Exploración de spiking neural networks para la generación de carácteres manuscritos

Aksel Serret Llopis

Resumen—Este trabajo explora el uso de *spiking neural networks* (SNN) para generar trazos manuscritos de caracteres, poniendo especial énfasis tanto en el aprendizaje de múltiples letras como en la capacidad de producir variaciones estilísticas dentro del mismo carácter. Se desarrollan experimentos con un escalado progresivo del número de carácteres (de 1 a 26), complementados con técnicas de entrenamiento por batches, suavizado de trayectorias e interpolación de estilos, especialmente itálicas. Además, se investiga cómo las SNN pueden aprender variaciones del mismo carácter generando trazos consistentes a partir de spiketrains diferentes. El uso del algoritmo e-prop hace posible un aprendizaje eficiente sin recurrir a Back propagation through time, y las SNN demuestran una buena capacidad de generalización incluso con pocos datos. A pesar de que el error tiende a aumentar al introducir más caracteres o estilos, el modelo es capaz de escalar, generar variaciones, interpolaciones, concatenado de caracteres y Aprender un concepto abstracto y generalizarlo.

Palabras clave. Spiking neural network, spikes, procesamiento temporal, eficiencia energética, generación manuscrita, variabilidad, interpolación, estilo itálico, e-prop, trazas de elegibilidad, NEST, PyNEST.

Abstract— This work explores the use of Spiking Neural Networks (SNNs) to generate handwritten character strokes, with a particular emphasis on learning multiple letters and the ability to produce stylistic variations within the same character. Experiments are conducted with a progressive scaling of the number of characters (from 1 to 26), complemented by training techniques such as batching, trajectory smoothing, and style interpolation—especially italics. Additionally, the study investigates how SNNs can learn variations of the same character by generating consistent strokes from different spike trains. The use of the e-prop algorithm enables efficient learning without relying on Backpropagation Through Time, and the SNNs show strong generalization capabilities even with limited data. Although the error tends to increase with the introduction of more characters or styles, the model is able to scale, generate variations, perform interpolations, concatenate characters, and learn abstract concepts for generalization.

Index Terms—. Spiking neural networks, spikes, temporal processing, energy efficiency, handwritten generation, variability, interpolation, italic style, e-prop, eligibility traces, NEST, PyNEST.

1 Introducción

La Inteligencia Artificial ha supuesto un cambio muy importante en el mundo de la ingeniería. La capacidad de estas arquitecturas ha revolucionado muchos procesos. Sin embargo, tienen un gran cuello de botella: el consumo energético necesario para su entrenamiento. Además, las redes tradicionales de *deep learning* presentan otro problema importante relacionado con los datos, ya que necesitan una gran cantidad de ellos para poder entrenarse correctamente y alcanzar los objetivos esperados como es el caso de las GAN [12]. Por ello, se están explorando nuevas alternativas dentro de la inteligencia artificial, como por ejemplo las *Spiking Neural Networks* (SNN). Estas redes representan una evolución en el diseño de redes neuronales artificiales, ya que se inspiran directamente en el funcionamiento biológico del cerebro humano. A diferencia de las redes tradicionales, que procesan la información mediante valores continuos, las SNN transmiten señales a través de impulsos eléctricos discretos llamados *spikes*, reproduciendo así de forma más realista la dinámica neuronal [1].

El creciente interés por estas redes se debe a su capacidad para manejar información temporal de forma eficiente y su potencial para reducir el consumo energético en comparación con los modelos clásicos. Estas características las convierten en una opción prometedora para tareas que requieren procesamiento temporal, como la clasificación de señales, el reconocimiento de patrones secuenciales o, como se plantea en este trabajo, la generación de caracteres manuscritos [1].

La motivación de este trabajo es evaluar en profundidad las capacidades de las *spiking neural networks* (SNN) para la generación de caracteres manuscritos, articulando tres objetivos principales:

1. Aprendizaje de caracteres individuales: entrenar la SNN para reproducir trazos específicos de diversas letras utilizando una sola red, analizando la precisión mediante

métricas tanto cualitativas como cuantitativas.

2. Generación de variaciones e interpolación de estilos: entrenar la red con múltiples versiones de un mismo carácter para evaluar su capacidad de producir variantes coherentes y nunca vistas. Además, se examinará su habilidad para interpolar gradualmente entre estilos diferentes, generando transiciones estilísticas suaves.

3. Aprendizaje de un estilo abstracto: Se intentará probar que las SNN son capaces de generalizar el estilo itálico a letras nunca vistas para poder generarlas en este estilo.

2 Planificación

La planificación del proyecto se estructura en varias fases que aseguran un desarrollo progresivo y organizado a lo largo del periodo de tiempo establecido. En primer lugar, se abordará una etapa de documentación técnica y biológica, donde se estudiarán los fundamentos neurocientíficos de las SNN y cómo estas simulan el comportamiento sináptico. Esta fase incluye también la familiarización con el entorno de simulación NEST y la ejecución de tutoriales para entender el funcionamiento de las redes, así como la reproducción del experimento original de generación de texto manuscrito con la palabra "chaos".

Una vez dominado el entorno, se procederá a realizar pruebas sobre el código original modificando distintos hiperparámetros para analizar su impacto en el rendimiento de la red. Paralelamente, se desarrollará un módulo que permita convertir texto manuscrito en coordenadas de trayectorias, necesarias para entrenar la SNN. Estas trayectorias serán transformadas en pulsos eléctricos (spikes) para que las neuronas aprendan el patrón espacial y temporal del trazo manuscrito.

Posteriormente, se modificará el código base para permitir que la misma red aprenda y genere más de un patrón, generación de variaciones y tranferencia de estilo. Finalmente, se analizarán los resultados mediante métricas de calidad de generación como el error cuadrático medio (MSE), valorando el rendimiento del sistema implementado.

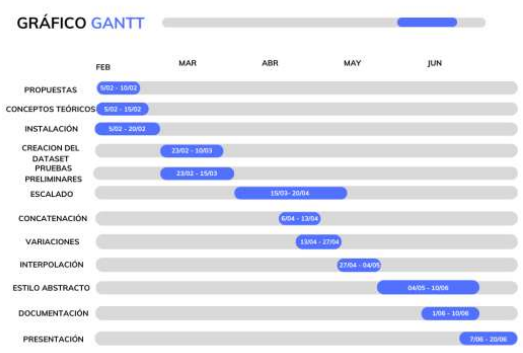


Figura 1: Gráfico gantt

3 Estado del arte y background

3.1 El concepto de spike

Las SNN se diferencian de las redes neuronales de *deep learning* tradicionales principalmente por la forma en la que procesan la información. Mientras que las redes clásicas de *Deep learning* utilizan valores continuos en sus activaciones, las SNN utilizan eventos discretos en el tiempo conocidos como *spikes*.

Un *spike* simula la activación de una neurona biológica. En el cerebro humano, una neurona permanece inactiva hasta que recibe suficientes estímulos acumulados. Cuando estos estímulos superan un cierto umbral de activación, la neurona emite un impulso eléctrico: un *spike*. Posteriormente, la neurona entra en un breve periodo de reposo antes de poder activarse nuevamente.

Estas activaciones se concatenan a lo largo del tiempo creando *spiketrains*, los cuales representan las activaciones que ha tenido una neurona en un periodo de tiempo definido [2].

En las SNN, este comportamiento se modela mediante neuronas artificiales que acumulan señal a lo largo del tiempo. Si la entrada acumulada excede el umbral definido, se produce un *spike*, codificado como un 1 en el *spike train*; si no, el valor permanece en 0 [2].

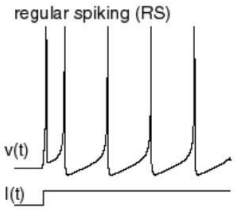


Figura 2 Spiketrain

3.2 Arquitectura de una SNN

La red neuronal utilizada en este proyecto se basa en una arquitectura típica de redes recurrentes espaciotempora-les, concretamente por las LSM (Liquit State Machine) [11], las cuales están compuestas por tres tipos principales de neuronas: neuronas de entrada, neuronas recurrentes y neuronas de salida. Cada uno de estos grupos cumple un rol especfico durante el procesamiento de la información y el aprendizaje.

Neuronas de entrada:

Las neuronas de entrada se encargan de recibir las señales codificadas como spiketrain. En este caso, estas señales representan las etiquetas del carácter a aprender. Se trata de secuencias de activaciones binarias (*spikes*) que varían en el tiempo. Las neuronas de entrada simplemente propagan estos *spikes* al resto de la red sin procesarlos, funcionando como un canal directo entre los datos y el sistema neuronal interno [9].

Neuronas recurrentes:

El núcleo de la red está formado por las neuronas recurrentes, que introducen una dinámica temporal interna al sistema. Estas neuronas están conectadas entre sí formando una red recurrente, lo que significa que pueden influirse mutuamente a través del tiempo [10]. Este tipo de conexión permite formar bucles internos, es decir, una neurona puede estar conectada a otra que a su vez vuelva a activarla en el futuro. Gracias a esta retroalimentación, la red puede almacenar memoria temporal, lo que resulta esencial para tareas como la generación de trayectorias, donde la información de un instante anterior influye en los pasos siguientes [11].

Las neuronas recurrentes acumulan los spikes recibidos tanto desde las neuronas de entrada como desde otras recurrentes. Cuando la señal acumulada supera el umbral de activación, se dispara un spike. Esta activación afecta a otras neuronas en pasos posteriores, generando una dinámica compleja y rica que imita las redes neuronales biológicas [10].

Neuronas de salida:

Finalmente, las neuronas de salida transforman la actividad de la red en señales que podemos interpretar, como las coordenadas derivadas en el caso de la escritura manuscrita. A diferencia de las neuronas anteriores, que emiten spikes binarios (0 o 1), estas neuronas generan una activación continua, es decir, valores numéricos que cambian con el tiempo y que se interpretan directamente como la salida.

Las coordenadas derivadas no representan posiciones absolutas, sino los pequeños desplazamientos en cada momento del tiempo [9]. Luego, sumando todos esos pequeños movimientos a lo largo del tiempo, se puede reconstruir la trayectoria completa de la letra escrita. Esta forma de representar el trazo permite que la red aprenda patrones de movimiento más suaves y naturales [4].

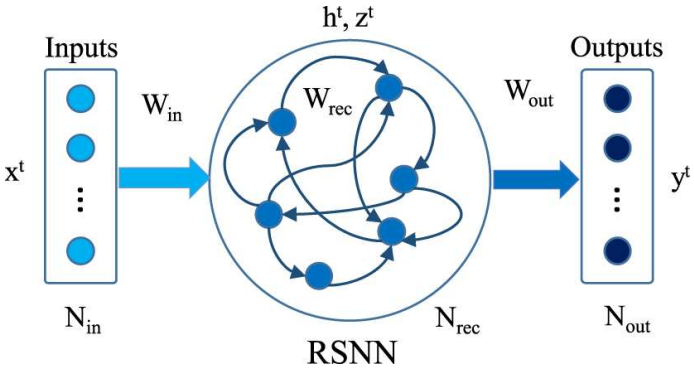


Figura 3 Arquitectura red SNN

3.3 e-prop

El algoritmo e-prop, abreviatura de eligibility propagation, constituye un método de aprendizaje diseñado específicamente para las SNN. Su funcionamiento se inspira en los mecanismos biológicos de aprendizaje que se observan en el cerebro, y presenta una alternativa más eficiente al tradicional algoritmo de Backpropagation Through Time (BPTT), especialmente en entornos donde las redes operan con señales temporales, como es el caso de los spikes [3].

En lugar de propagar el error hacia atrás a lo largo del tiempo completo, como ocurre en BPTT, e-prop emplea una estrategia más local. La clave reside en el uso de lo que se denomina trazas de elegibilidad, que son registros temporales de la actividad sináptica entre neuronas. Estas trazas indican si una neurona de entrada estuvo activa justo antes de que otra neurona se activara, simulando así el principio biológico por el cual una sinapsis puede fortalecerse si se produce una activación previa consistente [3].

El aprendizaje se produce mediante la combinación de estas trazas con una señal de error local, que refleja si la salida de la red se ha desviado de la esperada. Si la red comete un error al generar una trayectoria, las conexiones implicadas en dicho fallo reciben un ajuste negativo, debilitando la sinapsis. En cambio, si la red produce una salida más cercana a la deseada, las conexiones que han contribuido a ello se refuerzan. Este mecanismo de refuerzo o debilitamiento progresivo de las conexiones permite que la red aprenda de manera continua y eficiente, sin necesidad de almacenar todos los estados previos [3].

4 METODOLOGÍA

4.1 Uso del entorno NEST y herramientas en Python

Para llevar a cabo este proyecto se ha utilizado el entorno de simulación NEST, especialmente diseñado para modelar redes neuronales de tipo biológico, como las SNN. NEST [7] permite simular redes a gran escala con miles de neuronas y sinapsis, representando de forma precisa aspectos como el potencial de membrana, el umbral de disparo y la propagación de spikes a través de conexiones sinápticas.

La programación y configuración de las redes se ha realizado mediante PyNEST, la interfaz en Python que permite controlar completamente las simulaciones dentro de NEST [7]. Gracias a esta herramienta, se ha definido la arquitectura neuronal, gestionado el flujo de entrenamiento, procesado los spikes y visualizado los resultados.

Para complementar el desarrollo, se han empleado bibliotecas de Python como:

- NumPy, para realizar cálculos numéricos y manipular vectores y matrices [5].
- Matplotlib, para representar gráficamente los trazos y trayectorias generadas por las redes entrenadas [6].

Dado que actualmente NEST no cuenta con una versión estable con soporte para GPU, las simulaciones se han ejecutado utilizando supercomputadores basados en CPU. En concreto, se ha aprovechado el soporte para multithreading que ofrece NEST, lo cual permite paralelizar los cálculos y reducir significativamente los tiempos de entrenamiento. No se ha utilizado MPI, ya que en este proyecto se ha

trabajado con la versión multihilo de NEST que ya está optimizada para CPUs multinúcleo [5].

5 Desarrollo y resultados

5.1 Creación de data set

Para comenzar este proyecto, se analizó la estructura de las SNN aplicadas a la generación de texto manuscrito. Para poder entrenar una SNN, es necesario disponer de una señal objetivo que represente la trayectoria que debe aprender la red. Esta señal consiste en las derivadas de los puntos que componen el trazo de un carácter, es decir, la dirección del movimiento en cada instante.

Con el objetivo de generar este conjunto de datos de forma personalizada, se diseñó un pequeño programa utilizando las librerías *matplotlib.pyplot* [5] y numpy [4]. Este código permite al usuario dibujar manualmente los puntos de un trazo sobre una interfaz gráfica interactiva. Estos puntos son almacenados en una lista en el orden en que se dibujan.

Una vez capturados los puntos, se calculan sus derivadas para obtener la dirección del movimiento en cada paso y se genera así la señal que se usará como objetivo durante el entrenamiento. Este conjunto de derivadas se guarda en un archivo .txt, que será cargado posteriormente para entrenar la SNN.

A continuación, se muestra un ejemplo de varias letras del abecedario obtenidas con este sistema de captura de trazos:

Figura 4 Dataset de muestra.

5.2 Aprendizaje de más de un caràcter

En esta parte del proyecto, se partió del código base *NEST handwriting* [4], el cual permite generar la palabra "*Chaos*" aprendiendo únicamente una trayectoria. Sin embargo, este enfoque presenta una limitación importante: solo permite entrenar la red para reproducir un único carácter, por lo que no es escalable para tareas más complejas [4].

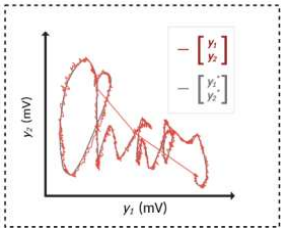


Figura 5 palabra chaos.

El objetivo fue, por tanto, adaptar dicho código para permitir que una misma red SNN pueda aprender y reproducir varios caracteres o trazos distintos. Para ello, se analizó primero el funcionamiento de estas redes.

Las SNN reciben como entrada *Spiketrains* (secuencias de activaciones neuronales distribuidas a lo largo del tiempo), y generan una salida que debe aproximarse a una señal objetivo, en este caso, la trayectoria de un carácter manuscrito.

Para empezar el entrenamiento se decidió realizar un estudio preliminar para observar si la red es capaz de aprender otro carácter diferente con puntos diferentes generado a partir del código de creación de dataset explicado anteriormente. Para ello se creo un *spiketrain* nuevo de forma aleatoria como se crea en el ejemplo de *"chaos"*. Posteriormente se entrenó la red con 20 neuronas de entrada y 5000 iteraciones igual que en el ejemplo anterior. A continuación, se muestran los resultados del aprendizaje del carácter "a".

MSE Promedio = 0.000189

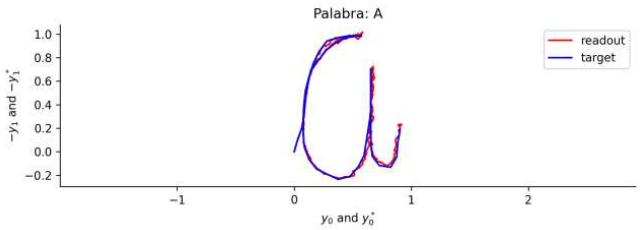


Figura 6 Aprendizaje del carácer *"a"*

En la figura numero 6 podemos observar el carácter generado por la red (trazo generado de color rojo) junto a su función objetivo (trazo generado de color azul). Como podemos observar, el trazo generado por nuestra red se ajusta con precisión al trazo objetivo el cual, ha sido utilizado en el entrenamiento de la red, por lo que podemos concluir que la arquitectura propuesta es capaz de aprender un carácter nuevo.

5.4 Estudio de la arquitectura

En esta sección se realizó un estudio preliminar de los hiperparámetros, partiendo del enfoque del tutorial que empleaba 20 neuronas de entrada, 200 neuronas recurrentes y 5000 iteraciones para trayectorias de 200 puntos, una configuración que resultaba poco óptima para nuestro caso ya que el objeitvo final era aprender 26 trazos distintos con el menor numero de iteraciones posible. Para mejorarla, se redujo el número de puntos a 50, usando la letra "a" como referencia, y se mantuvieron 1000 iteraciones por entrenamiento. Además, para que la red fuese capaz de aprender todos los caracteres se aumentaron a 500 neuronas recurrentes para que la misma tuviese la suficiente memoria para soportar todos los trazos.

A continuación, se fue incrementando el número de neuronas de entrada para evaluar su impacto en el rendimiento. La letra "a", codificada con una trayectoria de 50 puntos, sirvió como caso de estudio para comparar los

resultados obtenidos con distintos tamaños de entrada.

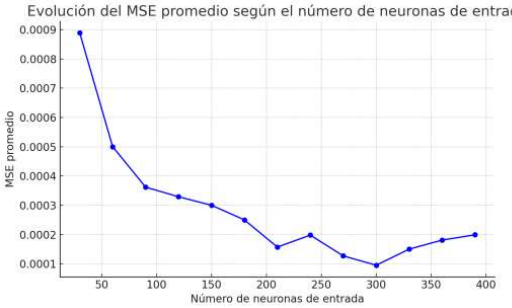


Figura 7 Evolución MSE por Neuronas de entrada

A partir del gráfico que muestra la evolución del MSE medio durante el entrenamiento de la letra "a", se observa una reducción significativa del error al aumentar el número de neuronas de entrada, especialmente hasta alcanzar aproximadamente las 300. Este comportamiento sugiere que una mayor resolución en la codificación de la entrada facilita el aprendizaje de la red.

No obstante, a partir de ese punto, el error comienza a incrementarse ligeramente. Esto se debe a que un número excesivo de neuronas de entrada genera una sobrecarga de *spiketrains*, introduciendo ruido en el sistema, en lugar de aportar más información útil dificultando así el entrenamiento.

5.5.1 Escalado a más de un carácter

En esta sección, el objetivo principal es explicar cómo se amplía el aprendizaje de una SNN para que reconozca múltiples caracteres. Se plantearon varias hipótesis sobre la manera en que el modelo podría manejar diferentes trayectorias, hasta que se decidió adoptar la concatenación de *spiketrains* diferentes en cada una de las iteraciones a través del tiempo

En cada iteración, se presenta a la red un *spiketrain* completo correspondiente a la trayectoria de un carácter específico para posteriormente comparar la salida generada con su trayectoria objetivo correspondiente. Al alternar estos *spiketrains* y actualizar la función objetivo acorde al carácter que representa cada uno, la red va aprendiendo progresivamente distintos caracteres, avanzando hasta abarcar todo el abecedario.

Después de hacer algunas pruebas iniciales con la red, vimos que necesitábamos muchas iteraciones si actualizamos los pesos en cada iteración para que pudiera aprender correctamente las diferentes trayectorias. Esto hacía que el proceso de entrenamiento fuera lento y menos eficiente. Por eso, decidimos cambiar a un enfoque basado en *batches*.

Este enfoque consiste en agrupar varias letras y actualizar los pesos de la red solo después de haber procesado todas las trayectorias de ese grupo, en lugar de hacerlo después de cada letra individualmente. Es decir, la red observa varias letras seguidas y luego realiza una única actualización de pesos teniendo en cuenta toda esa información. Este cambio trajo varias ventajas importantes:

- Mayor estabilidad en el aprendizaje: al procesar varias letras antes de ajustar los pesos, las actualizaciones se volvieron menos bruscas y más consistentes.
- Mejor capacidad de generalización: como la red veía diferentes letras antes de cada actualización, empezó a captar patrones comunes entre ellas, así como las diferencias características de cada una. Esto ayudó a que pudiera reproducirlas de forma más precisa.
- Más flexibilidad: al poder elegir el tamaño del batch, fue más fácil adaptar el entrenamiento según el número de caracteres disponibles o los recursos del equipo, consiguiendo un buen equilibrio entre rendimiento y eficiencia.

En cuanto a la arquitectura se optó por utilizar 500 neuronas recurrentes y 300 neuronas de entrada para escalar el modelo de forma optima. A continuación, se muestran los resultados de los entrenamientos:

5.5.2 Escalado: 2 carácteres

En este primer experimento se entrenó la SNN con únicamente dos letras: A y B, utilizando una configuración de 300 neuronas de entrada, 500 neuronas recurrentes y 1000 iteraciones por letra y un batch de 2.

Los resultados fueron los siguientes:

MSE PROMEDIO:

- Letra A: **0,000184**
- Letra B: **0,000219**

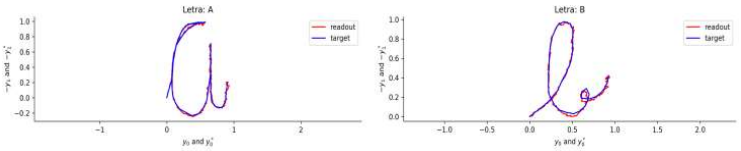


Figura 9 Caracteres "a" y "b" aprendidos por SNN.

Como se puede observar en la imagen vemos como la señal generada y aprendida por la red (de color rojo) se ha ajustado adecuadamente a la trayectoria objetivo de ambos caracteres en color azul, con un MSE muy bajo. A continuación, se muestra la LOSS curve la cual nos da información de como ha evolucionado la perdida a través de las diferentes iteraciones.

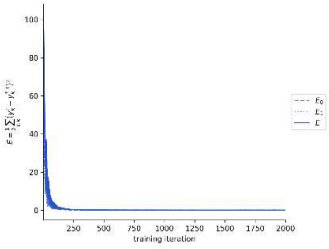


Figura 10 Curva Loss 2 caracteres.

En la imagen se observa una rápida disminución del error en las primeras iteraciones, alcanzando valores muy bajos hacia el final del entrenamiento. Esto indica que la

red aprende eficazmente a reproducir las trayectorias de los caracteres con gran precisión.

5.5.3 Escalado: 4 carácteres

En este experimento se entrenó la red con cuatro letras: "a", "b", "c" y "d", utilizando 300 neuronas de entrada, 500 neuronas recurrentes, y 1000 iteraciones por letra. El error cuadrático medio (MSE) obtenido para cada carácter fue el siguiente:

MSE PROMEDIO:

- Letra "a": **0.000785**
- Letra "b": **0.000457**
- Letra "c": **0.000722**
- Letra "d": **0.001278**

Observando los resultados nos damos cuenta de que el MSE aumenta ya que tiene que aprender más caracteres en cada entrenamiento.

5.5.4 Escalado: 8 carácteres

En este experimento, la red se entrenó con ocho letras —de la "a" a la "h"— utilizando 300 neuronas de entrada, 500 neuronas recurrentes y 1250 iteraciones por letra. En esta sección al observar que el error aumentaba se decidión aumentar el numero de iteraciones. En cuanto al MSE, se ha podido observar que ha obtenido valores diversos como: 0,000522 (letra "a") hasta 0,001012 (letra "d"), con una media global reportada de 0,001078.

Al comparar estos resultados con los obtenidos en experimentos anteriores con solo cuatro letras (donde el error medio fue de alrededor de 0,000811), se observa un aumento progresivo en el MSE. Esto confirma que, a medida que se añaden más ejemplos/caracteres al entrenamiento, el error general tiende a crecer, lo cual es coherente con el incremento de la complejidad del problema.

5.5.5 Escalado: 16 carácteres

En este nuevo experimento se entrenaron un total de 16 caracteres, manteniendo la misma arquitectura de red con 300 neuronas de entrada y 500 recurrentes, pero incrementando el número de iteraciones a 1 300 por letra. Esta estrategia buscaba frenar el aumento del error al ampliar la cantidad de caracteres.

Los resultados muestran que el MSE promedio por letra oscila entre aproximadamente 0,000449 (letra "h") y 0,001451 (letra "d").

Si comparamos estos datos con experimentos anteriores —con 2, 4 y 8 caracteres— se observa que el MSE global continúa aumentando progresivamente al agregar más caracteres, aunque este incremento no es tan drástico, gracias a la mayor cantidad de iteraciones. El ajuste fino del número de iteraciones ha ayudado a que, a pesar de la mayor complejidad, el modelo mantenga la capacidad de generalización, con errores en general por debajo de 10^{-3}. Sin embargo, sigue siendo evidente que ciertos caracteres, por su forma o dinámica más compleja, requieren un mayor esfuerzo de aprendizaje y resultan en mayor error.

5.5.6 Escalado: 26 carácteres

Para finalizar, en este experimento se entrenó la red neuronal para reconocer las 26 letras del alfabeto, manteniendo una arquitectura de 300 neuronas de entrada, 500 recurrentes y 1 300 iteraciones por carácter, igual que en el caso anterior, como en los casos anteriores, solo se utiliza un solo ejemplo por letra para el entrenamiento. El propósito fue evaluar cómo se comporta el modelo cuando tiene que aprender el conjunto completo de caracteres.

Los resultados indican que el MSE promedio por letra varía entre aproximadamente 0,000708 (letra "n") y 0,001690 (letra "z"). Las letras con trayectorias más complejas o menos lineales, como "v", "z", "j" y "g", presentan los errores más altos con valores que alcanzan hasta 0,001690. Por el contrario, letras de formas más simples como "c", "n", "o" o "u" obtiene mejores resultados, con errores por debajo de 0,00080.

Comparando este experimento con los anteriores, se observa que el MSE global sigue incrementándose al am-pliar el número de caracteres entrenados. Aunque el aumento no es abrupto, demuestra que el modelo necesi-ta más recursos o configuraciones adicionales para man-tener la precisión. En particular, las letras con movimien-tos más complejos evidencian el desafío creciente que representa escalar el aprendizaje. La consistencia en la arquitectura y en el número de iteraciones ha permitido que, aun con 26 caracteres, el error promedio se manten-ga por debajo de 2×10^{-3}, aunque claramente algunas letras requieren atención especial para mejorar su rendi-miento.

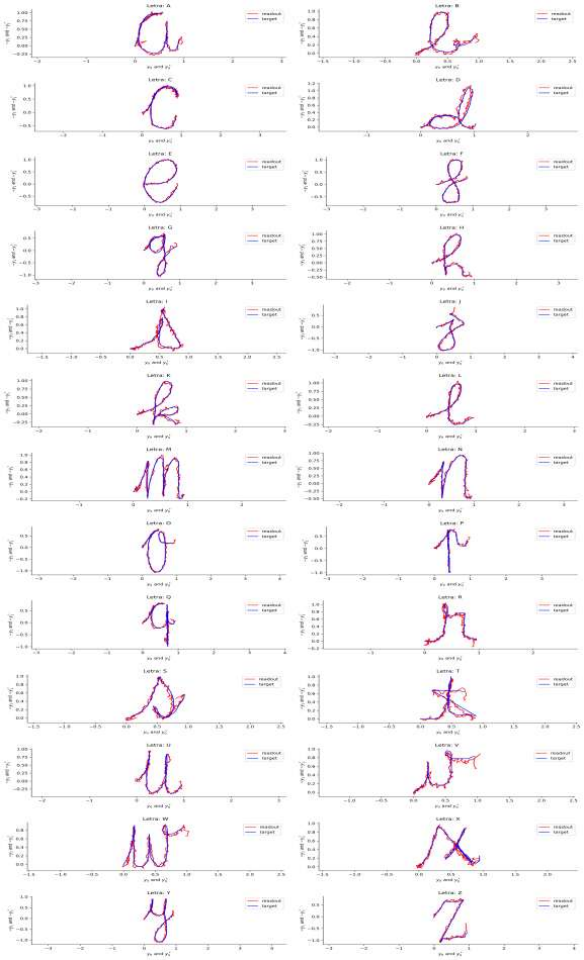


Figura 11: Letras generadas 26 batch

Estos resultados evidencian que el modelo ha logrado aprender todo el abecedario de forma precisa con tan solo un ejemplo de trayectoria por trazo. A pesar de la tendencia al alza en el error conforme se amplía el conjunto de caracteres, la red neuronal sigue reproduciendo trayectorias con gran exactitud incluso cuando se enfrenta al conjunto completo. A diferencia de otros sistemas, como redes GAN [8] o RNN tradicionales, que suelen necesitar enormes volúmenes de datos para generar variaciones realistas, las SNN demuestran una sorprendente capacidad de generalización con tan solo 26 ejemplos. Gracias a su codificación temporal y dinámica interna, estas redes resultan una alternativa muy prometedora cuando los datos son limitados o se busca eficiencia energética.

5.6 Concatenación de carácteres

Una vez se había aprendido correctamente la representación de todo el abecedario mediante una sola SNN, se decidió realizar un experimento adicional para observar el comportamiento de la red al generar letras concatenadas en una misma secuencia, con el objetivo de simular escritura manuscrita continua.

Para ello, se reutilizaron las trayectorias generadas por la SNN para cada letra, seleccionando la última aparición de cada carácter en el entrenamiento. A continuación, las trayectorias de salida (readout) y los objetivos (target) se concatenaron, respetando su estructura temporal, añadiendo un pequeño desplazamiento horizontal constante entre letras para facilitar su visualización.
Este procedimiento permitió construir una figura que representa todas las letras del abecedario dispuestas de forma secuencial:

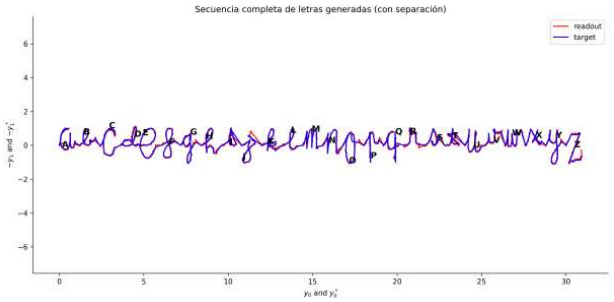


Figura 12: Caracteres Concatenados

Como se observa en la figura, las letras aparecen concatenadas de forma continua, no porque la SNN haya aprendido explícitamente a hacer transiciones entre caracteres, sino porque genera todos los puntos de forma seguida. Es decir, la red simplemente comienza a dibujar la siguiente letra justo donde terminó la anterior, sin reiniciar el trazo ni separar artificialmente las trayectorias, lo que da lugar a una concatenación visual fluida y natural.

5.7 Variaciones de un mismo carácter

Tras entrenar la red con diferentes caracteres utilizando configuraciones independientes de entrada para cada uno, el siguiente paso consistió en explorar una nueva línea de investigación: verificar si la red es capaz de aprender y generar variaciones dentro de un mismo carácter. En lugar de enfocarse en letras distintas.

El objetivo principal de este experimento es comprobar si, tras entrenarse con varias versiones distintas de la letra “a”, la red es capaz de reconocer patrones comunes entre ellas y captar qué elementos definen una “a”, incluso cuando el trazo presenta pequeñas variaciones.

La intención no es que la red memorice un único ejemplo, sino que sea capaz de aprender el concepto general del carácter. Para lograrlo, se utilizaron 5 ejemplos del carácter “a” diferentes.

Para conseguir este entrenamiento se decidió realizar un *spiketrain* combinado. Por un lado, un *spiketrain* base, el cual será completmente igual para 280 neuronas de entrada y un *spiketrain* de estilo que será diferente para las 20 neuronas restantes y diferentes para cada uno de los estilos de “a”.

Al hacerlo de este modo la red tendría que ser capaz de poder aprender que los *spiketrains* base corresponden al concepto de la letra “a” y que el *spiketrain* restante que corresponde a las 20 neuronas, sea capaz de asignarlo al concepto de variación del carácter

A continuación, se muestran los resultados generados por la red tras entrenarse con estas diferentes versiones del carácter” a”:

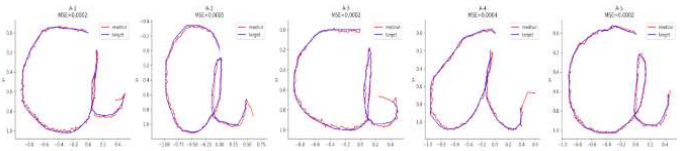


Figura 13: Variaciones entrenadas de “a”

Como se puede observar en la imagen, a pesar de tener un *spiketrain* similar, compartiendo una *spiketrain* de 280 neuronas en comun, hemos conseguido que el aprendizaje se haya realizado correctamente, adaptando cada uno de los *spiketrains* a sus correspondientes variaciones.

Una vez entrenado el modelo con las diferentes versiones de la letra “a”, el siguiente paso fue comprobar si la red era capaz de generalizar. Es decir, se quería ver si podía generar nuevas “a” que nunca había visto antes durante el entrenamiento.

Para ello, se relizo un programa el cual era capaz de cargar el modelo ya entrenado y poder pasarles *spiketrain* con la base de “a” concatenado con un spketrain para 20 neuronas completamente aleatorio que la red no había visto nunca. A continuación, se muestran los resultados de la creación de esste carácter con variaciones:

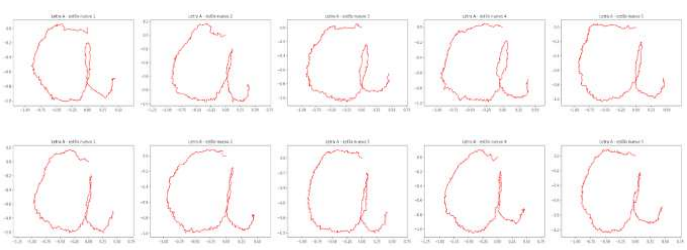


Figura 14 y 15: Variaciones de "a" generadas

Como se puede ver en los resultados, la red ha sido capaz de generar diferentes tipos de letra "a" que no había visto nunca durante el entrenamiento. A pesar de los nuevos *spiketrains* en las neuronas variables, los trazos generados mantienen la forma general del carácter, demostrando que la red no solo ha sido capaz de memorizar, sino que ha aprendido a generalizar el carácter "a". Esto confirma que una red SNN puede reproducir variaciones coherentes de un mismo carácter incluso habiendo entrenado únicamente con 5 ejemplos diferentes del mismo carácter. Esto marca una gran diferencia con los estudios tradicionales utilizando redes como las GAN que necesitan miles de ejemplos para poder generar un resultado similar. Esto demuestra una vez más el potencial de las SNN en este campo [13].

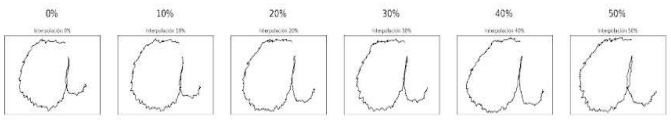
Tras comprobar que es posible generar variaciones de un mismo carácter habiendo entrenado la red con tan solo cinco ejemplos, se planteó un nuevo experimento orientado a evaluar si la SNN es capaz de entender la transición entre estilos de escritura.

El objetivo fue verificar si la red no solo memoriza patrones, sino que interioriza las diferencias entre estilos y puede interpolar entre ellos. Para ello, se diseñó un experimento basado en la interpolación progresiva entre el estilo 1 y el estilo 4 mostrados anteriormente.

La estrategia consistió en modificar aleatoriamente los *spiketrain* temporales del estilo 1, reemplazando algunos de sus instantes de activación por los correspondientes al estilo 4 de forma completamente aleatoria. De esta forma, se generaron versiones mixtas que combinaban características de ambos estilos.

La interpolación se hizo de forma gradual: al principio estaba 100 % en el estilo 1 y 0 % en el estilo 4. Luego, con cada paso intermedio, se fue aumentando poco a poco la proporción del estilo 4, reduciendo al mismo tiempo la del estilo 1. Finalmente, se alcanzó el punto invertido: 100 % estilo 4 y 0 % estilo 1.

A continuación, se muestran los resultados de este experimento:



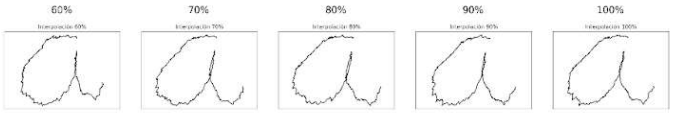


Figura 16: Interpolación estilos de "a"

La red modifica la forma de manera gradual al pasar del estilo 1 al estilo 4. En los primeros pasos, con un porcentaje bajo de estilo 4 (10–20 %), las letras conservan la estructura principal del estilo 1 pero ya muestran sutiles diferencias. Al llegar al 50 %, el resultado es una mezcla equilibrada. Cuando se superan los 60–70 %, el trazo comienza a parecerse claramente al estilo 4, y finalmente, al alcanzar el 100 % de estilo 4, la letra refleja completamente este estilo.

Este comportamiento refleja que la red no solo ha aprendido cada estilo por separado, sino que también puede generar transiciones fluidas entre ellos. En resumen, la SNN demuestra una notable capacidad de interpolación, lo que abre la puerta a diseñar variantes suaves y continuas entre distintos patrones de escritura.

5.8 Aprendizaje de un concepto abstracto

Para finalizar se realizo un estudio para poder comprobar si la red ya no es solo capaz de entender un estilo como hemos observado en el apartado anterior sino ser capaz de entrenar a una SNN para que aprenda el concepto itálico y esta sea capaz de poder generalizar este estilo a letras normales que haya aprendido anteriormente.

Para ello se ha realizado dos datasets, uno de letras normales y otro para letras Italicas (letras normales con inclinación). Para ello se crearán *spiketrains* base para 290 neuronas de entrada para cada una de las letras, para que la red identifique cada base con un carácter y un *spiketrain* de estilo (normal o itálico). De esta forma la red identificará cada uno de los estilos de letra. Una vez creado el dataset y los *spiketrain* se concatenarán y se entrenarán en un *batch* de 17, ya que para el entrenamiento se utilizarán 10 letras normales (de la "a" a la "l" sin tener en cuenta las letras "i" ni "j") y 7 letras italicas. El objetivo de este apartado es observar si después del entrenamiento y juntando la base de una letra con el estilo itálico para observar si la red es capaz de generarlas de forma itálica sin haber visto esa combinación anteriormente. A continuación, se muestran los resultados del entrenamiento: En este análisis, se observa que cuando se combinan estilos (normal e itálica) de una misma letra, los errores promedio (MSE) aumentan de forma notable. Los valores medios son:

Letra	Italica	Normal
A	0.001032	0.001125
B	0.000908	0.001347
C	0.001329	0.000796
D	0.000860	0.001108

E	0.001102	0.000650
F	0.001808	0.000978
G	0.003975	0.001053
H	-	0.001117
K	-	0.001026
L	-	0.000798

Figura 17: Tabla MSE

En general, estos promedios alcanzan valores más altos que los obtenidos al entrenar caracteres de forma aislada. Esto es esperable, ya que trabajar con varios estilos simultáneamente introduce más variabilidad en los patrones de entrada. No obstante, la red sigue mostrando su capacidad para generalizar y representar trayectorias variadas, aunque con un error ligeramente superior debido a la complejidad añadida.

En este apartado se ha aplicado un suavizado por media móvil para reducir el ruido de las trayectorias generadas por la red, lo que facilita comparar de forma visual los resultados con las trayectorias objetivo. Usamos una ventana de 20 muestras para atenuar las oscilaciones rápidas en las señales out_x y out_y, obteniendo formas más claras y coherentes sin alterar su estructura principal.

A continuacion se muestran los resultados suavizados generados junto a las señales objetivo del entrenamiento (Figura 18).

En la imagen se pueden observar en la fila superior los caracteres de estilo normal, mientras que en la fila inferior se pueden observar los caracteres Itálicos.

Como podemos observar, las trayectorias suavizadas se han ajustado a la señal objetivo después del entrenamiento, a pesar de que todos los *spiketrain* tanto de los caracteres normales y los caracteres itálicos tienen un estilo en común para cada tipo de carácter. A esto hay que añadirle que podemos concluir que la red ha sido capaz de aprender con éxito los caracteres itálicos y normales de cada una de las letras utilizando la base y ambos estilos.
En esta imagen (figura 18), observamos que la “h”, “k” y “l” no tienen la función objetivo. Estas letras se han generado a partir de juntar la combinación del *spiketrain* base de sus respectivas letras concatenado con el del estilo itálico, formando así una combinación nunca vista para la red. En este casó la red ha sido capaz de generalizar un estilo abstracto con tan solo siete ejemplos siendo capaz de generalizarlo a letras que no había visto de esta forma con tansolo 7 ejemplos de letras itálicas. Por lo tanto, podemos concluir que supone un avance para el aprendizaje de generación manuscrito con respecto a otras redes neuronales tradicionales como en el caso de las GAN que necesitan miles de ejemplos para poder generalizar un estilo abstracto.

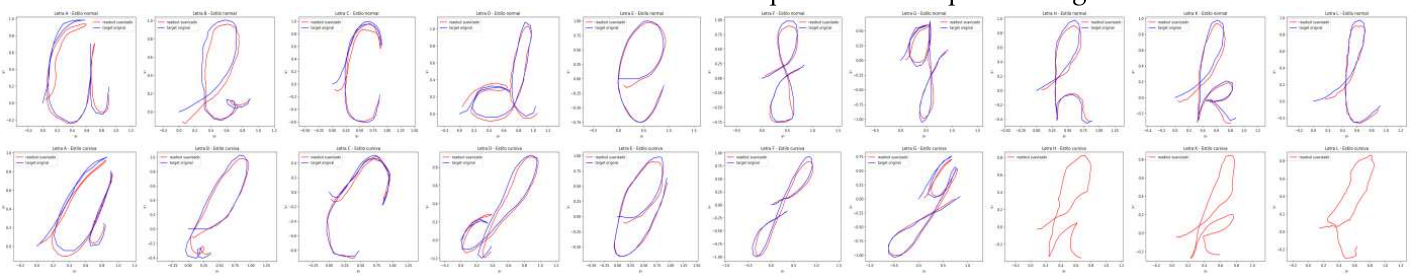


Figura 18: Generalización de estilo itálica

6. Conclusión

En conclusión, este Trabajo de Fin de Grado ha demostrado con éxito que las SNN pueden aprender y reproducir trayectorias manuscritas de todo el abecedario, mostrando grandes ventajas frente a otras arquitecturas tradicionales como las RNN o las GAN. En primer lugar, las SNN destacan por su eficiencia energética y su capacidad para procesar información temporal de forma natural, ya que trabajan directamente con *spikes* y codificación basada en tiempo. Gracias a esto, son capaces de generalizar patrones con pocos datos de entrenamiento, algo especialmente relevante en escenarios donde los recursos son limitados.

Además, el uso del algoritmo e-prop, inspirado en mecanismos biológicos como las trazas de elegibilidad, permite entrenar las SNN de manera local en el tiempo sin necesidad de almacenar largas secuencias como ocurre en BPTT. Esto se traduce en un modelo más escalable y potencialmente más eficiente para sistemas con restricciones computacionales. Asimismo, la introducción de técnicas de entrenamiento por *batches* y de suavizado de trayectorias ha demostrado ser efectiva para reducir ruido y mejorar la estabilidad y precisión de la salida.

No obstante, este estudio también pone de manifiesto algunas limitaciones inherentes a las SNN. Por una parte, el MSE se incrementa gradualmente conforme se añaden más caracteres o estilos, evidenciando que, aunque la red mantiene una buena capacidad de generalización, su rendimiento puede verse afectado cuando las trayectorias se vuelven más complejas o variables. Además, la falta de soporte completo para GPU en entornos como NEST limita la velocidad de entrenamiento, lo que podría suponer un obstáculo en aplicaciones a gran escala. Finalmente, aunque se ha logrado interpolar estilos y generar variaciones coherentes, el aprendizaje de combinaciones más diversas aún presenta desafíos, como se ha observado en el caso de caracteres itálicos más complejos.

En resumen, las SNN han demostrado ser una alternativa prometedora para la generación de escritura

manuscrita, ofreciendo eficiencia, interpretabilidad temporal y buena generalización con pocos datos. Sin embargo, su aplicación a problemas más complejos requerirá optimización de recursos, mejora del soporte de hardware y mayor exploración de parámetros para ajustar correctamente a la creciente variabilidad de los patrones.

Por todas estas características, este trabajo supone un gran avance para la inteligencia artificial conocida hoy en día y abre una nueva línea de exploración con arquitecturas mucho mas sostenibles.

AGRADECIMIENTOS

Este trabajo está dedicado a todas aquellas personas que me han apoyado a lo largo de todos estos años de carrera, y que han hecho posible que hoy llegue hasta aquí.

En especial, a mis padres, Salomé y Alfonso, por haberme dado la oportunidad de realizar este camino. Sin ellos, nada de esto habría sido posible.

A mis hermanos, Eric y Marc, por estar siempre ahí y apoyarme cuando lo he necesitado.

A mi pareja, Laura, por su paciencia, cariño y apoyo incondicional durante este periodo de tiempo.

A mis tutores, Xavier y Alicia, por todo lo que me han enseñado y por su valiosa ayuda durante el desarrollo de este proyecto.

Y, por último, a mi abuela Enriqueta, que, aunque no pueda estar hoy aquí, sé que estaría muy orgullosa de mi trabajo.

BIBLIOGRAFIA

[1] Kugele A, Pfeil T, Pfeiffer M and Chicca E (2020) Efficient Processing of Spatio-Temporal Data Streams With Spiking Neural Networks. *Front. Neurosci.* 14:439. doi: 10.3389/fnins.2020.00439Neurons and simple neural networks https://nest-simulator.readthedocs.io/en/v2.20.1/tutorials/pynest_tutorial/part_1_neurons_and_simple_neural_networks.html

[2] Bellec, G., Scherr, F., Subramoney, A. *et al.* A solution to the learning dilemma for recurrent networks of spiking neurons. *Nat Commun* **11**, 3625 (2020). https://doi.org/10.1038/s41467-020-17236-y

[3] Tutorial on learning to generate handwritten text with e-prop — NEST Simulator Documentation. (n.d.). https://nest-simulator.readthedocs.io/en/v3.8/auto_examples/eprop_plasticity/eprop_supervised_regression_handwriting.html

[4] *NumPy.* (n.d.). https://numpy.org/

[5] *Matplotlib.* (n.d.). https://matplotlib.org/

[6] *NEST simulador*. (n.d.). https://www.nest-simulator.org/

[7] Gan, J., Wang, W., Leng, J., & Gao, X. (2022). HiGAN+: Handwriting Imitation GAN with Disentangled Representations. *ACM Transactions on Graphics,* 42(1), 1–17. https://doi.org/10.1145/3550070

[8] Wikipedia contributors. (2025, June 17). *Spiking neural network*. Wikipedia. https://en.wikipedia.org/wiki/Spiking_neural_network

[9] Maes, A., Barahona, M., & Clopath, C. (2020). Learning spatio-temporal signals using a recurrent spiking network that discretizes time. *PLoS Computational Biology*, 16(1), e1007606. https://doi.org/10.1371/journal.pcbi.1007606

[10] Oladipupo, G. G. (2019, October 8). *Research on the concept of liquid state machine*. arXiv.org. https://arxiv.org/abs/1910.03354

[11] Bellec, G., Scherr, F., Subramoney, A., Hajek, E., Salaj, D., Legenstein, R., & Maass, W. (2020b). A solution to the learning dilemma for recurrent networks of spiking neurons. *Nature Communications,* 11(1). https://doi.org/10.1038/s41467-020-17236-y

[12] L.Kang, P.Riba, M.Rusiñol, A.Fornés, M.Villegas. Content and Style Aware Generation of Text-line Images for Handwriting Recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence* (TPAMI), 2021.

[13] K.Nikolaidou, G.Retsinas, G.Sfikas, M.Liwicki. "DiffusionPen: Towards Controlling the Style of Handwritten Text Generation". European Conference on Computer Vision (ECCV), 2024