Simulación de redes neuronales basadas en sistemas memristivos para aprendizaje supervisado

Walter Javier Quiñonez
Director: Dr. Diego Rubi
Co-Directora: Dra. María José Sánchez

Tesis de Licenciatura en Ciencias Físicas

Facultad de Ciencias Exactas y Naturales

Universidad de Buenos Aires

Diciembre 2021

Índice general

Resumen

En este trabajo se estudia, mediante simulaciones numéricas, la implementación por *hardware* de redes neuronales sencillas para reconocimiento de caracteres en base a dispositivos memristivos con geometría de barras cruzadas. Se analiza el proceso de convergencia del algoritmo de aprendizaje para distintas curvas de potenciación y depreciación sinápticas, tanto medidas experimentalmente, en distintos sistemas manganita-metal, como sintéticas. En ambos casos, se incorporaron en las simulaciones las limitaciones físicas asociadas a la implementación de las redes en sistemas físicos, entre las que mencionamos: bandas de pesos sinápticos discretas y acotadas, número de estados posibles, linealidad de las curvas de potenciación y depreciación. Las curvas de potenciación y depreciación sintéticas permiten variar de manera controlada las características anteriormente mencionadas y determinar cómo afecta cada uno de estos parámetros a la velocidad de convergencia y al *accurary* del algoritmo.

Para la simulación del entrenamiento y la actualización iterativa de los pesos sinápticos se implementó un algoritmo basado en el método del descenso por el gradiente y la regla de Manhattan. Esta regla facilita la actualización de los pesos sinápticos en una red neuronal implementada físicamente y evita gran parte de la electrónica periférica necesaria para actualizar los pesos de manera exacta a partir del descenso por el gradiente estándar.

Para cada caso, se determinaron los parámetros óptimos asociados a la función de activación utilizada, los cuales garantizan una correcta convergencia y maximizan la *accuracy*. Se concluyó que las curvas de potenciación y depreciación no lineales mejoran la velocidad de la convergencia del algoritmo. También se observó una mejora en la velocidad de convergencia al aumentar la banda de conductividades permitidas y al reducir el número de estados asociados a las curvas de potenciación y depreciación. Estas características del proceso de aprendizaje se relacionan con el método de actualización de pesos sinápticos utilizado.

Por último, se implementaron métodos que introducen estocasticidad en la convergencia del proceso de aprendizaje de la red; en particular, se utilizaron *mini-batches* y la suma de componentes aleatorios en las actualizaciones de los pesos sinápticos. Se concluyó que la implementación de estos métodos aumentan la velocidad de convergencia del proceso de aprendizaje (la situación óptima se tiene al combinar ambos métodos) y permite mejorar el *accuracy* aún para funciones de activación con parámetros no óptimos.

Los resultados obtenidos en esta tesis contribuyen a determinar y optimizar la respuesta eléctrica que deben tener los sistemas memristivos para su implementación en redes neuronales físicas.

Introducción

La Sociedad de la Información en la que vivimos hace un uso cada vez más intensivo de procesos de Inteligencia Artificial, lo cual incluye los métodos llamados de aprendizaje automático (*Machine Learning*) para la resolución de problemas complejos como el reconocimiento de imágenes o del habla. Dentro de estos métodos se destacan, entre otros, las redes neuronales. El estado del arte en redes neuronales se asocia a algoritmos (*software*) que corren sobre computadoras con arquitecturas estándar, del tipo de Von Neumann, donde la unidad de procesamiento está físicamente separada de la memoria. Sin embargo, la complejidad creciente tanto de las redes utilizadas (por ejemplo, redes profundas, convolucionales, GANs, etc. [1]) como del tamaño de los conjuntos de datos utilizados para su entrenamiento (*big data*) provocan un aumento exponencial en la cantidad de energía requerida para la implementación de los procesos de aprendizaje e inferencia, con la consiguiente huella de carbono que impacta sobre un medio ambiente que ya está siendo afectado por el cambio climático. En 2019 se estimó que el proceso de entrenamiento de una red neuronal profunda puede generar emisiones de CO_2 equivalentes a la vida útil de 5 automóviles [2]. Por otra parte, otro estudio reciente afirma que la energía asociada a cómputos de *machine learning* se duplica cada 3 meses, y se ha incrementado 300.000 veces entre 2012 y 2018 [1].

Lo anterior sugiere que la implementación masiva de estos algoritmos, en todos los aspectos de la vida moderna, implica un consumo energético no sostenible, siendo imperativo el desarrollo de máquinas significativamente más eficientes para la implementación de procesos de Inteligencia Artificial. Para lograr esta tarea, podemos tomar como modelo el cerebro humano, sistema extremadamente eficiente en la realización de tareas complejas con un consumo de energía órdenes de magnitud más bajo que el correspondiente a un microprocesador moderno [3]. Su eficiencia radica en su compleja arquitectura de neuronas y sinapsis interconectadas, la que permite la realización de cómputo en paralelo, junto a sus propiedades de plasticidad.

En los últimos años, se ha desatado una carrera de I+D para el desarrollo de plataformas de *hardware* capaces de emular las capacidades de aprendizaje de los sistemas biológicos, lo que se ha dado en llamar "cómputo neuromórfico" [4]. Entre los sistemas con este potencial podemos mencionar los sistemas "memristivos", que son estructuras metal-óxido-metal capaces de cambiar su resistencia eléctrica entre distintos estados no volátiles ante la aplicación de estímulo eléctrico externo [5]. En algunos casos, esta respuesta es multinivel y replica por lo tanto el comportamiento analógico de las sinapsis cerebrales. Se han reportado sistemas memristivos con distintas funcionalidades neuromórficas como *short-term-plasticity* [6] y *spike-time-dependant plasticity* [7].

Hace algunos años, Prezioso et al. demostraron que es posible el uso de sistemas de memristores con geometría de barras cruzadas (*cross-bars*) para la implementación de redes neuronales simples para reconocimiento de caracteres [8]. La red clasifica 3 clases de 10 imágenes cada una (3x3 píxeles), donde cada clase son pequeñas variaciones de las letras "z", "v" y "n". Las entradas de la red son los píxeles de las imágenes vectorizados y codificados en voltajes. La conductancia de los memristores del *cross-bar* se relaciona con los pesos sinápticos que conectan las neuronas de entrada/salida y las 3 neuronas de salida se activan si la imagen de entrada corresponde a la clase correcta. El proceso de aprendizaje consiste en la adaptación iterativa de los pesos sinápticos (conductividades) para minimizar una función de costo y maximizar la precisión de las clasificaciones.

Este trabajo se enfoca en la simulación de perceptrones simples para aprendizaje supervisado (reconocimiento de imágenes), como el descripto anteriormente, utilizando propiedades de potenciación y depreciación sináptica de sistemas memristivos reales basados en heteroestructuras manga-

nita/metal, con las limitaciones físicas que estos presentan (bandas de conductividades acotadas y discretas, no linealidades en la respuesta de potenciación y depreciación sináptica, etc.). Estas propiedades eléctricas fueron medidas en el Laboratorio de Ablación Láser, dirigido por el Dr. Rubi, en experimentos previos a la irrupción de la pandemia y a la suspensión de las actividades presenciales. Se simuló, mediante código desarrollado en el lenguaje de programación Python, el proceso de aprendizaje para el reconocimiento de imágenes de perceptrones simples construidos a partir de arreglos de barras cruzadas de memristores que presentan las limitaciones físicas antes detalladas.

A través de las simulaciones se buscó estudiar cómo influyen en el proceso de aprendizaje aspectos tales como:

- Variación de parámetros característicos de la función de activación utilizada.
- Ancho de las bandas de conductividad de cada sistema memristivo.
- Linealidad y no linealidad en la respuesta de potenciación y depreciación sináptica.
- Número de niveles de conductividad en las curvas de potenciación y depreciación.
- Inclusión de estocasticidad en el proceso de aprendizaje de la red.

Esta tesis se estructura de la siguiente forma: en el capítulo 1 se explican conceptos generales sobre *Machine learning*, redes neuronales y el método del descenso por el gradiente. En el capítulo 2 se describen los dispositivos memristivos, su fabricación y sus aplicaciones. En el capítulo 3 se explica el modelo simulado y se describe el algoritmo implementado. En el capítulo 4 se presentan los resultados de las simulaciones obtenidos utilizando las curvas de potenciación y depreciación sináptica experimentales, mientras que en el capítulo 5 se muestran los resultados utilizando curvas de potenciación y depreciación sináptica sintéticas. En el capítulo 6 se presentan los resultados obtenidos al introducir métodos estocásticos en el algoritmo descripto en el capítulo 3. Por último, en el capítulo 7 se discuten los resultados obtenidos, se exponen las conclusiones extraídas del trabajo realizado a lo largo de esta tesis y se plantean las expectativas a futuro para esta línea de investigación.

Capítulo 1

Machine learning y redes neuronales artificiales

1.1. Motivación

Debido al aumento acelerado de la tasa de generación de datos, la capacidad de cómputo con la que contamos para analizar esta cantidad cada vez mayor de información se ve completamente rebasada y no se prevé que los microprocesadores disponibles actualmente en el mercado puedan suplir esta necesidad de cómputo en los próximos años.

Entre los algoritmos de aprendizaje automático que más potencia han demostrado para el análisis de grandes conjuntos de datos (*big data*) se encuentran los que emulan la arquitectura y forma de procesamiento de la información del cerebro humano, como es el caso de las redes neuronales. Estos algoritmos son capaces de llevar a cabo tareas tales como clasificación de imágenes, reconocimiento del habla, procesamiento de lenguaje natural, bioinformática y, en general, están basados en redes con arquitectura jerárquica (redes profundas, convolucionales, GANs, etc. [1]), también conocidos como algoritmos de aprendizaje profundo (*deep learning*).

Si bien estos algoritmos gozan de gran éxito y difusión en el ámbito de la ciencia de datos, no se puede ignorar el consumo energético que conlleva el entrenamiento de redes para implementaciones de *deep learning*. Un ejemplo de esto es el caso de Google, que implementó un algoritmo conocido como *stacked autoencoder*, con el que se pudo reconocer de forma exitosa rostros de gatos. Para esto se utilizaron 10 millones de imágenes tomadas al azar de videos de Youtube. El proceso de entrenamiento requirió de un cluster de 16000 núcleos de procesamiento y consumió aproximadamente 100kW de potencia a lo largo de 3 días [9].

Lo anterior pone en evidencia la necesidad del desarrollo de nuevo *hardware* que permita aumentar considerablemente la eficiencia y la velocidad de ejecución de las tareas asociadas a algoritmos de *deep learning*.

El *hardware* de las computadoras modernas se basan en la tecnología de semiconductores CMOS (*complementary metal-oxide-semiconductor*) y la arquitectura de Von Neumann. Uno de los problemas que afronta la tecnología CMOS es el limite físico que se está alcanzando en la miniaturización de los transistores en los microprocesadores, lo que implica un problema para la escalabilidad de estos sistemas a futuro [10]. Por otro lado, la propia arquitectura de Von Neumann presenta intrínsecamente un problema que limita el desempeño del *hardware*, conocido como cuello de botella de Von Neumann [11] (*Von Neumann bottleneck* o *memory wall*). Este problema se debe a la comunicación entre el procesador (CPU) y los dispositivos de memoria (Fig. 1.1.(a)) [4], ya que la información debe ser transmitida de uno al otro a través de un *hardware* (*bus*) que, en general, transmite los datos almacenados en memoria a una frecuencia menor que la frecuencia de trabajo del CPU. Este flujo de datos entre el CPU y la memoria es responsable de gran parte del consumo de energía de una computadora.

Una de las soluciones al cuello de botella de Von Neumann son las arquitecturas basadas en redes neuronales biológicas (Fig. 1.1.(b)) [4], las que se inscriben dentro del llamado "cómputo neu-

romórfico", con las que se pretende desarrollar una nueva plataforma de computadoras inspiradas en los sistemas biológicos, formados por una red altamente interconectada de neuronas y sinapsis adaptables.

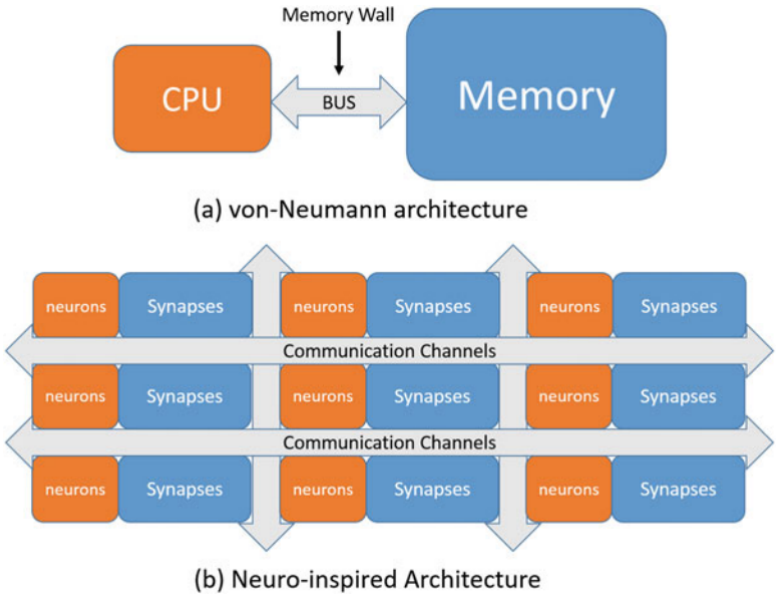

Figura 1.1: *(a) Diagrama de flujo de la arquitectura de Von Neumann en la que se basan las computadoras modernas. (b) Arquitectura inspirada en la red neuronal del cerebro humano (neuromórfica). Imagen tomada de [4].*

El cerebro humano cuenta con una serie de características que son deseables en una red neuronal artificial: es robusto y tolerante ante fallas, ya que puede lidiar con el hecho de que algunas de sus células mueren día a día sin afectar su funcionamiento de una manera apreciable; es flexible y puede ajustarse fácilmente a nuevos entornos "aprendiendo"; puede manejar información difusa, probabilística, ruidosa o inconsistente. Por último, el cerebro es compacto, consume poca energía y tiene una gran capacidad para procesar información en paralelo.

Dado que los algoritmos de *deep learning* implican operaciones con matrices de muy alta dimensionalidad, la implementación sobre *hardware* de arquitecturas neuromórficas se vuelve interesante debido a que este tipo de arquitecturas pueden procesar de manera eficiente información en paralelo. En este paradigma, el cómputo se distribuye entre todas las neuronas, que se comportan como unidades de cómputo simples, y el almacenamiento de la información se da en las sinapsis que están altamente interconectadas entre sí. Esta posible implementación de cómputo neuromórfico no reemplazaría totalmente a la arquitectura de Von Neumann, sino que se busca que complemente a esta última en tareas específicas que serían llevadas a cabo de forma más eficiente [4].

Por lo tanto, podemos considerar que la mayor motivación para el desarrollo por *hardware* de redes neuronales artificiales sería poder emular las características del cerebro humano en tareas donde las computadoras actuales no se muestran eficientes.

1.2. *Machine learning*

A grandes rasgos, los algoritmos de *Machine learning* (ML) constan de 3 partes [12]:

1. Un proceso de decisión: a partir de los datos de entrada, que pueden o no estar etiquetados, el algoritmo hace una estimación basada en un conjunto de características o *features* de los datos.

2. Una función de error: esta sirve para evaluar las predicciones del modelo. Si hay ejemplos conocidos (datos etiquetados), a partir de la función de error se puede evaluar la precisión del modelo.

3. Un proceso de optimización para el modelo: los pesos sinápticos se ajustan para reducir la discrepancia entre la estimación del modelo y los ejemplos conocidos. Este proceso se repite de forma iterativa y automática hasta que se alcanza un valor deseado de precisión.

La forma en que se pueden categorizar los distintos métodos de ML depende de la presencia o ausencia de intervención humana en los datos de entrada, si hay una recompensa o castigo, si se da un *feedback* específico o si los datos están etiquetados. Algunos de estos métodos son:

- Aprendizaje supervisado: el conjunto de datos utilizado en el aprendizaje fue etiquetado y clasificado previamente con intervención humana, de forma que es posible determinar la precisión del algoritmo en sus predicciones.
- Aprendizaje no supervisado: el conjunto de datos utilizado no fue etiquetado y un algoritmo identifica patrones y relaciones en los datos sin intervención humana.
- Aprendizaje semi-supervisado: el conjunto de datos contiene datos estructurados y no estructurados. Este método guía al algoritmo para llegar a sus propias conclusiones de forma independiente. La combinación de estos dos tipos de datos en un mismo conjunto permite a los algoritmos aprender a etiquetar datos no etiquetados.
- Aprendizaje reforzado: el conjunto de datos usa un sistema de "recompensa/castigo" dando al algoritmo un *feedback* para que aprenda en base a prueba y error.

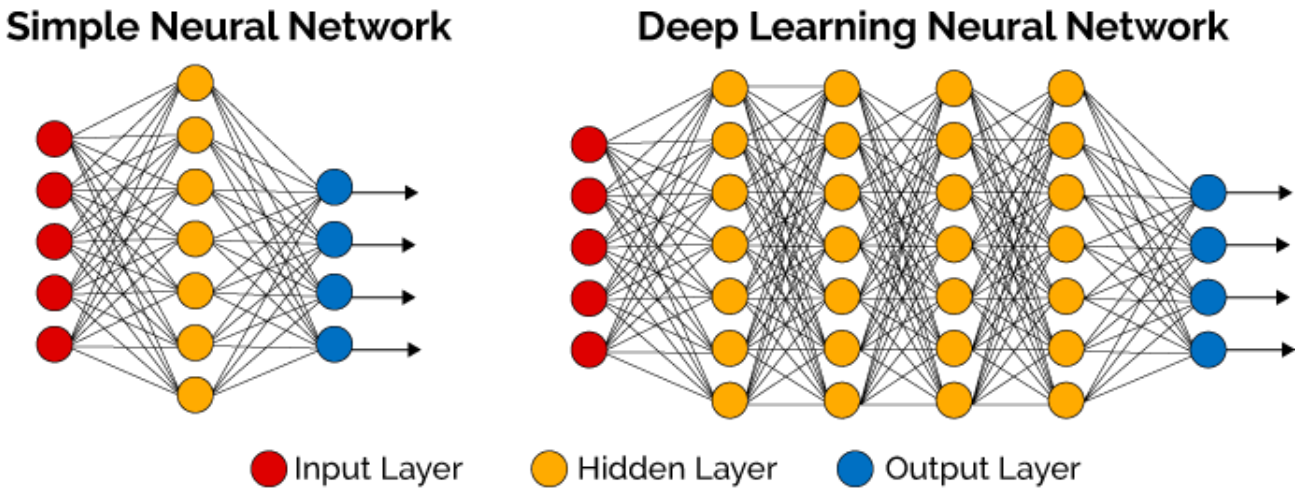

Figura 1.2: *Izquierda: Red neuronal simple con una capa de entrada, una de salida y una capa oculta. Derecha: Red profunda o de tipo deep learning; en este caso, adicionalmente a las capas de entrada y de salida, la red cuenta con cuatro capas ocultas. En los dos casos, las líneas que conectan neuronas entre sí representan a las sinapsis, que tienen un peso asociado. Imagen tomada de https://deepai.org/machine-learning-glossary-and-terms/hidden-layer-machine-learning.*

Las redes neuronales artificiales, o simplemente redes neuronales, están compuestas de capas de nodos o neuronas (Fig. 1.2), cuentan con una capa de entrada, una o más capas ocultas (intermedias) y una capa de salida. Cada nodo o neurona artificial se conecta a otras neuronas y tiene un peso asociado (peso sináptico) y un umbral. Si la salida de algún nodo individual supera el valor umbral, ese nodo se activa enviando información a la siguiente capa de la red; de lo contrario, la información no pasa a la siguiente capa. Una red neuronal que consta de más de tres capas, incluyendo las capas de entradas y salidas, se considera una red neuronal profunda. Una red neuronal que solo consta de dos o tres capas es una red neuronal simple.

Para tener una mejor distinción entre los campos de inteligencia artificial (IA), ML y *deep learning* (DL), veamos como éstos se relacionan entre sí (Fig. 1.3). IA incluye como subcampo a ML y, al

mismo tiempo, DL esta contenido como campo de estudio dentro de ML. IA se puede pensar como el conjunto de técnicas que permiten imitar algunos aspectos del comportamiento humano, por parte de máquinas, software, redes, etc; ML se enfoca en el uso de datos, métodos estadísticos y algoritmos, de forma que el aprendizaje se hace a partir de ejemplos, lo que conduce a que las tareas realizadas por las máquinas aumenten su precisión de manera gradual. En DL los métodos con los que se lleva a cabo el aprendizaje (en base a ejemplos) son los mismos que se utilizan en ML, pero implementando redes neuronales profundas.

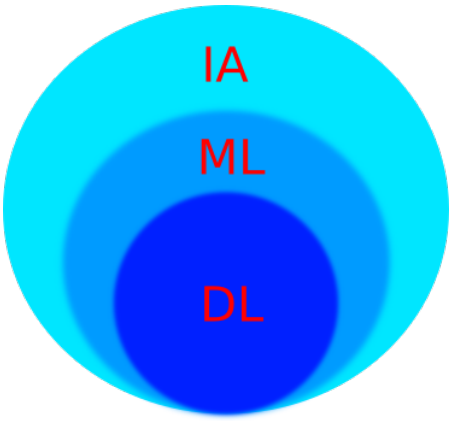

Figura 1.3: *Esquema que muestra gráficamente la relación entre los campos de IA, ML, y DL.*

Una de las diferencias más relevantes entre ML y DL es que este último automatiza gran parte de la extracción de características del proceso, eliminando parte de la intervención humana requerida y permitiendo el uso de grandes conjuntos de datos. En cambio, ML en su forma estándar depende de la intervención humana para aprender, dado que es necesario que un humano determine el conjunto de características para entender las diferencias entre los datos de entrada, requiriendo usualmente datos más estructurados.

DL puede aprovechar conjuntos de datos etiquetados (aprendizaje supervisado) pero esto no es una condición necesaria. Puede recibir datos sin estructura en forma "cruda" (*raw data*) como texto o imágenes, por ejemplo, y puede determinar de forma autónoma el conjunto de características que distingue diferentes categorías de datos entre sí.

1.3. Neurona artificial

El modelo de neurona artificial como unidad de cómputo surge del estudio de las neuronas biológicas y las redes que se forman a partir de estas. Si bien el modelo es extremadamente simple desde un punto de vista neurofisiológico, permite ganar conocimiento sobre los principios fundamentales de funcionamiento asociados a las redes neuronales biológicas, ya que no es necesario conocer todos los detalles y mecanismos de una célula individual para entender el comportamiento colectivo de toda una red de células.

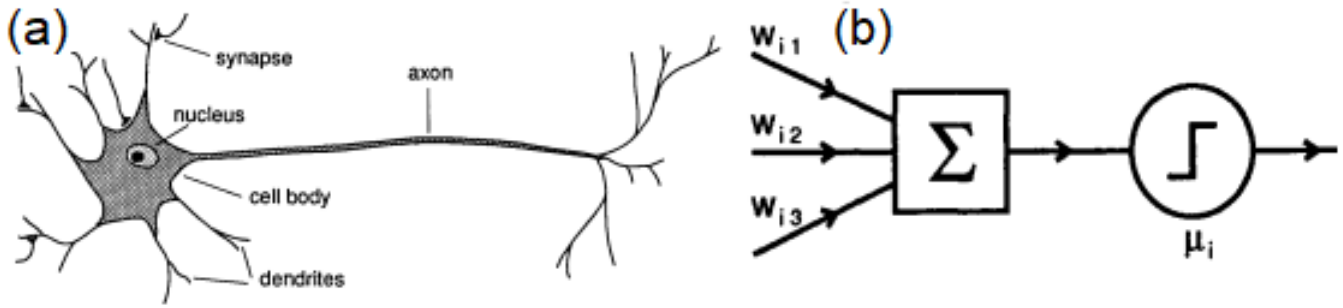

Figura 1.4: *(a) Esquema de una neurona biológica típica. (b) Esquema de una neurona artificial de McCulloch-Pitts, donde se representa la suma de los pesos a la entrada w_{ij} que da como resultado la activación o no activación de la neurona al comparar con un valor umbral μ_i. Imagen tomada de [13].*

El cerebro está compuesto por alrededor de 10^{11} neuronas de distintos tipos, en la Fig. 1.4.a se observa un esquema de una neurona típica. Las fibras nerviosas, llamadas dendritas, se ramifican y se conectan al cuerpo de la célula, donde se encuentra el núcleo. Desde el cuerpo de la célula se extiende una fibra larga llamada axón, que se ramifica en hebras y subhebras. Al final de estas se encuentran las terminaciones transmisoras de las uniones sinápticas o sinapsis, con otras neuronas. Los receptores de estas uniones en otras células pueden encontrarse tanto en las dendritas como en el cuerpo de las células mismas. Cada neurona está conectada, a través de 10^3-10^4 sinapsis, con otras neuronas.

La transmisión de una señal entre dos células, a través de una sinapsis, es un proceso químico y eléctrico complejo en el que se liberan sustancias llamadas neurotransmisores del lado de la unión que envía la señal. El efecto es aumentar o bajar el potencial eléctrico dentro del cuerpo de la célula receptora. Si este potencial llega a un valor umbral, un pulso o potencial de acción (*spike*), de aproximadamente 100 mV de amplitud y una duración de 0.1 a 1 ms, se envía hacia el axón. En este momento se dice que la célula se "dispara". El *spike* se ramifica a través del árbol axonal hacia las uniones sinápticas de otras células. Después de disparar, la célula debe esperar por un cierto tiempo, llamado periodo refractario, antes de poder ser disparada nuevamente [13].

La actividad de los *spikes*, generados por las neuronas y transmitidos a través de las sinapsis, es la responsable del flujo de información y los procesos de cómputo complejos realizados por el cerebro; por esto, se considera a las neuronas y sinapsis como las unidades básicas de cómputo. Las neuronas integran las entradas que llegan desde otras neuronas, generando *spikes* como resultado. Las sinapsis contribuyen al cómputo cambiando la intensidad de sus conexiones como resultado de la actividad neuronal, lo que se conoce como plasticidad sináptica (SP por sus siglas en inglés). Decimos que una sinapsis se potencia cuando aumenta su peso sináptico y que se deprecia cuando disminuye su peso sináptico. Se cree que el mecanismo de SP es el responsable del aprendizaje y la memoria en el cerebro [14].

El mecanismo SP se puede clasificar, según la duración de los eventos de potenciación y depreciación, como *short-term plasticity* (STP) y *long-term plasticity* (LTP). El primero corresponde a cambios que ocurren durante un periodo corto de tiempo y los segundos ocurren durante un tiempo mayor, de forma que el cerebro puede almacenar información de manera persistente. Una forma de LTP es la conocida como *spike-timing-dependent plasticity* (STDP), donde la intensidad de la conexión entre las neuronas se modifica en base a la actividad neuronal presináptica y postsináptica (Fig. 1.5)(izquierda) [4], teniendo en cuenta el orden temporal de los *spikes*. La plasticidad depende del tiempo relativo entre los *spikes* pre y postsinápticos (Fig. 1.5)(centro).

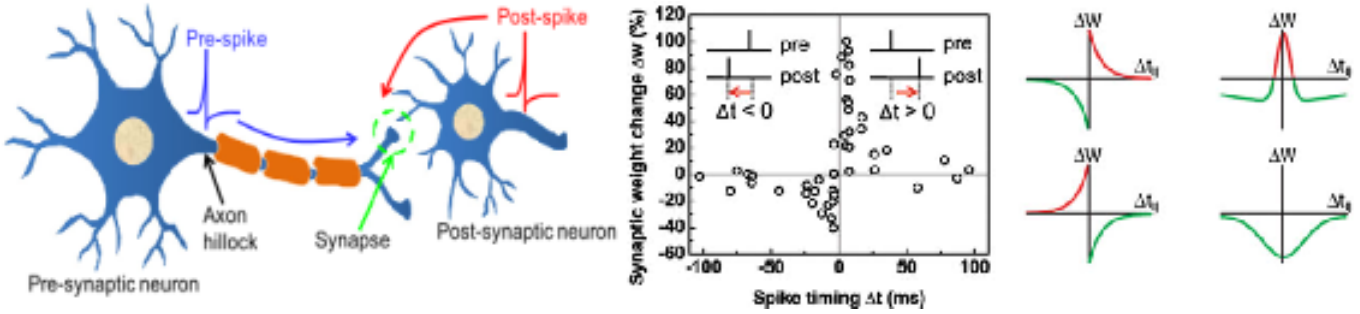

Figura 1.5: *Izquierda: Esquema de la regla de aprendizaje STDP. La intensidad de la conexión, entre dos neuronas, depende del tiempo relativo entre la actividad presináptica y postsináptica. Centro: Gráfico del cambio en los pesos sinápticos en función del timing relativo entre los spikes pre y post-sinápticos. Derecha: Distintas formas de STDP. Imagen tomada de [4].*

La sinapsis se potencia si un *spike* presináptico precede a un *spike* postsináptico; contrariamente, la sinapsis se deprecia si un *spike* postsináptico precede a un *spike* presináptico. El porcentaje de incremento del peso sináptico se determina de acuerdo a la diferencia de tiempo entre los *spikes* pre y postsinápticos.

El mecanismo STDP no es la única forma de SP, ni el único mecanismo asociado al cómputo a nivel celular del cerebro; la tasa de disparo, el orden de los *spikes* y la ubicación de las dendritas son algunos de los factores que afectan a la plasticidad.

En 1943, McCulloch y Pitts propusieron un modelo simple de neurona como una unidad de umbral binaria (Fig. 1.4.b). Este modelo de neurona computa una suma pesada de sus entradas, que se conectan a otras unidades, y da como resultado un estado de activación o no activación de la neurona si la suma es mayor o menor que un cierto valor umbral, respectivamente. El estado de la i-ésima neurona n_i que puede valer 1 o 0 se calcula como

$$n_i(t+1) = \Theta\left(\sum_j w_{ij} n_j(t) - \mu_i\right), \tag{1.1}$$

donde w_{ij} representa el peso que tiene la conexión (peso sináptico) entre la neurona j y la neurona i en la red. Este peso puede ser mayor que cero (sinapsis excitatoria) o menor que cero (sinapsis inhibitoria). Si el peso es cero no hay sinapsis entre las neuronas i y j. El parámetro μ_i es el valor umbral para la neurona i. El tiempo t se considera discreto, tomando como unidad de tiempo un paso de procesamiento o cálculo. La función $\Theta(x)$ es la función escalón o de Heaviside.

McCulloch y Pitts mostraron que un arreglo de estas neuronas simples es capaz de llevar a cabo cálculos similares a los de las computadoras digitales (eligiendo de forma adecuada los pesos w_{ij}), solo que no tan rápido como estas.

Las neuronas reales presentan algunas características que son omitidas en este modelo de neurona simple, como señales de salida con un continuo de valores posibles y que, además, combinen de manera no lineal distintas señales de entrada. Otra característica que se ignora es que no todas las neuronas se actualizan al mismo tiempo t, ni tienen el mismo retardo ($t \to t+1$) para actualizaciones consecutivas [13].

Una generalización simple del modelo de neurona de McCulloch y Pitts que incluye algunas de estas características de la neuronas reales es la siguiente

$$n_i = g\left(\sum_j w_{ij} n_j - \mu_i\right), \tag{1.2}$$

donde ahora n_i toma valores continuos y se llama estado o activación de la neurona i. La función escalón ahora se reemplaza con una función no lineal $g(x)$, que se llama función de activación. El tiempo t no se tiene en cuenta ya que es muy común hacer las actualizaciones de forma asincrónica y a tiempos al azar.

El modelo representado por la Ec. (1.2) se puede pensar en términos de la computación estándar (Von Neumann), de forma que cada neurona artificial juega el papel de un procesador que ejecuta un programa simple al computar la suma de los pesos de su entrada, provenientes de otros procesadores, y entrega a su salida un número que se calcula a partir de la función no lineal g, que depende de la suma de los pesos. Este número que se obtiene a la salida se envía a los demás procesadores que ejecutan el mismo tipo de programa aunque pueden usar distintos valores para los pesos e incluso otras funciones de activación. Los pesos y las funciones de activación se pueden pensar como datos almacenados localmente por los procesadores.

Debido a la gran cantidad de conexiones entre las neuronas en una red o, lo que es lo mismo, a la gran cantidad de términos en la suma que se calcula en la Ec. (1.2), los errores que se den en unos pocos términos no serán muy significativos. Esto sugiere que esta clase de sistemas pueden ser robustos y su desempeño no se degradará en gran medida frente a errores o ruido.

1.4. Perceptrón simple y proceso de aprendizaje

Nos interesa estudiar el aprendizaje supervisado en el contexto de una arquitectura de red en particular, el de las redes llamadas *feed forward*. Esto implica que cada neurona en una capa solo se conecta con las neuronas de la capa siguiente, a diferencia de, por ejemplo, las redes recurrentes donde las neuronas pueden tener *loops* de conexiones sobre sí mismas. Las redes *feed forward* fueron llamadas perceptrones en los trabajos iniciales donde fueron estudiadas [15-17]. Dependiendo del autor, la capa de entrada se cuenta o no como una capa de la red. En nuestro caso, no la contaremos como una capa ya que consideramos que las entradas no realizan ninguna clase de cómputo. En la Fig. 1.6 se muestran ejemplos de perceptrones. Llamaremos perceptrón simple a las redes *feed forward* de una sola capa.

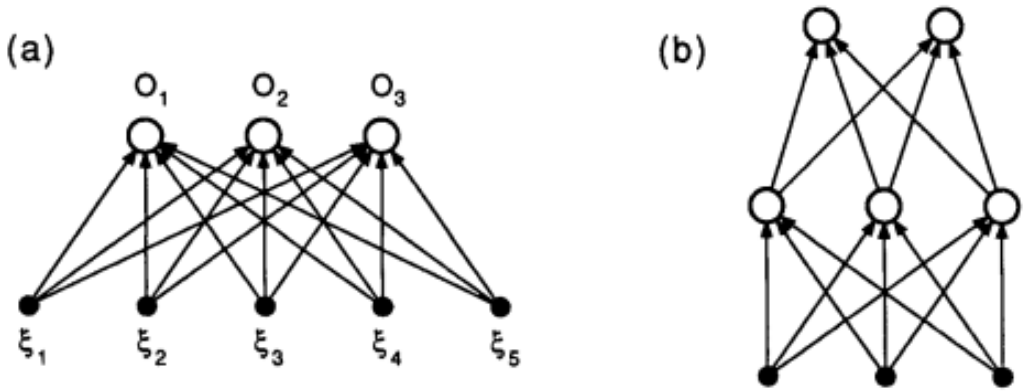

Figura 1.6: *(a) Perceptrón simple (una sola capa) con 5 neuronas de entrada ξ_i y una capa de salida con tres neuronas O_i. (b) Perceptrón de dos capas con 3 neuronas de entrada, una capa oculta de 3 neuronas y una capa de salida de 2 neuronas. Imagen tomada de [13].*

Un perceptrón simple consta de N entradas, donde notamos sus entradas como ξ_k y sus salidas como O_i. Estas últimas se computan como:

$$O_i = g(h_i) = g\left(\sum_{k=0}^{N} w_{ik}\xi_k\right), \tag{1.3}$$

donde $g(h)$ es la función de activación, y en su forma más general, suele ser no lineal. La salida depende explícitamente de las entradas, lo cual es válido para toda red *feed forward*. El valor umbral θ_i se omite ya que se puede tratar como conexiones a una neurona de entrada que tiene el valor fijo a $\xi_0 = -1$ y pesos $w_{i0} = \theta_i$ (a este valor fijo se le suele llamar *bias*). Entonces se tiene

$$O_i = g\left(\sum_{k=0}^{N} w_{ik}\xi_k\right) = g\left(\sum_{k=1}^{N} w_{ik}\xi_k - \theta_i\right). \tag{1.4}$$

Lo que buscamos es comparar la salida O_i^μ, que se obtiene a partir del patrón de entrada ξ_k^μ, con la salida objetivo ζ_i^μ, donde μ es el índice que corresponde a los patrones de entrada y salida de un elemento en particular del conjunto de entrenamiento. Idealmente, deberíamos tener, luego del proceso de aprendizaje, que $O_i^\mu = \zeta_i^\mu$ para cada i y μ.

Para el perceptrón simple cada patrón de salida O_i^μ esta dada por

$$O_i^\mu = g(h_i^\mu) = g\left(\sum_{k=0}^{N} w_{ik}\xi_k^\mu\right). \tag{1.5}$$

Las entradas, salidas y los patrones objetivos pueden tomar valores booleanos o continuos. Los valores posibles que pueden tomar las salidas dependen de la función de activación $g(h)$. Un caso común a tener en cuenta es cuando las salidas son continuas pero los patrones objetivos son booleanos. En esos casos podemos esperar que las salidas O_i^μ estén dentro un rango cercano a los valores objetivo.

Para los perceptrones simples, si existe un conjunto de pesos w_{ik} con los que la red puede llevar a cabo la tarea deseada luego del proceso de aprendizaje, entonces estos pueden ser hallados mediante alguna regla que parte de una serie de valores arbitrarios que son mejorados de forma sucesiva y que convergen a la respuesta correcta en una cantidad de pasos finita [13].

1.5. Método del descenso por el gradiente

Si bien es posible calcular analíticamente los valores de los pesos w_{ik} [13] que permiten obtener, para cada entrada de la red, la salida deseada, nos interesa encontrar una regla de aprendizaje que nos permita encontrar el conjunto de valores de los pesos w_{ik} de forma iterativa, partiendo de valores arbitrarios.

Para esto comenzamos definiendo una medida del error o función de costo como

$$E[w] = \frac{1}{2}\sum_{i\mu}(\zeta_i^\mu - O_i^\mu)^2 = \frac{1}{2}\sum_{i\mu}\left[\zeta_i^\mu - g\left(\sum_k w_{ik}\xi_k^\mu\right)\right]^2, \tag{1.6}$$

donde g es una función de activación diferenciable. Esta función de costo se minimiza cuando el conjunto de pesos w_{ik} permiten que los valores de las salidas se acercan a los valores deseados. Además, la función de costo es definida positiva, y converge a cero a medida que nos acercamos a la solución correcta. También hay que notar que depende sólo de los pesos w_{ik} y de los valores objetivo ζ_i^μ del problema.

A partir de esta función de costo, se puede hallar el conjunto de pesos w_{ik} al movernos en la dirección contraria a la dirección de crecimiento de la hipersuperficie definida en el espacio de los w, como se muestra en la Fig. 1.7.

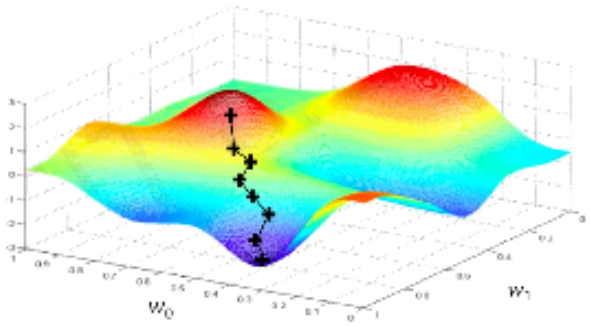

Figura 1.7: *Esquema del método del descenso por el gradiente donde se ven los pasos que se calculan y realizan sobre una superficie hasta llegar a un mínimo.*

Este método se conoce como descenso por el gradiente (*gradient descent*) y la idea básica es cambiar los valores w_{ik} en una cierta cantidad Δw_{ik} que es proporcional a menos el gradiente de la función de costo $E[w]$, de la siguiente forma:

$$\Delta w_{ik} = -\eta \frac{\partial E}{\partial w_{ik}} \tag{1.7}$$

$$= \eta \sum_{\mu} \Big[\zeta_i^{\mu} - g(h_i^{\mu})\Big] g'(h_i^{\mu}) \xi_k^{\mu}, \tag{1.8}$$

donde η se conoce como tasa de aprendizaje y es un valor que se elige para controlar la "velocidad" con la que el método busca los mínimos de la hipersuperficie y, por lo tanto, marca el paso o tasa de aprendizaje del algoritmo. Si calculamos estas correcciones para cada patrón de entrada ξ_k^{μ} (es decir, para un dado μ) se tiene

$$\Delta w_{ik}^{\mu} = \eta \delta_i^{\mu} \xi_k^{\mu}, \tag{1.9}$$

con la cantidad δ_i^{μ} dada por

$$\delta_i^{\mu} = \Big[\zeta_i^{\mu} - O_i^{\mu}\Big] g'(h_i^{\mu}). \tag{1.10}$$

Las Ecs. (1.9) y (1.10) se conocen como regla delta, regla de Adaline, regla de Widrow-Hoff o regla LMS (*least mean square*) [13].

Como se mencionó anteriormente, la función de costo depende de los pesos y los patrones objetivo, los que al mismo tiempo determinan la hipersuperficie y, por lo tanto, el mínimo al que el método del descenso por el gradiente debe converger para que el algoritmo realice la tarea de la forma más eficiente posible. Al utilizar la función de costo Ec. (1.6), en el caso más general la superficie es suave (diferenciable) pero presenta mínimos locales en los que el algoritmo puede estabilizarse. Elegir una tasa de aprendizaje baja permite hallar mínimos de forma más precisa, pero sin distinguir entre mínimos locales o absolutos. Usando una tasa de aprendizaje alta el método puede escapar de los mínimos locales, pero su precisión para llegar al valor del mínimo absoluto no es tan buena. Para evitar estos casos se implementan distintas estrategias para la elección de la tasa de aprendizaje, de forma que los mínimos locales son saltados rápidamente y los mínimos absolutos son hallados con buena precisión.

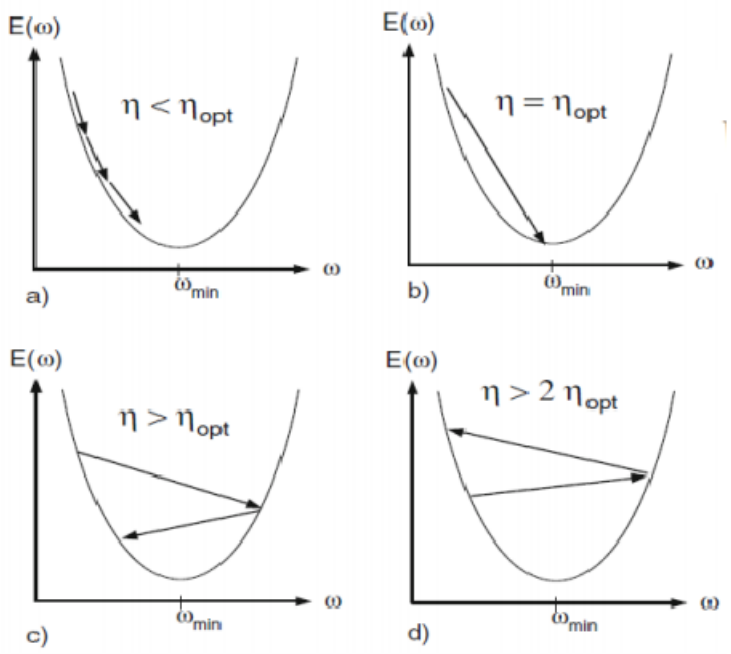

Figura 1.8: *Distintos comportamientos del método del descenso por el gradiente según la tasa de aprendizaje* η.

En la Fig. 1.8 se muestran varios ejemplos de cómo distintos valores de la tasa de aprendizaje η repercuten en la forma en que el método converge. Suponemos que existe un valor óptimo η_{opt}, que sería el paso ideal; al tomar valores menores a este, como en el caso a), el método converge al mínimo pero necesita de varios pasos de aprendizaje para alcanzarlo; para el caso b), $\eta = \eta_{opt}$ y el método converge en un paso. Para valores más grandes que el η_{opt}, casos c) y d), en cada paso de aprendizaje el método pasa de un lado al otro de la zona del mínimo y, para algunos valores, incluso puede llegar a alejarse del mínimo que estamos buscando.

Al aplicar todos los elementos de un conjunto de entrenamiento finito se dice que se completó un *epoch* de entrenamiento. Para el entrenamiento completo de la red típicamente se requieren de múltiples *epochs*. Si las actualizaciones de los pesos sinápticos se obtienen para cada *epoch* completo, se suele decir que se implementa un método *batch gradient descent*. Cuando el tamaño del conjunto de entrenamiento es muy grande la utilización de *batch gradient descent* implica un gran costo de cómputo y, además, el tiempo de cálculo de cada actualización de los pesos aumenta con el tamaño del conjunto. Una alternativa es el uso de *mini-batches*, que consiste en dividir el conjunto de entrenamiento en varios subconjuntos, de forma que al procesar cada subconjunto se calcula la actualización de los pesos sinápticos. Esto implica que por cada *epoch* completo se obtienen varias actualizaciones de los pesos y la cantidad de actualizaciones por *epoch* está determinada por el tamaño de los *mini-batches* que se toman.

Estos métodos presentan distintas ventajas y desventajas, que están relacionadas con el tamaño de los conjuntos de entrenamiento y con el costo de cómputo necesario para su implementación. La desventaja del uso de *mini-batches* es su alto costo computacional, además de que la alta frecuencia de las actualizaciones puede generar ruido en el cálculo del error [18, 19].

Capítulo 2

Sistemas físicos para cómputo neuromórfico

2.1. Memristores y conmutación resistiva

Las posibles implementaciones de *hardware* para cómputo neuromórfico requieren del desarrollo de sistemas físicos que emulen el comportamiento eléctrico de neuronas y sinapsis biológicas. Para este último caso, se considera a los dispositivos denominados "memristores" como uno de los candidatos prometedores sobre los que basar dicha implementación [4].

El concepto de memristor (nombre que proviene de la contracción de "memoria" y "resistor") fue descripto por primera vez en 1971 por Leon Chua [20, 21]. A partir de las ecuaciones generales de la teoría de circuitos, propuso que la resistencia eléctrica de este elemento es función de la cantidad de carga que ha circulado por él, es decir, depende de la historia de la corriente que ha fluido a través del mismo. Esto implica un comportamiento no lineal e histerético en las curvas de corriente-tensión (propiedades de memoria). El memristor es capaz de variar su resistencia eléctrica en base a la magnitud y polaridad del voltaje de una señal aplicada, además de poder retener su estado de resistencia aún al interrumpir la alimentación eléctrica externa. En la Fig. 2.1 se ven los tres componentes pasivos usuales: resistencia, capacitor e inductancia. Además se observan las cuatro magnitudes fundamentales de cualquier circuito electrónico: carga (q), corriente (i), tensión (v) y flujo (ϕ_m).

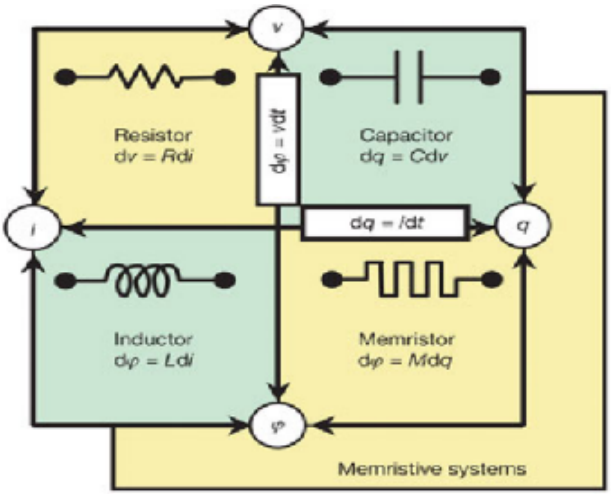

Figura 2.1: *Esquema que representa la simetría entre los componentes pasivos conocidos y las magnitudes físicas asociadas a un circuito eléctrico. El memristor surge de considerar esta simetría.*

Cada uno de estos elementos relaciona dos de estas magnitudes a través de sus parámetros característicos: la resistencia relaciona la tensión con la corriente ($R = dV/di$), la capacitancia relaciona la tensión con la carga ($C = dq/dV$) y la inductancia relaciona la corriente con el flujo

magnético ($L = d\phi_m/di$). Es fácil notar la falta de un cuarto elemento pasivo que relacione la carga con el flujo magnético. Ese elemento es el memristor ($M(t) = d\phi_m/dq$), donde $M(t)$ se denomina "memristancia" y es esencialmente un resistor con memoria.

Los dispositivos memristivos son estructuras tipo metal/óxido/metal (MIM) como la que se observa en la Fig. 2.2 [22], no siendo necesariamente los mismos metales los que constituyen los electrodos a ambos lados del óxido aislante. El electrodo donde se aplican los estímulos eléctricos se denomina electrodo superior o *top electrode* (TE) y el electrodo que se conecta a tierra se denomina electrodo inferior o *bottom electronde* (BE).

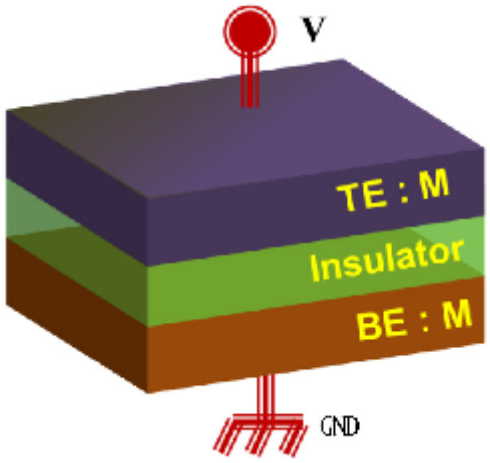

Figura 2.2: *Esquema de la geometría metal/óxido/metal (MIM) usual de los dispositivos memristivos. Imagen tomada de [22]*

La propiedad más destacable de estos dispositivos es la posibilidad de cambiar sus valores de resistencia, de forma no volátil (el estado de resistencia alcanzado se mantiene cuando el suministro eléctrico es interrumpido), aplicando estrés eléctrico. A este efecto se lo conoce como conmutación resistiva (*resistive switching* o RS) [23] y se ha reportado en una gran variedad de óxidos simples (TaO_x [24],TiO_2 [25], HfO_x [26], WO_x [27]) y complejos (manganitas y cupratos [28]) de metales de transición. Las ventajas que presentan los dispositivos memristivos son su gran escalabilidad (se han reportado dispositivos por debajo de la escala de los 10 nm), alta velocidad en los cambios de resistencia (RS por de bajo de los ns) y bajo consumo de energía (en el orden de los pJ) [29]. Otras características que se buscan en la fabricación de estos dispositivos son altos tiempos de retención, que es la capacidad de mantener en el tiempo un estado de resistencia dado (se buscan tiempos del orden de los 10 años) y la durabilidad, que es la capacidad de cambiar entre estados resistivos de manera estable ante la aplicación sucesiva de estrés eléctrico ($> 10^9$ ciclos).

Los mecanismos que rigen el efecto RS dependen fuertemente de los óxidos constitutivos, de los metales utilizados en los electrodos y de las propiedades interfaciales. Entre los mecanismos más aceptados se encuentra el efecto de RS filamentario [30]. En éste, la transición de un estado de alta resistencia a otro de baja resistencia (transición de R_{high} a R_{low}), que se denomina SET, se atribuye a la formación de nanofilamentos (usualmente formado por vacancias de oxígeno) conductores que cortocircuitan, total o parcialmente, el material dieléctrico. Por otro lado, la transición de estado de baja resistencia a otro de alta resistencia (transición de R_{low} a R_{high}) se denomina RESET y se asocia a la ruptura de los filamentos. La migración de los iones de oxígeno está asistida mayormente por un campo eléctrico y el *gap* en la distancia entre el filamento conductivo y el electrodo determina la resistencia del dispositivo [14]. En la Fig. 2.3.a se esquematiza el proceso de creación y ruptura de los filamentos que determinan los estados resistivos R_{low} y R_{high}, respectivamente [5].

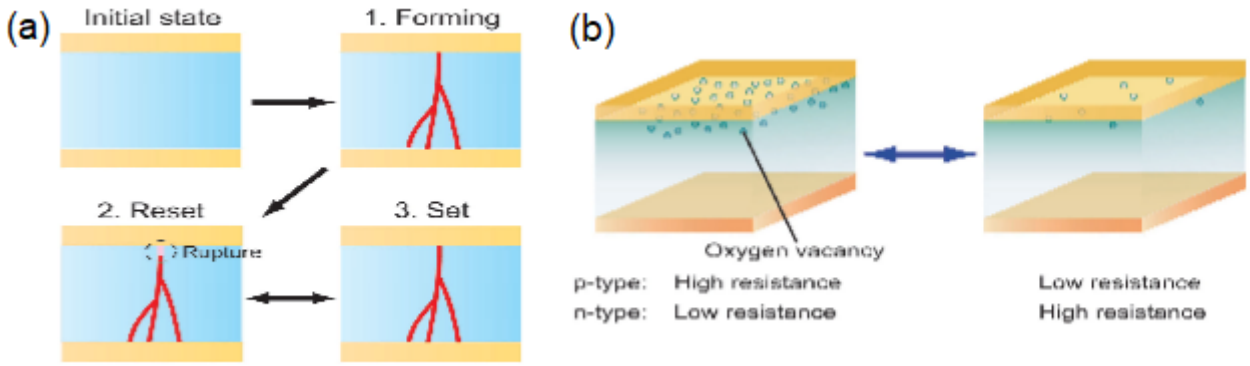

Figura 2.3: *(a) Proceso de SET y RESET para el mecanismo de creación y ruptura de filamentos conductores. (b) Proceso de SET y RESET para el mecanismo de conducción interfacial. Imagen tomada de [5].*

Otro mecanismo propuesto es el RS de tipo interfacial, relacionado con la modulación de la barrera de potencial (tipo Schottky) que se forma en la interfaz aislante-metal debido a la migración de las vacancias de oxígeno [5]. Esta se genera por la presencia de un campo eléctrico inducido por el estímulo aplicado, lo que al mismo tiempo cambia la resistencia del dispositivo en la zona de la interfaz donde se produce el cambio en la barrera (Fig. 2.3.b).

Los electrodos metálicos tienen un papel fundamental en el RS interfacial. Para que la interfaz tenga un rol activo se requiere que la altura de la barrera de potencial sea alta, mientras que barreras bajas dan lugar a interfaces cuasi-óhmicas que no presentan RS. La altura de la barrera puede ser controlada mediante la elección del metal del electrodo: cuando el óxido es de tipo-p, para lograr que la altura de la barrera sea mayor se requieren metales con función trabajo pequeña. Lo opuesto ocurre para óxidos tipo-n [31].

En la Fig. 2.4 se muestra como cambian las curvas I-V al variar los metales utilizados para el electrodo. Los cuatro dispositivos fueron fabricados utilizando como óxido aislante la manganita PCMO [4]. En todos los casos se observan curvas I-V histeréticas, que ponen de manifiesto el efecto de RS y muestran el carácter memrisitivo de estos dispositivos. Por ejemplo, en la curva (a) se observa que el sistema Ni/PCMO muestra altos valores de corriente pero un cociente R_{high}/R_{low} pequeño, mientras que en (d) se observa que el sistema Al/PCMO tiene un cociente R_{high}/R_{low} más grande pero valores de corriente más bajos.

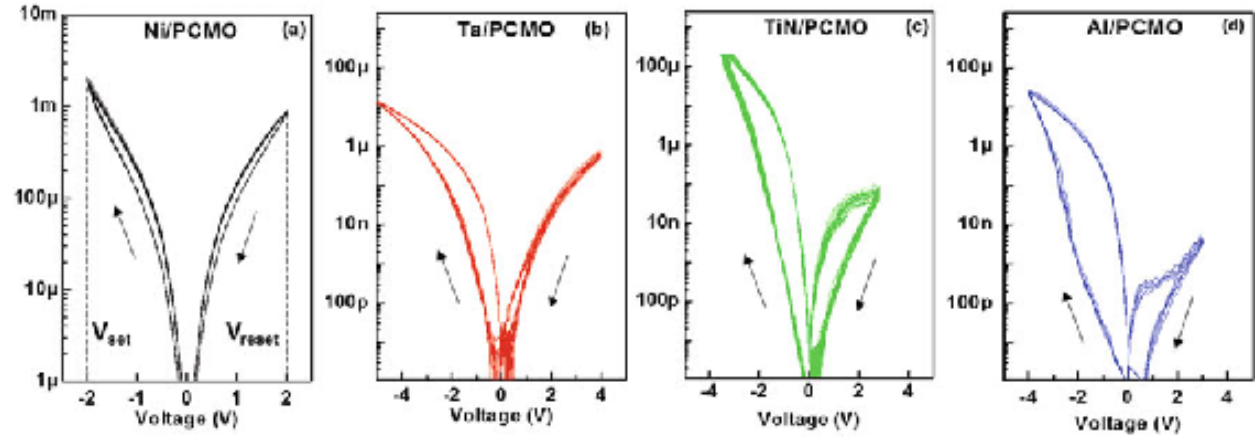

Figura 2.4: *Curvas I-V correspondientes a cuatro dispositivos de manganita PCMO con electrodos de metales distintos. En todos los casos se observan curvas que evolucionan con sentido horario. Imagen tomada de [4].*

2.2. Fabricación

En este trabajo utilizamos datos de mediciones memristivas realizadas en manganitas con estructura de perovskita crecidas en forma de films delgados. Se trabajó con tres sistemas memristivos: $La_{1/3}Ca_{2/3}MnO_3$ con electrodo superior de Ti (LCMO/Ti) [32, 33], la misma manganita con electrodo superior de Ag (LCMO/Ag) [34] y la maganita-cobaltita $La_{1/2}Sr_{1/2}Mn_{1/2}Co_{1/2}O_3$ crecida sobre el substrato conductor $Nb : SrTiO_3$ (NSTO/LSMCO) [35]. Estos materiales han mostrado excelentes propiedades memristivas cuando se fabrican estructuras MIM con ellas. Una ventaja de trabajar con *films* delgados es que los campos eléctricos que se producen en estos espesores a voltajes bajos (del orden del volt) son lo suficientemente intensos (cercano a 100 kV/cm) como para activar los mecanismos de RS con un bajo costo energético. Adicionalmente, los sistemas basados en films delgados permiten reducir las dimensiones laterales hasta la escala nanométrica mediante el uso de técnicas de micro/nanofabricación.

Una de las técnicas más utilizadas (en el contexto de investigaciones científicas) para el crecimiento de *films* delgados es el depósito por láser pulsado (PLD por sus siglas en ingles). Esta técnica se basa en el impacto de un pulso láser de alta potencia en blancos sólidos densos, produciendo la ablación de estos y la formación de un plasma de material ionizado. El plasma incide sobre un substrato enfrentado al blanco, condensa y da lugar al crecimiento del film. Las propiedades de las películas depositadas dependen fuertemente de las características del sustrato, del material a depositar y de los parámetros de crecimiento. Esta técnica ha demostrado ser capaz de producir películas delgadas de diversos materiales con propiedades físicas y estructurales superiores a las obtenidas por otras técnicas como la de evaporación térmica, evaporación por haz de electrones (*e-beam*), *sputtering* o métodos químicos [31, 36].

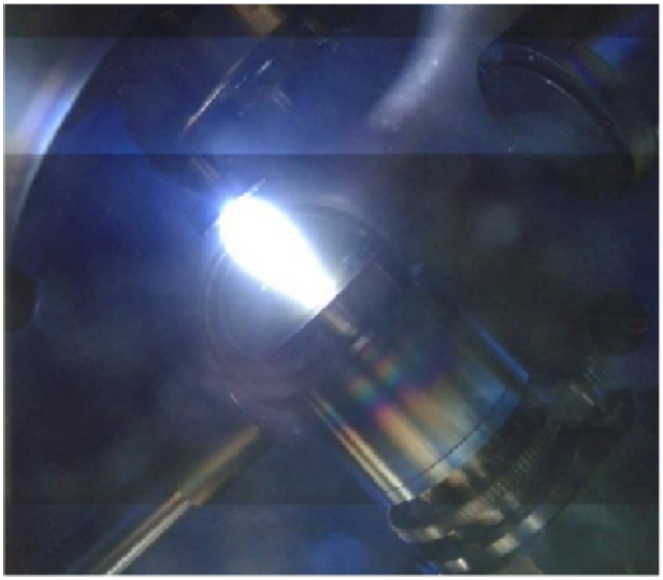

Figura 2.5: *Foto de la pluma obtenida durante el depósito de material. Imagen tomada de [31].*

La forma estándar en la que se lleva a cabo el depósito por PLD consiste en focalizar un haz láser ultravioleta (UV) sobre un blanco del material a crecer, produciendo un plasma denso compuesto por átomos, moléculas, iones y electrones de alta energía que se dirigen hacia el sustrato en forma de "pluma" (Fig. 2.5) [31]. La pluma generada está orientada en dirección normal a la superficie del sustrato. La temperatura del plasma es elevada y las partículas que lo conforman poseen alta energía cinética. Luego de cada pulso láser, las especies ablacionadas se depositan sobre el sustrato y de este modo crece la película. Todo el proceso se lleva a cabo dentro de una cámara conectada a un sistema de bombas de vacío; al mismo tiempo, es posible el ingreso de gases para poder realizar el depósito en atmósferas reactivas, lográndose así optimizar la congruencia del film fabricado. La cámara cuenta con ventanas que permiten el ingreso del láser, que es dirigido y enfocado mediante un sistema óptico que consta de espejos dieléctricos y lentes. Por último, en los sistemas utilizados

para la fabricación de films cristalinos (policristalinas o epitaxiales) como los considerados en este trabajo, se cuenta con un equipo calefactor (*heater*) que permite calentar el substrato y facilita la cristalización del material.

Las propiedades de las películas depositadas por PLD dependen de parámetros tales como la fluencia del haz láser, la presión de oxígeno en la cámara, la temperatura del sustrato y la distancia blanco-substrato, entre otros [31].

En la Fig. 2.6 se aprecian imágenes tomadas por microscopía electrónica de barrido (SEM) de un dispositivo memristivo basado en un *film* delgado de LCMO [34].

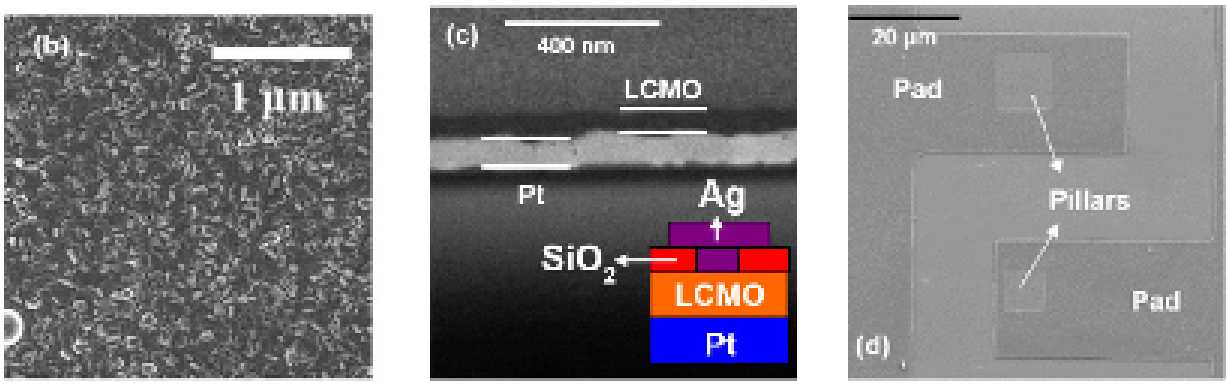

Figura 2.6: *(Izquierda) Imagen SEM de la superficie de un film de LCMO. (Centro) Sección transversal, medida también mediante SEM, del mismo film. En el inserto se observa un esquema del dispositivo fabricado. (Derecha) Vista superior tomada con SEM del dispositivo final microfabricado. Imagen tomada de [34].*

En la imagen SEM de la izquierda se observa una red densa de nanogranos en la superficie del *film*. En el centro se tiene una imagen SEM de la sección transversal del *film*, donde se puede apreciar el substrato de $Si/SiO_2/Ti/Pt$ que actúa como electrodo inferior y la capa de LCMO. En el inserto se observa un esquema de la geometría utilizada para la fabricación del electrodo superior, consistente en pilares de plata embebidos en una matriz de aislante de SiO_2. Esto se realizó mediante técnicas de microfabricación (litografía electrónica, *sputtering*, *e-beam*, etc). En la imagen de la derecha se ve una vista aérea del dispositivo luego del proceso de microfabricación; los pilares de plata terminan en *pads* macroscópicos que se utilizan para contactar el dispositivo con los instrumentos de medición eléctrica.

2.3. Aplicaciones en cómputo neuromórfico

Entre las posibles aplicaciones de los dispositivos memristivos se destacan su uso como memorias no volátiles, compuertas lógicas y la aplicación en la que nos enfocaremos en este trabajo: su uso en posibles implementaciones en *hardware* para cómputo neuromórfico [31]. El interés en el uso de memristores para esta aplicación en particular surge fundamentalmente de su comportamiento memristivo multinivel, emulando eléctricamente el comportamiento analógico de las sinapsis biológicas, lo que permite emular los pesos adaptables de las redes neuronales.

El comportamiento memristivo multinivel está estrechamente asociado a distintas funcionalidades neuromórficas de los memristores, como STP, LDP y STDP [6, 7, 37].

En la Fig. 2.7 [38] se observan datos experimentales de un sistema memristivo basado en un material ferroeléctrico [38] que presenta distintas características de plasticidad. En (a) se observa que un *spike* de 50 μs (gráfico superior) produce un aumento de la conductancia, que se mantiene brevemente en el tiempo, lo que corresponde a STP. En (b) se aumenta la duración del *spike* a 500 μs (gráfico superior) resultando en un cambio en la conductancia que se mantiene durante mucho más tiempo, lo que corresponde a LTP. En (c), con una serie de *spikes* de 2 μs y tensión de $3.2\ V$ (en el gráfico superior y en su inserto se muestran la forma de los *spikes* y como fueron aplicados de forma

consecutiva), la conductancia del dispositivo muestra un incremento (multinivel), que se denomina *long-term potentiation* o simplemente potenciación, mientras que con *spikes* negativos ($-3.5\ V$) la conductancia del dispositivo decrece gradualmente, a esto se lo llama *long-term depreciation* o depreciación.

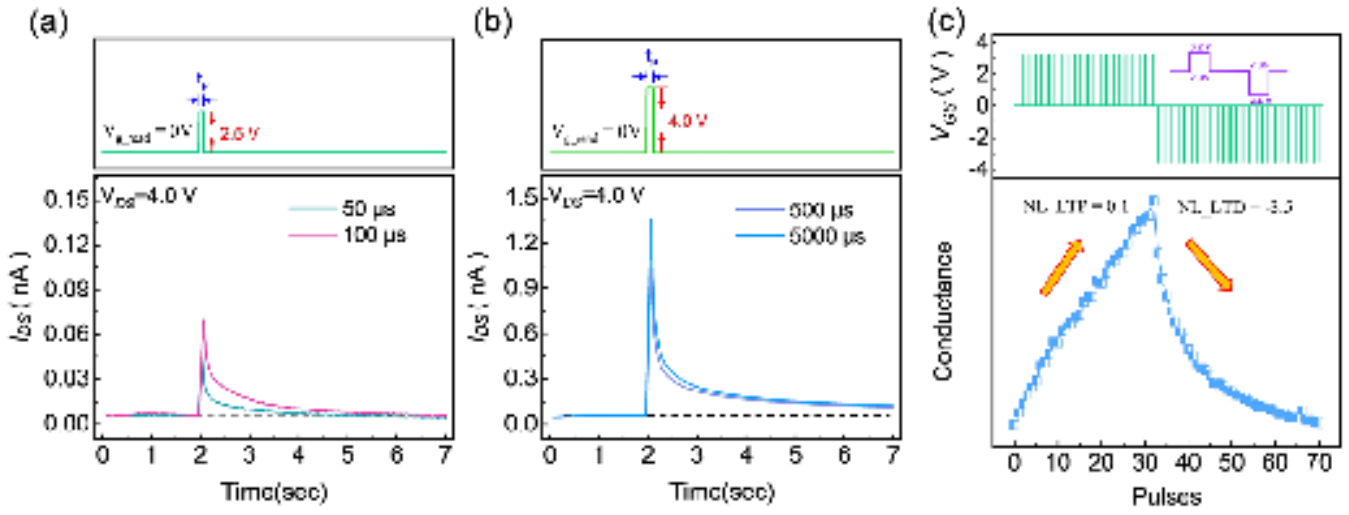

Figura 2.7: *(a) STP con diferentes tiempos para el spike. (b) LTP con diferentes tiempos para el spike. (c) Curva del proceso de potenciación y depreciación a partir de LTP, con trenes de spikes de distinta polaridad. Imagen tomada de [38].*

En los dispositivos memristivos que presentan características de plasticidad [7, 34, 37, 39] se pueden encontrar una gran variedad de comportamientos en sus curvas de potenciación y depreciación, como linealidad y no linealidad, distintos rangos o bandas de conductancia asociados a estos procesos, distinta cantidad de niveles dentro de las bandas anteriormente mencionadas y, por último, distinta cantidad de pulsos relacionados con los procesos de potenciación y depreciación. Estas características físicas presentan distintas ventajas y desventajas al momento de implementar y/o simular *hardware* para cómputo neuromórfico, siendo necesario la adopción de estrategias para mitigar los problemas que puedan surgir.

Una particularidad de los memristores es que es posible fabricar arreglos de estos dispositivos con arquitecturas denominadas de "barras cruzadas". Estas estructuras permiten la integración y escalabilidad de circuitos, lo cual es necesario para implementaciones a gran escala de hardware para cómputo neuromórfico [29]. Esta arquitectura es equivalente a la de las redes neuronales implementadas por software y, hasta cierto punto, a la de los sistemas biológicos reales.

En estos arreglos, se disponen filas y columnas de electrodos metálicos de forma perpendicular y superpuestos entre sí; en las intersecciones de los electrodos se deposita el material aislante, de forma que en cada intersección se obtiene una estructura MIM y, por lo tanto, un dispositivo memristivo (Fig. 2.8) [31]. Los dispositivos memristivos permiten la implementación de funciones neuronales y sinápticas básicas, como por ejemplo el mapeo de los pesos sinápticos de la red en sus valores de conductancia, mientras que la conectividad es proporcionada por los electrodos metálicos de la red. En años recientes se ha reportado la fabricación de sistemas de barras cruzadas (como el que se observa en la Fig. 2.9 [8]) basados en memristores para el reconocimiento de caracteres [8, 26, 29].

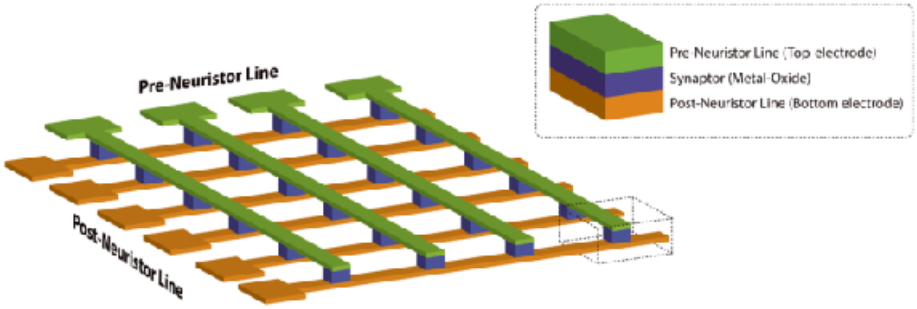

Figura 2.8: *Esquema de un arreglo de barras cruzadas utilizado en la implementación de hardware para cómputo neuromórfico. En el inserto se muestra un detalles de las estructuras MIM que se forman en las intersecciones de los electrodos inferiores y superiores. Imagen tomada de [31].*

El cómputo se lleva a cabo de forma paralela; para esto se aplica una tensión de lectura (tensión con intensidad baja que permite la medición de la resistencia del dispositivo sin perturbarlo) en las filas y luego se multiplica la tensión de lectura por la conductancia de los dispositivos en cada intersección. De acuerdo a las leyes de Kirchoff, se obtiene como resultado una corriente en cada columna. En este caso, las filas actúan como neuronas de entrada (presinápticas) mientras que las columnas actúan como neuronas de salida (postsinápticas). Típicamente, se utiliza electrónica periférica a la salida de cada columna, con el fin de convertir la corriente analógica que se obtiene a la salida en un *spike* de salida [4, 29].

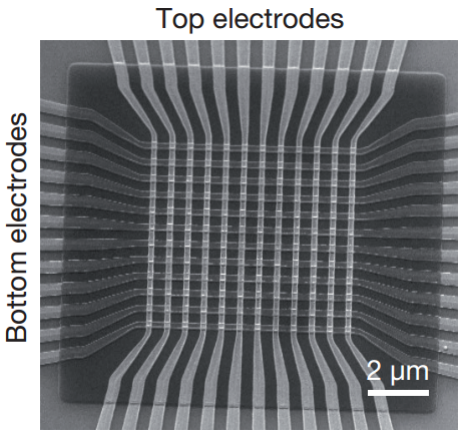

Figura 2.9: *Imagen de un sistema de barras cruzadas de 12 x 12 memristores. Imagen tomada de [8]*

Este tipo de implementaciones presenta una serie de problemas, algunos de los cuales son intrínsecos a los dispositivos memrisitivos y otros a la propia arquitectura de barras cruzadas. Para mencionar algunos, podemos comenzar por el hecho de que las conductancias de los memristores del arreglo, que juegan el papel de pesos sinápticos, solo toman valores positivos, lo que impone una limitación al momento de implementar algoritmos de aprendizaje en este tipo de redes, ya que en general estos algoritmos consideran la posibilidad de obtener pesos sinápticos tanto positivos como negativos.

Por otro lado, los memristores suelen tener asociadas curvas de potenciación y depreciación sináptica (conductancias) discretizadas (Fig. 2.7.(c)), lo que implica que las actualizaciones de los pesos sinápticos solo pueden realizarse de forma aproximada. La precisión de las actualizaciones de los pesos y, por lo tanto, la precisión del cómputo de la red, se ven afectadas por la cantidad de estados intermedios que se tienen entre los estados de máxima y mínima conductancia en cada uno de los dispositivos que forman el arreglo de barras cruzadas. La no linealidad en las curvas de potenciación y depreciación también dificulta el proceso de actualización de los pesos al requerir de electrónica y programación periférica para determinar el número y amplitud del estímulo a aplicar en cada caso.

Otro efecto que se debe mitigar es la existencia de *sneak paths* [40], que se dan cuando se establecen flujos de corrientes parásitas por caminos del circuito que no son los deseados. Por ejemplo,

si queremos aplicar tensión a un memristor particular del arreglo, la corriente puede circular por más de un memristor a la vez (involucrando dispositivos vecinos al que se desea acceder), sumando una resistencia en paralelo a la resistencia del memristor al que se quiere acceder. El efecto de *sneak paths* resulta en un incremento en los voltajes que se deben aplicar en los electrodos del arreglo de barras cruzadas para acceder o modificar el estado de cada elemento del arreglo, con el consiguiente aumento de energía eléctrica consumida, lo que limita las posibilidades de miniaturización de la red. Estas corrientes parásitas también pueden perturbar los estados resistivos de los memristores vecinos. Para reducir los efectos que se generan a partir de los *sneak paths* se suele agregar un selector en cada juntura memristiva. Estos selectores puede ser un transistor en serie o una interfaz con respuesta tensión-corriente no lineal [4].

Capítulo 3

Implementación del algoritmo de aprendizaje

3.1. Modelado de la red neuronal

Como punto de partida tomamos los trabajos realizados por F. Alibart et al. [29] y M. Prezioso et al. [8], en los que se implementaron redes neuronales en base a sistemas memristivos con arquitectura de barras cruzadas, realizando el proceso de entrenamiento de dichas redes para tareas de reconocimiento de imágenes. En la Fig. 3.1 [8] se observa un esquema eléctrico del sistema físico simulado en la presente tesis, el cual fue propuesto en los trabajos mencionados. Este arreglo de barras cruzadas consta de 10 neuronas de entrada, 6 líneas de salida que se combinan para dar 3 neuronas de salida y 60 memristores, donde las conductancias de cada memristor están dadas por el valor $G_{ij}^{\pm}$ ($i = 1, 2, 3$ y $j = 1, 2, ..., 10$) y los valores V_j corresponden a los voltajes de la j-ésima neurona de entrada.

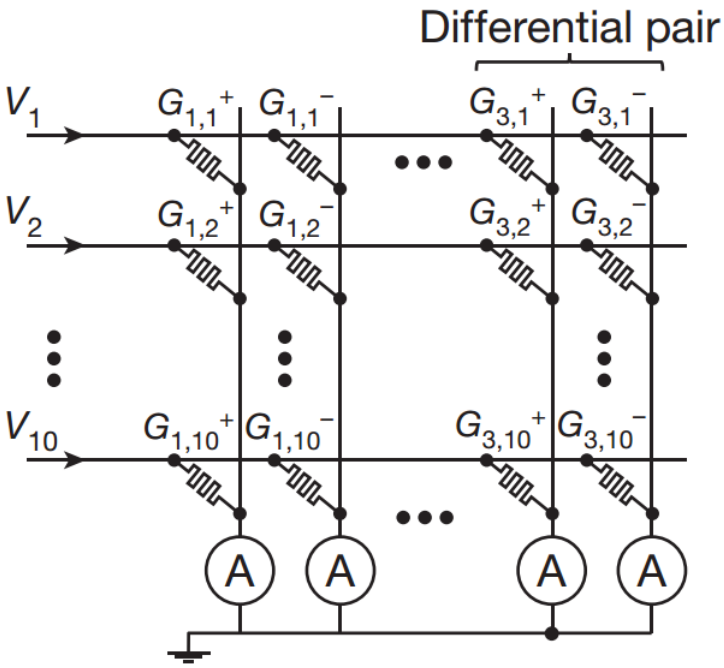

Figura 3.1: *Esquema del arreglo de barras cruzadas de 10x6 memristores que fue simulado numéricamente en este trabajo. Imagen tomada de [8].*

Las imágenes que esta red es capaz de clasificar (esquematizadas en la Fig. 3.2.(a)) están formadas por 3x3 píxeles; cada píxel puede tomar dos estados, claro u oscuro. Un píxel claro corresponde a un valor de $-100\ mV$ y un píxel oscuro a $+100\ mV$. La entrada V_{10} de la última neurona se usa como *bias* constante de $-100\ mV$.

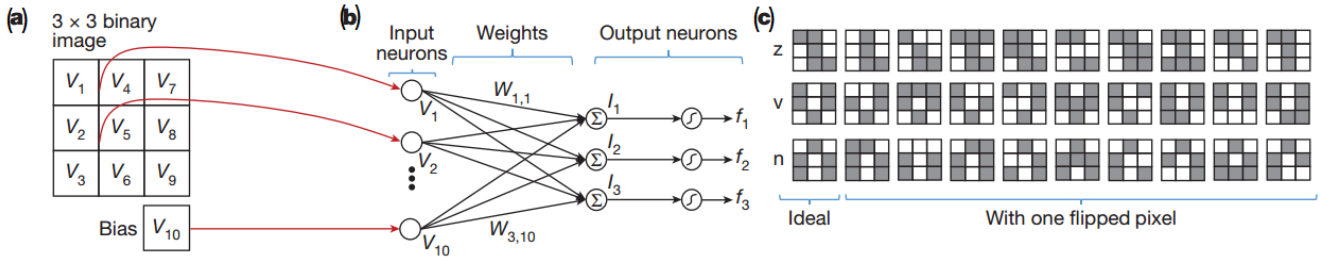

Figura 3.2: *(a) Esquema de una imagen de entrada. (b) Perceptrón simple con el que fue modelada la red neuronal formada por un arreglo de barras cruzadas de memristores. (c) Conjunto de imágenes utilizado para el entrenamiento de la red. Imagen tomada de [8].*

Según las leyes de Kirchoff, las corrientes en cada neurona de salida están dadas por

$$I_i = \sum_{j=1}^{10} w_{ij} V_j, \tag{3.1}$$

donde w_{ij} es el peso sináptico de la conexión entre la neurona de salida i con la neurona de entrada j. Los pesos sinápticos w_{ij} se definen a partir de las conductancias $G_{ij}^{\pm}$ de los memristores, al tomar un par de dispositivos (par diferencial) para obtener un peso sináptico "efectivo" de la siguiente manera

$$w_{ij} = G_{ij}^{+} - G_{ij}^{-}. \tag{3.2}$$

De esta forma es posible obtener pesos sinápticos w_{ij} positivos y negativos a partir de los valores de conductancia $G_{ij}^{\pm}$; de lo contrario, estaríamos limitando el proceso de aprendizaje de la red. Con esta estrategia pasamos de un arreglo de 10x6 valores de conductancia $G_{ij}^{\pm}$ a un arreglo de 10x3 pesos sinápticos w_{ij}.

Sobre cada corriente de salida I_i se aplica una función de activación f_i,

$$f_i = tanh(\beta I_i), \tag{3.3}$$

que define la salida de cada neurona de salida. Idealmente, los valores que las neuronas de salida pueden tomar deberían ser booleanos, ya que buscamos realizar tareas de clasificación con resultados binarios. Al tener salidas con valores continuos es necesario tener en cuenta algún criterio de clasificación *ad hoc* para determinar cuando los resultados obtenidos son correctos o incorrectos. La tangente hiperbólica como función de activación implica que una clasificación exitosa positiva tendrá asociado un valor de f_i positivo y cercano a 1, mientras que una clasificación exitosa negativa un valor de f_i más cercano a -1. El parámetro β (que en un sistema físico real posee unidades de A^{-1}) determina la curvatura de la tangente hiperbólica y, como mostraremos en el Cap. 4, es un parámetro crítico cuyo valor óptimo depende del rango de corrientes que circulan por la red. A lo largo de esta tesis se tomará, por simplicidad, a β como adimensional.

Para realizar las simulaciones modelamos el arreglo de barras cruzadas como un perceptrón simple con 10 neuronas de entrada, 3 neuronas de salida y 30 pesos sinápticos, como se observa en la Fig. 3.2.(b) [8]. Las entradas de la red son los píxeles de las imágenes vectorizados y codificados según los valores de voltaje que corresponden al brillo de los píxeles. A cada una de las primeras nueve neuronas de entrada le corresponde un píxel de la imagen; a la décima neurona de entrada le corresponde el *bias* constante.

El conjunto de entrenamiento esta formado por 30 imágenes, como las que se observan en la Fig. 3.2.(c) [8], separadas en 3 clases. La primera clase (clase 0) corresponde a la letra "z" más nueve variaciones de ella, la segunda clase (clase 1) corresponde a la letra "v" más nueve variaciones y, por último, la tercera clase (clase 2) corresponde a la letra "n" y sus nueve variaciones. Debido al tamaño limitado del conjunto de entrenamiento, también lo usamos como conjunto de testeo.

Las 3 neuronas de salida (enumeradas de 0 a 2) están asociadas a cada una de las clases del conjunto de imágenes, respectivamente. Para las salidas objetivo consideramos que la neurona que se debe activar tiene una salida de +0.85 (la salida de la función de activación es adimensional), mientras que las demás neuronas tienen una salida de -0.85. Por ejemplo, la salida objetivo para la clase 1 será el vector (+0.85;-0.85;-0.85).

A grandes rasgos, el proceso de clasificación de las imágenes consiste en ingresar una imagen vectorizada al perceptrón, calcular las funciones de activación de cada neurona de salida y determinar si la neurona que se activa es la correspondiente a la clase de la imagen de entrada o no.

El criterio de clasificación más simple que se puede implementar consiste en calcular las funciones de activación asociadas a una imagen en particular, comparar los valores que se obtuvieron para las tres neuronas de salida, y determinar si el valor más alto corresponde a la neurona asociada a la clase de la imagen de entrada. Por ejemplo, la neurona de salida 0 va a reconocer la letra "z" y va a originar una señal cercana al valor objetivo (+0.85) si se ingresa cualquier de las imágenes de la clase "z" y una señal cercana al valor definido como clasificación exitosa negativa (-0.85) si se ingresa una imagen de las otras clases.

Un criterio más exigente consiste en calcular las funciones de activación para todas las imágenes del conjunto de entrenamiento, comparando la salida de la neurona asociada a la clase de la imagen que estamos clasificando con las salidas de la misma neurona para todas las imágenes de las otras clases. Si la salida de la imagen que estamos clasificando es mayor (menor) que las de las imágenes de las otras clases, consideramos la clasificación como correcta (incorrecta) [8]. Por ejemplo, si queremos clasificar una imagen que pertenece a la clase 1, tenemos que comparar la función de activación de la neurona 1 de dicha imagen con las funciones de activación de la neurona 1 de todas las imágenes de las clases 0 y 2. En los capítulos que describen los resultados obtenidos en este trabajo, el criterio de clasificación corresponde al descripto en este párrafo.

3.2. Algoritmo propuesto para el entrenamiento de la red

Debido al mapeo que hacemos entre los pesos sinápticos w_{ij} del perceptrón simple con el que fue modelada la red neuronal y las conductancias $G_{ij}^{\pm}$ de los memristores que forman el arreglo de barras cruzadas (Ec. (3.2)), la implementación del descenso por el gradiente usando la regla delta, como fue descripta en la Sec (1.5), es poco práctica. Esto se debe principalmente a que las variaciones de las conductancias de los memristores que forman la red neuronal están regidas por procesos de potenciación y depreciación sináptica no lineales. Como se mencionó en la Sec 2.3, esto provoca que la secuencia de pulsos que hay que aplicar para ajustar la conductancia de un dado memristor dependa del valor inicial de conductancia, lo que físicamente implica la integración de *hardware* adicional a la red neuronal.

Para redes neuronales basadas en dispositivos memristivos como la que fue simulada, la forma más sencilla de implementar el proceso de aprendizaje requiere adoptar la regla de Manhattan [8, 29]. Esta regla consiste en actualizar los pesos sinápticos exclusivamente en base al signo de las correcciones Δw_{ik} que se calculan con el método del gradiente.

Siguiendo la propuesta realizada por Prezioso [8], la actualización de la conductancia $G_{ij}^{\pm}$ de cada memristor se realizó mediante la aplicación de pulsos predeterminados de SET o RESET -de polaridad inversa y voltaje fijo-, donde la elección entre uno o otro pulso lo determina el signo de la actualización calculada mediante el método estándar de descenso por el gradiente. El cambio de conductancia ante la aplicación de estos pulsos se obtiene de las curvas de potenciación y depreciación experimentales de tres sistemas: LCMO/Ti, LCMO/Ag y NSTO/LSMCO, obtenidas a partir de la aplicación repetida de pulsos de SET y RESET, respectivamente. Esta curvas se muestran, para los 3 sistemas estudiados, en la Fig. 3.3.

La curva de la Fig.3.3.a) corresponden al sistema LCMO/Ti; las curvas se construyeron aplicando pulsos de $+2V$ (RESET) para el proceso de depreciación y $-2V$ (SET) para el de potenciación. La Fig. 3.3.b) corresponde al sistema LCMO/Ag; en este caso, se aplicaron pulsos de $-1.5V$ (RESET) para el proceso de depreciación y $+1.5V$ (SET) para el proceso de potenciación. La Fig. 3.3.c)

corresponde al sistema NSTO/LSMCO, en el que se aplicaron pulsos de $+5.5V$ (RESET) para el proceso de depreciación y pulsos de $-2.5V$ (SET) para el proceso de potenciación.

La regla de aprendizaje implementada consiste en aplicar el método del descenso por el gradiente dado por la Ec. (1.9) y obtener, en primera instancia, las actualizaciones exactas. A partir de estas, se procede de la siguiente manera:

- Si $\Delta w_{ij} > 0$, aplicamos un pulso de SET a G_{ij}^{+} y un pulso de RESET a G_{ij}^{-}, obteniendo ΔG_{ij}^{+} y ΔG_{ij}^{-} respectivamente. En base a esto calculamos las $G_{ij}^{\pm}$ actualizadas y con la Ec. (3.2) los w_{ij} actualizados.
- Si $\Delta w_{ij} < 0$, aplicamos un pulso de RESET a G_{ij}^{+} y pulso de SET a G_{ij}^{-}, obteniendo ΔG_{ij}^{+} y ΔG_{ij}^{-} respectivamente. En base a esto calculamos las $G_{ij}^{\pm}$ actualizadas y con la Ec. (3.2) los w_{ij} actualizados.

De esta forma se obtienen las actualizaciones de los pesos sinápticos w_{ij} en base a las curvas de potenciación y depreciación experimentales.

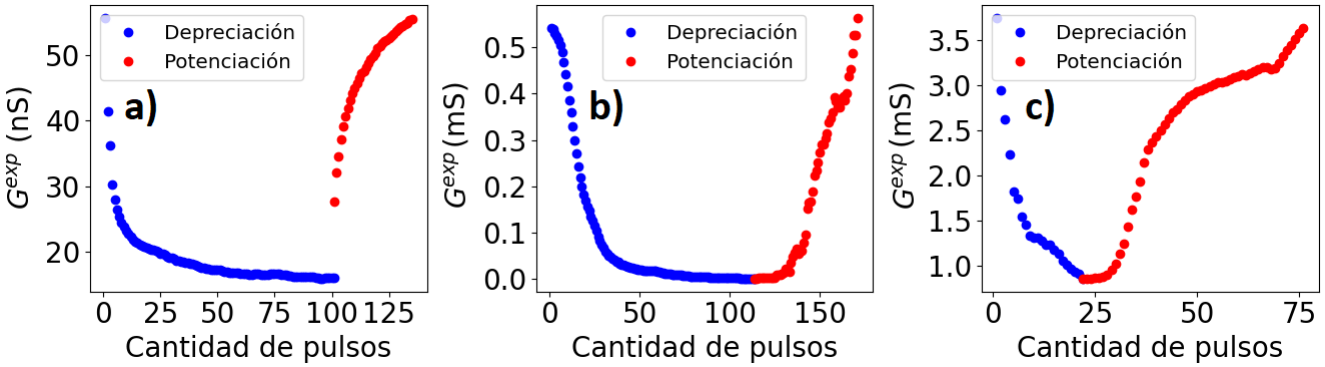

Figura 3.3: *Curvas de potenciación y depreciación sináptica obtenidas experimentalmente para: a) LCMO/Ti, b) LCMO/Ag y c) NSTO/LSMCO.*

La principal ventaja de la regla de Manhattan sobre la regla delta es su simplicidad, que lleva a implementaciones por *hardware* compactas y con tiempos de entrenamiento más rápidos. La desventaja de este método es que estamos sacrificando precisión en las clasificaciones en favor de la velocidad en el aprendizaje [41].

Teniendo en cuenta lo expuesto previamente, el algoritmo implementado para simular el aprendizaje del arreglo de barras cruzadas consiste en adaptar de manera iterativa los pesos sinápticos w_{ij}, en base a las conductancias $G_{ij}^{\pm}$ de los 60 memristores del arreglo, hasta converger a la situación deseada. Una descripción del algoritmo es la siguiente:

1. Se inicializa la matriz de conductancias $G_{ij}^{\pm}$ (10x6), esto determina la matriz de pesos sinápticos w_{ij} (10x3) por la Ec. (3.2).
2. Se vectorizan las imágenes del conjunto de entrenamiento.
3. Se selecciona una imagen de entrada y se calcula I_i y f_i según las Ecs. (3.1) y (3.3), para cada una de las neuronas de salida (1,2 y 3).
4. Se calcula para cada w_{ij} la actualización (parcial) dada por:

$$\Delta_{ij} = V_j[f_i^{(g)} - f_i]\frac{df}{dI}|(I = I_i),$$

donde $f_i^{(g)}$ es la salida deseada para la imagen correspondiente ($+0.85$ si la imagen pertenece a la clase/neurona correcta y -0.85 en caso de que no), f_i es la salida real calculada en el punto anterior y el último término es la derivada de la función de activación evaluada en I_i.

5. Se repite el proceso anterior para el resto de las imágenes.

6. Luego de pasar las 30 imágenes (*epoch* completo), se calcula la actualización exacta de cada peso sináptico w_{ij}

$$\Delta w_{ij} = \eta \sum_{T.I.} \Delta_{ij}$$

donde η es la tasa de aprendizaje, que en nuestro caso tomamos igual a 1 por simplicidad (normalmente, en algoritmos que actualizan con el valor exacto obtenido del descenso por el gradiente, es menor a 1) y la sumatoria es sobre todas las imágenes (T.I.).

7. Dependiendo del signo de Δw_{ij}, se actualizan las conductancias $G_{ij}^{\pm}$ según la regla de aprendizaje previamente descripta.

8. Se calcula la *accuracy* del clasificador (cantidad de imágenes reconocidas correctamente / total de imágenes) luego del *epoch*.

9. Se calcula el error del *epoch* según la Ec. (1.6).

10. Se repite el proceso anterior el número de *epochs* necesarios hasta lograr convergencia según algún criterio (cantidad de clasificaciones por arriba de un umbral y/o alcanzar un mínimo estable en el error).

3.3. Determinación de los incrementos de $G_{ij}^{\pm}$ en base a los datos experimentales

Para poder llevar a cabo el paso 7 del algoritmo descripto en la sección anterior, necesitamos procesar las curvas de potenciación y depreciación, ya que estas relacionan la conductancia G^{exp} vs. cantidad de pulsos eléctricos en una curva que tiene valores de conductancia discretizados.

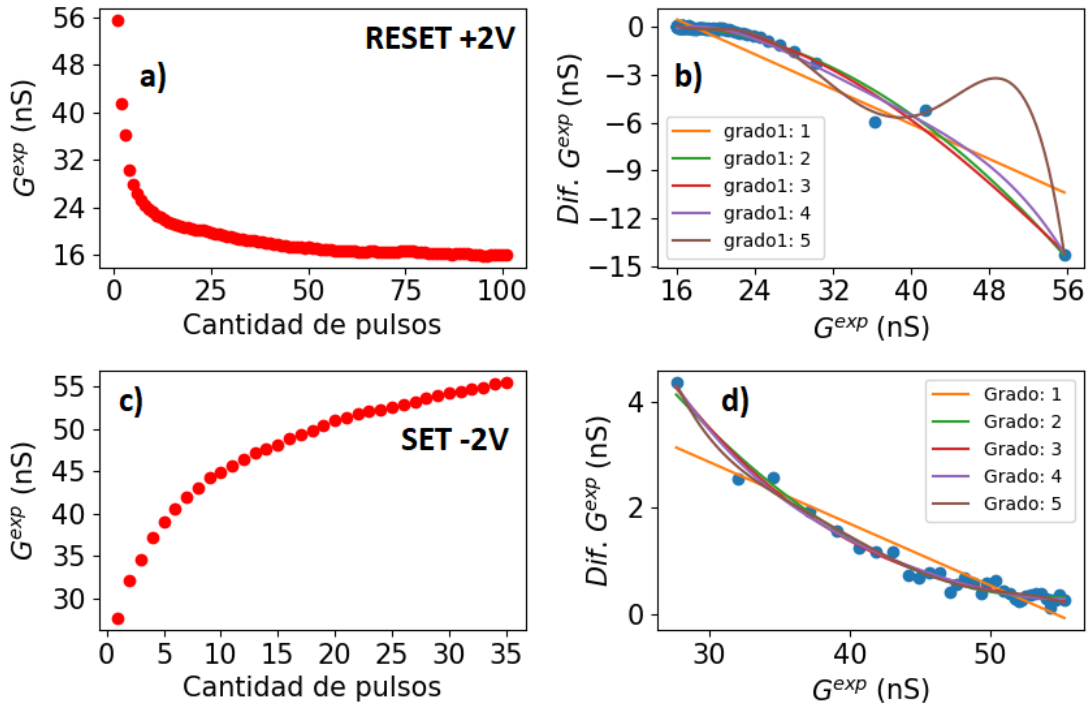

Figura 3.4: *a) Conductancia vs. cantidad de pulsos para el proceso de depreciación para el sistema LCMO/Ti. b) Curva de diferencia de conductancia experimental vs. conductancia experimental. En el mismo gráfico se observan ajustes polinómicos de distintos grados. c) Idem que a) para el proceso de potenciación. d) Idem que b) para el proceso de potenciación.*

A partir de estas curvas de potenciación y depreciación sinápticas medidas, el estímulo a aplicar durante el proceso de actualización de la conductancias $G_{ij}^{\pm}$ depende del estado previo a la actualización. Esto dificulta determinar el valor que le corresponde a la variación ΔG^{exp}, a partir de G^{exp}, ante un pulso de RESET o SET para un estado previo arbitrario.

Para realizar las actualizaciones se adoptó el siguiente método:

1. Se calcularon numéricamente los incrementos ΔG^{exp} ante la aplicación de pulsos de SET y RESET.

2. Se graficó ΔG^{exp} vs. G^{exp} tanto para el SET como para el RESET.

3. Se realizó un ajuste de la curva ΔG^{exp} vs. G^{exp} para obtener una función continua que nos permite calcular, a partir de un valor arbitrario de G^{exp}, la variación ΔG^{exp} que le corresponde. Se probaron ajustes con polinomios de distintos grados y se eligió, para cada caso, el grado que mejor ajusta a los puntos experimentales sin caer en una situación de sobre-ajuste. Esto se hace tanto para las curvas de potenciación como de depreciación.

En la Fig. 3.4 se observa la implementación de este método para las curvas asociadas al dispositivo LCMO/Ti.

Como estamos considerando bandas de conductancia acotadas para todos los sistemas modelados, se impusieron condiciones de contorno para las actualizaciones para garantizar que luego de la actualización la conductancia de cada memristor se mantenga dentro de la banda permitida.

La condición que se implementó consiste en calcular el paso 7 del algoritmo y determinar si los pesos actualizados, dados por $w_{ij} + \Delta w_{ij}$, con w_{ij} el valor del peso sináptico antes de actualizar, se encuentran dentro de la banda de conductancias experimentales. A partir de esto, se pueden dar tres situaciones:

- Si el valor del peso actualizado está en el rango permitido adoptamos el valor calculado.

- Si el peso actualizado es mayor que el máximo valor de conductancia G_{max} de la banda, no se realiza la actualización y se toma como peso actualizado $w_{ij} = G_{max}$.

- Si el peso actualizado es menor que el mínimo valor de conductancia G_{min} de la banda, no se realiza la actualización y se toma como peso actualizado $w_{ij} = G_{min}$.

Capítulo 4

Simulación del entrenamiento del perceptrón utilizando las curvas de potenciación y depreciación experimentales

En este capítulo se presentan los resultados obtenidos a partir de la implementación del algoritmo de entrenamiento descripto en la Sec (3.2), en el que se utilizaron los datos experimentales de las curvas de potenciación y depreciación correspondientes a los sistemas LCMO/Ti, LCMO/Ag y NSTO/LSMCO para calcular las actualizaciones de los pesos sinápticos.

Una realización del proceso de entrenamiento consiste en la elección de valores iniciales para la matriz de conductancia $G_{ij}^{\pm}$ (en el rango permitido correspondiente) y su adaptación sucesiva durante una cierta cantidad de *epochs*. Para cada realización se inicializó la matriz de conductancias $G_{ij}^{\pm}$ tomando valores aleatorios en el rango de conductancias experimentales, de forma que cada realización es independiente de las demás. Esta forma de inicialización es la que se utilizó en todas las simulaciones llevadas a cabo en esta tesis.

Haciendo una estimación a partir de la función de costo (Ec. (1.6)) vemos que la relación $(f_i^{(g)} - f_i)^2$ en esta ecuación toma un valor máximo ≈ 3.42 (tomando $f_i^{(g)} = +0{,}85$ y $f_i = -1$) y un valor mínimo de 0 (para $f_i = +0{,}85$). Si en estos dos casos tenemos en cuenta que sumamos sobre cada función de activación f_i con $i = 1, 2, 3$ y luego sumamos sobre las 30 imágenes y dividimos por 2, obtenemos como cota superior para el error ≈ 154 y como cota inferior 0. En la Fig. 4.1 se muestra, como ejemplo, la distribución de probabilidad del error inicial para el sistema LCMO/Ag al inicializar las matrices $G_{ij}^{\pm}$ al azar, en la banda de conductancias permitidas, 10^6 veces, lo cual se ajusta bien por una distribución normal con $\mu = 73{,}5$ y $\sigma = 19$. Se observa que los errores se mantienen dentro del rango estimado previamente. Para los sistemas LCMO/Ti y NSTO/LSMCO el error también presenta una distribución normal dentro del rango estimado.

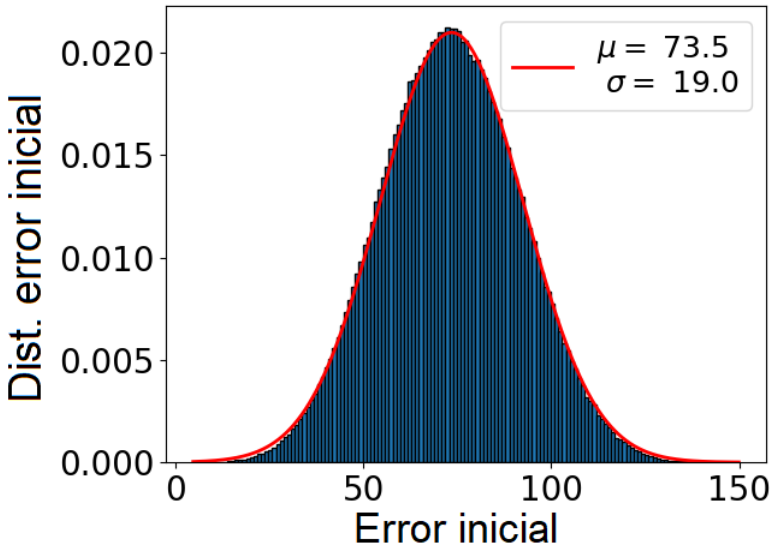

Figura 4.1: *Distribución de probabilidad normal ($\mu = 73.5$, $\sigma = 19$) del error inicial para el rango de conductancias del sistema LCMO/Ag.*

Como condición de corte para cada realización se puede fijar una cantidad de *epochs* máxima (sin ningún tipo de condición sobre el valor del error) o fijar un umbral de error por debajo del cual se considera que el algoritmo convergió. Para este último criterio, se corrieron alrededor de 20 realizaciones con un alto número de *epochs* (1000) y se determinó el promedio de los errores alcanzados luego del último epoch, verificándose que en todos los casos el error prácticamente no varía a partir del *epoch* 500. A este valor lo consideramos como el error asociado al mínimo "absoluto" de la función de costo. Dado que en los siguientes capítulos se buscará obtener resultados estadísticamente significativos, la implementación de 1000 *epochs* por realización representa un costo de cómputo alto. Debido a esto, se eligieron valores umbrales cercanos al mínimo absoluto que aseguren que el proceso de aprendizaje alcanzó el valor máximo del *accuracy* $= 1$ y que al mismo tiempo requiera un tiempo de cómputo del orden de 1 s por realización.

En lo que sigue se hablará de "velocidad de convergencia" en relación al número de *epochs* necesarios para lograr la convergencia del proceso de aprendizaje. Dicha velocidad no se relaciona con la velocidad de cómputo del algoritmo, la que obviamente depende de las prestaciones de la computadora con la que se realiza la simulación.

4.1. Resultados para el sistema LCMO/Ti. Elección del parámetro β óptimo

Las curvas de potenciación y depreciación correspondientes al sistema LCMO/Ti se mostraron previamente en la Fig. 3.3. El rango de conductancias para este sistema se encuentre entre 1.59×10^{-8} S y 5.55×10^{-8} S.

Lo primero que se estudió fue el efecto que tiene en el proceso de aprendizaje la elección de distintos valores del parámetro β presente en la función de activación (Ec. (3.3)). Para esto se realizaron una serie de simulaciones utilizando los valores $\beta = 10^5, 10^6, 10^7, 10^8$ y 10^9, y fijándose para cada realización 200 *epochs* de entrenamiento. En la Fig. 4.2 se grafican los resultados obtenidos. En la fila superior se observan las curvas de evolución del error en función de los *epochs* de entrenamiento para cada valor de β y en la fila inferior se grafica la *accuracy* en función de los *epochs* correspondiente a cada gráfico de la fila superior. Para cada β se simularon 20 realizaciones.

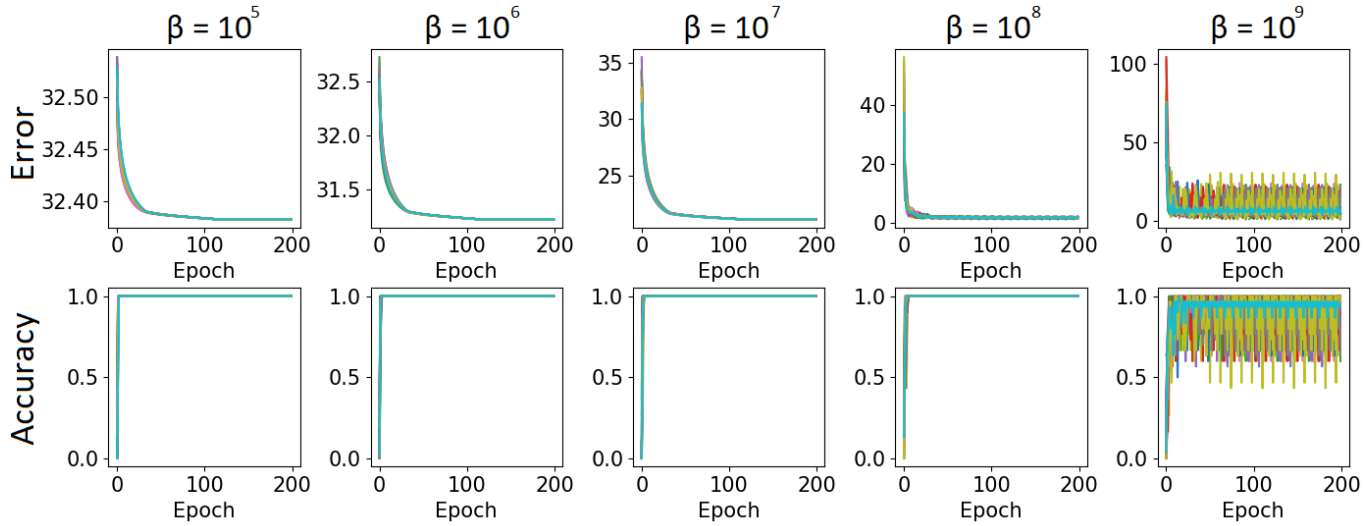

Figura 4.2: *Comportamiento de las curvas de error y accuracy para distintos valores del parámetro β. El valor al que converge el error disminuye al aumentar β. Para $\beta = 10^8$ se observa la aparición de ruido. Para cada β se calcularon 20 realizaciones.*

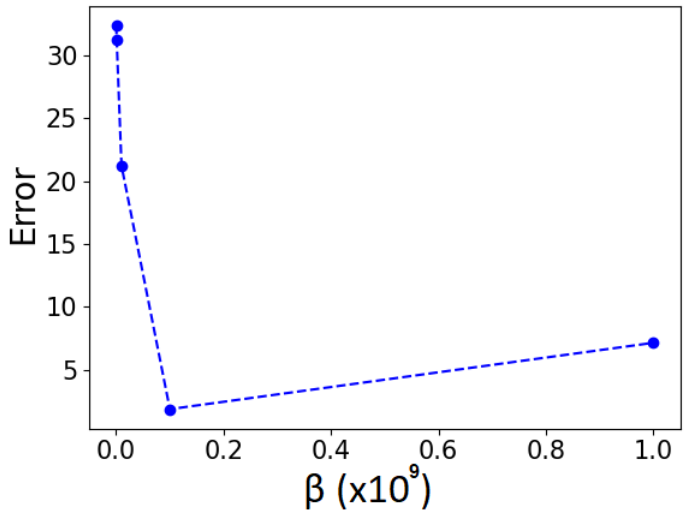

Figura 4.3: *Variación del error en función de β obtenida promediando los errores en el último epoch de cada serie de simulaciones de la Fig. 4.2. Se observa que el error tiende a disminuir con el aumento de β pero, para $\beta > 10^7$, el ruido introducido por las actualizaciones de los pesos sinápticos provoca un aumento en su valor promedio.*

De estos resultados se observa que el mínimo al que converge el error disminuye al aumentar el valor de β, pero a partir de $\beta = 10^8$ se aprecia la aparición de ruido (esto se observa mejor en la Fig. 4.4) tanto en las curvas de error como en las de *accuracy*. Para valores mayores a $\beta = 10^8$ la magnitud del ruido aumenta. También es evidente que la *accuracy* alcanza el valor máximo en muy pocos *epochs*.

En la Fig. 4.3 se muestra la evolución del error convergido (*epoch* 200) como función de β, promediado para las distintas realizaciones. Se observa, en primera instancia, una disminución del error al aumentar β. Para $\beta > 10^7$ se aprecia un incremento del error, relacionado con la aparición del ruido en las curvas de error y *accuracy*.

En la Fig. 4.4 se graficó (para una realización) el promedio de la función de activación (FA) de todas las imágenes pertenecientes a cada una de las clases. En cada gráfico se observan los promedios de las FA que se obtienen para cada neurona de salida (N_i) y cada columna de gráficos corresponde a las imágenes de cada clase. Por ejemplo, en el gráfico a) se graficó el promedio de todas las FA que

se obtienen en cada neurona de salida para todas las imágenes que pertenecen a la clase 0. La fila superior corresponde a los promedios de las FA para valores de $\beta = 10^7$ y la fila inferior corresponde a $\beta = 10^8$.

En cada caso, las FA correspondientes a la clase de las imágenes sobre las que se promedia aumentan con los *epochs* hasta converger a un valor máximo y las FA de las clases restantes disminuyen hasta converger en un valor mínimo. El valor máximo al que convergen las FA aumenta al aumentar β, mientras que el mínimo disminuye al disminuir β. Esto explicaría la disminución del mínimo del error al aumentar β, ya que se tomaron como valores objetivo de las FA ± 0.85 y, por lo tanto, al aumentar β aumentan en promedio los máximos de las FA acercándose al valor objetivo, disminuyendo la diferencia entre las salidas objetivos y las salidas obtenidas. Esto implica que los términos de la función de error (Ec. (1.6)) se hacen cada vez más cercanos a cero, minimizando el error.

El ruido que se observa para $\beta = 10^8$ se debe a que la tangente hiperbólica comienza a asemejarse a una función escalón a medida que dicho parámetro crece, lo que provoca que el ajuste de las conductancias -y las correspondientes corrientes de salida- durante el proceso de aprendizaje conduce a variaciones importantes en los valores de FA, reflejadas en el ruido en la convergencia. Por el contrario, para valores de $\beta < 10^8$ el comportamiento de la tangente hiperbólica es más suave y las variaciones de FA durante el entrenamiento son menores, produciendo una convergencia también más suave.

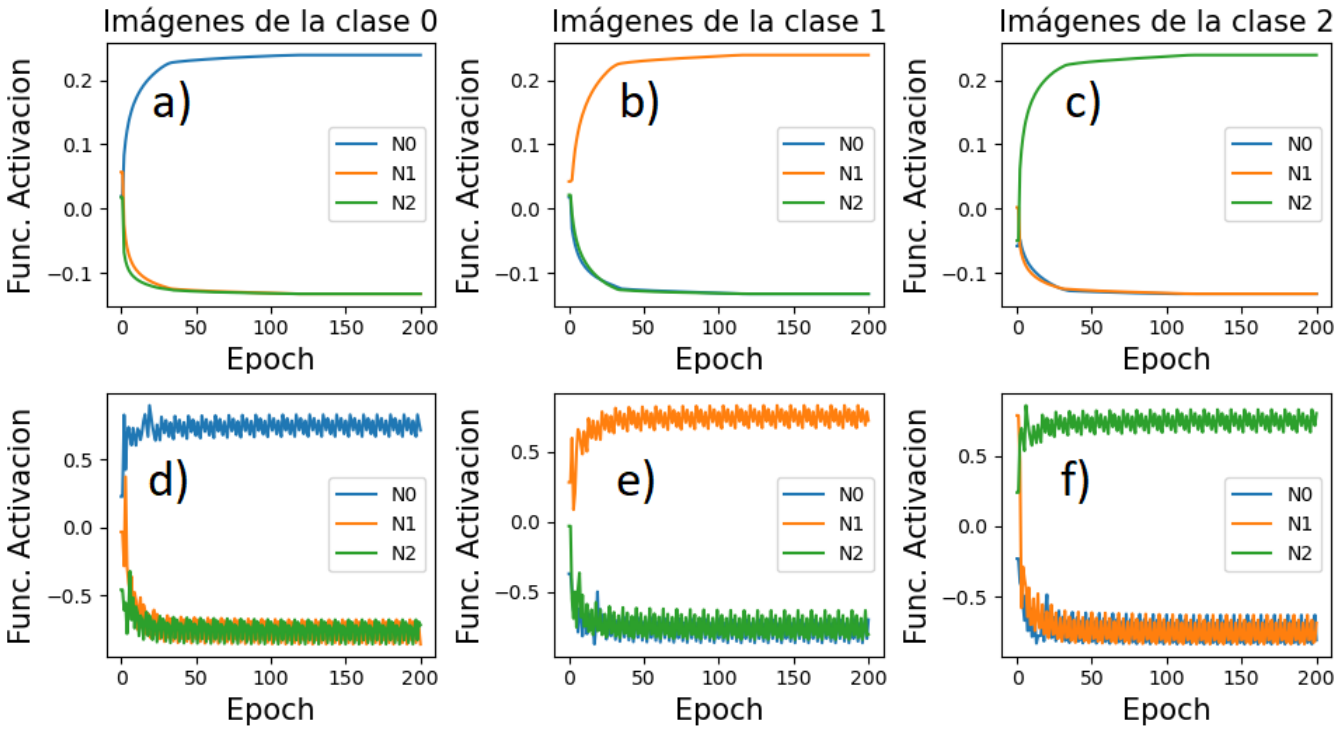

Figura 4.4: *Promedio de las funciones de activación (FA) de todas las imágenes pertenecientes a cada una de las clases (en cada columna se graficó el promedio correspondiente a cada clase). En la fila superior los promedios corresponden a $\beta = 10^7$ y en la fila inferior a $\beta = 10^8$. NO corresponde a la neurona cero, N1 a la neurona uno y N2 a la neurona 2.*

Debido a que se utilizaron datos experimentales de sistemas memristivos con bandas de conductancia acotadas y de distintos órdenes de magnitud, las corrientes asociadas a cada sistema también tendrán distintos órdenes de magnitud. Esto implica que, dado el argumento de la FA (βI), la elección del parámetro óptimo β_o para cada sistema va a depender del rango de corrientes (y conductancias) asociado. El criterio en el que nos basaremos para elegir un valor de β_o consiste en determinar el valor de este parámetro que da el mínimo error convergido, antes de la aparición de ruido. Para el caso del sistema LCMO/Ti el valor que adoptamos es $\beta_o = 10^7$.

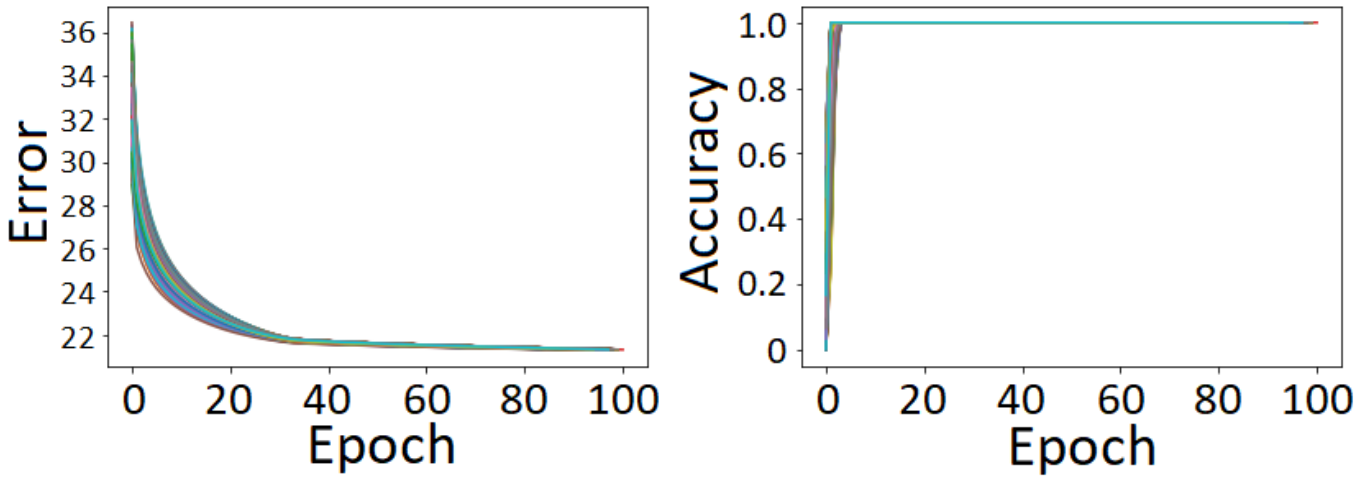

Figura 4.5: *Curvas de error (izquierda) y accuracy (derecha) en función de los epochs para el sistema LCMO/Ti. Se tomaron 2000 realizaciones del proceso de aprendizaje con* $\beta_o = 10^7$.

En la Fig. 4.5 se muestran las curvas de error y *accuracy* para 2000 realizaciones. Para cada realización se adoptó como condición de corte que el error alcance el valor 21.3 dentro de los 200 epochs. Este valor se eligió teniendo en cuenta que el mínimo absoluto para este sistema es de 21.2, por lo que la diferencia entre el valor de corte y el mínimo absoluto es de menos del $0.5\,\%$. Se observa poca variabilidad entre las distintas curvas a asociadas a cada realización y, en todos los casos, la *accuracy* converge rápidamente a 1. En el histograma de la Fig. 4.6 se muestra la distribución de probabilidades de convergencia para distintos números de *epochs*, la cual se puede describir con una distribución normal de media μ y desviación estándar σ. Para el sistema LCMO/Ti, se obtuvo $\mu = 97.5$ *epochs* y $\sigma = 1.6$ *epochs*.

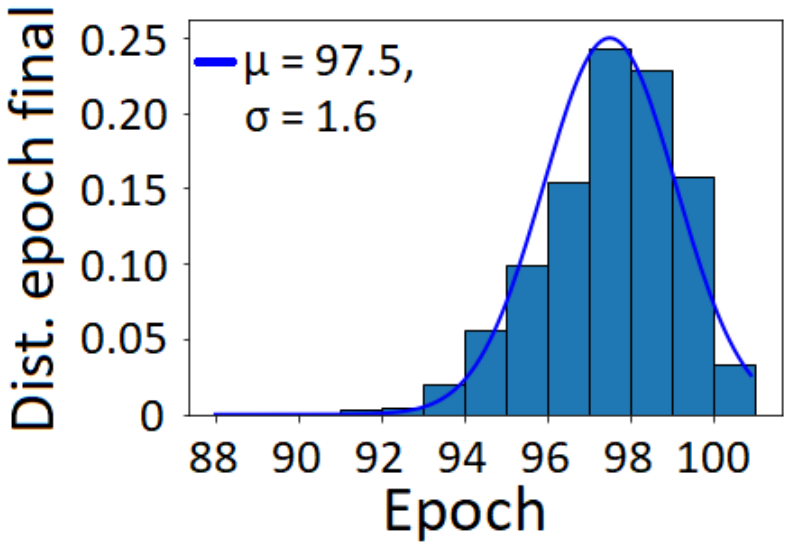

Figura 4.6: *Distribución de probabilidades de convergencia para distintos números de epochs para el sistema LCMO/Ti. La distribución de probabilidad de convergencia que se obtuvo es una normal con* $\mu = 97.5$ *epochs y* $\sigma = 1.6$ *epochs.*

4.2. Resultados para el sistema LCMO/Ag

Las curvas de potenciación y depreciación correspondientes al sistema LCMO/Ag se mostraron previamente en la Fig. 3.3. El rango de conductancias para este sistema se encuentra entre 7.9×10^{-7} S y 5.41×10^{-4} S. Para este sistema se tomó el parámetro $\beta_o = 10^3$.

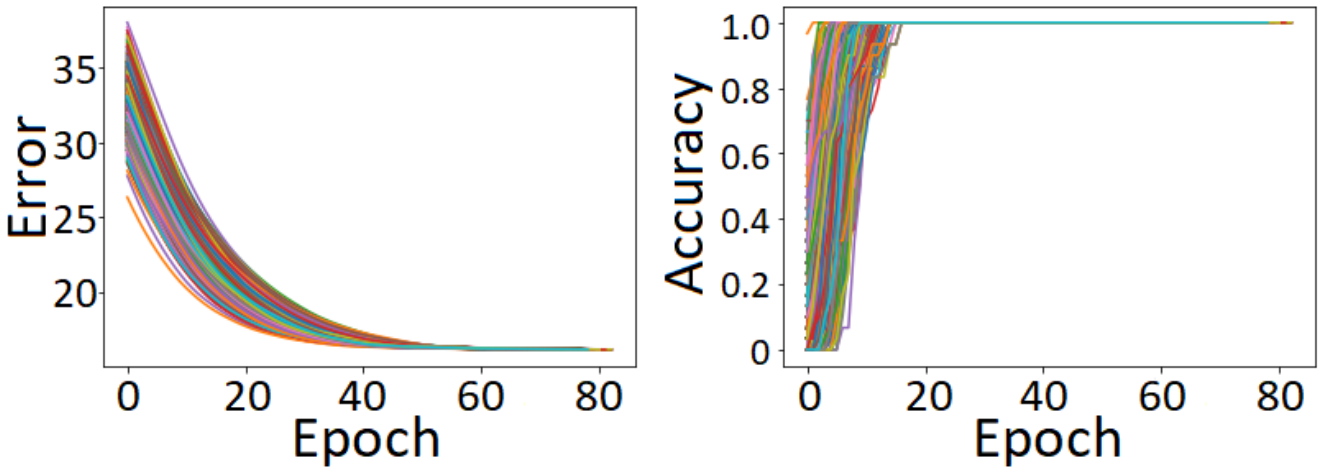

Figura 4.7: *Curvas de error (izquierda) y accuracy (derecha) en función de los epochs para el sistema LCMO/Ag. Se hicieron 2000 realizaciones del proceso de aprendizaje para* $\beta_o = 10^3$.

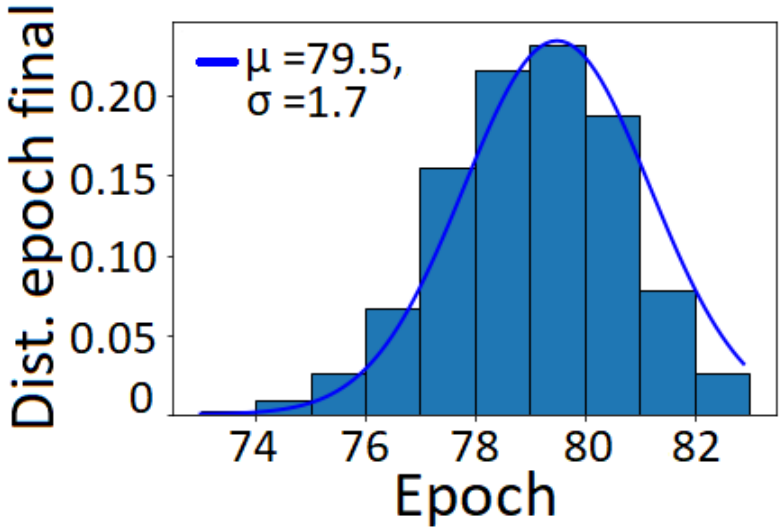

Figura 4.8: *Distribución de probabilidades de convergencia para distintos números de epochs para el sistema LCMO/Ag. La distribución de probabilidad de convergencia que se obtuvo es una normal con* $\mu = 79.5$ *epochs y* $\sigma = 1.7$ *epochs.*

En la Fig. 4.7 se muestran las curvas de error y *accuracy* en función de los *epochs* para 2000 realizaciones. En este sistema se observa un poco más de variabilidad comparado con el sistema LCMO/Ti, en particular para las curvas de *accuracy*, ya que en algunos casos son necesarios más de 15 *epochs* para alcanzar el valor de *accuracy* igual a 1. Como condición de corte para cada realización se tomó que el error alcance un valor de 16.2 dentro de los 200 *epochs*. Este valor umbral difiere en un 0.6 % del mínimo absoluto que se encuentra en 16.1.

En la Fig. 4.8 se ve que la distribución de probabilidades de convergencia obtenida se caracteriza por $\mu = 79.5$ *epochs* y $\sigma = 1.7$ epochs.

En la Fig. 4.9 se observan los promedios de las FA para cada neurona de salida asociadas a las imágenes de cada clase. Nuevamente, se observa que el promedio de las salidas asociadas a la clase de las imágenes sobre las que promediamos aumentan desde el estado inicial hasta converger a un valor máximo, mientras que el promedio de las salidas asociadas al resto de las clases disminuyen hasta converger a un valor mínimo.

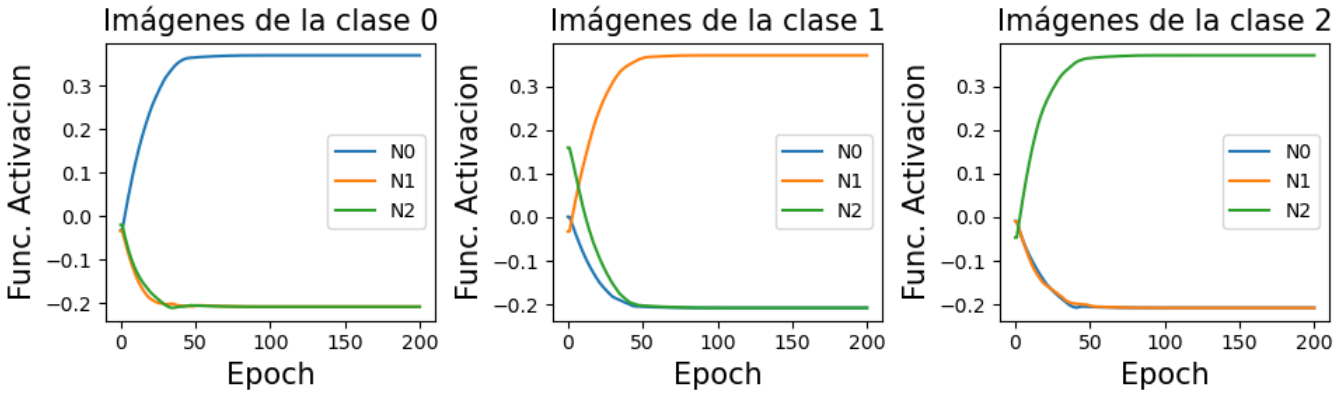

Figura 4.9: *Promedio de las FA de todas las imágenes pertenecientes a cada una de las clases. Las curvas graficadas corresponden a una realización usando el parámetro* $\beta = 10^3$. *NO corresponde a la neurona cero, N1 a la neurona uno y N2 a la neurona 2.*

4.3. Resultados para el sistema NSTO/LSMCO

Las curvas de potenciación y depreciación correspondientes al sistema NSTO/LSMCO se mostraron previamente en la Fig. 3.3. El rango de conductancias para este sistema se encuentra entre 8.59×10^{-4} S y 3.5×10^{-3} S. Para este sistema el parámetro es $\beta_o = 10^2$.

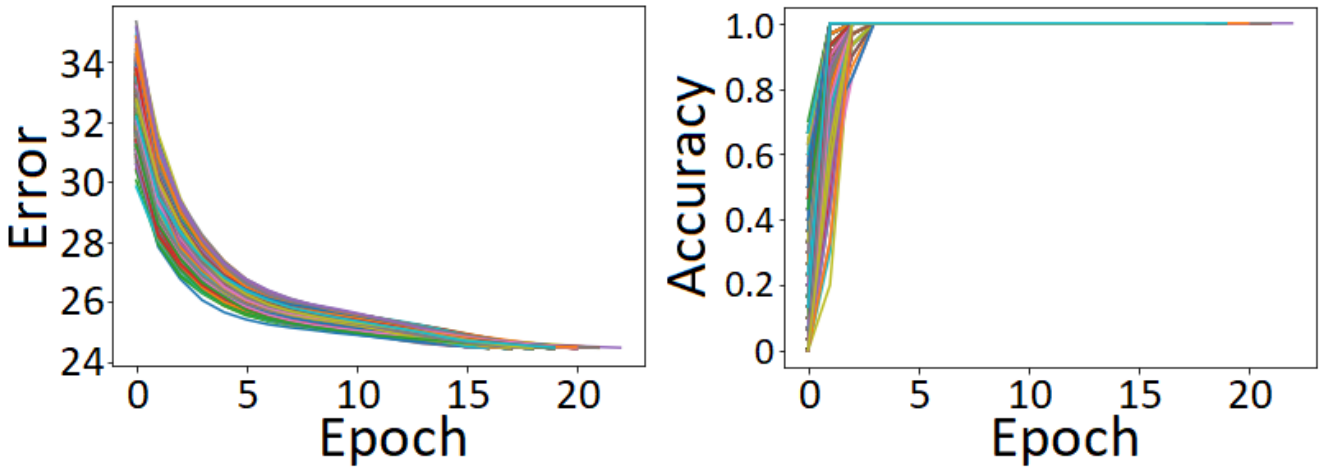

Figura 4.10: *Curvas de error (izquierda) y accuracy (derecha) en función de los epochs para el sistema NSTO/LSMCO. Se hicieron 2000 realizaciones del proceso de aprendizaje para* $\beta_o = 10^2$.

En la Fig. 4.10 se muestran las curvas de error y *accuracy* en función de los *epochs* para 2000 realizaciones, donde se observa que las curvas de *accuracy* alcanzan el valor 1 rápidamente (en menos de 5 *epochs*). Para cada realización se adoptó como condición de corte que el error alcance el valor 24.5 dentro de los 200 epochs. Este valor umbral difiere en $0.4\,\%$ del mínimo absoluto del error, que se encuentra en 24.4.

En el histograma de la Fig. 4.11 se muestra que la distribución de probabilidad de convergencia que se obtuvo es una normal con $\mu = 19.5$ *epochs* y $\sigma = 0.9$ *epochs*

En la Fig. 4.12 se observan que los promedios de las FA para cada neurona de salida asociadas a las imágenes de cada clase sigue el mismo comportamiento que para los sistemas LCMO/Ti y LCMO/Ag. Las salidas promediadas asociadas a la clase correcta aumentan desde el estado inicial hasta converger a un valor máximo, mientras que las salidas promediadas asociadas al resto de las clases disminuyen hasta converger a un valor mínimo.

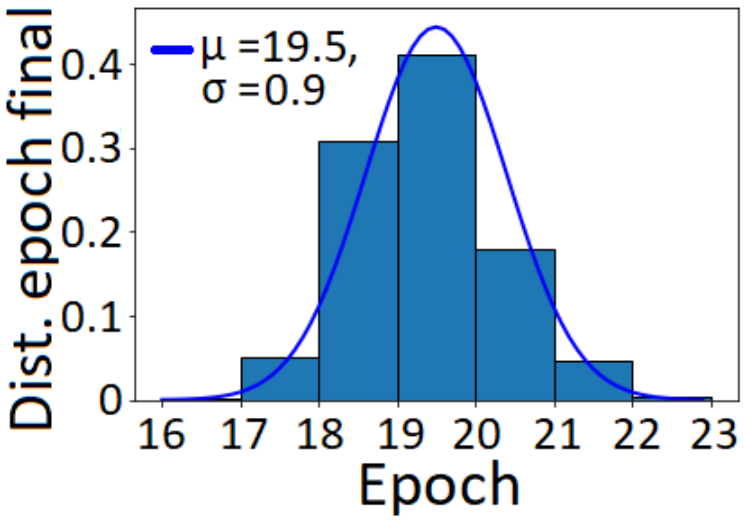

Figura 4.11: *Distribución de probabilidades de convergencia para distintos números de epochs para el sistema NSTO/LSMCO. La distribución de probabilidad de convergencia que se obtuvo es una normal con* $\mu = 19.5$ *epochs y* $\sigma = 0.9$ *epochs*

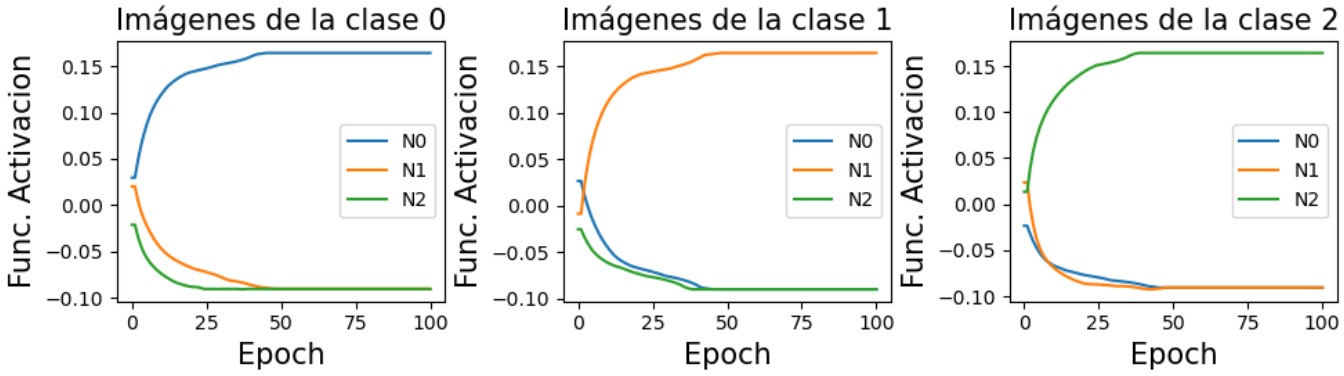

Figura 4.12: *Promedio de las FA de todas las imágenes pertenecientes a cada una de las clases. Las curvas graficadas corresponden a una realización usando el parámetro* $\beta = 10^2$*. NO corresponde a la neurona cero, N1 a la neurona uno y N2 a la neurona 2.*

4.4. Discusión

De los resultados obtenidos hasta ahora, podemos concluir que en general la *accuracy* alcanza el valor máximo (igual a 1) en pocos *epochs*, lo cual suele ser más rápido (en término de *epochs*) que la cantidad de *epochs* necesarios para alcanzar la saturación del error en un mínimo, para los tres sistemas considerados. Esto sugiere que la evolución del error es un indicador más apropiado para analizar la robustez de la convergencia, dado el tamaño reducido del conjunto de entrenamiento que estamos considerando (que al mismo tiempo es nuestro sistema de testeo), ya que permite discriminar mejor entre situaciones donde los datos están, en promedio, más próximos o más alejados de la frontera de decisión, determinada por los pesos sinápticos de la red luego de la convergencia.

Para los tres sistemas considerados, existe un β_o que minimiza el error y a la vez evita la aparición de "ruido" en la convergencia. Por otra parte, el β determina también el valor máximo (mínimo) que alcanza la función de activación ante una clasificación correcta (incorrecta); en general, se tiene que dichos valores se incrementan con el valor de β, concomitantemente con la disminución del error. Si consideramos los valores medios de los rangos de conductancia $G_{cen} = 3.5 \times 10^{-8}$ S para el sistema LCMO/Ti, $G_{cen} = 2.7 \times 10^{-4}$ S para el sistema LCMO/Ag y $G_{cen} = 1.3 \times 10^{-3}$ S para el sistema NSTO/LSMCO y los multiplicamos por los valores de β_o que optimizan la convergencia en cada caso

(10^7 A^{-1}, 10^4 A^{-1} y 10^3 A^{-1} respectivamente, donde ahora explicitamos las unidades de β), se tiene en los 3 casos valores en el orden de 10^{-1} $A^{-1}S$. Esto sugiere una forma de determinar a *priori* el β_o óptimo para cualquier sistema, dado su rango de conductividades permitidas, que garantice que la función de activación variará de manera suave durante el proceso de entrenamiento, evitando saltos abruptos que introducirán ruido en la convergencia, como se mostró en la Fig. 4.2 para valores de $\beta > \beta_o$.

La diferencia más notable que se encuentra entre los tres sistemas es el valor medio de las distribuciones de probabilidades de convergencia . Dado que los tres sistemas estudiados difieren en su rango de conductividades permitidas, en el número de pulsos de SET y RESET asociados a las curvas de potenciación y depreciación o en la linealidad de estas, es difícil saber a priori cuál es el parámetro más relevante que controla la velocidad de la convergencia del entrenamiento. Para intentar clarificar esto, recurrimos a curvas de potenciación y depreciación sintéticas a partir de las cuáles es posible variar estos parámetros uno a uno y de manera controlada, según se describirá en lo que sigue.

Capítulo 5

Simulación del entrenamiento del perceptrón utilizando las curvas de potenciación y depreciación sintéticas

5.1. Curvas sintéticas utilizadas

Con el objetivo de poder estudiar los cambios que se producen en los procesos de entrenamiento de la red al manipular algunas características de las curvas de potenciación y depreciación sinápticas, se reemplazaron las curvas de estos procesos medidas experimentalmente por curvas sintéticas.

El primer tipo de curvas que se consideró, por su simplicidad, fueron curvas lineales tanto para la potenciación como para la depreciación sináptica, como se observa en la Fig. 5.1.(a). Solo consideraremos curvas simétricas, es decir, curvas que se construyen con la misma cantidad de pulsos de SET y RESET, de forma que la pendiente de las dos rectas es la misma en valor absoluto, pero con signo negativo para la depreciación y positivo para la potenciación. Las curvas de potenciación y depreciación lineales en dispositivos memristivos presentan la ventaja de que los cambios de conductancia que se obtienen a partir de ellas son independientes del estado previo de conductividad, por lo que la actualización de los pesos sinápticos de la red se vuelve mucho más sencilla.

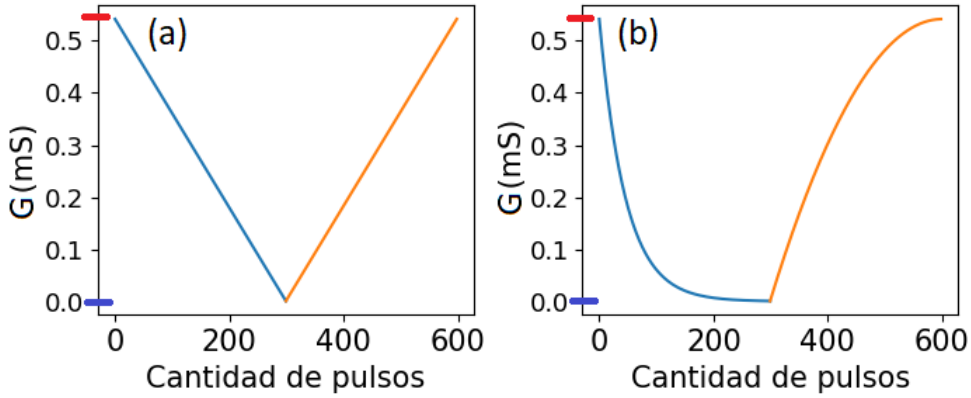

Figura 5.1: *(a) Curvas de potenciación y depreciación sintéticas lineales. (b) Curvas de potenciación y depreciación sintéticas no lineales. En ambos casos, las líneas horizontales roja y azul, marcadas sobre el eje vertical, representan el máximo y mínimo de conductancia respectivamente. Tanto en (a) y (b) se tomaron 570 pulsos de SET y RESET para la potenciación y depreciación. El rango de conductancia corresponde al del sistema LCMO/Ag.*

En la Fig. 5.1.(b) se muestran curvas de potenciación y depreciación no lineales. Para la depreciación se utilizó función exponencial decreciente y para la potenciación una función cuadrática creciente. Al igual que con las curvas lineales, se toma la misma cantidad de pulsos de SET y RESET para la potenciación y depreciación, respectivamente. Este tipo de curvas no lineales se eligieron

dado que emulan de manera cualitativa la no linealidad que presentan las curvas de potenciación y depreciación experimentales del sistema LCMO/Ti.

Sistema	Mínimo conductancia	Máximo conductancia
LCMO/Ti	1.59×10^{-8} S	5.55×10^{-8} S
LCMO/Ag	7.9×10^{-7} S	5.41×10^{-4} S
NSTO/LSMCO	8.59×10^{-4} S	3.5×10^{-3} S

Tabla 5.1: *Rangos de conductancia de cada sistema memristivo considerado en las simulaciones basadas en curvas sintéticas.*

En las curvas de la Fig. 5.1 se tomó, a modo de ejemplo, el rango de conductancia experimental del sistema LCMO/Ag y se utilizaron 570 pulsos de SET y de RESET. Los resultados que se presentan en las siguientes secciones se consiguieron al variar la cantidad de pulsos (de forma simétrica) y los rangos de conductancias, lo que determina una familia de distintas curvas potenciación y depreciación lineales y no lineales. Los rangos de conductancia utilizados (Tabla 5.1) corresponden a los rangos experimentales de los tres dispositivos que consideramos hasta ahora.

5.2. Determinación del parámetro β_o

La Fig. 5.2 muestra la evolución de la convergencia al variar el parámetro β de la función de activación, para curvas sintéticas lineales y no lineales en el rango de conductividades del sistema LCMO/Ag.

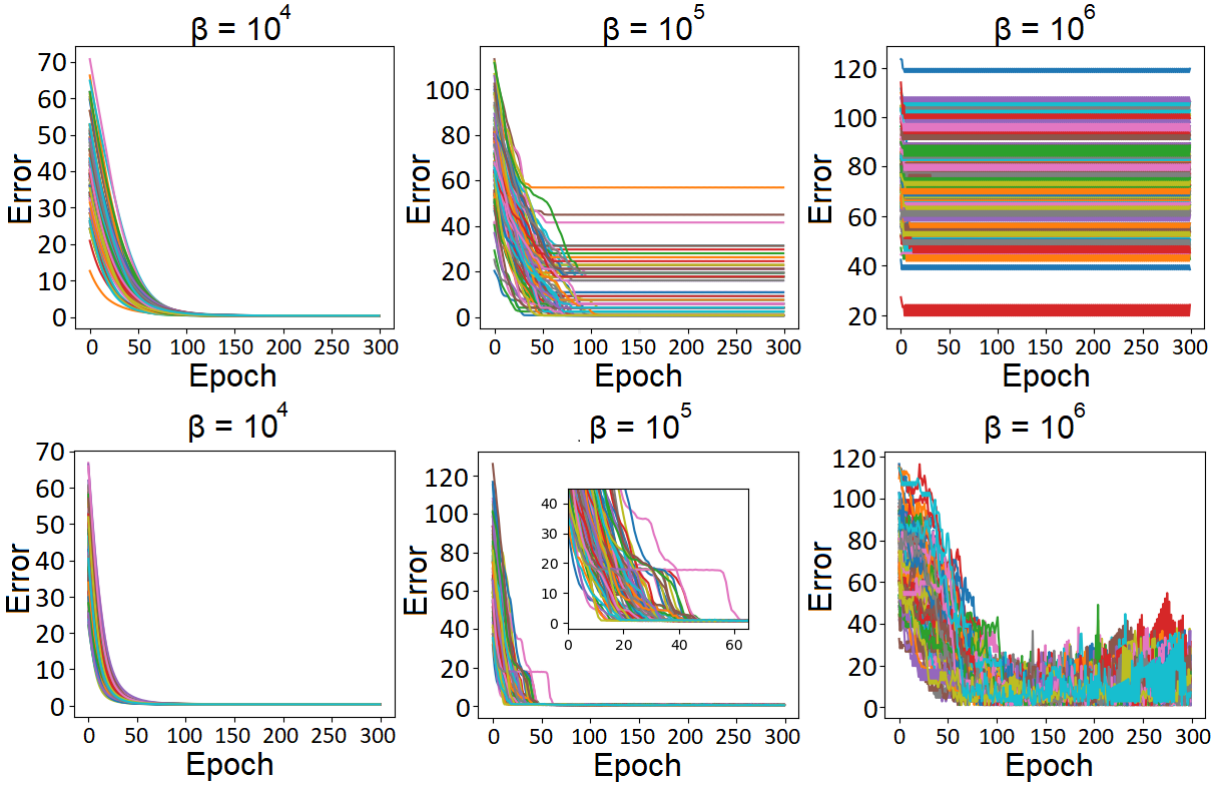

Figura 5.2: *Evolución del error para distintos valores de β. En los gráficos superiores se utilizaron curvas de potenciación y depreciación lineales, en las inferiores no lineales. En el inserto del gráfico inferior, para $\beta = 10^5$, se observa un zoom en los primeros 60 epochs.*

Los gráficos superiores corresponden a curvas de potenciación y depreciación sintéticas lineales, mientras que los inferiores corresponden a curvas no lineales. En ambos casos se utilizaron 200 pulsos

de SET y RESET y para las 100 realizaciones que conforman cada gráfico se llevaron a cabo 300 *epochs* de entrenamiento, suficientes para lograr la convergencia.

El criterio para elegir el β_o es el mismo que se utilizó para las curvas experimentales, donde se habían tomando 200 *epochs*. De estos gráficos notamos que el β_o óptimo es de 10^4, un orden de magnitud mayor que el valor 10^3 determinado en el capítulo anterior para las curvas experimentales. Esta diferencia se debe probablemente al mayor número de pulsos de SET y RESET asociados a las curvas sintéticas discutidas aquí. Del mismo gráfico se observa que la no linealidad del las curvas de potenciación y depreciación aumenta la velocidad de la convergencia.

Cuando subimos el β a 10^5, para las curvas lineales notamos que para algunas realizaciones el error estabiliza en terrazas de mayor valor que no permiten la convergencia al mínimo absoluto. En el caso de las curvas no lineales estas terrazas se manifiestan en los primeros 60 *epochs*, pero son inestables ya que para una cantidad suficientemente grande de *epochs* todas las realizaciones convergen al mínimo absoluto. Para $\beta = 10^6$ se aprecia ruido de gran amplitud tanto para el caso lineal como en el no lineal, consistentemente con lo discutido en el capítulo anterior. Se observó un comportamiento similar al utilizar los rangos de conductancias restantes, y se determinaron como óptimos en esos casos los valores $\beta_o = 10^8$ para el rango de conductividades del sistema LCMO/Ti y $\beta_o = 10^3$ para caso del sistema NSTO/LSMCO.

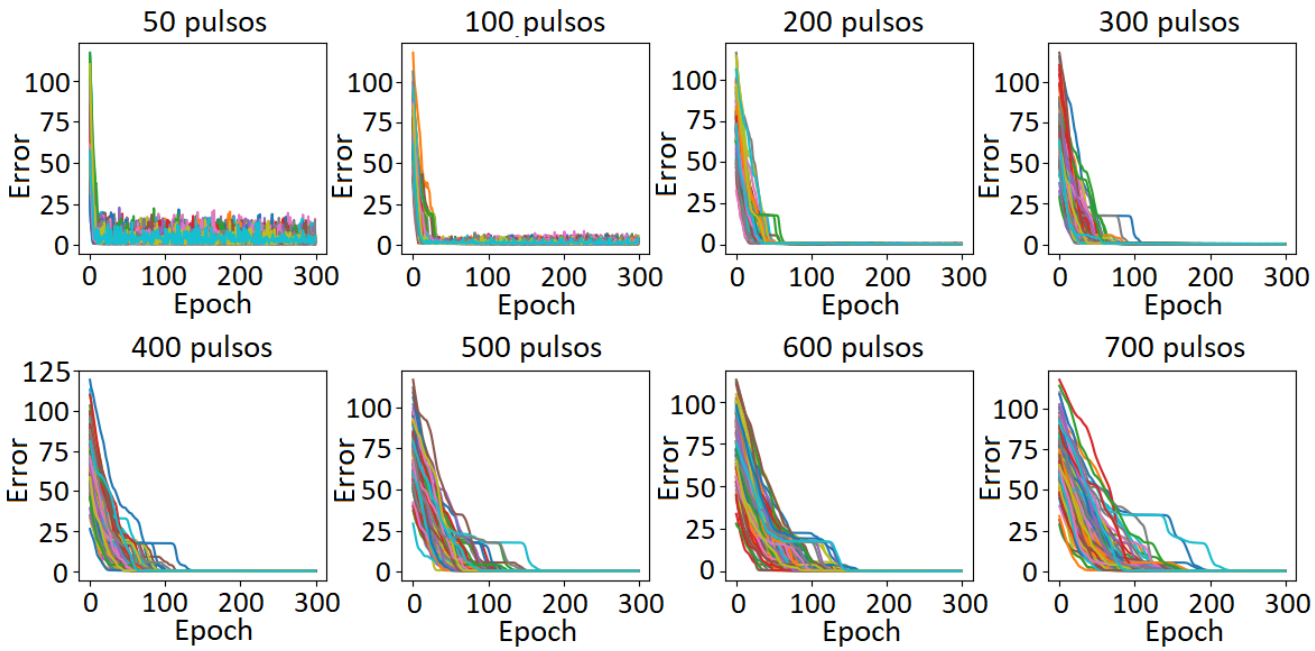

Figura 5.3: *Evolución del error vs. el número de epochs al variar el número de puntos que conforman las curvas de potenciación y depreciación sintéticas no lineales, en el rango del sistema LCMO/Ag. Se tomó $\beta = 10^5$. Cada gráfico consiste de 100 realizaciones.*

En la Fig. 5.3 se observa la evolución del error como función del número de epochs para un β mayor al óptimo y para distintos número de pulsos asociados a curvas de potenciación y depreciación no lineales, en el rango de LCMO/Ag. El valor del parámetro β se fijó en 10^5 y se observa que al pasar de 50 a 200 pulsos, el ruido en la evolución del error disminuye, pero para una cantidad mayor de pulsos las terrazas asociadas a la estabilización del error en mínimos locales parecen ser más persistentes. A partir de estos gráficos se observa que al elegir un valor del parámetro β no óptimo, no es posible obtener una convergencia monótona y sin ruido a un mínimo absoluto, como la mostrada en la Fig. 5.2. Lo mismo se observó en el caso de las curvas lineales y para los rangos de conductancias restantes.

5.3. Variación de cantidad de pulsos de SET y RESET

Determinados los valores β_o para cada rango de conductancias, se estudió la forma en la que cambia la velocidad de convergencia al variar la cantidad de pulsos de SET y RESET en las curvas de potenciación y depreciación sintéticas lineales y no lineales.

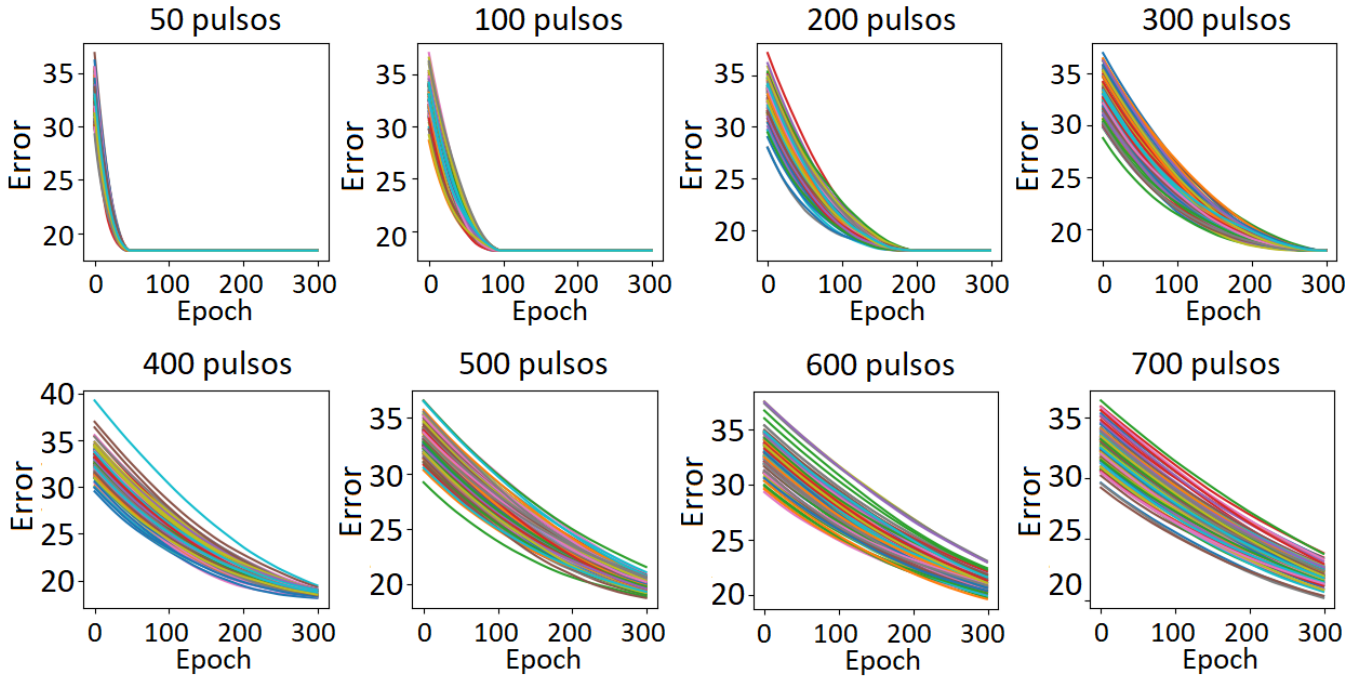

Figura 5.4: *Evolución del error vs. el número de epochs al variar el número de puntos que conforman las curvas de potenciación y depreciación sintéticas lineales, en el rango del sistema LCMO/Ag. Se hicieron 100 realizaciones en cada caso.*

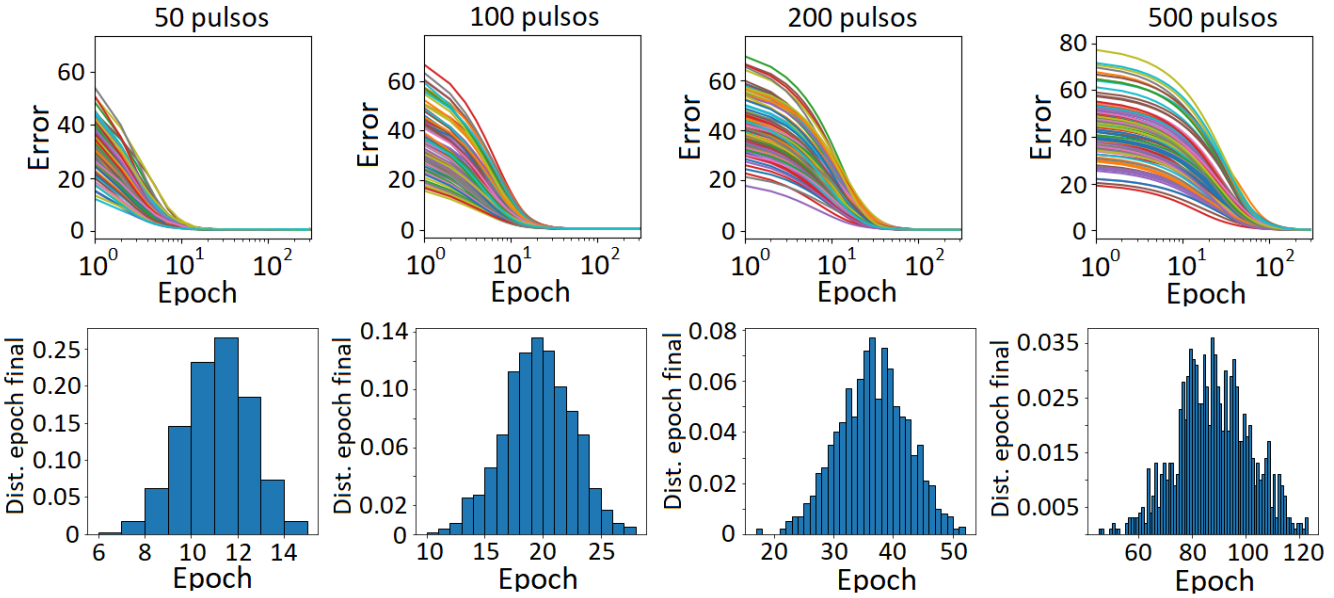

Figura 5.5: *(Fila superior) Evolución del error vs. el número de epochs para 50, 100, 200 y 500 pulsos en las curvas de potenciación y depreciación sintéticas no lineales, en el rango del sistema LCMO/Ag. Los gráficos se muestran en escala semi-log para apreciar mejor las diferencias en las velocidades de convergencia. (Fila inferior) Distribuciones de probabilidades de convergencia en función del número de epochs, para cada uno de los gráficos de la fila superior.*

Utilizando el rango de conductividades del sistema LCMO/Ag se calcularon las curvas de convergencia del error para 50, 100, 200, 300, 400, 500, 600 y 700 pulsos de SET y RESET, fijando la cantidad de *epochs* en 300. En la Fig. 5.4 se muestra una selección de los resultados obtenidos para curvas lineales. Por cada valor de pulsos se llevaron a cabo 100 realizaciones. A simple vista, se observa que la cantidad de *epochs* necesarios para alcanzar el mínimo de error aumenta con la cantidad de pulsos asociados a las curvas de potenciación y depreciación. Esto se debe a que para el algoritmo utilizado, un mayor número de pulsos redunda en una menor magnitud de las modificación de los valores de las actualizaciones de los pesos sinápticos, lo que naturalmente retarda el proceso de convergencia.

El mismo estudio se llevó a cabo utilizando curvas de potenciación y depreciación no lineales. En este caso se utilizaron 50, 100, 200 y 500 pulsos. Los resultados se muestran en la Fig. 5.5. En la fila superior se muestra la evolución del error en función de los *epochs* y como cambia esta al aumenta la cantidad de pulsos. Se tomaron 300 *epochs* y 100 realizaciones. El eje horizontal se muestra en escala logarítmica para que se aprecie mejor la convergencia. Al aumentar la cantidad de pulsos, la cantidad de *epochs* necesarios para alcanzar el mínimo parece aumentar, pero en este caso no es tan evidente como para las curvas lineales.

Para obtener más detalles sobre la convergencia se construyeron los histogramas a partir de los cuales se determinó la distribución de probabilidades de convergencia en función del número de *epochs*. Para cada realización se adoptó como condición de corte que el error alcance el valor 2 dentro de los 300 *epochs* y por cada valor de pulsos se llevaron a cabo 1000 realizaciones. En la fila inferior de la Fig. 5.5 se muestran los histogramas (para cada valor del número de pulsos) que ponen en evidencia que el valor medio de la distribución de probabilidad normal de convergencia aumenta al aumentar la cantidad de pulsos. El comportamiento es similar para los rangos de conductancias restantes, por lo que podemos concluir que al fijar el rango de conductancias y la linealidad de las curvas de potenciación y depreciación, el valor medio de la distribución de probabilidad normal de convergencia se puede controlar variando la cantidad de pulsos presentes en las curvas de potenciación y depreciación.

5.4. Variación del rango de conductancias

Por último se estudió la convergencia del error dejando la cantidad de pulsos de SET y RESET fijos en 200. Se comparó la convergencia de las curvas lineales en los rangos de conductancia de los sistemas LCMO/Ti y LCMO/Ag. Se hizo la misma comparación para las curvas no lineales. En todos los casos, la condición de corte para el error se tomó en 2, la cantidad de *epochs* máximos se fijo en 500 y se llevaron a cabo 3000 realizaciones.

En la Fig. 5.6 se muestran los resultados obtenidos para las curvas lineales. Los gráficos de la fila superior corresponden al rango de conductancias del sistema LCMO/Ti y la fila inferior al rango del LCMO/Ag. Comparando los histogramas observamos que para el rango LCMO/Ag la distribución de probabilidad normal de convergencia obtenida es μ =73.5 *epochs*, mientras que para el rango LCMO/Ti es $\mu = 92.2$ *epochs*.

De forma similar, utilizando las curvas no lineales (Fig. 5.7) se obtuvo $\mu = 36$ *epochs* para el rango LCMO/Ag (fila inferior) y $\mu = 76$ *epochs* para el rango LCMO/Ti (fila superior). Tanto para el caso lineal como para el no lineal, se observa que la velocidad de convergencia es mayor para el rango de conductividades del sistema de LCMO/Ag, siendo este rango mayor que el correspondiente al de LCMO/Ti.

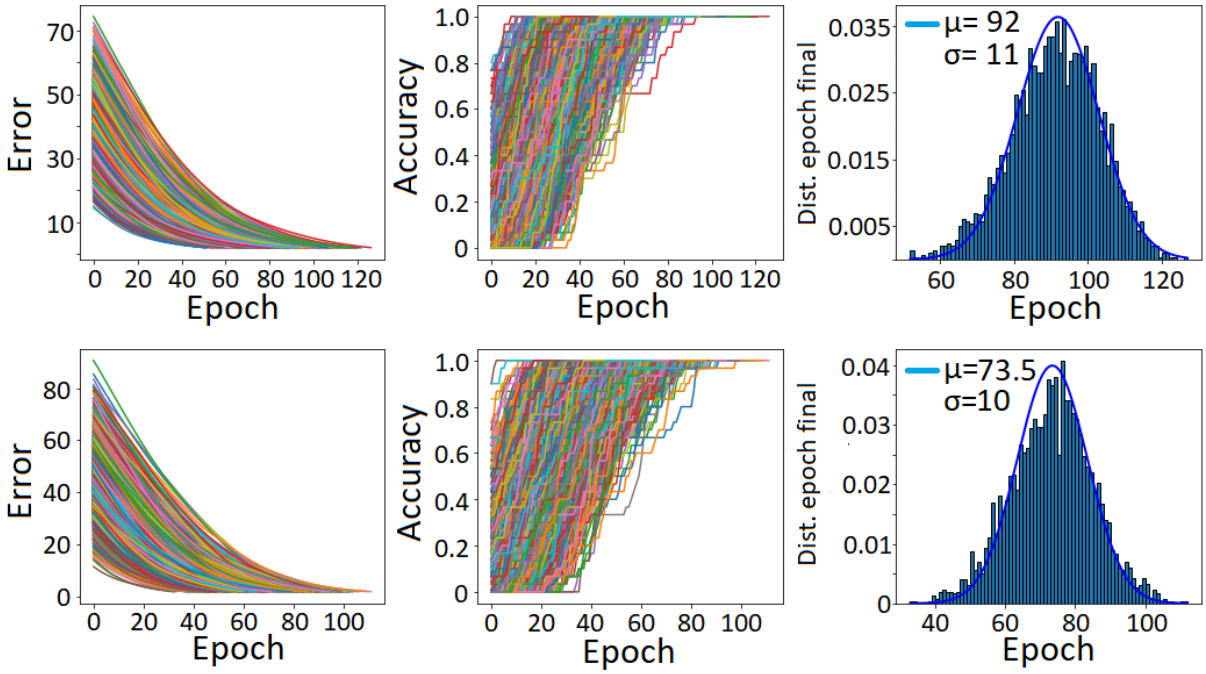

Figura 5.6: *Comparación de la convergencia del error usando curvas de potenciación y depreciación lineales en los rangos de conductancia de los sistemas LCMO/Ti (fila superior) y LCMO/Ag (fila inferior). En ambos sistemas se observa una gran variabilidad en la cantidad de epochs necesarios para que la accuracy alcance el valor máximo igual a 1. Las distribuciones de probabilidad de convergencia se ajustan por normales. Para el sistema LCMO/Ti se obtuvo* $\mu = 92$ *epochs y* $\sigma = 11$ *epochs, mientras que para el sistema LCMO/Ag se obtuvo* $\mu = 73.5$ *epochs y* $\sigma = 10$ *epochs.*

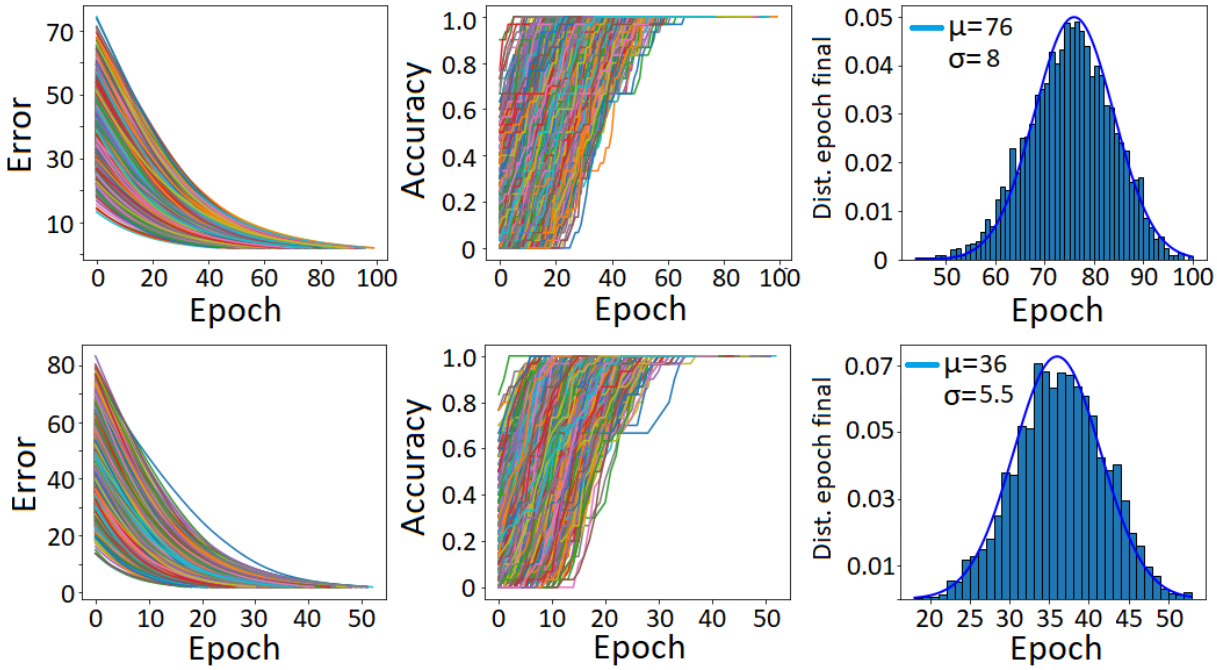

Figura 5.7: *Comparación de la convergencia del error usando curvas de potenciación y depreciación no lineales en los rangos de conductancia de los sistemas LCMO/Ti (fila superior) y LCMO/Ag (fila inferior). Al igual que en el caso de las curvas sintéticas lineales, en ambos sistemas se observa una gran variabilidad en la cantidad de epochs necesarios para que la accuracy alcance el valor máximo igual a 1. Las distribuciones de probabilidad de convergencia se ajustan por normales. Para el sistema LCMO/Ti se obtuvo* $\mu = 76$ *epochs y* $\sigma = 8$ *epochs, mientras que para el sistema LCMO/Ag se obtuvo* $\mu = 36$ *epochs y* $\sigma = 5.5$ *epochs.*

5.5. Discusión

A partir del análisis del proceso de aprendizaje utilizando curvas de potenciación y depreciación sintéticas, se observa que el parámetro β es altamente relevante en el proceso de convergencia. Para valores de β por encima de su valor óptimo, el sistema estabiliza su error en terrazas asociados a mínimos locales. Este efecto es menos relevante para curvas de potenciación y depreciación no lineales. Este último tipo de curvas también parece favorecer la velocidad de la convergencia en todos los casos estudiados. Cabe resaltar que la literatura es contradictoria en ese punto, ya que si bien recientemente se ha reportado que las curvas no lineales mejoran el proceso de aprendizaje de sistemas de barras cruzadas de memristores [42], otros trabajos previos sugieren lo contrario [43]. Esta discrepancia puede tener origen en diferencias en las estrategias de actualización de los pesos y en otras particularidades del método de aprendizaje, como la elección de la función de costo. Por otra parte, nuestras simulaciones muestran que tanto al subir el número de pulsos asociados a las curvas de potenciación y depreciación, como al disminuir el rango de conductividades permitidas, el proceso de convergencia del error se ralentiza. Esto se debe a que, de acuerdo al método de actualización de pesos utilizado, las magnitudes de las actualizaciones correspondientes disminuyen.

Todos los resultados obtenidos hasta el momento se basan en el análisis o la manipulación de las curvas de potenciación y depreciación, las cuales son una característica física de cada sistema memristivos considerado. En el próximo capítulo se discute la implementación de estrategias adicionales que permitan aumentar la convergencia del error en una menor cantidad de *epochs*, como el uso de *mini-batches* o la suma de componentes aleatorias en las actualizaciones de los pesos.

Capítulo 6

Aumento de la velocidad de convergencia del aprendizaje a partir de la introducción de estocasticidad

En este capítulo se estudiarán estrategias de aceleración de la convergencia mediante la introducción de estocasticidad en el proceso de aprendizaje. Esto incluye el uso de *mini-batches* [18, 19] y por otro lado, la adición de componentes aleatorias en las actualizaciones de pesos sinápticos [44]. Esta última estrategia implica que la actualización de pesos se hará de acuerdo a:

$$G_{ij}^{act} = G_{ij} + \Delta G_{ij} + C_{ij}^{rand}, \tag{6.1}$$

donde G_{ij}^{act} es el valor de la conductancia después de actualizar, G_{ij} es la conductancia antes de actualizar, ΔG_{ij} el incremento calculado con el algoritmo propuesto y C_{ij}^{rand} es una componente aleatoria que se genera en el intervalo $[-\Delta G_{ij}, \Delta G_{ij}]$, de forma que será siempre una fracción de la actualización exacta ΔG_{ij}. Esto es equivalente a sumar una componente de ruido con distribución uniforme al cálculo de las actualizaciones. Todos los resultados presentados en este capítulo se calcularon usando curvas de potenciación y depreciación, tanto experimentales como sintéticas, correspondientes al sistema LCMO/Ag, en el rango de conductividades permitidas de dicho sistema.

6.1. Resultados usando curvas de potenciación y depreciación experimentales

Para comenzar, se implementó el uso de dos *mini-batches* y se utilizaron los datos experimentales de potenciación y depreciación para calcular la evolución del error. Para esto se fijó en 200 la cantidad de *epochs*, se usaron 2 mini-batches (se dividió el conjunto de entrenamiento en 2 conjuntos de 15 imágenes cada uno) y se llevaron a cabo 1000 realizaciones. En la Fig. 6.1 se muestran los resultados obtenidos, en donde se observa que el ruido es predominante. Esto indica que, dado el tamaño pequeño del conjunto de datos utilizado, el grado de estocasticidad introducido en el proceso de aprendizaje mediante la implementación de *mini-batches* es demasiado grande para garantizar una convergencia suave.

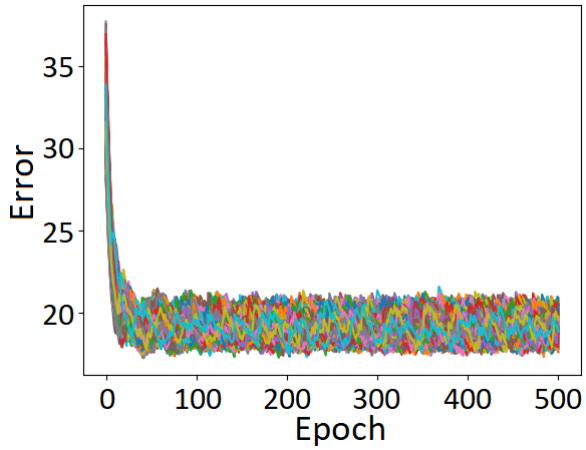

Figura 6.1: *Evolución del error utilizando los datos experimentales del sistema LCMO/Ag e implementando dos mini-batches.*

Lo siguiente fue implementar la suma de componentes aleatorias en las actualizaciones de los pesos sinápticos, nuevamente utilizando las curvas de potenciación y depreciación experimentales del sistema LCMO/Ag. Se fijo como condición de corte que el error alcance un valor de 16.2 (este valor se eligió con el criterio explicado en el Cap. 4), se tomaron 200 *epochs* como máximo y se llevaron a cabo 3000 realizaciones. En el histograma de la Fig. 6.2 se observa que la distribución de probabilidad de convergencia es normal con $\mu = 39.5$ *epochs* y $\sigma = 0.8$ *epochs*. En las curvas de error no se observó ruido como en el caso de la implementación de *mini-batches*. Comparando estos resultados con los que se obtuvieron al simular el proceso de aprendizaje sin utilizar ningún método estocástico (Fig. 4.8), vemos que la cantidad de *epochs* necesarios para lograr la convergencia se redujo un 50 %.

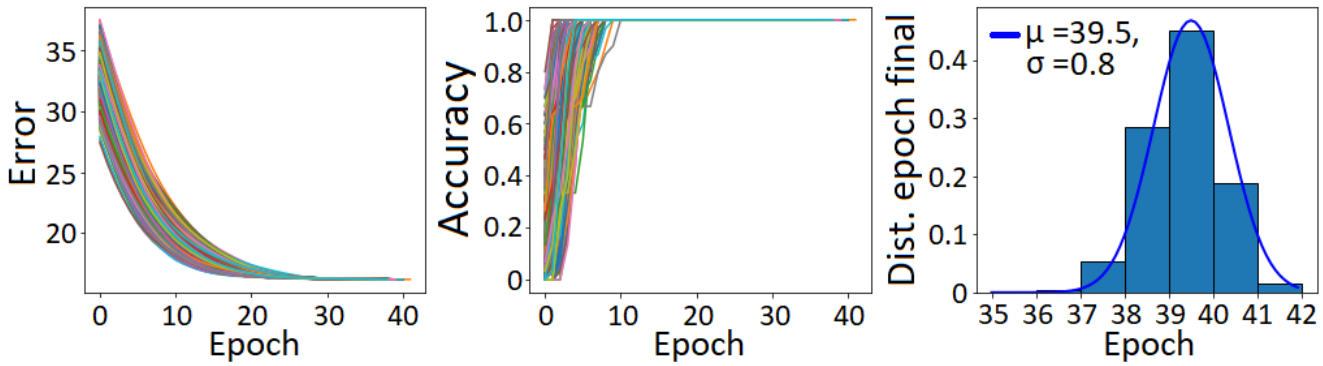

Figura 6.2: *Curvas de error (izquierda) y accuracy (centro) simuladas utilizando los datos experimentales e implementando la suma de componentes aleatorias en las actualizaciones de los pesos sinápticos. En el caso de la accuracy, el valor máximo se alcanza en menos de 10 epochs. (Derecha) La distribución de probabilidad de convergencia obtenida es normal con $\mu = 39.5$ epochs y $\sigma = 0.8$ epochs.*

6.2. Análisis del ruido introducido en la convergencia por los métodos estocásticos

Es aceptado que la implementación de métodos que introducen estocasticidad en el entrenamiento de redes neuronales generan ruido en la evolución del error [45, 46]. Por esto, en la implementación de este tipo de estrategias se debe lograr un compromiso entre el aumento de la velocidad de la convergencia y el umbral de ruido "aceptable" en la evolución de la función de costo durante el entrenamiento.

En este apartado se estudia el impacto que tiene la elección de la cantidad de pulsos que generan las curvas de potenciación y depreciación sintéticas en la magnitud del error introducido por este tipo de estrategias. Se realizó una serie de simulaciones a partir de las cuales fue posible comparar de forma cuantitativa (para una cantidad de pulsos fija) la magnitud del ruido introducido al usar *mini-batches* y suma de componentes aleatorias, respectivamente. En las primeras dos series se implementaron *mini-batches* con curvas de potenciación y depreciación sintéticas lineales para la primera y no lineales para la segunda. En las tercer y cuarta series de simulaciones se implementó la suma de componentes aleatorias, usando curvas de potenciación y depreciación sintéticas lineales para la tercera y no lineales para la cuarta. Para cada serie de simulaciones, se obtuvo la evolución de las curvas de error variando la cantidad de pulsos entre 50, 100, 200 y 400 respectivamente. Se tomaron 500 *epochs* de entrenamiento y 100 realizaciones para cada una de las situaciones ya descriptas. En cada serie y para cada valor de la cantidad de pulsos, se calculó la media y la desviación estándar entre los *epochs* 300 y 500 (donde se consideró que el error convergió) y se tomó a la desviación estándar obtenida como una medida de la amplitud del ruido (bajo la hipótesis de que el ruido tiene distribución gaussiana).

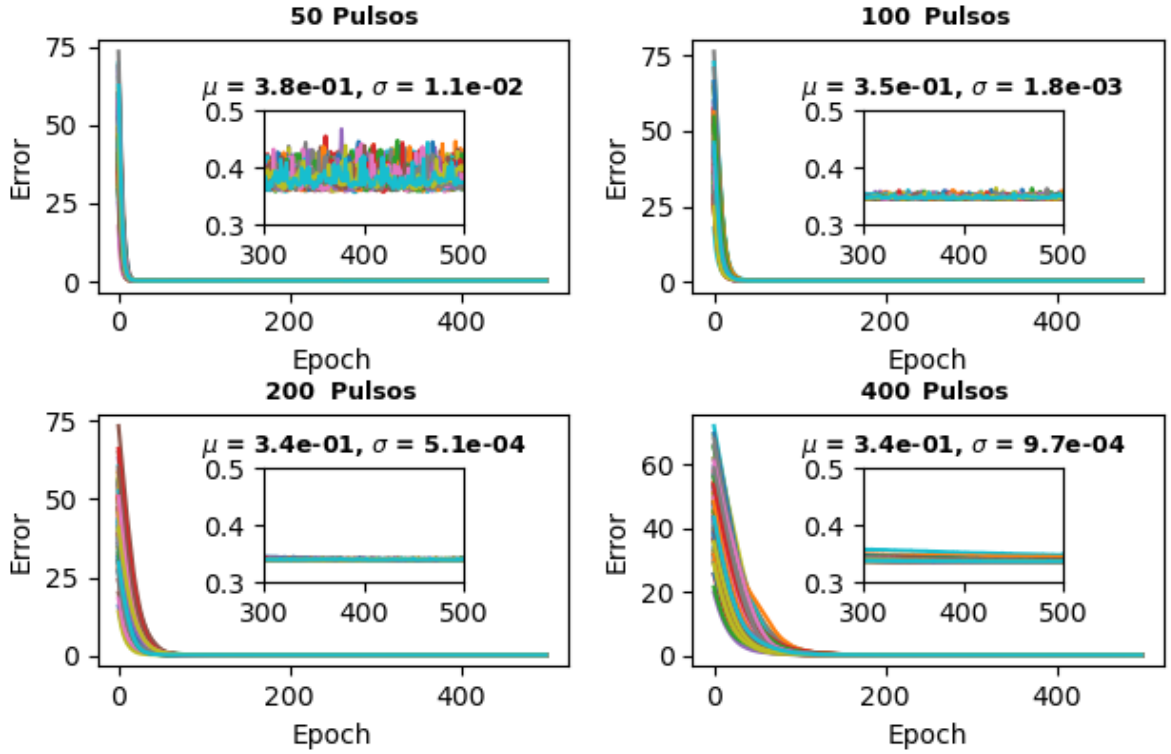

Figura 6.3: *Curvas de error vs. número de epochs para distintas cantidad de pulsos en las curvas de potenciación y depreciación, utilizando curvas sintéticas lineales y mini-batches de tamaño 15. En los insertos se observa un zoom sobre las curvas de error entre los epochs 300 y 500. En este intervalo se calculó la media μ y la desviación estándar σ asociada a distintas realizaciones del error convergido.*

En la Fig. 6.3 se muestran los resultados obtenidos para la primera serie en la que se usaron 2 *mini-batches* de 15 imágenes cada uno y se utilizaron curvas de potenciación y depreciación sintéticas lineales. En los insertos de cada gráfico se muestra un zoom en el intervalo de 300 a 500 *epochs*, en donde se calculó la media μ y la desviación estándar σ de todas las curvas de error. Lo que se observa es que a medida que aumenta la cantidad de pulsos el ruido disminuye, ya que la desviación estándar se reduce en 2 órdenes de magnitud al pasar de 50 a 400 pulsos.

Los resultados de la segunda serie de simulaciones se muestran en la Fig. 6.4. En este caso se usaron curvas sintéticas no lineales para el proceso de potenciación y depreciación y, al igual que en la serie anterior, se usaron 2 *mini-batches* de 15 imágenes cada uno. En este caso, el ruido solo se reduce en un orden de magnitud al pasar de 50 a 400 pulsos.

Comparando los resultados obtenidos entre las curvas sintéticas lineales (primer serie) y no lineales (segunda serie), podemos concluir que al variar la cantidad de pulsos la media μ del ruido no cambia

significativamente, pero las fluctuaciones del error, caracterizadas por σ, disminuyen. En particular, se observa que el valor de σ es mayor para las curvas sintéticas no lineales que para las lineales (comparando para una misma cantidad de pulsos).

Los resultados de la tercera serie de simulaciones se muestran en la Fig. 6.5. En este caso se implementó la suma de componentes aleatorias y se usaron curvas sintéticas lineales. Al igual que en la primera serie de simulaciones (en la que también se usaron curvas sintéticas lineales), la amplitud del ruido se reduce en dos órdenes de magnitud al pasar de 50 a 400 pulsos. Para cada valor de los pulsos, el orden de magnitud del ruido es el mismo para la primera y tercera serie de simulaciones.

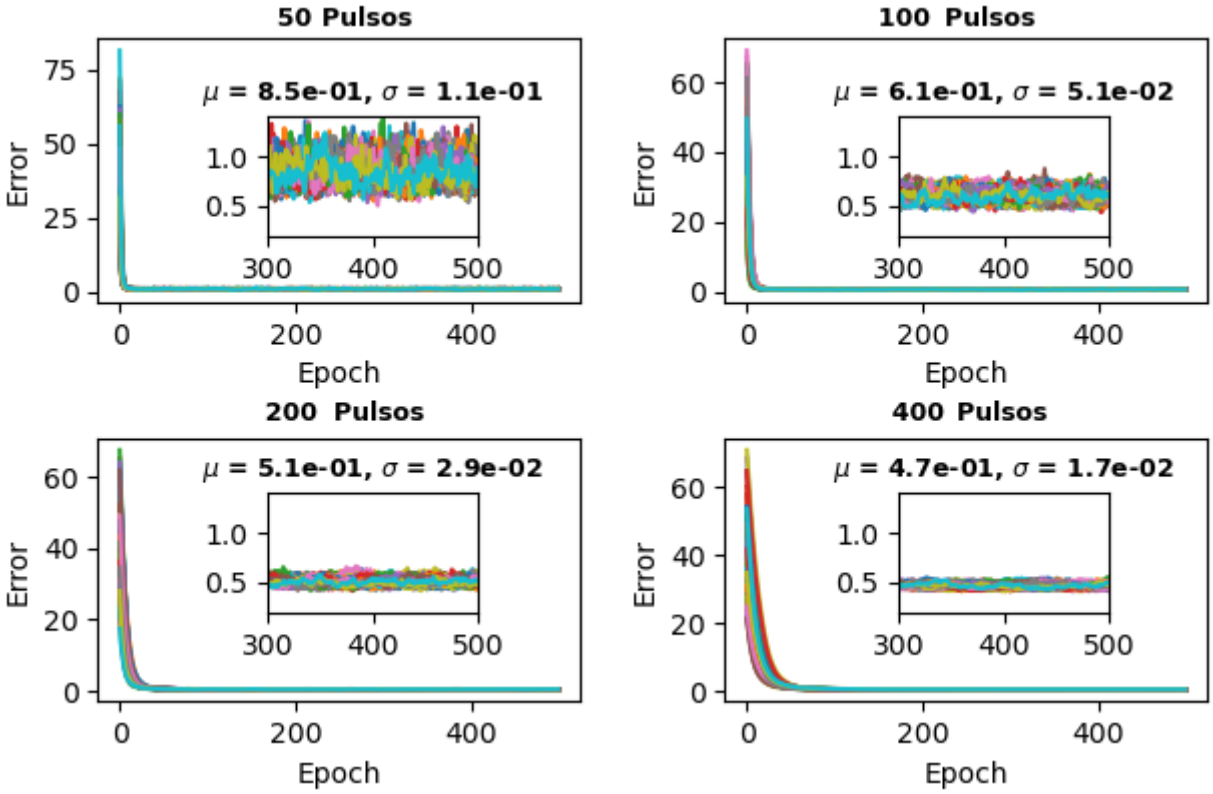

Figura 6.4: *Curvas de error vs. número de epochs para distintas cantidad de pulsos en las curvas de potenciación y depreciación, utilizando curvas sintéticas no lineales y mini-batches de tamaño 15. En los insertos se observa un zoom de sobre las curvas de error entre los epochs 300 y 500. En este intervalo se calculó la media μ y la desviación estándar σ asociada a distintas realizaciones del error convergido.*

Por último, los resultados de la cuarta serie se muestran en la Fig. 6.6, donde se implementó la suma de componentes aleatorias y se usaron curvas sintéticas no lineales. Se observa que la amplitud del ruido se reduce en un orden de magnitud al pasar de los 50 pulsos a los 4001 pero, comparando estos resultados con los obtenidos en la segunda serie de simulaciones (donde también se usaron curvas sintéticas no lineales), podemos concluir que el ruido inducido en este caso es menor en un orden de magnitud para cada valor de número de pulsos.

Al comparar los resultados de la tercera serie con los de la cuarta, observamos que implementar la suma de componentes aleatorias genera ruido cuya amplitud decrece cuando aumenta la cantidad de pulsos en las curvas de potenciación y depreciación sintéticas. Para los 50 pulsos, el orden de las fluctuaciones σ del ruido de las curvas sintéticas lineales (tercera serie) y no lineales (cuarta serie) es comparable. Para las curvas lineales, σ es un orden de magnitud menor a partir de los 100 pulsos que para las curvas no lineales. En el caso de la media μ de las curvas de error, para 50 pulsos, el caso lineal es un orden de magnitud menor que para el caso no lineal, pero a medida que la cantidad de pulsos aumenta μ se vuelve comparable en ambos casos y no se observan diferencias significativas.

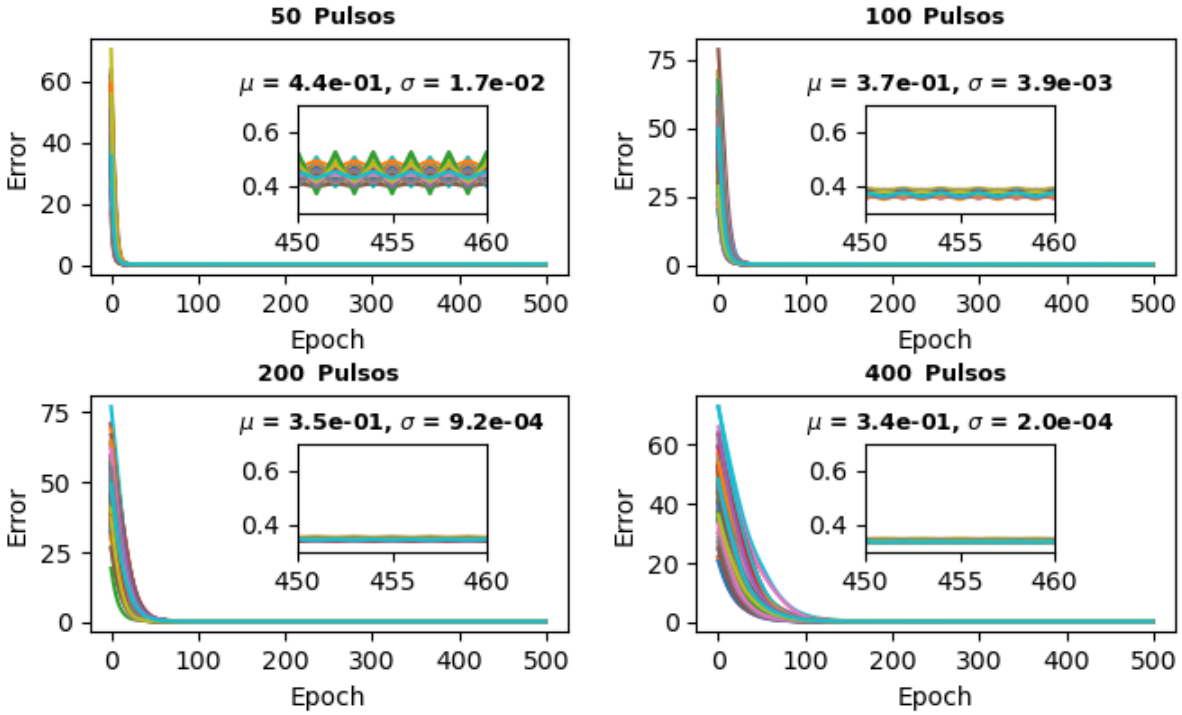

Figura 6.5: *Curvas de error vs. número de epochs para distintas cantidad de pulsos en las curvas de potenciación y depreciación, utilizando curvas sintéticas lineales y sumando componentes aleatorias en las actualizaciones de los pesos. En los insertos se observa un zoom sobre las curvas de error entre los epochs 450 y 460. En el intervalo de 300 a 500 epochs se calculó la media μ y la desviación estándar σ asociada a distintas realizaciones del error convergido.*

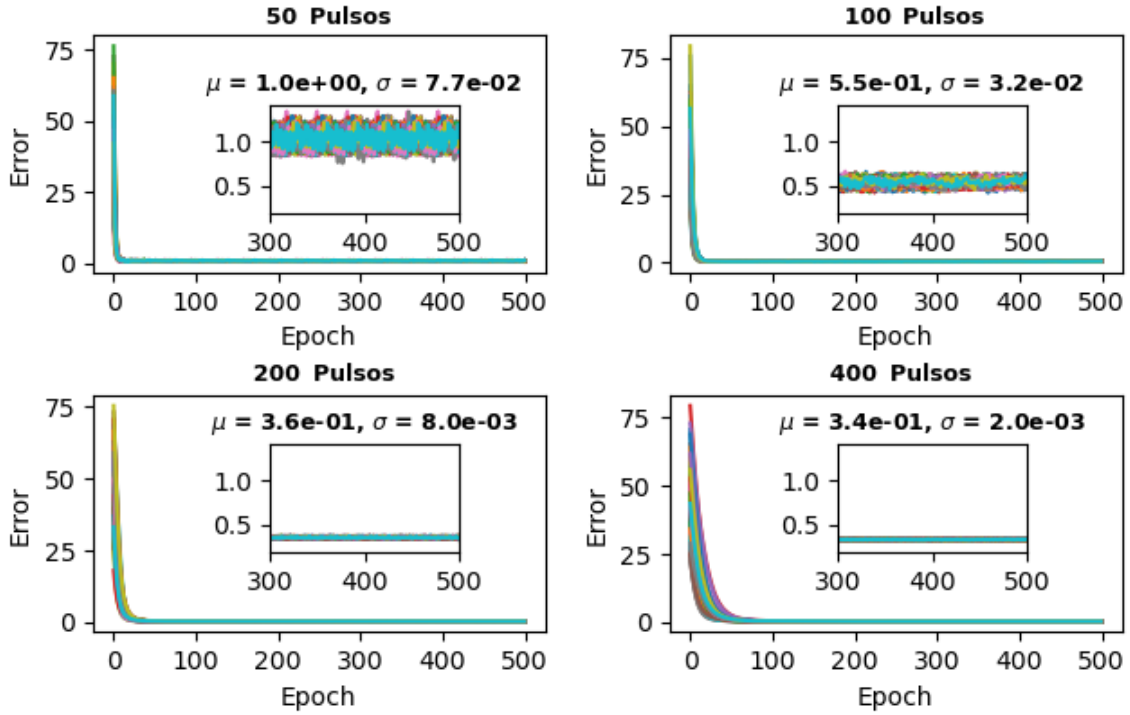

Figura 6.6: *Curvas de error vs. número de epochs para distintas cantidad de pulsos en las curvas de potenciación y depreciación, utilizando curvas sintéticas no lineales y sumando componentes aleatorias en las actualizaciones de los pesos. En los insertos se observa un zoom sobre las curvas de error entre los epochs 300 y 500. En este intervalo se calculó la media μ y la desviación estándar σ asociada a distintas realizaciones del error convergido.*

Podemos concluir que las fluctuaciones del ruido en la convergencia del proceso de aprendizaje,

inducido por la implementación de *mini-batches* y la implementación de la suma de componentes aleatorias en las actualizaciones de pesos sinápticos, disminuyen con la cantidad de pulsos con los que generan las curvas potenciación y depreciación. En particular, se observó que para curvas de potenciación y depreciación lineales, las fluctuaciones del ruido son comparables al implementar cualquiera de los métodos considerados. En el caso de curvas de potenciación y depreciación no lineales, las fluctuaciones del ruido al implementar *mini-batches* son mayores que al implementar la suma de componentes aleatorias (comparando para una misma cantidad de pulsos). Esto último está de acuerdo con lo observado en la Fig. 6.1 en la que se implementaron *mini-batches* utilizando los datos experimentales de las curvas de potenciación y depreciación sinápticas. Por último, en ningún caso se observaron variaciones significativas en el valor de la media μ del ruido, lo que esta de acuerdo con los resultados previos en donde se concluyó que el valor del mínimo del error (donde consideramos la convergencia) depende fundamentalmente de la elección del parámetro β.

6.3. Convergencia utilizando *mini-batches* y componentes aleatorias

A continuación, se analizó para curvas de potenciación y depreciación sintéticas lineales (se tomó 200 pulsos y $\beta_o = 10^4$) la convergencia del error utilizando *mini-batches* y suma de componentes aleatorias en las actualizaciones de los pesos. Para esto se realizaron tres series de simulaciones: en la primera solo se usaron *mini-batches*, en la segunda solo se usó la suma de componentes aleatorias y en la tercera se combinaron ambas. En todas las series se tomaron 3000 realizaciones, 200 pulsos para curvas lineales y se fijó como condición de corte que el error alcance un valor de 2. Además se fijó en 300 la cantidad máxima de *epochs*.

Los resultados de la primera serie se muestran en la Fig. 6.7, en donde se observa que la distribución de probabilidad de convergencia es normal con $\mu = 40$ *epochs* y $\sigma = 6$ *epochs*. Comparando estos resultados con los presentados en la Fig. 5.6, en donde no se utilizó ningún método estocástico, vemos que la cantidad de *epochs* necesarios para lograr la convergencia se reduce en un 45 %.

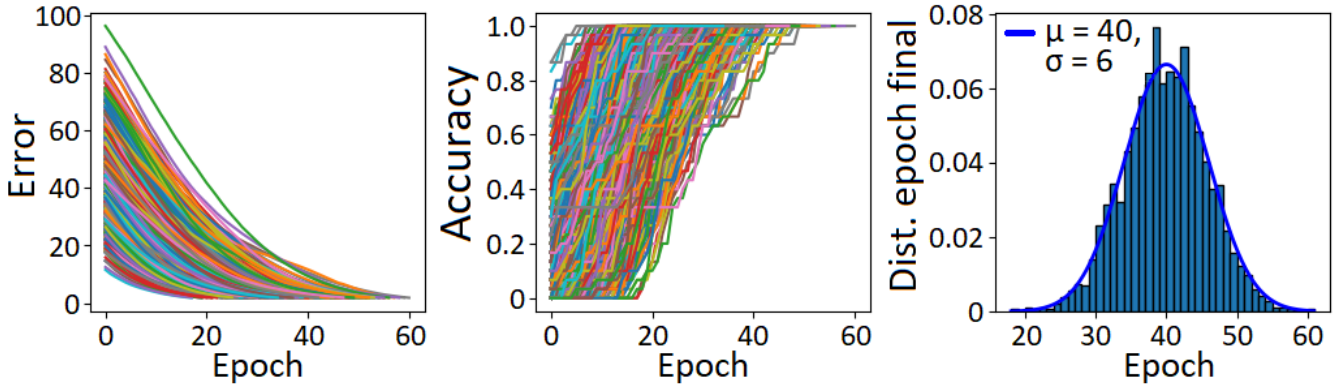

Figura 6.7: *Curvas de error (izquierda) y accuracy (medio) vs. el número de epochs y distribución de probabilidad normal ($\mu = 40$ epochs, $\sigma = 6$ epochs) de convergencia como función del número de epochs (derecha), obtenidas utilizando 2 mini-batches.*

Los resultados de la segunda serie se muestran en la Fig. 6.8. La distribución de probabilidad de convergencia es normal con $\mu = 38$ *epochs* y $\sigma = 6$ *epochs*. Si comparamos estos resultados con los presentados en la Fig. 5.6, en donde no se utilizó ningún método estocástico, vemos que la cantidad de *epochs* necesarios para lograr la convergencia se reduce en un 48 %. Comparando con los resultados cuando solo se implementan *mini-batches* (Fig. 6.7) vemos que en ambos casos se obtiene $\sigma = 6$ *epochs*, por lo que podemos concluir que utilizar cualquiera de estos métodos, reduce la cantidad de *epochs* necesarios para lograr la convergencia en una cantidad comparable.

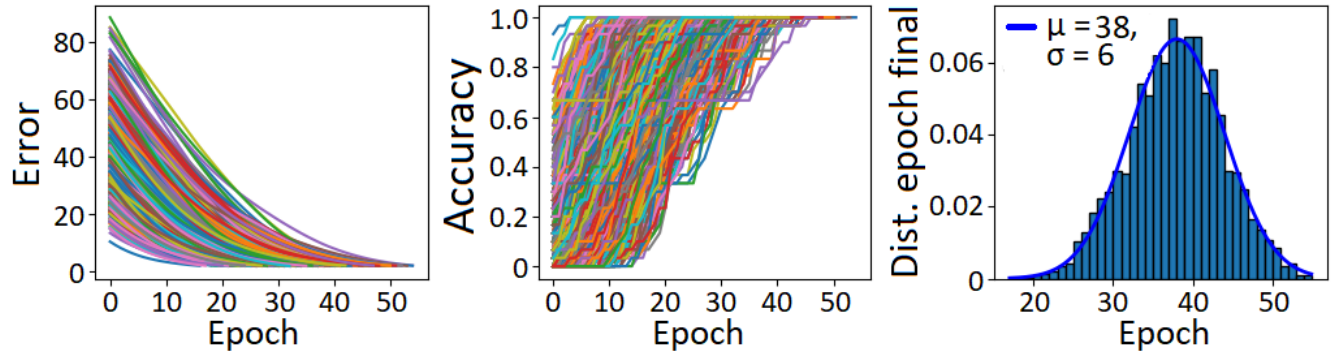

Figura 6.8: *Curvas de error (izquierda) y accuracy (medio) vs. el número de epochs y distribución de probabilidad normal ($\mu = 38$ epochs, $\sigma = 6$ epochs) de convergencia como función del número de epochs (derecha), obtenidas utilizando suma de componentes aleatorias.*

Los resultados de la tercera serie se muestran en la Fig. 6.9, en donde se observa que la distribución de probabilidad normal es $\mu = 21$ *epochs* y $\sigma = 3$ *epochs*. En este caso, al comparar estos resultados con los presentados en la Fig. 5.6, en donde no se utilizó ningún método estocástico, vemos que la cantidad de *epochs* necesarios para lograr la convergencia se reduce en un 71 %, lo que es cerca de un 22 % menor que el valor obtenido al implementar cada método por separado.

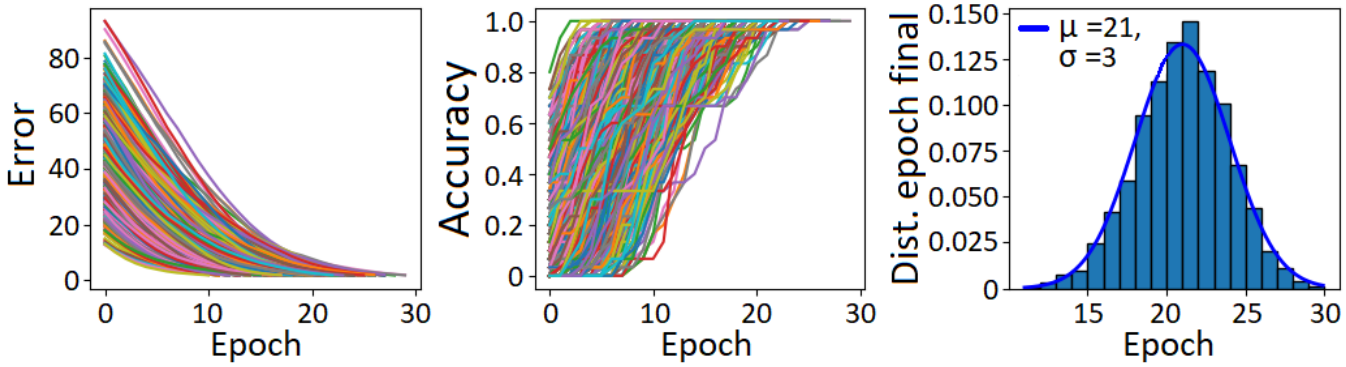

Figura 6.9: *Curvas de error (izquierda) y accuracy (medio) vs. el número de epochs y distribución de probabilidad normal ($\mu = 21$ epochs, $\sigma = 3$ epochs) de convergencia como función del número de epochs (derecha), obtenidas utilizando 2 mini-batches y sumando componentes aleatorias.*

6.4. Mejora en la convergencia del *accuracy* para β no óptimo usando métodos estocásticos

Como se observó en la Fig. 5.2, al utilizar curvas de potenciación y depreciación sintéticas lineales y tomando un valor $\beta = 10^5$ no óptimo, se generan terrazas en las curvas de error vs. *epochs* que dificultan la convergencia del proceso de aprendizaje. En la Fig. 6.10 se muestran los resultados al tomar 3000 realizaciones del proceso de aprendizaje en las condiciones descriptas previamente, donde se tomó como condición de corte que el error alcance el valor 2 dentro de los 500 *epochs*. En el gráfico de la izquierda se observan las terrazas en las curvas de error vs. *epochs*, en el gráfico del centro también vemos terrazas en las curvas de *accuracy* vs. *epochs* donde claramente muchas realizaciones no llegan a alcanzar el valor máximo de *accuracy*. En el gráfico de la derecha se muestra un histograma en donde se grafica la frecuencia de cada valor de la *accuracy* en el último *epoch* (500). Se observa que la frecuencia de la *accuracy* $= 1$ es de 1350, pero sumando la frecuencia para valores de *accuracy* < 1 se obtiene 1650, por lo que es mayor la cantidad de realizaciones que no llegan al valor ideal de la *accuracy* que las que sí lo logran.

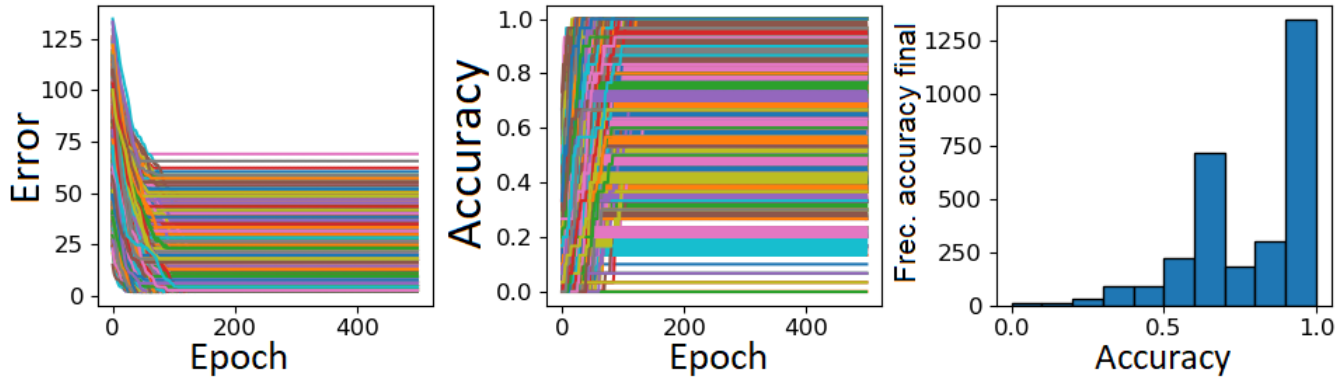

Figura 6.10: *Izquierda: Curvas de error vs. epochs. Centro: Curvas de accuracy vs. epochs. Para las curvas de error y accuracy se observa la presencia de gran cantidad de terrazas que no permiten la convergencia. Derecha: Histograma de frecuencia de cada valor de la accuracy en el último epoch. No se utilizó ningún método estocástico.*

Implementando la suma de componentes aleatorias y 2 *mini-batches* se simuló nuevamente el proceso de aprendizaje bajo las mismas condiciones que en la Fig. 6.10. En este caso se observa que la frecuencia de la *accuracy* $= 1$ es 2678, mientras que para *accuracy* < 1 la frecuencia es 322, lo que representa una mejora significativa en la convergencia de la *accuracy* al valor ideal 1 respecto de los resultados de la Fig. 6.10 en la que no se utilizaron métodos estocásticos.

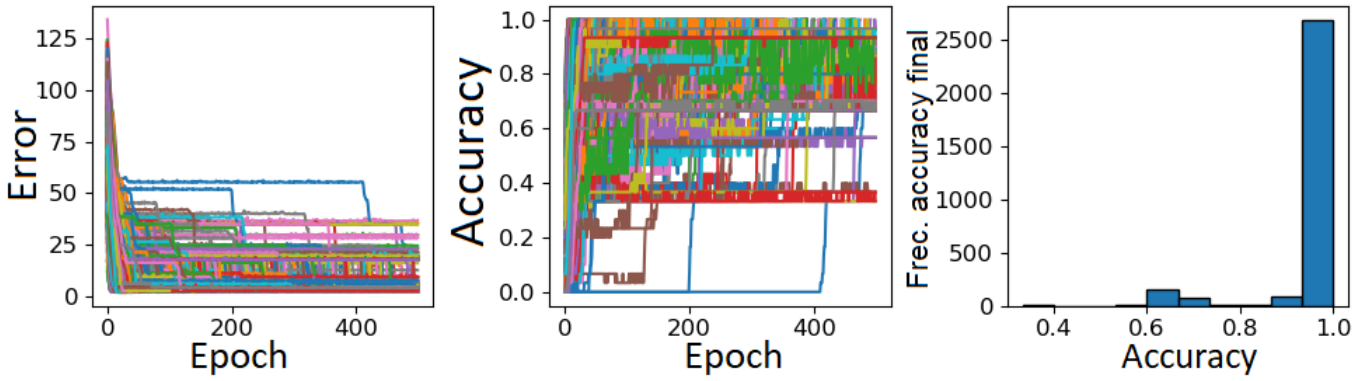

Figura 6.11: *Izquierda: Curvas de error vs. epochs. Centro: Curvas de accuracy vs. epochs. Para las curvas de error y accuracy se observa la presencia de algunas terrazas que no permiten la convergencia. Derecha: Histograma de frecuencia de cada valor de la accuracy en el último epoch. En este casó se utilizaron 2 mini-batches y suma de componentes aleatorias.*

6.5. Discusión

A partir de los resultados presentados en este capítulo, podemos concluir que el ruido en la convergencia del error se reduce al aumentar el número de puntos que forman las curvas de potenciación y depreciación, tanto para la implementación de *mini-batches* como para la suma de componentes aleatorias en las actualizaciones de pesos sinápticos. Se observó que al utilizar curvas de potenciación y depreciación lineales, el ruido generado al implementar los dos métodos mencionados previamente es comparable, pero al utilizar curvas de potenciación y depreciación no lineales, el ruido generado al implementar *mini-batches* es mayor que al implementar la suma de componentes aleatorias. Esto último explicaría las diferencias en el ruido que se observaron al utilizar las curvas de potenciación y depreciación experimentales utilizando *mini-batches* (Fig. 6.1) y la suma de componentes aleatorias (Fig. 6.2). Estas curvas cuentan con 175 pulsos en total (sumando pulsos de SET y RESET) y son no lineales, lo que favorece la generación de ruido de amplitud considerable al implementar *mini-batches*.

Otro resultado relevante es que la introducción de estocasticidad en la actualización de pesos acelera la convergencia del error; esto es, se reduce la cantidad de *epochs* necesaria para alcanzar su mínimo. Se obtiene una mayor velocidad de convergencia al combinar el uso de *mini-batches* con la suma de componentes aleatorios en las actualizaciones de los pesos. Eso es relevante dada la estocasticidad intrínseca que poseen los sistemas memristivos (reflejado, por ejemplo, en la variabilidad de los estados de alta y baja resistencia ante ciclados sucesivos). Por lo tanto, nuestros resultados sugieren que esta propiedad de los memristores no sólo no sería detrimental para su uso en redes neuronales implementadas por *hardware*, sino que podría incluso mejorar la prestación de estos sistemas si se hace una ingeniería adecuada de los dispositivos para optimizar la variabilidad de los cambios resistivos de cada memristor de la red [47].

Por último, se mostró que utilizando curvas de potenciación y depreciación sintéticas lineales y eligiendo un valor β que no sea el óptimo, la aparición de terrazas en las curvas de error y *accuracy* vs. *epochs* pueden reducirse, lo que confirma que la introducción de estocasticidad en el proceso de actualización de pesos sinápticos mejora la convergencia del algoritmo del proceso de aprendizaje.

Capítulo 7

Conclusiones

En esta tesis se simuló el proceso de aprendizaje de una red neuronal sencilla implementada por *hardware* a partir de arreglos de barras cruzadas de memristores, focalizándose en cómo influyen en la convergencia de la red las limitaciones físicas que poseen esos dispositivos. Se utilizó como punto de partida curvas de potenciación y depreciación sináptica medidas experimentalmente en tres sistemas memristivos basados en manganitas, con distintas bandas de conductividades asociadas. Se utilizó como función de activación la función $\tanh(\beta I)$, caracterizada por el parámetro β. Se encontró para cada sistema el valor óptimo β_o que minimiza el nivel al cual converge el error durante el proceso de entrenamiento y, a la vez, garantiza una convergencia suave sin que se aprecie la aparición de ruido en la evolución del error. Para los sistemas NSTO/LSMCO, LCMO/Ag y LCMO/Ti los β_o hallados son 10^2, 10^3 y 10^7 respectivamente. Se determinó de manera fenomenológica que la convergencia óptima se da para $G_{cen}\beta_o \approx 10^{-1}$, donde G_{cen} es el valor central de la banda de conductividades. Se encontró que conforme aumenta el parámetro β el error convergido estabiliza en valores menores. Por otra parte, para β mayor al valor óptimo β_0 se aprecia la aparición de ruido en la evolución del error. Se observó en todos los casos que la métrica utilizada (*accuracy*) converge a 1 en pocos *epochs*, lo que vinculamos al tamaño pequeño del conjunto de datos utilizado (30 imágenes correspondientes a 3 clases de caracteres). El error, en cambio, converge más lentamente a su mínimo y parece ser, por lo tanto, más indicado para caracterizar la robustez de la convergencia.

Al implementar diferentes realizaciones del proceso de convergencia para cada sistema, se encontró que el *epoch* de convergencia (definido por la estabilización del error cerca de su mínimo) sigue una distribución normal. Se utilizó el valor medio de esta distribución para comparar las velocidades de convergencia para distintas curvas de potenciación y depreciación sináptica. Se apreciaron diferencias en las velocidades de convergencia para los 3 sistemas experimentales utilizados.

Dado que las curvas de potenciación y depreciación difieren en más de un parámetro (bandas de conductividades, número de puntos de las curvas, separación entre puntos contiguos, linealidad, etc.) es difícil determinar cuales son los que determinan la velocidad de convergencia; para eludir esta dificultad, se decidió trabajar con curvas de potenciación y depreciación sintéticas, que permiten variar uno a uno y de manera controlada estos parámetros. De este estudio se determinó que al aumentar la cantidad de pulsos que forman las curvas de potenciación y depreciación sináptica la velocidad de convergencia del algoritmo se reduce. Sucede lo mismo al disminuir el rango de conductividades permitidas. Esto se debe a que en ambos casos la magnitud de las actualizaciones de los pesos se reduce, como consecuencia del método utilizado (regla de Manhattan) para actualizar los pesos sinápticos. Un resultado interesante se encontró al estudiar el comportamiento de las curvas de error vs *epochs* para valores $\beta > \beta_o$, dado que se observó que al utilizar curvas sintéticas lineales se generan terrazas en dichas curvas que están asociadas a la estabilización del error en mínimos locales. Para las curvas sintéticas no lineales este efecto no es tan apreciable. Por último, se encontró que la velocidad de convergencia es mayor para curvas de potenciación y depreciación no lineales, lo cuál es un punto controversial en la literatura existente a la fecha [4, 42, 43].

Como una alternativa para conseguir mejoras en la velocidad de convergencia que no impliquen la manipulación de aspectos físicos de los dispositivos memristivos, como lo es variar las curvas

de potenciación y depreciación sinápticas, se implementaron métodos que introducen estocasticidad en el cálculo de las actualizaciones de los pesos sinápticos. Para esto se propusieron dos métodos: *mini-batches* y la suma de componentes aleatorias en las actualizaciones de los pesos sinápticos. Con ambos métodos fue posible aumentar la velocidad de convergencia del algoritmo al utilizar curvas de potenciación y depreciación experimentales, pero al mismo tiempo, estos métodos generan ruido en las curvas de error vs. *epochs*. Se observó que al usar curvas de potenciación y depreciación sintéticas lineales, para ambos métodos, el ruido es comparable, pero al utilizar curvas sintéticas no lineales el ruido al implementar *mini-batches* es mayor que al implementar la suma de componentes aleatorias. Fue posible combinar ambos métodos de manera de obtener un aumento en la velocidad de convergencia con respecto a cada método individualmente.

Utilizando las curvas de potenciación y depreciación sintéticas, y comparando con el caso en el que no se utiliza ningún método estocástico, al implementar por separado *mini-batches* y la suma de componentes aleatorias se observa, en ambos casos, una reducción de $\approx 45\,\%$ en la cantidad de *epochs* necesarios para lograr la convergencia. Por otra parte, al combinar los dos métodos se consigue una reducción en la cantidad de *epochs* necesarios de $\approx 71\,\%$. Además, con estos métodos fue posible mejorar la convergencia al utilizar curvas de potenciación y depreciación sintéticas lineales y funciones de activación no óptimas con $\beta > \beta_o$, de manera que la inclusión de estocasticidad parece evitar la estabilización del error en mínimos locales (terrazas en la evolución del error vs. número de *epochs*). Dada la estocasticidad intrínseca que presenta la respuesta eléctrica de sistemas memristivos, la cual origina por ejemplo variaciones aleatorias en los estados resistivos ante ciclados sucesivos, las mejoras en la convergencia observadas al implementar métodos estocásticos se vuelven relevantes [48]. Esta característica de los memristores podría utilizarse para mejorar el funcionamiento de redes neuronales implementadas por *hardware* basadas en estos dispositivos.

Este trabajo constituye un primer acercamiento al tema de estudio, el cual prevemos continuar desarrollando durante el transcurso de mi tesis doctoral. El principal interrogante en estos momentos es si las estrategias de aceleración de la convergencia mediante la introducción de estocasticidad - verificadas en la red simple simulada aquí- se extienden a redes neuronales más complejas, utilizando conjuntos de datos más grandes. Concretamente, se propone como trabajo futuro:

- Implementación del algoritmo de aprendizaje utilizando otras funciones de costo. Por ejemplo, la *cross-entropy* multiclase [49, 50] podría ser más adecuada para problemas de clasificación como el reconocimiento de imágenes. Por otra parte, se estudiará cómo afectan a la convergencia y *accuracy* el uso de otras funciones de activación.

- Estudiar otros métodos para determinar el incremento de los pesos sinápticos a partir de las curvas de potenciación y depreciación experimentales, ya que en este trabajo solo se consideró el ajuste de las curvas ΔG^{exp} vs. G^{exp}. Un método alternativo puede ser el descripto en la Ref. [42], donde implementan una función logística para describir la evolución de la conductancia con el número de pulsos, tanto para la potenciación como la depreciación.

- Implementación de curvas de potenciación y depreciación sintéticas no lineales con distinto grado de no-linealidad y diferente concavidad, dado que el efecto de estas características en el desempeño de la red no está completamente determinado.

- Escalar el tamaño de la red para procesar el conjunto de datos de dígitos manuscritos del MNIST, el cual incluye decenas de miles de imágenes y se considera un estándar en el procesamiento de imágenes mediante métodos de ML. Adicionalmente, el uso de este conjunto de datos permitirá la separación de los datos en conjuntos de entrenamiento y testeo, la cual es una práctica usual de validación de algoritmos de ML.

- Agregar capas profundas a la red, lo cual en redes neuronales tradicionales redunda en una mejora en sus prestaciones (por ejemplo, incremento de la *accuracy*). Esto implica la modificación del algoritmo de *back-propagation* utilizado.

- Implementación, en sistemas reales de barras cruzadas de memristores, de las estrategias de optimización obtenidas mediante simulaciones. Para tal fin se propone utilizar memristores basados en TaO_x, los cuales, debido a sus prestaciones [51], se posicionan como uno de los sistemas más prometedores para implementaciones de cómputo neuromórfico.

Agradecimientos

A Diego Rubi, Majo Sánchez y los compañeros del grupo de Ablación láser por su dirección y ayuda durante este trabajo.

A Gustavo Lado y Enrique Segura por sus sugerencias e interés en esta tesis.

A mis amigos y compañeros de la facultad que siempre estuvieron para dar una mano o simplemente para hablar boludeces.

A Sabri, gracias por aguantarme todo este tiempo y por ser una maniática de las faltas de ortografía.

A mi familia, Babi, Miriam, Leo, Daniel y Agustina. A mis tíos Reina y Laly, y a mis primos.

A mi papá, Joaquin, nada esto hubiese sido posible sin tu apoyo. A pesar de que nunca entendías una goma cuando te contaba sobre de mi carrera o las cosas que hacia, siempre estabas dispuesto a escuchar y hacerme entender que esto era importante para vos también. Simplemente gracias por todo.

Bibliografía

[1] A. Geron, *Hands-On Machine Larning with Scikit-Learn, Keras and TensorFlow* (O'Reilly Media, Inc., 2019).

[2] E. Strubell, A. Ganesh y A. McCallum, en Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics (jun. de 2019), págs. 3645-3650.

[3] P. A. Merolla, J. V. Arthur, R. Alvarez-Icaza, A. S. Cassidy, J. Sawada, F. Akopyan, B. L. Jackson, N. Imam, C. Guo, Y. Nakamura, B. Brezzo, I. Vo, S. K. Esser, R. Appuswamy, B. Taba, A. Amir, M. D. Flickner, W. P. Risk, R. Manohar y D. S. Modha, Science **345**, 668 (2014).

[4] S. Yu, *Neuro-inspired Computing Using Resistive Synaptic Devices* (Springer, Cham, 2017).

[5] A. Sawa, Mater. Today **11**, 28 (2008).

[6] R. Berdan, E. Vasilaki, A. Khiat, G. Indiveri, A. Serb y T. Prodromakis, Sci. Rep. **6** (2016).

[7] Z. Wang, S. Joshi, S. E. Savel'ev, H. Jiang, R. Midya, P. Lin, M. Hu, N. Ge, J. P. Strachan, Z. Li, Q. Wu, M. Barnell, G.-L. Li, H. L. Xin, R. S. Williams, Q. Xia y J. J. Yang, Nat. Mater. **16** (2017).

[8] M. Prezioso, F. Merrikh-Bayat, B. D. Hoskins, G. C. Adam, K. K. Likharev y D. B. Strukov, Nature **521** (2015).

[9] Q. V. Le, R. Monga, M. Devin, G. Corrado, K. Chen, M. Ranzato, J. Dean y A. Y. Ng, CoRR **abs/1112.6209** (2011).

[10] K. J. Kuhn, IEEE Trans. Electron Devices **59**, 1813 (2012).

[11] S. A. McKee, en Proceedings of the 1st Conference on Computing Frontiers, CF '04 (2004), pág. 162.

[12] I. C. Education, *What is Machine Learning?*, jul. de 2020.

[13] J. Hertz, A. Krogh y R. G. Palmer, *Introduction to the theory of neural computation* (CRC Press, 1991).

[14] D. Kuzum, S. Yu y H.-S. P. Wong, Nanotechnology **24**, 382001 (2013).

[15] H. D. Block, Rev. Mod. Phys. **34**, 123 (1962).

[16] H. D. Block, B. W. Knight y F. Rosenblatt, Rev. Mod. Phys. **34**, 135 (1962).

[17] F. Rosenblatt, *Principles of neurodynamics. perceptrons and the theory of brain mechanisms*, inf. téc. (Cornell Aeronautical Lab Inc Buffalo NY, 1961).

[18] Y. Bengio, *Practical recommendations for gradient-based training of deep architectures*, 2012.

[19] L. Bottou, F. E. Curtis y J. Nocedal, SIAM Rev. **60**, 223 (2018).

[20] L. Chua, IEEE Transactions on Circuit Theory **18**, 507 (1971).

[21] L. Chua y S. M. Kang, Proceedings of the IEEE **64**, 209 (1976).

[22] A. Prakash, D. Jana y S. Maikap, Nanoscale Res. Lett. **8**, 28 (2013).

[23] R. Waser, R. Dittmann, G. Staikov y K. Szot, Adv. Mater. **21**, 2632 (2009).

[24] C. Ferreyra, M. J. Sánchez, M. Aguirre, C. Acha, S. Bengió, J. Lecourt, U. Lüders y D. Rubi, **31**, 155204 (2020).

[25] K. Seo, I. Kim, S. Jung, M. Jo, S. Park, J. Park, J. Shin, K. P. Biju, J. Kong, K. Lee, B. Lee y H. Hwang, **22**, 254023 (2011).

[26] Y. Kim, W. H. Jeong, S. B. Tran, H. C. Woo, J. Kim, C. S. Hwang, K.-S. Min y B. J. Choi, AIP Adv. **9**, 045131 (2019).

[27] R. Yang, K. Terabe, G. Liu, T. Tsuruoka, T. Hasegawa, J. Gimzewski y M. Aono, English, ACS Nano **6**, 9515 (2012).

[28] N. Ghenzi, M. J. Sánchez, M. J. Rozenberg, P. Stoliar, F. G. Marlasca, D. Rubi y P. Levy, J. Appl. Phys. **111**, 084512 (2012).

[29] F. Alibart, E. Zamanidoost y D. B. Strukov, Nat. Commun. **4**, 2072 (2013).

[30] G. Sassine, S. La Barbera, N. Najjari, M. Minvielle, C. Dubourdieu y F. Alibart, Journal of Vacuum Science & Technology B **34**, 012202 (2016).

[31] C. Ferreyra, "Fabricación, caracterización y modelado de films delgados con propiedades memristivas" (Departamento de Física. Facultad de Ciencias Exactas y Naturales. Universidad de Buenos Aires, 2021).

[32] W. R. Acevedo, C. Ferreyra, M. J. Sánchez, C. Acha, R. Gay y D. Rubi, J. Phys. D: Appl. Phys. **51**, 125304 (2018).

[33] C. Ferreyra, W. R. Acevedo, R. Gay, D. Rubi y M. J. Sánchez, J. Phys. D: Appl. Phys. **53**, 015302 (2019).

[34] E. Miranda, W. Román Acevedo, D. Rubi, U. Lüders, P. Granell, J. Suñé y P. Levy, J. Appl. Phys. **121**, 205302 (2017).

[35] W. Román Acevedo, C. A. M. van den Bosch, M. H. Aguirre, C. Acha, A. Cavallaro, C. Ferreyra, M. J. Sánchez, L. Patrone, A. Aguadero y D. Rubi, Appl. Phys. Lett. **116**, 063502 (2020).

[36] J. E. Crowell, J. Vac. Sci. Technol. A **21**, S88 (2003).

[37] J. Chen, C.-Y. Lin, Y. Li, C. Qin, K. Lu, J.-M. Wang, C.-K. Chen, Y.-H. He, T.-C. Chang, S. M. Sze y X.-S. Miao, IEEE Electron Device Lett. **40**, 542 (2019).

[38] G. Zhong, M. Zi, C. Ren, Q. Xiao, M. Tang, L. Wei, F. An, S. Xie, J. Wang, X. Zhong, M. Huang y J. Li, Appl. Phys. Lett. **117**, 092903 (2020).

[39] X. Mou, J. Tang, Y. Lyu, Q. Zhang, S. Yang, F. Xu, W. Liu, M. Xu, Y. Zhou, W. Sun, Y. Zhong, B. Gao, P. Yu, H. Qian y H. Wu, Sci. Adv. **7**, eabh0648 (2021).

[40] D. Roclin, O. Bichler, C. Gamrat y J.-O. Klein, en Proceedings of the 2014 IEEE/ACM International Symposium on Nanoscale Architectures, NANOARCH '14 (2014), págs. 13-18.

[41] E. Zamanidoost, F. M. Bayat, D. Strukov e I. Kataeva, en 2015 IEEE 9th International Symposium on Intelligent Signal Processing (WISP) Proceedings (2015), págs. 1-6.

[42] A. Gutsche, S. Siegel, J. Zhang, S. Hambsch y R. Dittmann, Front. Neurosci. **15**, 723 (2021).

[43] Y. Pyo, J.-U. Woo, H.-G. Hwang, S. Nahm y J. Jeong, Nanomaterials **11**, `10.3390/nano11102684` (2021).

[44] G. Lado y E. Segura, International Journal on Soft Computing, Artificial Intelligence and Applications **8**, 37 (2019).

[45] D. Kafka y D. Wilke, *Traversing the noise of dynamic mini-batch sub-sampled loss functions: A visual guide*, 2020.

[46] U. Simsekli, L. Sagun y M. Gurbuzbalaban, *A Tail-Index Analysis of Stochastic Gradient Noise in Deep Neural Networks*, 2019.

[47] V. Joshi, M. Le Gallo, S. Haefeli, I. Boybat, S. R. Nandakumar, C. Piveteau, M. Dazzi, B. Rajendran, A. Sebastian y E. Eleftheriou, Nat. Commun. **11**, 1 (2020).

[48] G. S. Snider, Nanotechnology **18**, 365202 (2007).

[49] C. M. Bishop, *Pattern Recognition and Machine Learning (Information Science and Statistics)* (Springer Science + Business Media, LLC, 2006).

[50] P. Y. Simard, D. Steinkraus, J. C. Platt y col., en Icdar, vol. 3, 2003 (2003).

[51] J. J. Yang, M.-X. Zhang, J. P. Strachan, F. Miao, M. D. Pickett, R. D. Kelley, G. Medeiros-Ribeiro y R. S. Williams, Appl. Phys. Lett. **97**, 232102 (2010).