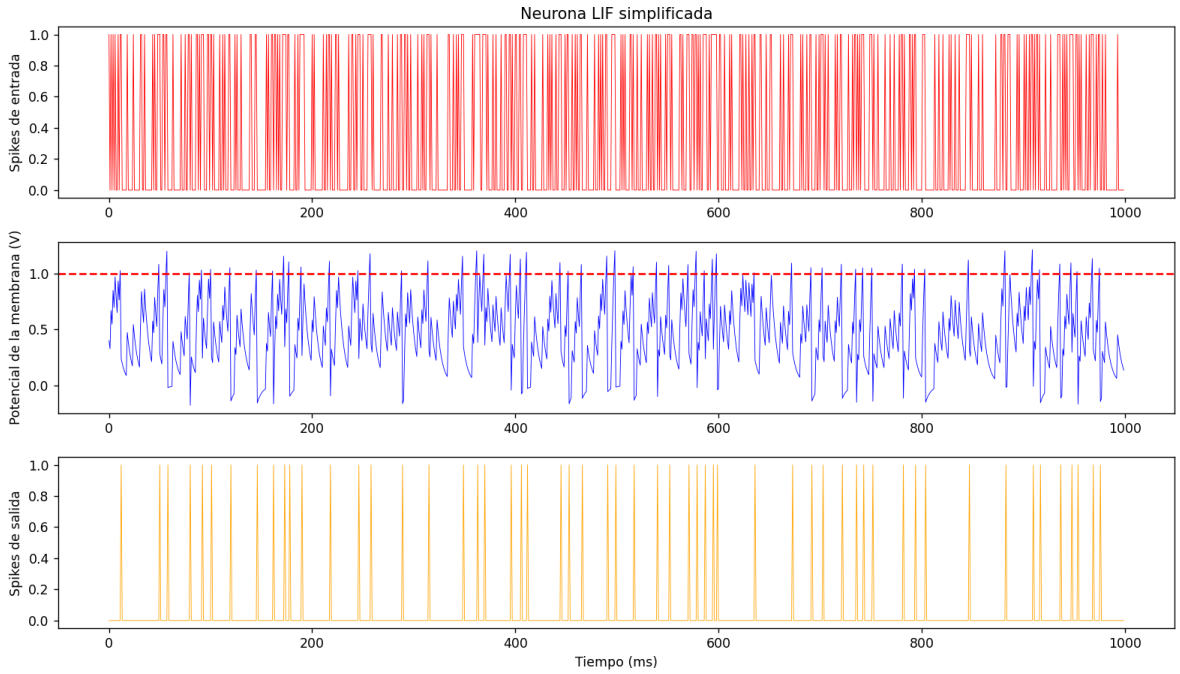

Figura 3.6: Comportamiento de una neurona LIF simplificada.

Adicionalmente se realizaron algunas modificaciones menores a funciones generadas en el capítulo anterior, esto para asegurar el funcionamiento con el modelo LIF simplificado.

3.3.3. Implementación del sustituto del gradiente

Como un primer ejemplo de entrenamiento de una SNN, se desarrolló una red que siga una referencia, es decir que a partir de una condición inicial x_0 establecida por el usuario la red pueda estimar y llegar al valor deseado x_d. La red implementada tiene la siguiente estructura:

- Capa de entrada con 1 neurona.

- Una capa oculta, con 10 neuronas.
- Capa de salida con 1 neuronas.

A manera de analogía se consideró un sistema tipo péndulo, el cual puede posicionarse en un rango de 360°, por lo que para el escalamiento se toma como $x_{max} = 180$ y $x_{min} = -180$ y de esta manera cubrir el rango de los 360°.

Al hacer uso del modelo de codificación *Rate coding*, es necesario establecer el intervalo de tiempo en que se estarán codificando los resultados de la SNN. En este caso se toma un intervalo de tiempo para la codificación se estableció de 100 ms, con 100 pasos de tiempo, es decir, las neuronas pueden emitir spikes cada 1 ms. El vector de tiempo total se considera de 20 s. Por lo tanto, la red realizara un total de 20,000 cálculos.

El propósito es generar un entrenamiento en línea, para que de esta manera, cuando se implemente la red neuronal sobre un sistema dinámico, se puedan compensar incertidumbres. Por esa razón en la red se retroalimentará, haciendo que el estado actual x sea ahora la condición inicial x_0 en el paso siguiente. De manera general, la implementación de la red y el algoritmo de entrenamiento se describen en el siguiente pseudocódigo.

	Pseudocódigo para el entrenamiento de una SNN.
1	Definir la función del sustituto del gradiente
2	Definir los parámetros de la red (neuronas y capas)
3	Definir los intervalos de tiempos (Rate coding Ts y global Tg)
4	Definir el valor deseado x_d y la condición inicial x_0
5	Escalar los valores x_d y x_0 e igualar x=x_0
6	Inicialización de los pesos de manera aleatoria
7	Para cada t en el intervalo Tg:
8	Codificar x (convertir a un vector de spikes)
9	Realizar el proceso de propagación hacia adelante
10	Decodificar la salida de la red x_out (convertir el vector a un valor real)
11	Realizar la retro propagación con el sustituto del gradiente
12	Actualizar los pesos
13	x=x_out
14	Guardar el valor de x des escalado para graficar
15	Graficar los spikes emitidos en los emitidos en la última codificación
16	Graficar la evolución de x des escalado en el tiempo

En la Figura 3.7 se pueden visualizar los resultados obtenidos de la aproximación de la red hacia el valor deseado $x_{des} = 45$.

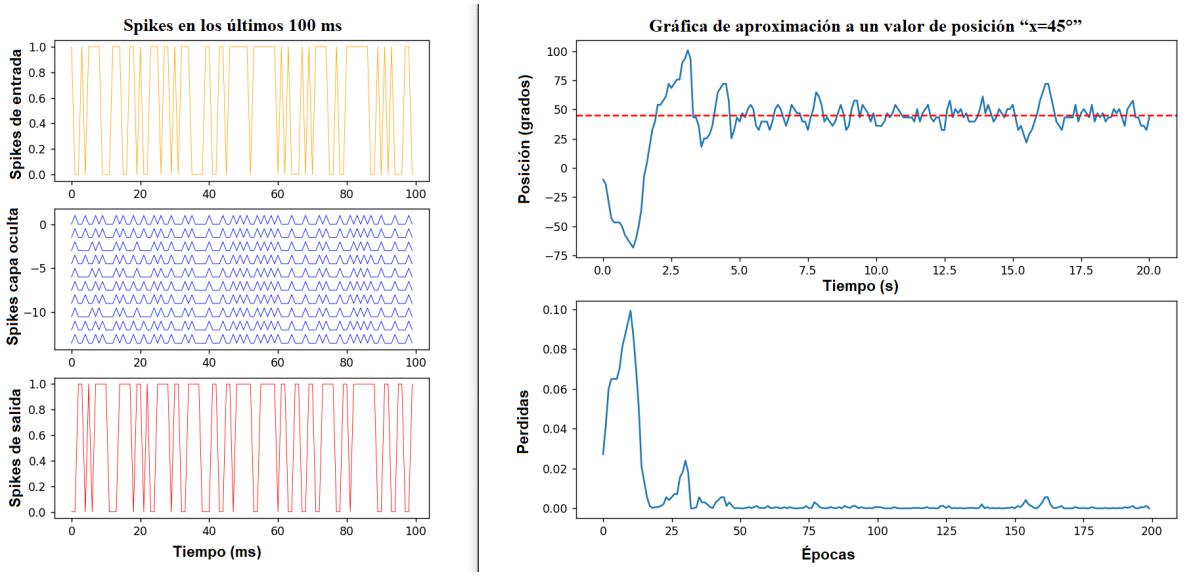

Figura 3.7: Aproximación a un valor deseado usando una SNN.

3.4. Resultados después del entrenamiento de una SNN

Se realizaron diferentes tipos de pruebas a la red con la finalidad de analizar y comprender su comportamiento. Las pruebas se dividen en las siguientes categorías: Aproximación a diferentes valores y optimización. A continuación, se presentan los resultados obtenidos para cada una de estas pruebas.

3.4.1. Aproximación a diferentes valores

De manera general, se puede decir que la red no es muy precisa pues presenta un gran número de oscilaciones. Debido a que la inicialización de los pesos se hace de manera aleatoria se pueden obtener mejores o peores resultados al correr la simulación en varias ocasiones. Este comportamiento se presenta en prácticamente cualquier posición, sin embargo, al aproximarse a algunas posiciones en específico parece costarle más. Se corrieron simulaciones para 4 diferentes posiciones, $x_d = -135, -40, 35, 140$, las cuales corresponde a las Figuras 3.8, 3.9, 3.10 y 3.11 respectivamente.

Se observa que particularmente, para valores negativos las oscilaciones son de mayor intensidad conforme el valor decrece, esto probablemente se debe a que conforme disminuye el valor, también lo hace de manera proporcional el número de spikes, lo que provoca que le cuente más trabajo a la red mantenerse activa. Por otro lado, para los valores positivos las oscilaciones son ligeramente menores, sin embargo, también le toma un poco más de tiempo a la red acercarse al valor deseado.

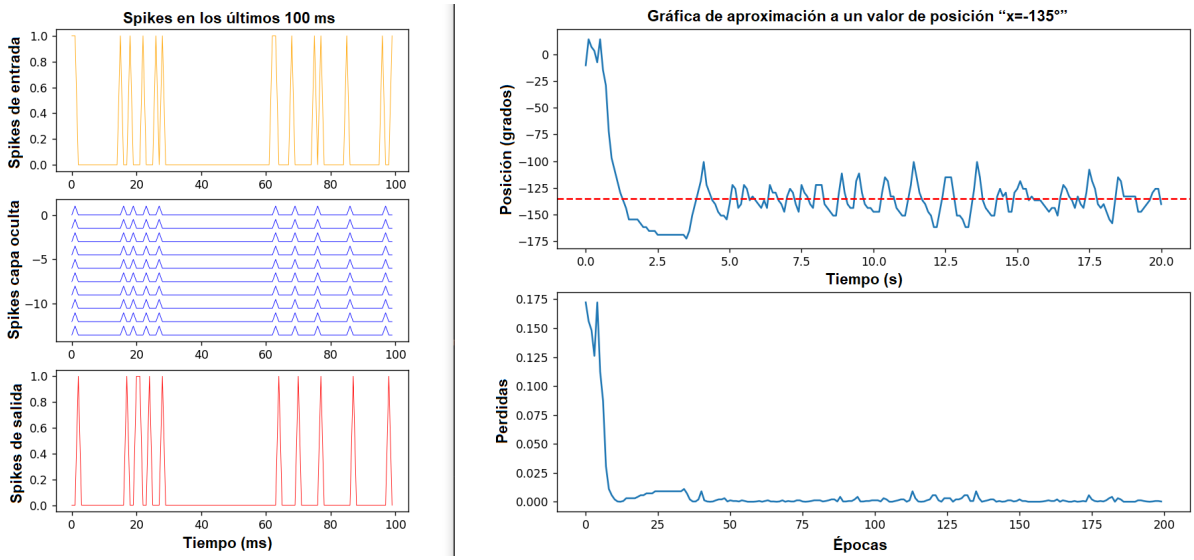

Figura 3.8: Aproximación con $x_d = -135$.

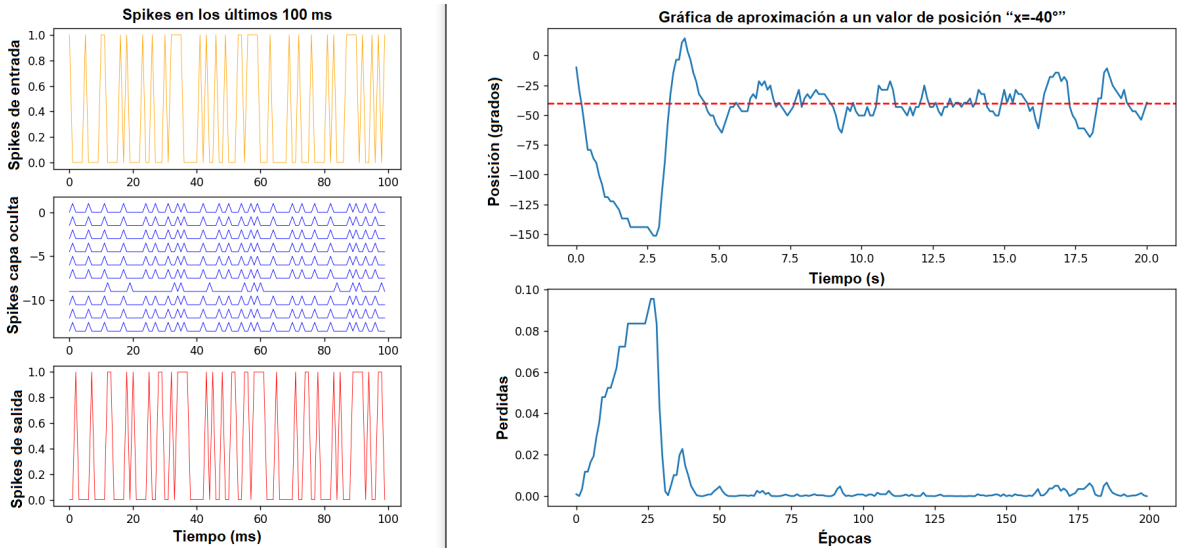

Figura 3.9: Aproximación con $x_d = -40$.

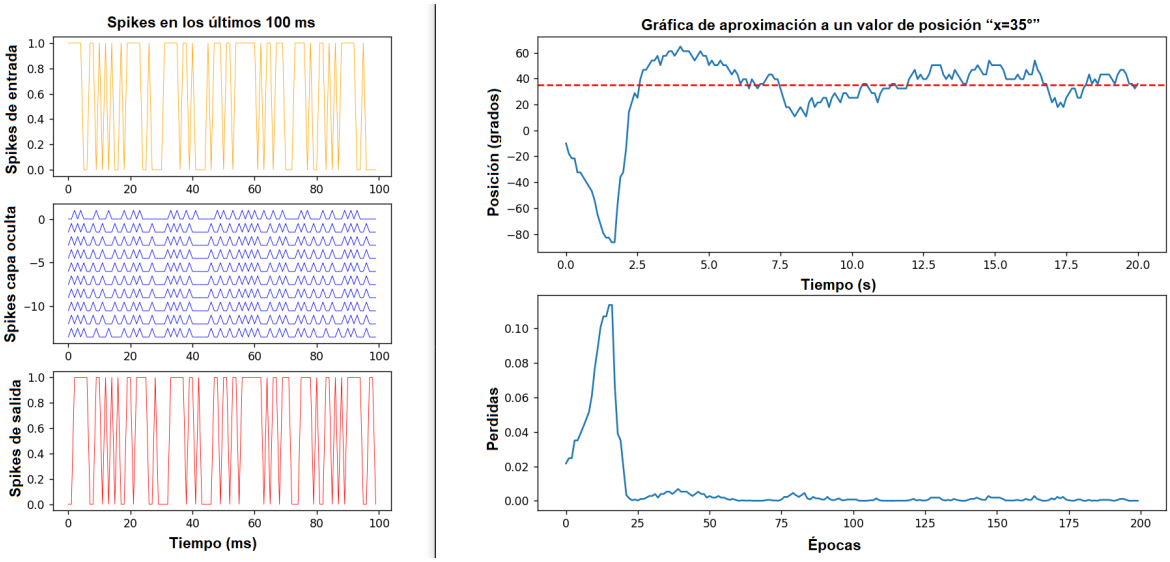

Figura 3.10: Aproximación con $x_d = 35$.

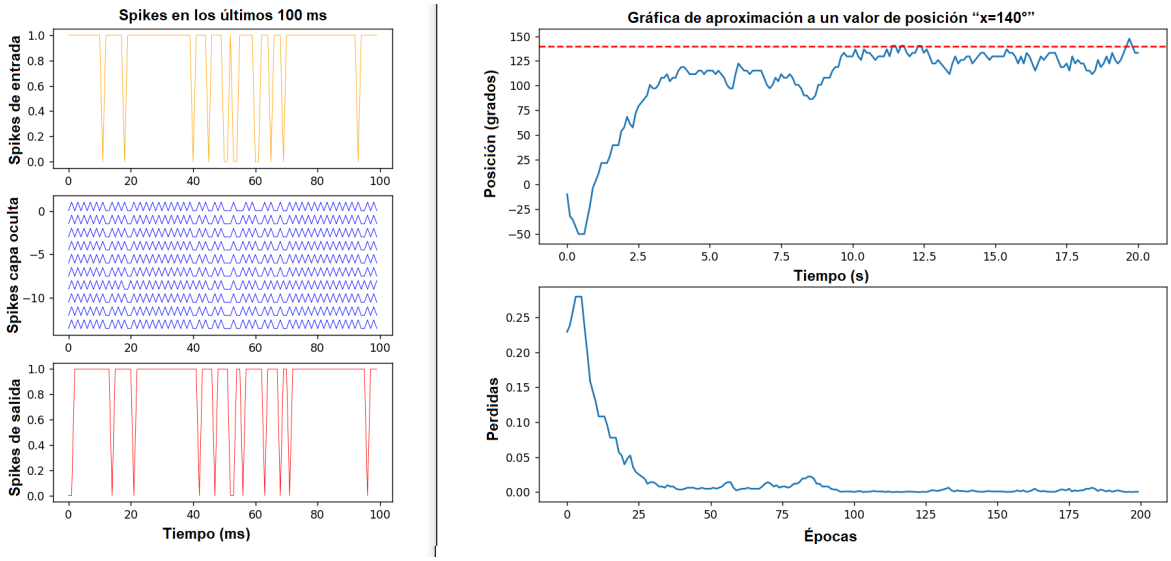

Figura 3.11: Aproximación con $x_d = 140$.

3.4.2. Optimización

Se buscaron diferentes maneras para optimizar el comportamiento de la red, se probó cambiando el gradiente sustituto, por uno del tipo arco tangente, recomendado en [17], el cual tiene la forma:

$$\sigma(x) = \frac{1}{\pi}\arctan(\pi x), \tag{3.24}$$

cuya derivada es:

$$\sigma'(x) = \frac{1}{\pi}\frac{1}{1+(\pi x)^2}, \tag{3.25}$$

sin embargo, se observó prácticamente el mismo comportamiento, que con la función sigmoide atenuada. Su comportamiento se puede observar en la Figura 3.12.

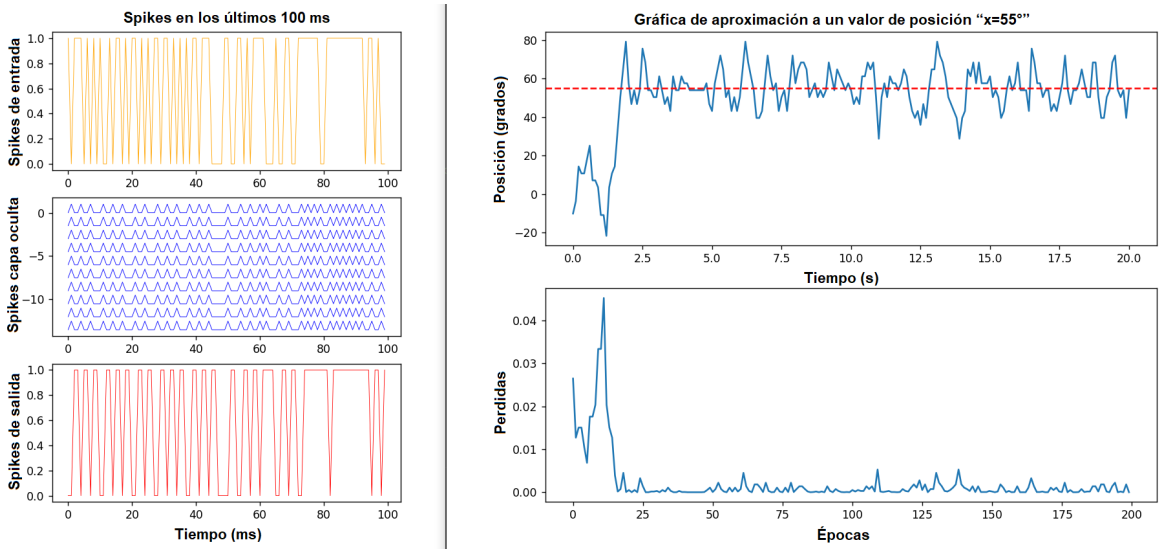

Figura 3.12: Aproximación utilizando la función arco tangente como sustituto del gradiente.

Por otro lado, se puede aumentar el número de spikes, aumentando la resolución del *rate coding*, pasando de 1ms a 0.1ms lo cual da mejores resultados, pero a su vez conlleva un mayor tiempo de simulación, pues se pasa de hacer 100 muestras por segundo a 1,000 muestras por segundo. En la Figura 3.13 se pueden observar los resultados.

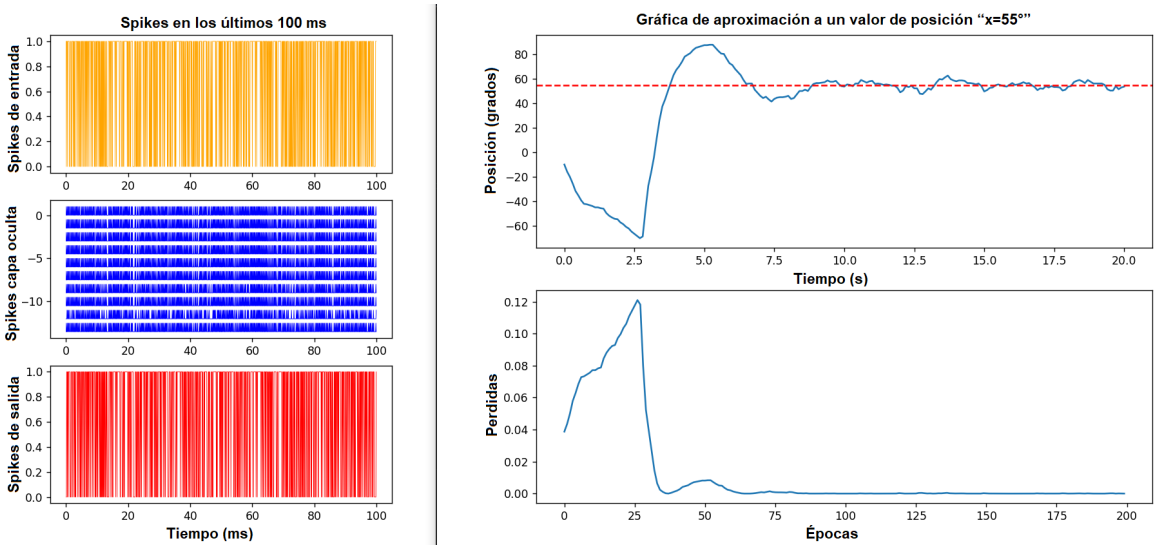

Figura 3.13: Aproximación aumentando la resolución del rate coding.

Capítulo 4

Implementación de un Control Neuro-Inspirado en un sistema no lineal de primer orden

Las redes neuronales artificiales se han aplicado ampliamente al control adaptativo de sistemas no lineales, su capacidad de aprendizaje, junto con su capacidad de capturar relaciones no lineales, les permite ajustar y optimizar el comportamiento del sistema de control para lograr una operación más eficiente desde el punto de vista energético [23] [24]. En algunos casos, el entrenamiento inicial de la red neuronal puede requerir una cantidad significativa de energía y recursos computacionales, sin embargo, una vez entrenada, la capacidad de las redes neuronales para adaptarse, aprender y optimizar su comportamiento puede contribuir significativamente a mejor la eficiencia energética en diversas aplicaciones de control [25].

En este capítulo se realiza la implementación mediante simulación de tres controles neuronales adaptativos, uno utilizando una red neuronal artificial tradicional (ANN) con función de activación sigmoide, otro con una red de tipo RBF y finalmente, usando una SNN. Todos se evalúan sobre un sistema no lineal de primer orden en presencia de dos tipos de incertidumbres, una perturbación contante y una perturbación variante en el tiempo. La simulación se lleva a cabo a utilizando el lenguaje de programación Python y se realiza un análisis comparativo de los controles neuro-inspirados propuestos.

4.1. Diseño de un Control Neuro-Inspirado

En la presente sección se describe el desarrollo de un control neuronal. Se realiza un análisis teórico, el diseño del control y el análisis de estabilidad por Lyapunov.

4.1.1. Planteamiento del problema

Considerando un sistema de naturaleza no lineal, el cual físicamente puede representar el movimiento de un objeto y puede ser representado mediante la siguiente expresión:

$$\dot{x}(t) = f(x(t)) + u(t) + \varsigma(t), \tag{4.1}$$

donde $f(\cdot) \in \mathbb{R}$ representa la dinámica no lineal del sistema, $u(t) \in \mathbb{R}$ es la entrada de control y $\varsigma(t) \in \mathbb{R}$ es una perturbación exógena variante en el tiempo. En la Figura 4.1 se presenta un diagrama ilustrativo del sistema.

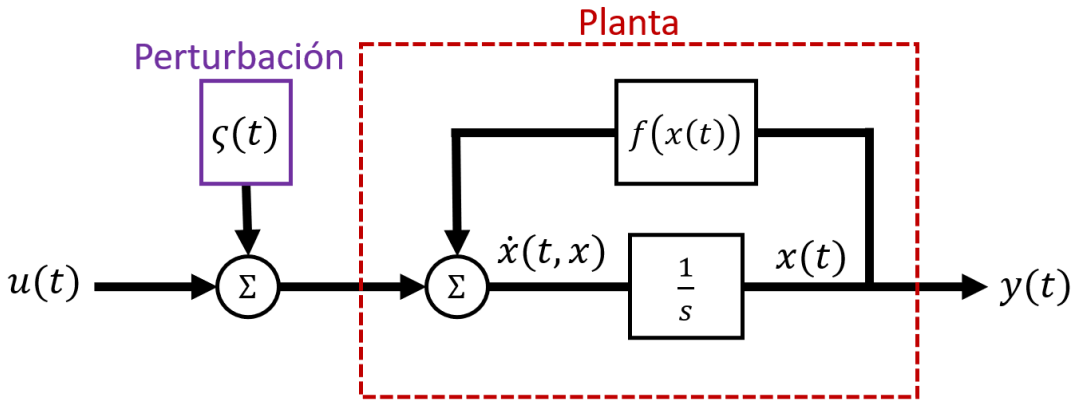

Figura 4.1: Diagrama representativo del sistema.

Considerando mediciones ideales, la salida del sistema es representada por $y(t) = x(t)$. Por lo que las perturbaciones se consideran en la entrada de control y estas son perturbaciones de velocidad, las cuales pueden representar fricción, viento u obstáculos que puedan modificar la velocidad del sistema. Sabiendo que $\varsigma(t) \neq 0$, la cual es desconocida, pero se puede delimitar dentro de un régimen de operación del sistema, siendo $\|\varsigma_i(t)\| \leq \varsigma_{max}$.

4.1.2. Aproximación basada en redes neuronales

Dado que las dinámicas no lineales $f(x)$ y $g(x)$ son desconocidas, por lo tanto, una combinación lineal finita de funciones de activación ponderadas para aproximar dichos términos no lineales dentro de un conjunto prescrito Ω.

Teorema 4.1.1. *(Aproximación universal) Sea $f(\chi(t))$ una función suave en el conjunto compacto Ω. Por lo tanto, existe una arquitectura de red neuronal de tres capas basada en una única capa intermedia de η neuronas con un vector de función de activación adecuado $\phi(\chi(t)) \in \mathbb{R}^\rho$ y un vector de pesos ideales $\omega \in \mathbb{R}^\rho$, tal que,*

$$f(x) = \boldsymbol{\omega}^T + \boldsymbol{\varphi}(\chi(t)) + \varepsilon, \tag{4.2}$$

donde ε representa el error de aproximación de la red neuronal. En cuanto a la función de activación, se puede seleccionar entre una tangente hiperbólica, sigmoide o gaussiana (RBF).

Suposición 4.1.1. *El peso ideal $\boldsymbol{\omega}(\cdot)$, la función de activación $\boldsymbol{\varphi}(\chi)$ y el error de aproximación ε tienen su límite superior respectivamente en $\|\boldsymbol{\omega}(\cdot)\| \leq \omega_{max}$, $\|\boldsymbol{\varphi}(\cdot)\| \leq \varphi_{max}$ y $\|\boldsymbol{\varepsilon}\| \leq \varepsilon_{max}$, con $\omega_{max}, \varphi_{max}, \varepsilon_{max} > 0 \in \mathbb{R}$.*

Observación 4.1.1. *El error de aproximación desconocido ε_i puede ser reducido incrementando el número de neuronas.*

El teorema 4.1.1 permite aproximar uniformemente $f(\xi)$ usando la siguiente expresión:

$$\hat{f}(\chi) = \hat{\boldsymbol{\omega}}^T + \boldsymbol{\varphi}(\chi(t)), \tag{4.3}$$

donde $\boldsymbol{\omega} = [\omega_1, \cdots, \omega_\rho]^T$ y $\boldsymbol{\varphi} = [\varphi_1, \cdots, \varphi_\rho]^T$, donde ρ representa el número de neuronas. En este trabajo, consideramos funciones de activación sigmoide, base radial para las RBF y el modelo LIF para las SNN.

4.1.3. Diseño del controlador

La política de control propuesta pretende alcanzar el objetivo de regulación mientras compensa la dinámica desconocida dependiente del estado, $f(x)$ y las perturbaciones externas $\varsigma(t)$. En otras palabras, el esquema de control de primer orden pretende minimizar el error de posición, es decir,

$$\lim_{t\to\infty} \|\chi(t) - \chi_d(t)\| \to 0. \tag{4.4}$$

En ese sentido, reescribimos el modelo en una dinámica de error de primer orden, obteniendo la siguiente expresión,

$$\dot{e}(t) = f(\chi(t)) + u(t) + \varsigma(t) - \dot{\chi}_d(t), \tag{4.5}$$

donde $e = \chi(t) - \chi_d(t)$. En cuanto al diseño del control, consideramos el siguiente termino global,

$$\bar{u}(t) = f(\chi(t)) + u(t) + \varsigma(t), \tag{4.6}$$

lo que permite reescribir la dinámica de error anterior como:

$$\dot{e}(t) = \bar{u}(t) - \dot{\chi}_d(t). \tag{4.7}$$

El controlador propuesto para estabilizar (4.5) se encuentra dado por la siguiente expresión:

$$\bar{u}(t) = -\boldsymbol{\omega}^T \boldsymbol{\varphi}(e(t)) + \varepsilon + \dot{\chi}_d(t). \tag{4.8}$$

y cuya regla de adaptación/aprendizaje se representa por la siguiente ecuación:

$$\dot{\hat{\omega}}(t) = -\alpha e(t) \boldsymbol{\varphi}(\boldsymbol{e(t)}), \tag{4.9}$$

donde $\alpha \in \mathbb{R}$ es el factor de aprendizaje.

Observación 4.1.2. *El peso estimado se calcula a partir de la minimización del error de estimación del peso, expresado como,*

$$\tilde{\boldsymbol{\omega}} = \boldsymbol{\omega} - \hat{\boldsymbol{\omega}}, \tag{4.10}$$

y su derivada temporal correspondiente es:

$$\dot{\tilde{\boldsymbol{\omega}}} = -\dot{\hat{\boldsymbol{\omega}}}. \tag{4.11}$$

4.2. Simulación de un Control Neuro-Inspirado

Partiendo de un sistema no lineal de primer orden, la red neuronal es diseñada para sustituir a un controlador clásico, por lo que utiliza como variable principal el error de posición, es decir la diferencia entre la posición deseada y la posición actual ($e(t) = x_d - x(t)$). Esta red realiza un aprendizaje en línea, por lo que no necesita de una etapa de entrenamiento previa o un conjunto de datos, esto permite que aún con el cambio de las perturbaciones el sistema siga adaptándose, minimizando el error a valores muy cercanos a cero. En la Figura 4.2 se presenta un diagrama a bloques en lazo cerrado de un control neuronal.

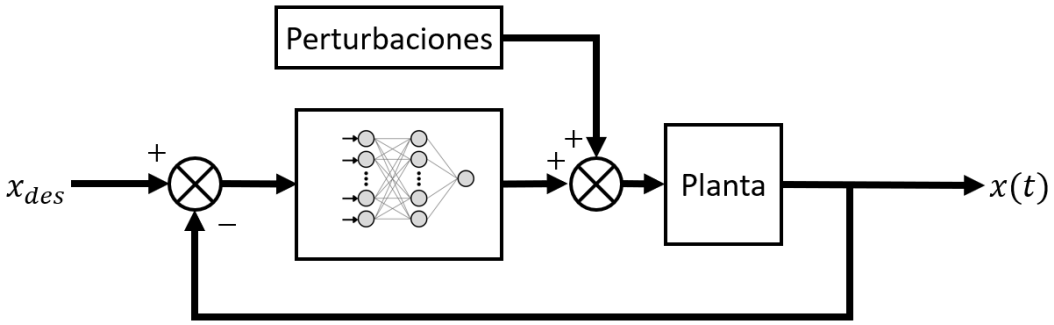

Figura 4.2: Diagrama a bloques del control neuronal adaptativo.

Para realizar la simulación, se utilizó el lenguaje de programación Python el cual permite obtener la respuesta de los controladores en presencia de diferentes tipos de perturbaciones a través del tiempo. Los casos analizados son:

1. Perturbación dependiente del estado,

$$\dot{x}(t) = u(t) + f(x(t)) = u(t) + 6\text{sen}(3x(t)).$$

2. Perturbación dependiente del estado más una contante,

$$\dot{x}(t) = u(t) + f(x(t)) + \delta = u(t) + 6\text{sen}(3x(t)) + 10.$$

3. Perturbación dependiente del estado más una variante en el tiempo,

$$\dot{x}(t) = u(t) + f(x(t)) + \varsigma(t) = u(t) + 6\text{sen}(3x(t)) + 10\text{sen}(2t).$$

Siendo $6\text{sen}(3x(t))$ el termino no lineal del sistema. A continuación, en la Figura 4.3 se presenta un diagrama de flujo general que se siguió en las simulaciones.

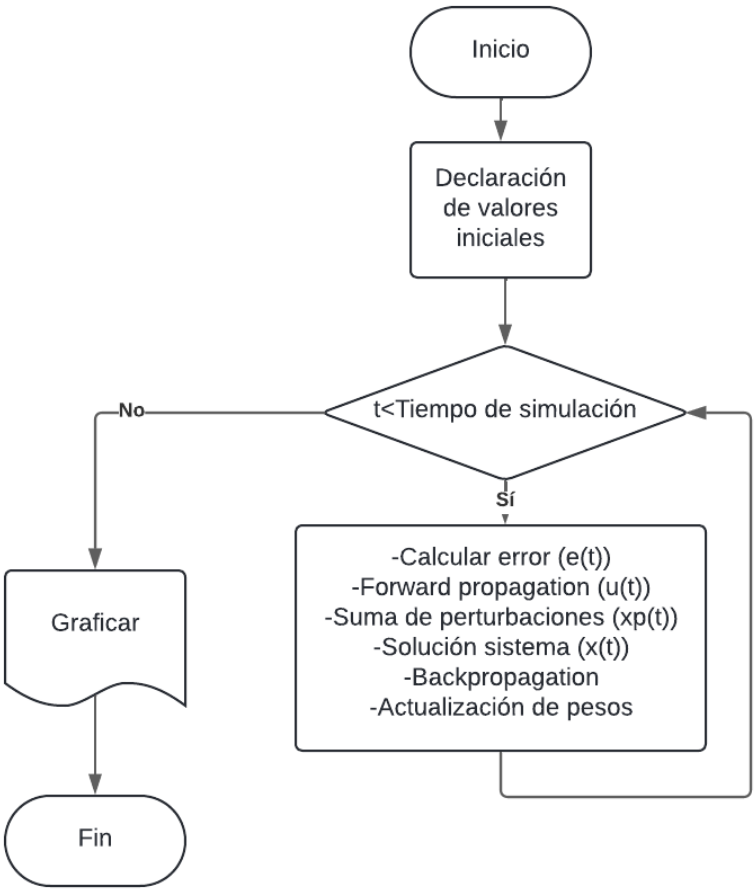

Figura 4.3: Diagrama de flujo general utilizado para la simulación.

4.2.1. Control Neuro-Inspirado empleando una ANN (función de activación sigmoide)

En la ecuación (4.12) se presenta la función de activación sigmoide empleada para la red neuronal tradicional,

$$\varphi = \frac{1}{1+e^{-x}}, \tag{4.12}$$

además, su derivada es representada por:

$$\varphi' = \frac{e^{-x}}{(1+e^{-x})^2}, \tag{4.13}$$

la función sigmoide tiene una forma más sencilla para su derivada, la cual es representada por:

$$\varphi' = \varphi(1-\varphi). \tag{4.14}$$

4.2.2. Control Neuro-Inspirado empleando una red de tipo RBF

En la ecuación (4.15) se presenta la función de activación utilizada para este controlador,

$$\varphi = \exp\left(-\frac{(x-\mu)^2}{\sigma^2}\right), \tag{4.15}$$

donde μ y σ representan la media y la varianza de las entradas x respectivamente. Además, esta red cuenta con 600 neuronas en la capa oculta.

4.2.3. Control Neuro-Inspirado empleando una SNN

En los capítulos anteriores se desarrollaron las funciones necesarias para simular las neuronas del modelo LIF, para el Forward propagation y el Backpropagation, por lo que se realizó una librería propia la cual se nombró *LIF_LIB* en donde se contienen todas estas funciones. Por lo tanto, para la simulación solo es necesario seguir el diagrama de flujo general de la Figura 4.3. Para la optimización de la red se decidió aumentar el número de spikes aumentando la resolución del *rate coding*, pasando de $1ms$ por spike a $0.1ms$ lo cual mejora considerablemente el comportamiento de la SNN.

4.3. Resultados de los controles Neuro-Inspirados

A continuación, se presentan y describen los resultados obtenidos de cada uno de los controles neuro-inspirados implementados en el sistema de primer orden con cada una de las incertidumbres anteriormente mencionadas. En todos los casos se considera una posición deseada $x_d = 10$ y una condición inicial $x_0 = -5$.

4.3.1. Perturbación dependiente del estado

En la Figura 4.4 se puede visualizar la respuesta para el sistema de la forma $\dot{x}(t) = u(t) + 6\text{sen}(3x(t))$, con una perturbación dependiente del estado. La primera gráfica representa la posición a través del tiempo y se puede observar que el control con la respuesta más rápida es el que utiliza una ANN (sigmoide), únicamente presentado ligeras oscilaciones. Por otro lado, el control RBF también es rápido pero presenta un sobre impulso de casi 20 % . Finalmente, el SNN no presenta sobre impulso, pero su respuesta es más lenta. La segunda gráfica corresponde al error de posición y se puede observar con mayor precisión que la red ANN reduce su error a 0.32 en el segundo 0.5, seguido por el SNN con un error de 0.68 en el segundo 2.38 y finalmente el RBF con un error de 0.95 a los 2.94 segundos. Por otro lado, en la Figura 4.5 se pueden ver los spikes de cada una de las neuronas de la SNN en los últimos $100ms$.

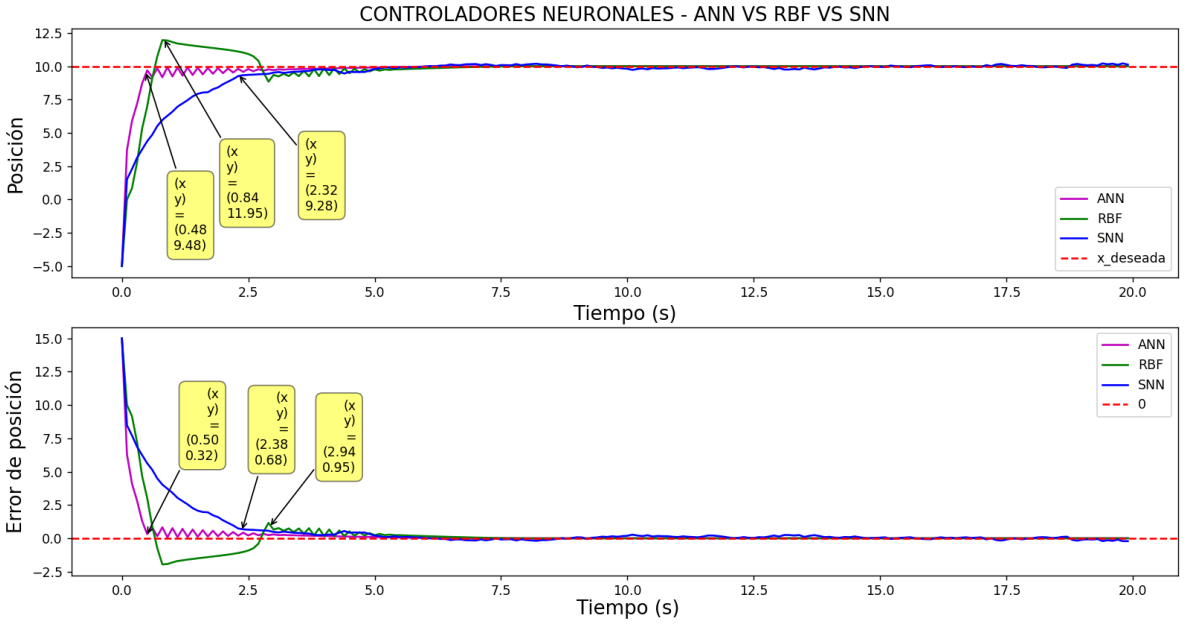

Figura 4.4: Resultados obtenidos para una perturbación dependiente del estado.

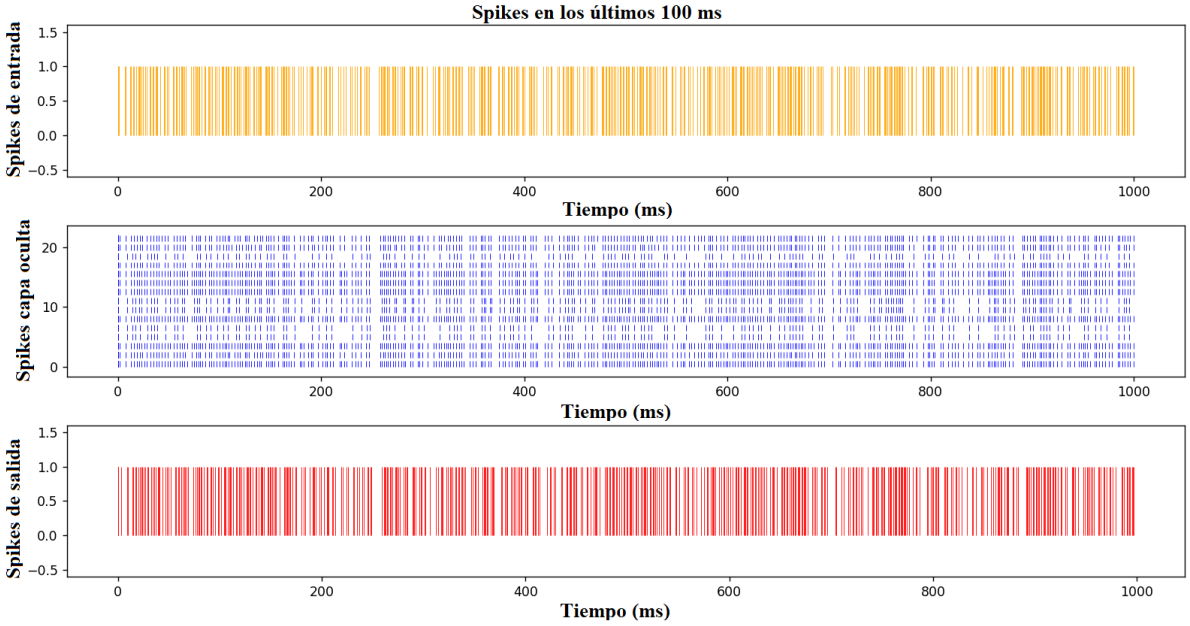

Figura 4.5: Spikes de la red SNN en los últimos $100ms$.

4.3.2. Perturbación dependiente del estado más una contante

En la Figura 4.6 se puede visualizar la respuesta para el sistema de la forma $\dot{x}(t) = u(t) + 6\text{sen}(3x(t)) + 10$, con una perturbación dependiente del estado más otra constante a

partir del segundo 10. La primera gráfica representa la posición a través del tiempo y se puede observar que el control neuronal SNN es más robusto ante las perturbaciones, pues es prácticamente imperceptible su efecto, el RBF compensa de manera rápida la perturbación constante, regresando prácticamente a la posición deseada a partir del segundo 9.41 mientras que el ANN la alcanza en el segundo 12.3. La segunda gráfica corresponde al error de posición y se puede observar con mayor precisión que el control neuronal SNN a pesar de ser el más robusto también es más lento. Por otro lado, en la Figura 4.7 se pueden ver los spikes de cada una de las neuronas de la SNN en los últimos $100ms$.

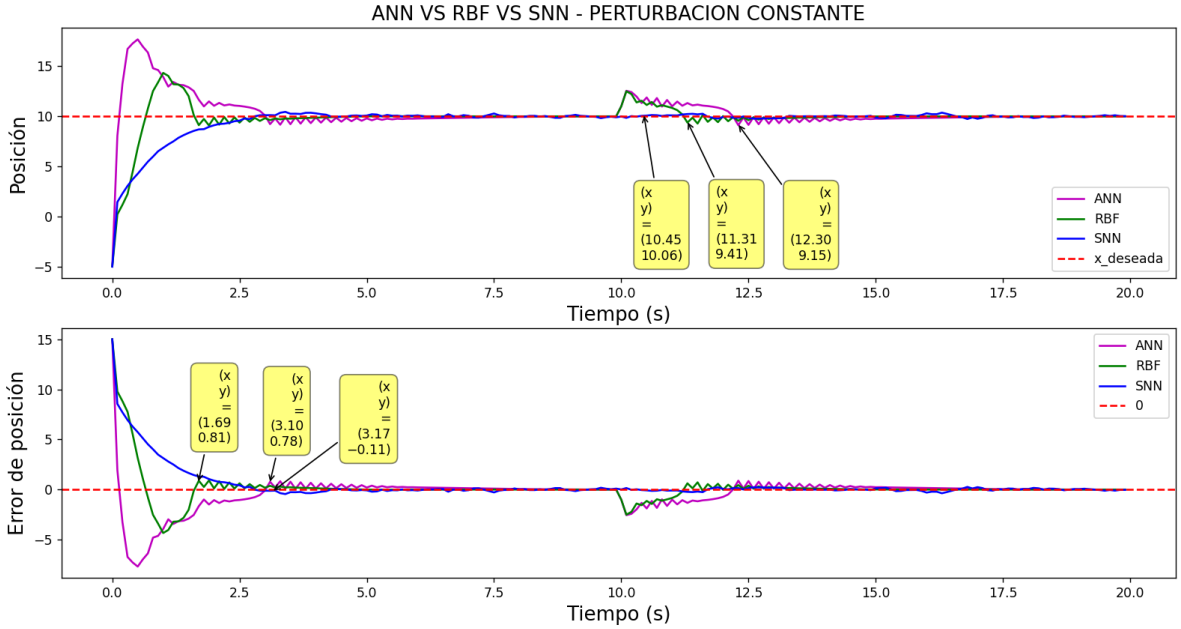

Figura 4.6: Resultados obtenidos para una perturbación dependiente del estado más una contante.

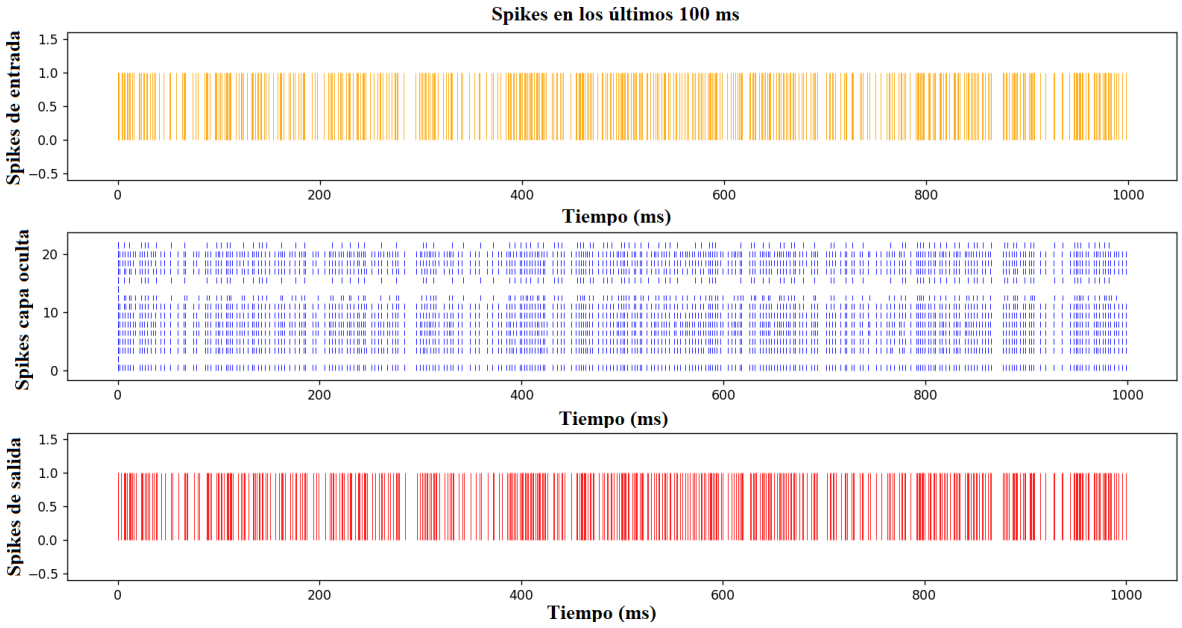

Figura 4.7: Spikes de la red SNN en los últimos $100ms$.

4.3.3. Perturbación dependiente del estado más una variante en el tiempo

En la Figura 4.8 se puede visualizar la respuesta para el sistema de la forma $\dot{x}(t) = u(t) + 6\text{sen}(3x(t)) + 10\text{sen}(2t)$, con una perturbación dependiente del estado más otra variante en el tiempo. La primera gráfica representa la posición a través del tiempo y se puede observar la robustez del control neuronal SNN ante las perturbaciones ya que se obtienen oscilaciones mínimas respecto a los demás, el RBF comienza a reducir ligeramente las oscilaciones a partir del segundo 16.35 mientras que el ANN las mantiene prácticamente en la misma magnitud durante toda la simulación. La segunda gráfica corresponde al error de posición y se puede observar que en esta ocasión el control ANN reduce el error inicial más rápido, sin embargo, el SNN mantiene una magnitud menor durante prácticamente toda la simulación. Por otro lado, en la Figura 4.9 se pueden ver los spikes de cada una de las neuronas de la SNN en los últimos $100ms$.

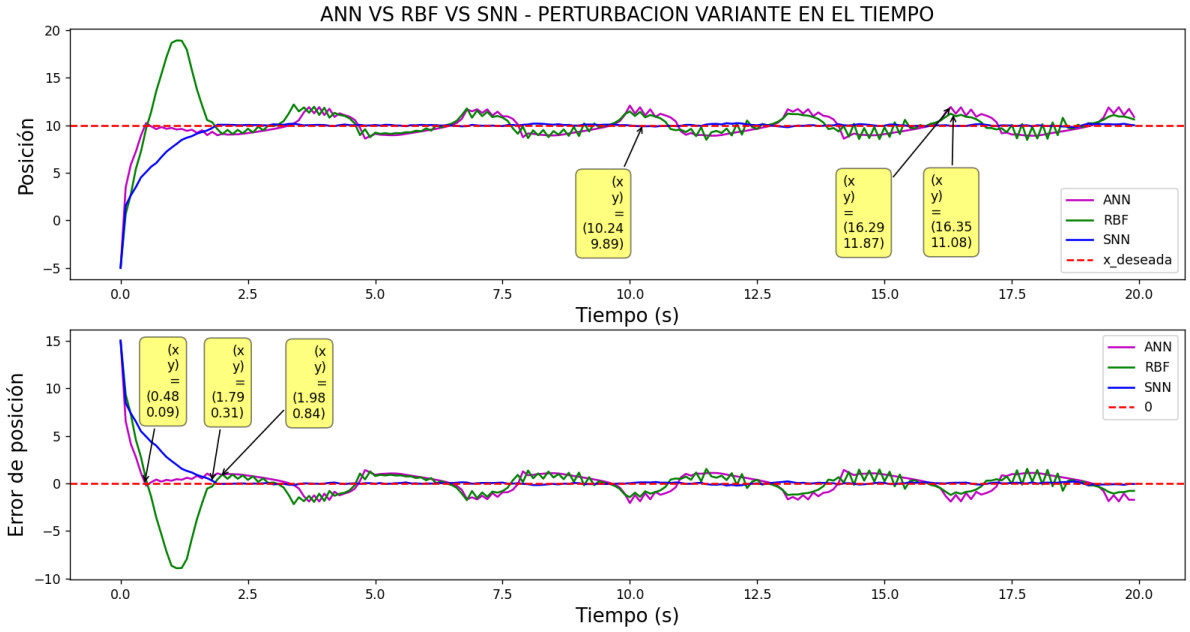

Figura 4.8: Perturbación dependiente del estado más una variante en el tiempo.

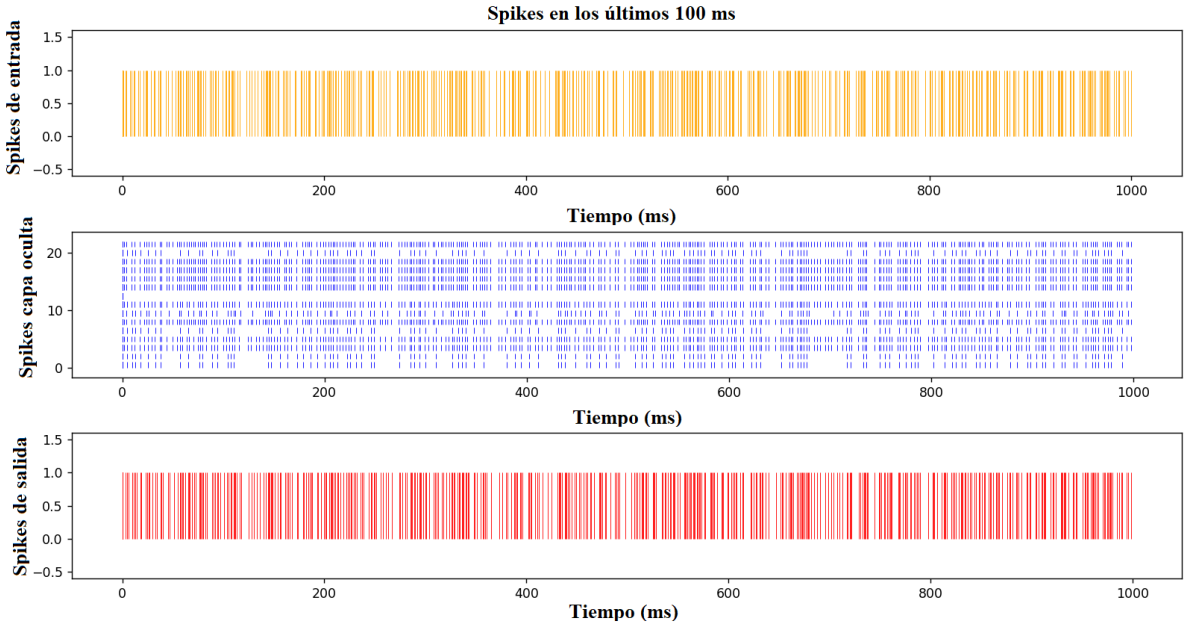

Figura 4.9: Spikes de la red SNN en los últimos $100ms$.

Capítulo 5

Implementación de un control Neuro-Inspirado en un sistema multiagente

El desarrollo de sistemas autónomos es un tópico bastante importante en la actualidad, su relevancia continúa creciendo en diversos ámbitos de la sociedad. Una categoría en pleno auge son los sistemas multiagente (SMA), los cuales permiten solucionar problemas complejos mediante la inteligencia distribuida y el trabajo colaborativo, sin embargo, su autonomía se ve limitada debido a los altos costos y tiempos computacionales, por lo que actualmente se buscan nuevas formas de conseguir una eficiencia energética. En este sentido, los algoritmos de control han evolucionado a fin de realizar tareas más complejas mostrando una autonomía elevada, en donde los sistemas de control neuronal se distinguen por la plasticidad y desempeño frente a condiciones inesperadas. Las redes neuronales tienen la capacidad de aprender, por lo que, si en un sistema conocemos la entrada, el valor de salida deseado y el valor de salida actual, un controlador integrado por una red neuronal es capaz de modificar sus parámetros aprendiendo la dinámica de la planta hasta conseguir un sistema confiable.

En este capítulo se realiza la implementación mediante simulación de un sistema multiagente básico conformado por dos agentes. Para ambos agentes se considera el modelo de primer orden no lineal con perturbaciones que se analizó en el capítulo anterior. Además se realiza un análisis comparativo contra un sistema multiagente que hace uso de una red neuronal tradicional (ANN) que utiliza una función de activación sigmoide. La simulación se lleva a cabo a utilizando el lenguaje de programación Python.

5.1. Simulación de un sistema Multi-Agente

Los sistemas multiagente (MAS) son un área central de la artificial contemporánea. Un sistema multiagente consta de múltiples agentes que interactúan en un entorno compartido para lograr objetivos comunes. La investigación de MAS abarca una variedad de problemas técnicos, cómo el diseñar algoritmos que permitan a uno o más agentes lograr objetivos específicos en un MAS. Además, el cómo se comunican y propagan información entre agentes [26].

Los sistemas multiagente y el flujo laplaciano están estrechamente relacionados a través de la teoría de grafos y la dinámica de redes. El flujo laplaciano proporciona una herramienta poderosa para el análisis y diseño de algoritmos de control distribuido y sincronización en sistemas multiagente [27]. El flujo laplaciano nos permite obtener la dinámica cooperativa entre los agentes, esto es el cómo los agentes se relacionan e interactúan entre ellos.

Generalmente el estudio de la dinámica de estos sistemas supone que cada agente es asociado con un estado que cambia dinámicamente. El estado de un agente cambia debido a múltiples causas y en particular a las interacciones que se producen cuando el agente entra en contacto con otro agente [28]. Sea G un dígrafo dirigido ponderado con n nodos. La dinámica colectiva de un grupo de agentes puede ser descrita por la siguiente expresión [29]:

$$\dot{\boldsymbol{x}} = -L\boldsymbol{x} \tag{5.1}$$

en donde L es la matriz laplaciana. Y dado el dígrafo G con matriz de adyacencia A y la matriz de grados externos D_{out}. La matriz laplaciana de G es [27]:

$$L = D_{out} - A \tag{5.2}$$

A es definida de la siguiente manera: para cada borde (i, j), la entrada (i, j) de A es igual al peso a_{ij} del borde (i, j) y todas las demás entradas de A son igual a cero. En otras palabras, $a_{ij} > 0$ si y solo si (i, j) es un borde de G y $a_{ij} = 0$ en cualquier otro caso. Finalmente, la matriz de grados externos D_{out} de un dígrafo G es la matriz diagonal definida por:

$$D_{out} = \begin{bmatrix} d_{out}(1) & 0 & 0 \\ 0 & \ddots & 0 \\ 0 & 0 & d_{out}(n) \end{bmatrix} \tag{5.3}$$

donde $d_{out}(n)$ es el grado de salida ponderado, definido por:

$$d_{out}(n) = \sum_{j=1}^{n} a_{ij}, \tag{5.4}$$

es decir, $d_{out}(n)$ es la suma de los pesos de todos los bordes de salida en G.

Para lograr una correcta simulación de un sistema multiagente, es necesario obtener la dinámica colectiva del sistema, la cual se encuentra con ayuda de la teoría de grafos. Posteriormente se realiza el diagrama a bloques del sistema y finalmente, se realiza la implementación en código.

5.1.1. Dinámica colectiva del sistema multiagente

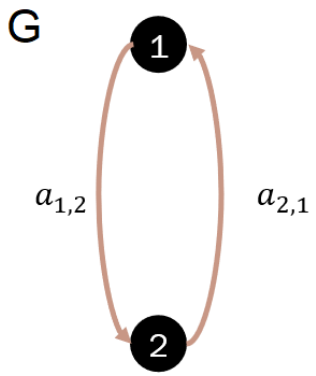

Figura 5.1: Dígrafo de dos agentes ponderados.

Considerando el dígrafo ponderado G, mostrado en la figura 5.1, cual es conformado por dos agentes. A es la matriz de adyacencia y es igual a el peso $a_{(i,j)}$ del borde (i, j) y todas las otras entradas de A son iguales a cero, la matriz A se representa de la siguiente manera:

$$A = \begin{bmatrix} 0 & a_{1,2} \\ a_{2,1} & 0 \end{bmatrix}. \tag{5.5}$$

D_{out} es la matriz de grados externos y es la suma de los pesos de todos los bordes externos, por lo que se obtiene:

$$D_{out} = \begin{bmatrix} a_{1,2} & 0 \\ 0 & a_{2,1} \end{bmatrix}. \tag{5.6}$$

L es la matriz Laplaciana y esta dada por $L = D_{out} - A$,

$$L = \begin{bmatrix} a_{1,2} & -a_{1,2} \\ -a_{2,1} & a_{2,1} \end{bmatrix}. \tag{5.7}$$

La dinámica colectiva del sistema esta dada por la siguiente expresión [27], $\dot{\boldsymbol{x}} = -L\boldsymbol{x}$,

$$\begin{bmatrix} \dot{x_1} \\ \dot{x_2} \end{bmatrix} = - \begin{bmatrix} a_{1,2} & -a_{1,2} \\ -a_{2,1} & a_{2,1} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}. \tag{5.8}$$

Por lo tanto, la dinámica colectiva del sistema es:

$$\begin{bmatrix} \dot{x_1} \\ \dot{x_2} \end{bmatrix} = \begin{bmatrix} -a_{1,2}x_1 + a_{1,2}x_2 \\ a_{2,1}x_1 - a_{2,1}x_2 \end{bmatrix}. \tag{5.9}$$

5.1.2. Diagrama a bloques

El objetivo principal para los agentes es seguirse mutuamente, en otras palabras, la posición deseada del Agente 1 será la posición actual del Agente 2 y viceversa. Además, se añade un *offset* el cual asegure que se mantenga una distancia de seguridad entre los agentes, por lo que:

$$x_1 - x_2 = offset. \tag{5.10}$$

En la figura 5.2 se muestra el diagrama a bloques de control obtenido para este sistema.

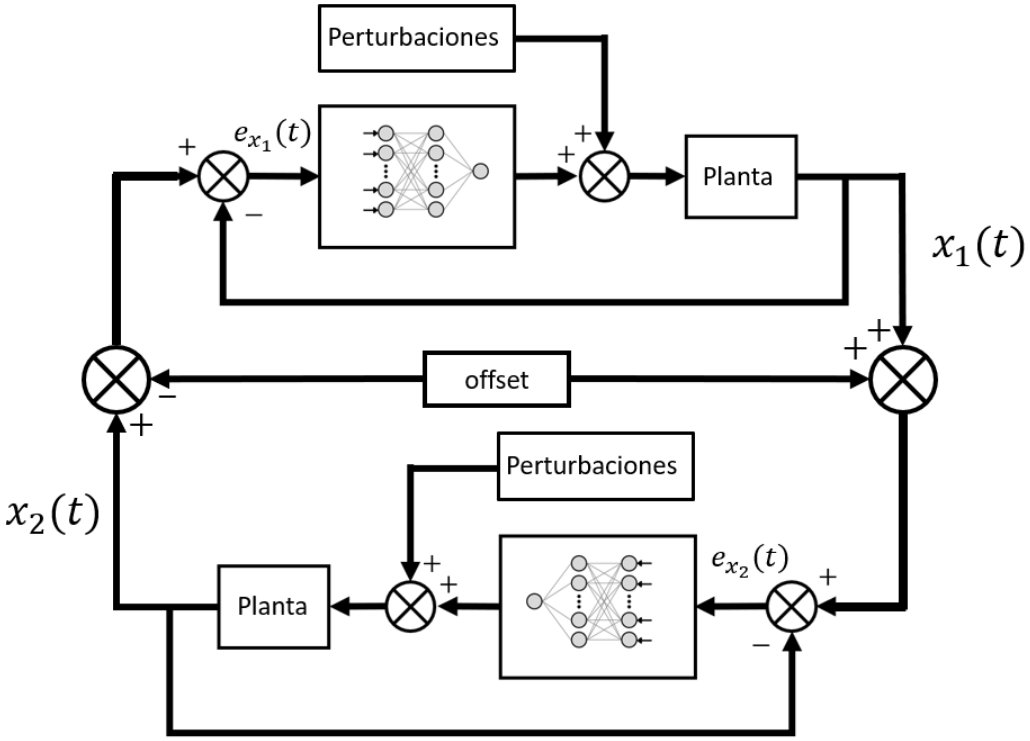

Figura 5.2: Diagrama a bloques del control multiagente.

5.2. Resultados de un control SNN en un contexto Multi-Agente

A continuación, se presentan y describen los resultados obtenidos. Se analizaron diferentes casos para poder comprobar el comportamiento del sistema multiagente, así como la efectividad del control neuro-inspirado. En todos los casos se considera una condición inicial $x_0^1 = 20$ para el agente 1 y $x_0^2 = -20$ para el agente 2.

5.2.1. Simulación de un SMA sin *offset* con perturbaciones

En la figura 5.3 se puede visualizar la respuesta para un SMA conformado por 2 agentes sin *offset*, bajo todas las perturbaciones analizadas en el capítulo anterior pero en distintos

instantes de tiempo. La primera gráfica representa la posición a través del tiempo y se puede observar que gracias a la robustez del control neuronal SNN ante las perturbaciones, los agentes parten de sus respectivas condiciones iniciales, se encuentran en el punto medio (0) y ahí se mantienen durante el resto de la simulación. Por otro lado, el SMA con ANN si es afectado por las perturbaciones, cambiando la posición de los agentes en casi cada instante de tiempo de la simulación. La segunda gráfica corresponde al error de posición y se puede observar que el control SNN lo reduce ligeramente más rápido.

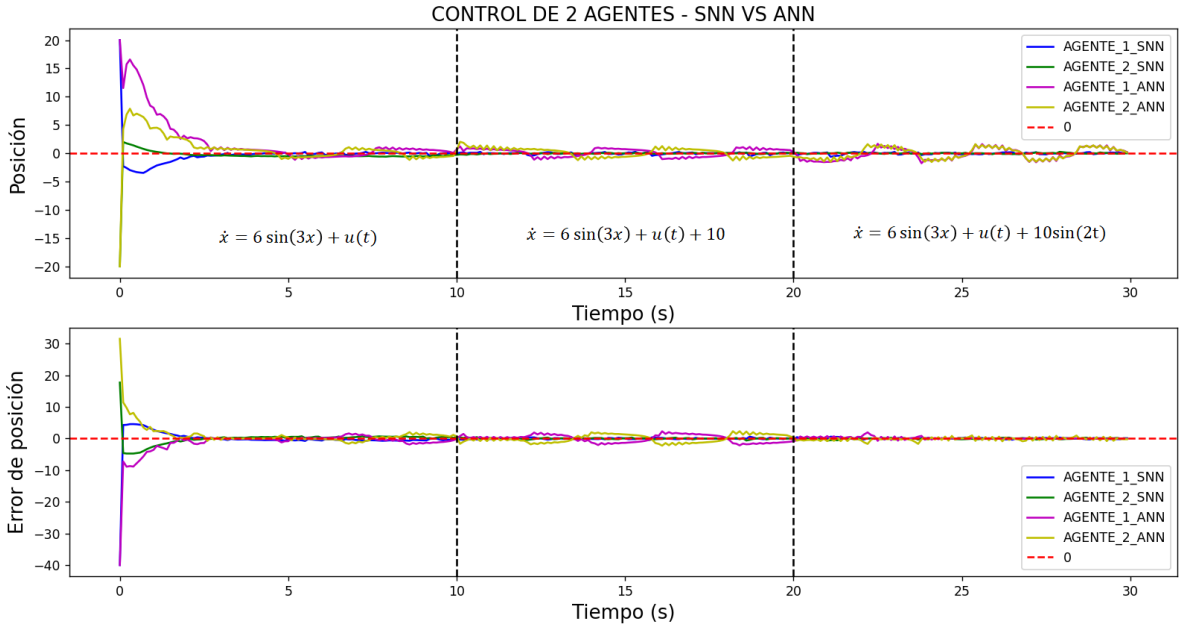

Figura 5.3: Simulación de un SMA sin offset ante perturbaciones.

5.2.2. Simulación de un SMA con *offset* con perturbaciones

En la figura 5.4 se puede visualizar la respuesta para un SMA conformado por 2 agentes con un *offset*= 10, bajo todas las perturbaciones analizadas en el capítulo anterior pero en distintos instantes de tiempo. La primera gráfica representa la posición a través del tiempo y se puede observar que los agentes con el control neuro-inspirado, los agentes parten de sus respectivas condiciones iniciales y se encuentran en un punto medio ($x^1 = 5$ y $x^2 = -5$) y ahí se mantienen durante el resto de la simulación. Por otro lado, los agentes con el controlador ANN son afectados en mayor medida por las perturbaciones. La segunda gráfica corresponde al error de posición y se puede observar que tanto en el control SNN como en el ANN, los agentes se encuentran después de los 3 segundos.

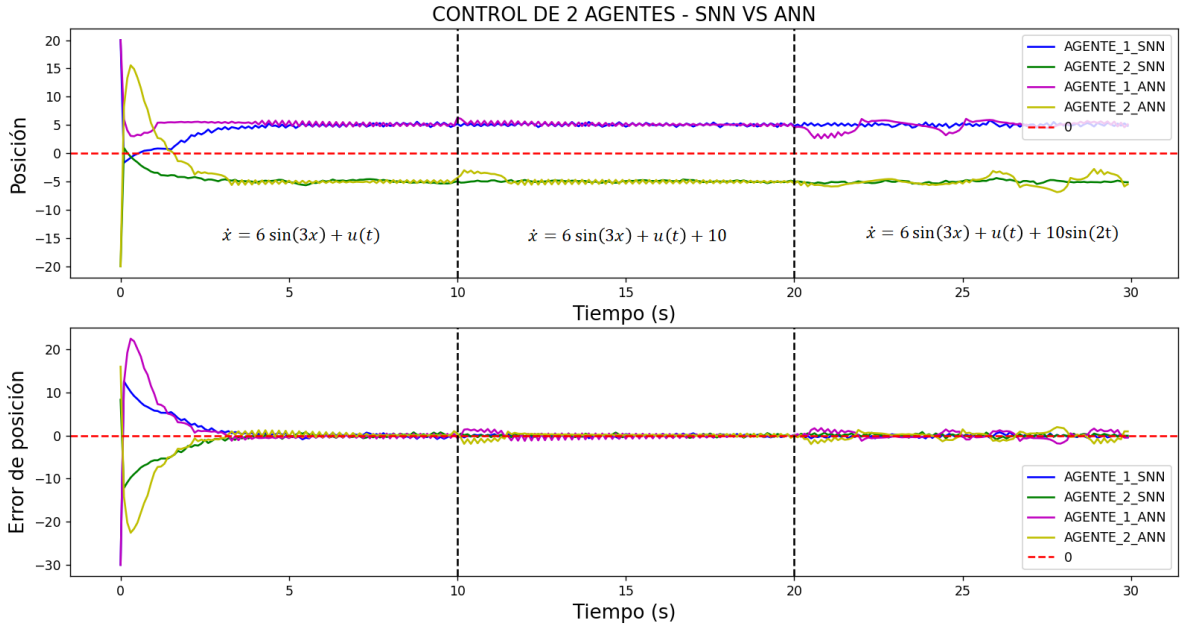

Figura 5.4: Simulación de un SMA con offset= 10 ante perturbaciones.

5.2.3. Simulación de un SMA con posición deseada constante con perturbaciones

En la figura 5.5 se puede visualizar la respuesta para un SMA conformado por 2 agentes con un *offset*= 10 y esta vez con una posición deseada $x_d^1 = 15$ para el agente 1, bajo todas las perturbaciones analizadas anteriormente pero en distintos instantes de tiempo. La primera gráfica representa la posición a través del tiempo y se puede observar que los agentes con el control SNN parten de sus respectivas condiciones iniciales y el agente 1 alcanza la posición deseada $x_d^1 = 15$ mientras que el agente 2 se mantiene a una distancia $x_d^2 = 5$ manteniendo siempre el *offset*= 10. Por otro lado, los agentes con el controlador ANN siguen siendo afectados por las perturbaciones. La segunda gráfica corresponde al error de posición y se puede observar que nuevamente al control neuronal SNN le toma más tiempo hacer que los agentes lleguen a su posición deseada, mientras que en el ANN, los agentes se encuentran ligeramente más rápido.

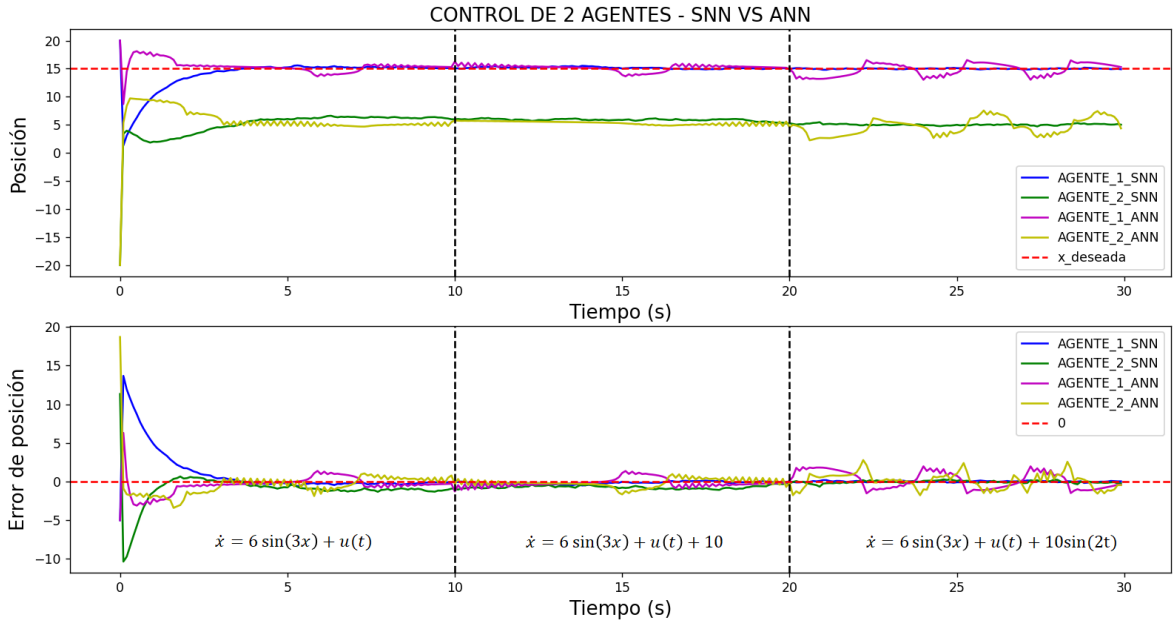

Figura 5.5: Simulación de un SMA con $x_d^1 = 15$ ante perturbaciones.

5.2.4. Simulación de un SMA con posición deseada variante con perturbaciones

En la figura 5.6 se puede visualizar la respuesta para un SMA conformado por 2 agentes con un *offset*= 10 y esta vez con una posición deseada x_d que varía en diferentes instantes de tiempo para el agente 1, bajo la perturbación dependiente del estado más una variante en el tiempo. La primera gráfica representa la posición a través del tiempo y se puede observar que los agentes con el control neuronal SNN parten de sus respectivas condiciones iniciales y el agente 1 alcanza la posición deseada x_d^1 mientras que el agente 2 se mantiene a una distancia $x^2 = x_d^1 - offset$, nuevamente, gracias a la robustez del control por SNN se aprecian movimientos más suaves dentro del SMA. Por otro lado, los agentes con el controlador ANN son afectados por las perturbaciones manteniendo oscilaciones de mayor magnitud. La segunda gráfica corresponde al error de posición y se puede observar que al control neuronal SNN le toma más tiempo hacer que los agentes lleguen a su primera posición deseada, mientras que en el ANN los agentes se encuentran rápidamente, sin embargo, para las siguientes posiciones deseadas, ambos controladores se comportan de manera similar en tanto a la reducción de error de posición entre los agentes.

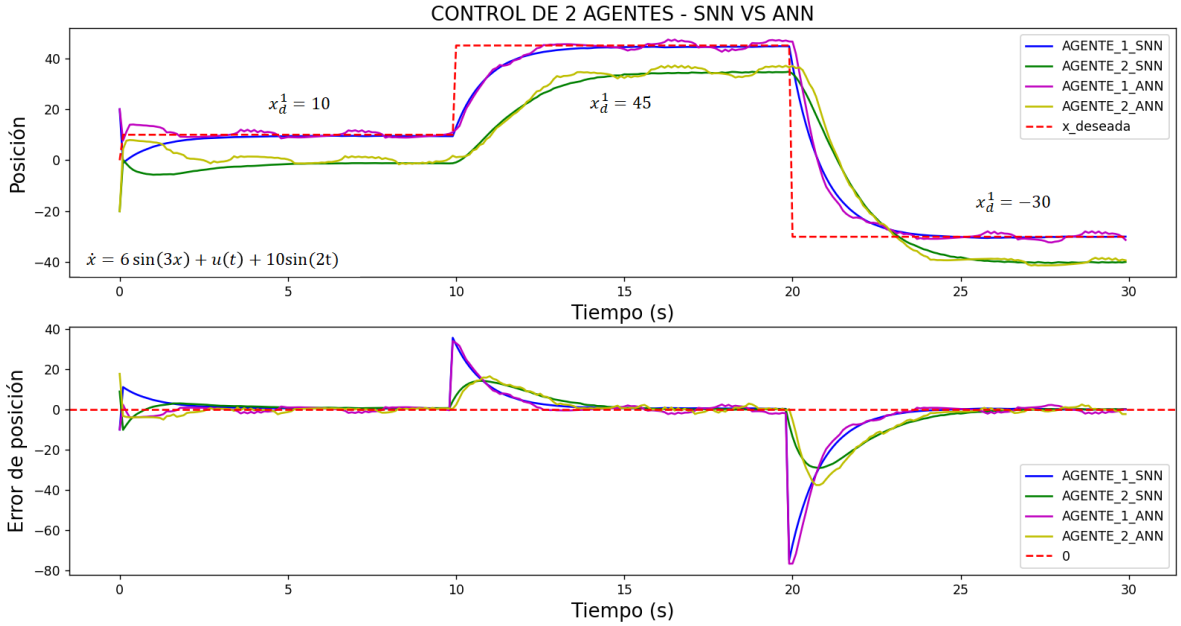

Figura 5.6: Simulación de un SMA con x_d^1 variante con perturbaciones.

Conclusiones generales

A partir de los resultados obtenidos a través de cada uno de los capítulos se puede concluir que se han cumplido los objetivos planteados en este proyecto de tesis.

- Utilizando el lenguaje de programación Python se comprobó el comportamiento de las redes neuronales de tipo Spiking tomando como base el modelo LIF. Se desarrolló una serie de códigos que permitieron realizar el *Forward propagation* de una SNN, además se logró establecer una metodología de aprendizaje en línea para las SNN. Está metodología permitió la implementación de las SNN dentro de un contexto de control, obteniendo un control neuro-inspirado el cual es capaz de compensar de manera robusta perturbaciones no lineales en un sistema de primer orden.

- A través de la dinámica colectiva, se implementaron controladores neuronales dentro de un sistema multiagente básico, conformado por dos agentes. Los resultados demostraron que el control neuro-inspirado con SNN es capaz de controlar de manera correcta la posición de los dos agentes sin que se vean radicalmente afectados por las perturbaciones presentes en el sistema. El control propuesto con SNN realiza una adaptación (aprendizaje) en línea, por lo que no necesita de una etapa de entrenamiento previa, lo que permite que aún con el cambio de las perturbaciones, el sistema siga adaptándose y consiguiendo errores muy cercanos a cero.

- Este trabajo muestra una de las muchas aplicaciones que pueden tener las redes neuronales Spiking, se analizó su comportamiento dentro de un lazo cerrado de control. Actualmente existen pocos trabajos que utilicen este enfoque en las SNN, por lo que se considera como un área de oportunidad en la investigación. Los resultados muestran un futuro prometedor para los controladores neuronales, especialmente para los que emplean SNN, por lo cual como trabajo futuro se plantea la posibilidad de ser implementados en sistemas de segundo orden, para de esta manera poder visualizar su comportamiento ante sistemas más complejos. Además, se busca compararlo contra otro tipo de controladores actuales, como lo son los controles fraccionarios o por modos deslizantes, los cuales son métodos de control robusto y no lineal.

Bibliografía

[1] LW Meng, GC Qiao, XY Zhang, J Bai, Y Zuo, PJ Zhou, Y Liu, and SG Hu. An efficient pruning and fine-tuning method for deep spiking neural network. *Applied Intelligence*, pages 1–14, 2023.

[2] Youngeun Kim and Priyadarshini Panda. Visual explanations from spiking neural networks using inter-spike intervals. *Scientific reports*, 11(1):19037, 2021.

[3] Kashu Yamazaki, Viet-Khoa Vo-Ho, Darshan Bulsara, and Ngan Le. Spiking neural networks and their applications: A review. *Brain sciences*, 12, 06 2022.

[4] Alexander Henkes, Jason K Eshraghian, and Henning Wessels. Spiking neural network for nonlinear regression. *arXiv preprint arXiv:2210.03515*, 2022.

[5] Ran Wang, Zhicheng Zhou, and Guangji Qu. Fuzzy neural network pid control based on rbf neural network for variable configuration spacecraft. In *2018 IEEE 3rd Advanced Information Technology, Electronic and Automation Control Conference (IAEAC)*, pages 1203–1207. IEEE, 2018.

[6] ZHANG Xin and QUAN Ying. Research on fixed time sliding mode control strategy based on rbf neural network. *Journal of Measurement Science & Instrumentation*, 14(2), 2023.

[7] Raquel Flórez and José Fernández. Las redes neuronales artificiales, fundamentos teoricos y aplicaciones practicas. *La Coruña, Oleiros, España: Netbiblo*, 2008.

[8] Juan Manuel Corchado. *Redes Neuronales Artificiales: un enfoque práctico*. Pearson Educación, 2004.

[9] F Lara Rosano. Fundamentos de redes neuronales artificiales, 1996.

[10] Damián Jorge Matich. Redes neuronales: Conceptos básicos y aplicaciones. *Universidad Tecnológica Nacional, México*, 41:12–16, 2001.

[11] Haiyan Wang and Fanwei Meng. Control method of nanomaterial numerical control electronic processing based on rbf neural network. *Advances in Materials Science and Engineering*, 2022(1):1904431, 2022.

[12] Jianhui Han, Zhaolin Li, Weimin Zheng, and Youhui Zhang. Hardware implementation of spiking neural networks on fpga. *Tsinghua Science and Technology*, 25(4):479–486, 2020.

[13] Nikola K Kasabov. *Time-space, spiking neural networks and brain-inspired artificial intelligence*. Springer, 2019.

[14] Wenzhe Guo, Mohammed E. Fouda, Ahmed M. Eltawil, and Khaled Nabil Salama. Neural coding in spiking neural networks: A comparative study for robust neuromorphic systems. *Frontiers in Neuroscience*, 15, 2021.

[15] Jason K Eshraghian, Max Ward, Emre Neftci, Xinxin Wang, Gregor Lenz, Girish Dwivedi, Mohammed Bennamoun, Doo Seok Jeong, and Wei D Lu. Training spiking neural networks using lessons from deep learning. *arXiv preprint arXiv:2109.12894*, 2021.

[16] Kinjal Patel, Eric Hunsberger, Sean Batir, and Chris Eliasmith. A spiking neural network for image segmentation. *arXiv preprint arXiv:2106.08921*, 2021.

[17] Jason K. Eshraghian, Max Ward, Emre O. Neftci, Xinxin Wang, Gregor Lenz, Girish Dwivedi, Mohammed Bennamoun, Doo Seok Jeong, and Wei D. Lu. Training spiking neural networks using lessons from deep learning. *Proceedings of the IEEE*, 111(9):1016–1054, 2023.

[18] Peter U Diehl, Guido Zarrella, Andrew Cassidy, Bruno U Pedroni, and Emre Neftci. Conversion of artificial recurrent neural networks to spiking neural networks for low-power neuromorphic hardware. In *2016 IEEE International Conference on Rebooting Computing (ICRC)*, pages 1–8. IEEE, 2016.

[19] Christoph Stöckl and Wolfgang Maass. Optimized spiking neurons can classify images with high accuracy through temporal coding with two spikes. *Nature Machine Intelligence*, 3(3):230–238, 2021.

[20] Shuiying Xiang, Yahui Zhang, Junkai Gong, Xingxing Guo, Lin Lin, and Yue Hao. Stdp-based unsupervised spike pattern learning in a photonic spiking neural network with vcsels and vcsoas. *IEEE Journal of Selected Topics in Quantum Electronics*, 25(6):1–9, 2019.

[21] Ashish Gautam and Takashi Kohno. An adaptive stdp learning rule for neuromorphic systems. *Frontiers in Neuroscience*, 15:741116, 2021.

[22] David Weinberg. Fantastic spiking neural networks and how to train them, 2021.

[23] Shuai Li, Yinyan Zhang, Shuai Li, and Yinyan Zhang. Neural networks for robot arm cooperation with a full distributed control topology. *Neural Networks for Cooperative Control of Multiple Robot Arms*, pages 49–74, 2018.

[24] M Revanesh, Sheetal S Gundal, JR Arunkumar, P Joel Josephson, S Suhasini, and T Kalavathi Devi. Artificial neural networks-based improved levenberg–marquardt neural network for energy efficiency and anomaly detection in wsn. *Wireless Networks*, pages 1–16, 2023.

[25] Shengnan Li, Chuankui Yan, and Ying Liu. Energy efficiency and coding of neural network. *Frontiers in Neuroscience*, 16:1089373, 2023.

[26] Stefano Albrecht. How can research in multi-agent systems help us to address challenging real-world problems?, 2024.

[27] Francesco Bullo, Jorge Cortés, Florian Dörfler, and Sonia Martínez. *Lectures on network systems*, volume 1. CreateSpace, 2018.

[28] Stefania Monica, Federico Bergenti, and Franco Zambonelli. A kinetic approach to investigate the collective dynamics of multi-agent systems. *International Journal on Software Tools for Technology Transfer*, 25(5):693–705, 2023.

[29] Reza Olfati-Saber, J Alex Fax, and Richard M Murray. Consensus and cooperation in networked multi-agent systems. *Proceedings of the IEEE*, 95(1):215–233, 2007.