OPTIMIZACIÓN DE REDES NEURONALES COMPLETAMENTE CONECTADAS EN SISTEMAS EMBEBIDOS

FULLY CONNECTED NEURAL NETWOR OPTIMIZATION FOR EMBEDDED SYSTEMS

Matías Vago Ancarola
Universidad Argentina de la Empresa, Argentina
mvagoancarola@uade.edu.ar

Leonardo Javier Amet
Universidad Argentina de la Empresa, Argentina
lamet@uade.edu.ar

Recepción: 18/noviembre/2024 **Aceptación:** 5/febrero/2025

Resumen

Este artículo examina la viabilidad de implementar Redes Neuronales de Capas Completamente Conectadas, en sistemas embebidos de bajos recursos. Hoy en día, el estado del arte de las redes neuronales esta principalmente ocupado por las redes de convolución o “Convolutional Neural Networks”. Sin embargo, las redes de capas completamente conectadas, un concepto más antiguo, aún tienen aplicabilidad en este contexto, y son más factibles de implementar en estos sistemas de recursos limitados gracias a su estructura más simple.

Se abordan las limitaciones de estos dispositivos y se proponen y prueban optimizaciones. Específicamente, las optimizaciones implementadas son dos. Por un lado, se utiliza el proceso de cuantificación de pesos y aritmética de punto fijo. Por otro lado, se aborda el concepto de aproximación de la función de activación.

El objetivo es demostrar la aplicabilidad de estos algoritmos en estos sistemas gracias a las optimizaciones, y hacer una breve mención a un dispositivo desarrollado a modo de prueba de concepto, el “Neural Pen”.

Palabras Clave: Embebidos, Neuronales, Redes, Sistemas.

Abstract

This article examines the viability of implementing Fully Connected Neural Networks in low resource embedded systems. Today, the state of the art of neural

networks is mainly occupied by Convolutional Neural Networks. Nonetheless, Fully Connected networks, an older concept, still have plenty of applicability in this context thanks to their simplicity in comparison to CNNs. The limitations of these devices are discussed, and optimizations are proposed and tested.

Specifically, the optimizations implemented are two. On one hand, weight quantization techniques and fixed-point arithmetic are utilized. On the other hand, the concept of activation function approximation is explored.

The objective is to demonstrate the viability of these algorithms in these systems thanks to the optimizations, and make a brief mention of a device developed as a proof of concept of the work carried out, the "Neural Pen".

Keywords: *Embedded, Networks, Neural, Systems.*

1. Introducción

En los últimos años, las redes neuronales se convirtieron en uno de los principales y más novedosos métodos de resolución de problemas. Es muy normal la asociación del concepto "redes neuronales" con los conceptos "alto poder de procesamiento" o "mucha capacidad de almacenamiento", ya que, es muy habitual su aplicación para resolver problemas muy complejos como, por ejemplo, el reconocimiento de imágenes. Esta aplicación en particular puede implicar redes neuronales con decenas de capas, cientos de miles de variables, sistemas capaces de realizar miles de millones de operaciones por segundo, y la capacidad de almacenar esa enorme cantidad de información. Lo que no fue tan habitual hasta más recientemente es la aplicación de esta tecnología en sistemas embebidos de recursos más limitados.

La industria del IoT (Internet of Things), por ejemplo, se encuentra en pleno crecimiento y está llena de dispositivos de estas características que pueden beneficiarse de este enfoque. En definitiva, las redes neuronales pueden ser tan grandes y costosas de procesar, como compleja sea la aplicación en particular. Si se pretende resolver un problema simple, una red con pocas variables y fácil de procesar puede ser suficiente. Las redes neuronales han sido el enfoque de mucha investigación y desarrollo desde su concepción. Hoy en día, el estado del arte de

esta tecnología está ocupado en gran parte por lo que se conoce como Redes Neuronales de Convolución o “Convolutional Neural Networks” (CNN). Este tipo de red neuronal ha llevado a enormes avances en el campo, y a aplicaciones revolucionarias como la detección de objetos en tiempo real. Entre los algoritmos específicos más implementados se destacan “Single Shot Detector” (SSD) [Liu, 2016], y “You Only Look Once” (YOLO) [Redmon, 2016] .

Estas tipologías de red neuronal implican una complejidad elevada en comparación con las redes más tradicionales de capas completamente conectadas. En la actualidad, el enfoque se encuentra en la optimización de estos algoritmos y el desarrollo de hardware específico para posibilitar la implementación de CNNs en sistemas embebidos. Un ejemplo reciente es el trabajo [Moosmann, 2023] donde los investigadores implementaron una versión de YOLO en sistemas embebidos, gracias al uso de hardware específicamente diseñado para ese propósito.

Sin embargo, las redes de capas completamente conectadas continúan siendo relevantes. Pueden ser menos poderosas, pero su complejidad reducida facilita su implementación en sistemas de recursos muy limitados, que no fueron diseñados con este tipo de aplicación en mente. Aun con la complejidad reducida de este tipo de redes, optimizaciones son necesarias para facilitar su implementación, sobre todo en aplicaciones que requieren predicciones en tiempo real. Para lograrlo, se plantea realizar dos optimizaciones puntuales.

Por un lado, se utilizará la cuantificación de pesos y el uso de un sistema de punto fijo. Esta es una optimización común y establecida [Hoffman, 2006] que tiene dos grandes ventajas. Trabajar con variables cuantificadas permite a los dispositivos operar con números enteros, algo que pueden hacer de manera mucho más rápida y eficiente en comparación a cuando utilizan números flotantes. Esto es particularmente cierto en sistemas que no poseen una unidad de punto flotante. Además, la cuantificación de variables permite reducir el peso en memoria de cada una de estas variables. Esto permite considerables ahorros de memoria en sistemas en los que éste es un recurso muy limitado.

Por otro lado, se propone utilizar una aproximación de la función de activación. Antecedentes a este enfoque se pueden encontrar en los trabajos [Timmons, 2020],

[Cotton, 2011] y [Jing, 2017]. La función de activación es una operación fundamental en la aritmética de redes neuronales, pero puede ocupar una gran cantidad de recursos computacionales cuando se trata de la función Sigmoide o la Tangente Hiperbólica. Esto es particularmente cierto cuando se considera la frecuencia con la que es necesario aplicar esta función al realizar una predicción.

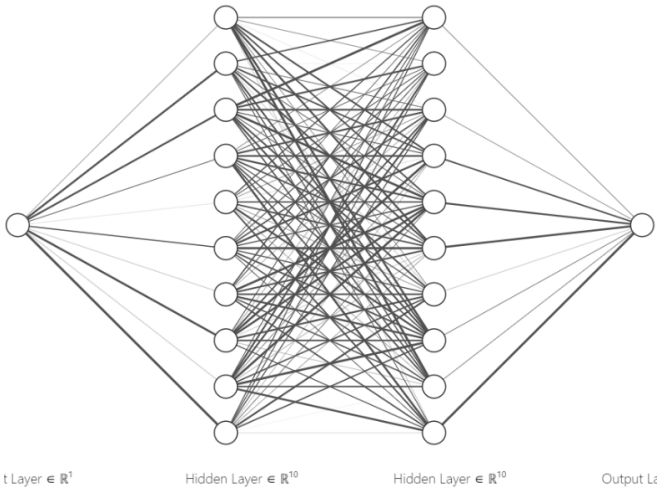
2. Métodos

Introducción a redes neuronales

Una red neuronal es un modelo computacional que consiste en una serie de nodos interconectados (neuronas artificiales) y un algoritmo de aprendizaje. Este sistema procesa valores de entrada para predecir o estimar el valor de salida más probable. Esta metodología de modelización es muy eficaz para sistemas complejos difíciles de representar con técnicas tradicionales. Es particularmente útil cuando se trata de implementar para resolver tareas “muy humanas”. El reconocimiento de imágenes es un excelente ejemplo de esto. Resolver este tipo de problemas con métodos de programación más tradicionales es una tarea muy desalentadora. Es en estos casos donde las redes neuronales realmente brillan.

Inspirado en el funcionamiento del cerebro biológico, este paradigma permite a las computadoras aprender a partir de datos observacionales, adaptándose y mejorando su rendimiento con la experiencia. Las redes neuronales pueden abordar una amplia gama de problemas, desde reconocimiento de patrones hasta toma de decisiones.

La estructura básica de una red neuronal de redes completamente conectadas se ilustra en la Figura 1, que muestra cómo se organizan las neuronas en capas y cómo fluye la información a través del sistema. Las redes neuronales modelan sistemas complejos mediante variables ficticias que se ajustan durante el entrenamiento para aproximarse al sistema real. Este proceso implica la modificación iterativa de parámetros como la parcialidad de los nodos y los pesos de las conexiones, con el objetivo de minimizar el error entre la salida obtenida y la deseada. Algoritmos de aprendizaje, como el gradiente descendiente estocástico, ajustan estos parámetros hasta lograr un desempeño aceptable.



Fuente: elaboración propia

Figura 1 Ejemplo de red neuronal.

En las redes neuronales, una mayor cantidad de nodos y conexiones suelen permitir aproximaciones de sistemas más complejos, pero también aumentan el costo computacional de entrenamiento y procesamiento. Por lo tanto, al implementar estas soluciones, especialmente en sistemas embebidos de bajos recursos, es crucial optimizar la red para obtener los resultados deseados con la menor cantidad de variables posibles.

El propósito de este artículo es explorar algunas de estas optimizaciones, particularmente relevantes para sistemas embebidos con limitaciones de recursos. Cabe destacar que el campo de las redes neuronales es extenso, y la aritmética detallada detrás de su funcionamiento excede el alcance de este artículo. Para una comprensión más profunda, se recomienda consultar [Nielsen, 2019].

Las limitaciones

Antes de presentar las optimizaciones realizadas, se repasan las principales y más comunes limitaciones de un sistema embebido:

- Menor poder de procesamiento: Los microcontroladores tienen un poder de cómputo significativamente menor que las PC modernas. Esto se debe principalmente a la diferencia en la velocidad del clock, que determina la cantidad de instrucciones ejecutadas por segundo. Mientras una PC promedio actual opera a unos $2.1\ GHz$, el microcontrolador utilizado en este proyecto

(STM32F103C8T6) alcanza un máximo de $72\,MHz$, aproximadamente 30 veces menos. Es importante notar que, a diferencia de las PC que distribuyen su capacidad entre múltiples procesos, los microcontroladores dedican todos sus recursos a la ejecución del código específico.

- Menor capacidad de almacenamiento: En una PC tenemos varios gigabytes de memoria disponible para almacenar las variables necesarias para la ejecución del programa. En cambio, el microcontrolador en cuestión sólo posee $20\,kb$ de memoria RAM (en el orden de 400,000 veces menos comparado con una PC moderna promedio).
- Ausencia de unidad de punto flotante (FPU): a diferencia de los microprocesadores de PC, muchos microcontroladores, incluido el utilizado en este proyecto, carecen de una Unidad de Punto Flotante (FPU). Estos dispositivos solo poseen una Unidad Lógica Aritmética (ALU) para cálculos con números enteros. Como consecuencia, las operaciones con números decimales deben realizarse por software, lo que aumenta significativamente el tiempo de procesamiento.

La limitación es particularmente relevante para las redes neuronales, que operan principalmente con números fraccionarios. Los valores de entrada en estas redes suelen normalizarse entre -1 y 1 o 0 y 1, lo que requiere cálculos con decimales. La ausencia de FPU impacta directamente en el rendimiento de la red neuronal en estos sistemas embebidos, afectando tanto la velocidad de procesamiento como la precisión de los cálculos. Esta restricción se vuelve crucial al implementar redes neuronales en sistemas de recursos limitados, presentando un desafío significativo para los desarrolladores en términos de optimización y eficiencia.

Aproximaciones de la función de activación

Uno de los principales métodos de optimización, y quizás el más interesante, es la aproximación de la función de activación. Esta función, en el caso de este proyecto se analiza la tangente hiperbólica, es uno de los cálculos más ejecutados por el microcontrolador, ya que es necesaria para calcular los valores de salida en

cada nodo. De la misma manera, su derivada es usada frecuentemente durante el proceso de entrenamiento. Las Ecuaciones 1 y 2 representan a la tangente hiperbólica y su derivada.

$$A_{(x)} = \tanh x \tag{1}$$

$$A'_{(x)} = \frac{1}{\cosh x} \tag{2}$$

El cálculo de estas funciones implica procesos computacionales complejos que demandan un alto costo en términos de procesamiento. Esta complejidad se vuelve particularmente problemática en escenarios que requieren la ejecución de estos cálculos con alta frecuencia, lo cual puede impactar significativamente en el rendimiento general del sistema. Si se busca mejorar el rendimiento en un microcontrolador de bajos recursos, no es posible utilizar la forma exacta de las Ecuaciones 1 y 2. Como solución a este problema se buscaron formas de aproximar estas funciones, equivalentes que sean más fáciles de procesar pero que conservan la funcionalidad. En [Timmons, 2020] se abordó este tema y consiguieron buenos resultados. Estos investigadores fueron capaces de hacer aproximaciones de las funciones de activación que eran más eficientes que las funciones originales, sin perder precisión en porcentajes de aciertos. De la misma manera en [Cotton, 2011] y [Jing, 2017] se discute este concepto. De todos modos, las aproximaciones utilizadas en estos trabajos son más complejas que lo buscado. Por lo tanto, se abordó el diseño una alternativa aún más sencilla y eficiente, adaptada específicamente a las necesidades y limitaciones de nuestro sistema.

Cuantización de variables y uso de punto fijo

Las variables de la red neuronal como los pesos, parcialidades, valor de cada nodo y desvíos son datos que deben ser almacenados en el microcontrolador. Para representar estas variables comúnmente, se utiliza números decimales en formato de coma flotante de simple precisión (estándar IEEE-754). Este tipo de dato se conoce como *float* en lenguaje C. En un escenario ideal, este sería el tipo de dato a utilizar dado que su precisión es más que suficiente para nuestra aplicación. El microcontrolador no dispone de FPU, por lo que las operaciones en punto flotante

requieren mayor procesamiento. Debido a esto, se propone representar las variables utilizando *punto fijo* y cuantizando las variables. De esta forma, se tiene datos del tipo entero que el microcontrolador puede procesar de forma nativa, pero que en realidad representan números fraccionarios. Además, en algunas aplicaciones, podrían utilizarse variables enteras que ocupan menos bits en memoria que su contraparte en coma flotante de precisión simple, lo cual implica un gran ahorro en memoria, además de un procesamiento más eficiente. Un antecedente de esta metodología puede encontrarse en [Hoffman, 2006].

La desventaja de este método radica en la necesidad de seleccionar cuidadosamente la posición del punto decimal, de modo que los bits asignados a la parte entera sean suficientes para representar el rango de valores necesario, a la vez que los bits de la parte fraccionaria proporcionen la precisión requerida.

Por otro lado, es necesario redefinir las operaciones teniendo en cuenta la aritmética de punto fijo para lograr resultados correctos. La multiplicación de dos números en punto fijo requiere un ajuste posterior. Se puede tomar el ejemplo de multiplicar los números 2.0 y 3.0 en punto fijo en un sistema de $8\ bits$, con $4\ bits$ asignados a la parte entera, y 4 a la parte decimal. La Tabla 1 muestra algunos números y sus representaciones en este sistema. El producto de 2.0 y 3.0 en punto fijo representados por 32 y 48, con el operador de multiplicación tradicional da como resultado 1536. Si se observa la Tabla 1, se ve que el resultado que se debería haber obtenido es el 96, quien representa al 6.0 decimal. Si m es el número de bits fraccionarios, el resultado correcto se obtiene multiplicando los números como enteros y luego desplazando el resultado m bits a la derecha (equivalente a dividir por 2^m). Para la división, por otra parte, se debe realizar el proceso inverso. Primero se desplaza el dividendo m bits a la izquierda (multiplicando por 2^m) y luego se divide por el divisor.

Tabla 1 Ejemplo representación en punto fijo.

Decimal	-8.0	-7.5	-2.0	0	1.0	2.0	3.0	6.0	7.9375
P. Fijo	-128	-120	-32	0	16	32	48	96	127

Fuente: elaboración propia

Respecto de las sumas y restas, se aplica aritmética saturada. En estas se debe verificar si el resultado excede el rango representable. Si esto ocurre, el resultado se "satura" al valor máximo o mínimo del rango, según corresponda. Esto previene el desbordamiento y mantiene el resultado dentro de los límites válidos. Evidentemente esto producirá resultados erróneos, pero serán menos erróneos que si no se los limita. De esta manera, implementando punto fijo, se consigue convertir todas las operaciones en punto flotante a operaciones con enteros, las cuales el microcontrolador puede realizar fácilmente.

Agregado de memoria adicional SD

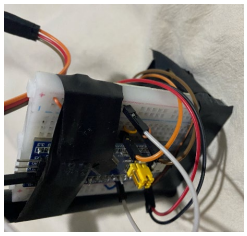
Otra de las grandes limitaciones es la falta de memoria en estos dispositivos de pocos recursos. Las redes pueden llegar a requerir cientos de nodos y varias capas para poder obtener resultados aceptables, según la aplicación. Esto implica una enorme cantidad de variables que deben ser almacenadas, y el microcontrolador debe tener rápido acceso a ellas para poder operar. Una solución posible es agregar memoria adicional mediante, por ejemplo, una memoria SD. El inconveniente de este enfoque es que el microcontrolador debe leer estas variables de la tarjeta SD para poder utilizarlas. Si la totalidad de las variables no entran en la memoria nativa del microcontrolador, son necesarias lecturas frecuentes donde las variables se sobrescriben unas a otras en función de cuales son necesarias en cada momento.

El dispositivo Neural Pen

A modo de prueba de concepto, y para poder probar la eficacia de las optimizaciones y visualizar los resultados, se desarrolló un dispositivo que implementa redes neuronales con las optimizaciones ideadas. Este dispositivo fue, en parte, la inspiración para el desarrollo de la investigación. Si bien este artículo se enfoca en los métodos de optimización desarrollados para la implementación de redes neuronales en sistemas embebidos, en paralelo, estuvo el desarrollo del dispositivo Neural Pen. Este periférico para ordenador fusiona las funciones de un ratón y un teclado en un único dispositivo gracias al uso de redes neuronales. Permite al usuario utilizarlo como un mouse tradicional, o escribir con él como si

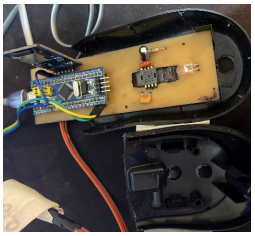
fuera un lápiz. Gracias al uso de redes neuronales, este es capaz de distinguir los símbolos dibujados mediante el análisis de los desplazamientos en X e Y detectados por un sensor óptico. Luego de predecir los símbolos dibujados, es capaz de enviarlos como caracteres de texto al igual que lo hace un teclado. Este desarrollo paralelo fue de suma importancia para permitir la prueba y puesta en práctica de todos los conceptos investigados durante el proyecto, y, por lo tanto, este apartado sirve de breve mención y muestra de los prototipos realizados.

Durante el desarrollo del proyecto, se diseñaron tres prototipos. Cada uno fue muy útil para detectar fallas, y llevaron al desarrollo de un tercer y último prototipo. Para esta tercera implementación, se diseñó placa de circuito impreso (PCB) a medida y se reutilizó la carcasa de un mouse para contener de manera segura y prolija todos los componentes dentro. Estos prototipos han desempeñado un papel crucial en la transición de la teoría a la práctica, permitiendo verificar hipótesis y resultados. Las Figuras 2, 3, 4 y 5 ilustran los tres prototipos elaborados.



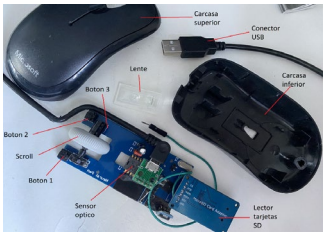
Fuente: elaboración propia

Figura 2 Primer prototipo.



Fuente: elaboración propia

Figura 3 Segundo prototipo.



Fuente: elaboración propia

Figura 4 Tercer prototipo - despiece.

Fuente: elaboración propia

Figura 5 Tercer prototipo – ensamblado.

La red neuronal ejecutada en el dispositivo final posee 60 entradas para los desplazamientos en los ejes X e Y (30 para cada eje), y 36 salidas para las letras del abecedario, y los dígitos del 0 al 9. Contiene 2 capas ocultas, cada una con 30 nodos.

3. Resultados y discusión

Aproximaciones de la función de activación

La tangente hiperbólica, función de activación elegida para este proyecto, fue objeto de un proceso de aproximación para optimizar su implementación en sistemas embebidos. Este proceso consistió en identificar funciones que, siendo considerablemente más simples, emularan el comportamiento de la tangente hiperbólica en el intervalo relevante. Se evaluaron diversas alternativas, sometiéndolas a pruebas rigurosas dentro de una red neuronal para analizar su funcionalidad, impacto en la precisión de las predicciones y tiempos de ejecución. De las múltiples opciones examinadas para aproximar la tangente hiperbólica, las Ecuaciones 3 y 4 demostraron ofrecer el mejor equilibrio entre precisión y eficiencia computacional, adaptándose óptimamente a las limitaciones de recursos del sistema objetivo.

$$A(x) \cong \begin{cases} -1, x \leq -2 \\ \frac{x^2}{4} + x, -2 < x \leq 0 \\ \frac{-x^2}{4} + x, 0 < x < 2 \\ 1, x \geq 2 \end{cases} \quad (3)$$

$$A'(x) \cong \begin{cases} 0, x \leq -2 \\ \frac{x}{2} + 1, -2 < x \leq 0 \\ \frac{-x}{2} + x, 0 < x < 2 \\ 0, x \geq 2 \end{cases} \quad (4)$$

La Figura 6 ofrece una comparación visual de las funciones originales y sus aproximaciones. En verde se muestra la función tangente hiperbólica original (Ecuación 1), mientras que su aproximación (Ecuación 3) se representa en gris. La derivada de la tangente hiperbólica (Ecuación 2) aparece en naranja, y su correspondiente aproximación (Ecuación 4) se visualiza en azul.

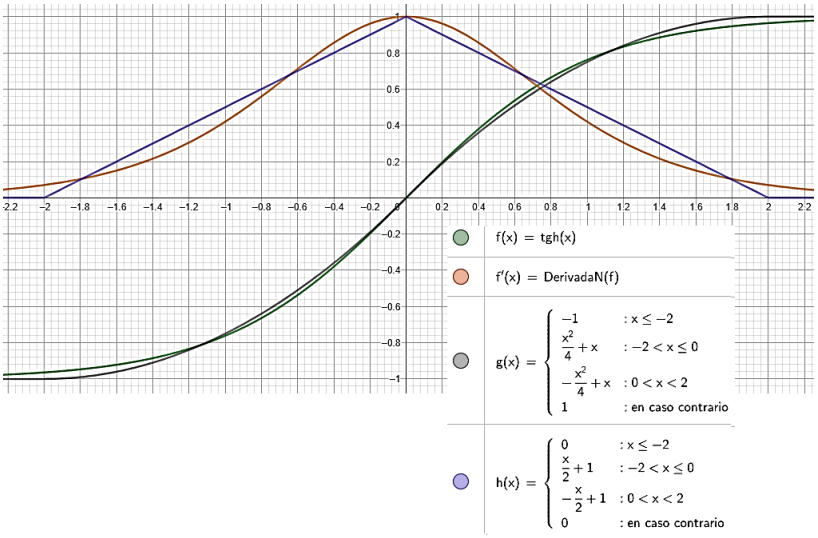
Para evaluar la eficiencia de las funciones exactas y sus aproximaciones, se diseñó una prueba comparativa. Esta consistió en un bucle que se ejecuta dos mil millones de veces, donde en cada iteración:

- Se asigna un valor aleatorio entre -1 y 1 a una variable.

- Se evalúa tanto la función como su derivada en ese valor.

Las aproximaciones propuestas son significativamente más eficientes para el microcontrolador, ya que involucran operaciones más simples:

- La función aproximada requiere solo una multiplicación ($x * x$), una suma y una división.
- La derivada aproximada es aún más sencilla, eliminando la necesidad de multiplicación.



Fuente: elaboración propia

Figura 6 Aproximación de función activación.

Además, los divisores en estas operaciones son potencias de 2, algo muy eficiente para procesar sistemas binarios como los microcontroladores. Los resultados de esta prueba comparativa se presentan en la Tabla 2, ilustrando cuantitativamente la mejora en eficiencia lograda con las aproximaciones. Los resultados de la implementación demuestran una mejora sustancial en el rendimiento. Con las aproximaciones propuestas, el programa se ejecuta aproximadamente 4.6 veces más rápido en promedio, lo que representa un aumento significativo en la eficiencia computacional. Notablemente, esta optimización no conlleva una pérdida apreciable en la precisión de las predicciones. La implementación final alcanza tasas de acierto del orden del 93%, tanto utilizando las funciones exactas como las aproximadas. Esta equivalencia en precisión, combinada con la considerable reducción en tiempo

de ejecución, valida la efectividad de las aproximaciones propuestas para su uso en sistemas embebidos de recursos limitados.

Tabla 2 Tiempos de ejecución reales vs aproximaciones.

	Tiempo con funciones reales [s]	**Tiempo con funciones aproximadas [s]**
1	227	49
2	229	51
3	275	60
4	226	49
5	224	47
Promedio	236.2	51.2

Fuente: elaboración propia

Punto fijo

La implementación del sistema de punto fijo resultó ser crucial para optimizar el rendimiento en el microcontrolador sin unidad de punto flotante. Inicialmente, se intentó utilizar variables de 16 bits (int16_t), buscando maximizar el ahorro de memoria. Sin embargo, esta aproximación resultó insuficiente para proporcionar el rango y la precisión decimal necesarios en la implementación particular del Neural Pen. Consecuentemente, se adoptó un formato de 32 bits, equivalente al tamaño de los floats estándar. Esta decisión proporcionó un rango y precisión adecuados para la aplicación. Aunque no se logró reducir el espacio en memoria respecto al uso de floats, la implementación en punto fijo disminuyó significativamente la carga computacional y los tiempos de ejecución del microcontrolador.

Es importante destacar que la viabilidad de utilizar 16 bits o una cantidad aún menor puede variar según el tamaño de la red y los requisitos específicos de la aplicación. Por lo tanto, en futuros proyectos, sería prudente considerar la posibilidad de emplear 16 bits si las condiciones lo permiten, lo cual podría resultar en un ahorro adicional de memoria.

Memoria SD

Una de las primeras implementaciones probadas excedía la capacidad de memoria del microcontrolador. Debido a esto, se debió añadir una memoria SD para poder guardar los pesos, que son las variables que más espacio ocupan. El resto

de las variables quedaban almacenadas en la memoria propia del controlador. El microcontrolador se comunicaba con la SD para leer los valores de los pesos a medida que los necesitaba, y poder realizar los cálculos. Para reducir la cantidad de lecturas, la idea fue en una sola lectura, recuperar todos los pesos necesarios para hacer, por ejemplo, el pasaje hacia adelante de una capa a otra, y esos pesos se guardaban en la memoria propia del controlador. Una vez que fueron usados y se necesitaban pesos nuevos, se leía nuevamente la próxima cadena de valores necesarios, y estos sobrescriben a los ya usados. De esta manera, nunca se supera la memoria total disponible e interna del microcontrolador, pero se evitaba hacer una lectura individual por cada valor necesario con el objetivo de ahorrar tiempo de ejecución. De todas formas, esta implementación resultó ser muy lenta, el principal contribuyente a este problema siendo la enorme cantidad de variables y, por lo tanto, la cantidad de operaciones necesarias. El dispositivo no era capaz de predecir los símbolos dibujados en tiempo real.

En implementaciones siguientes, se consiguió reducir la cantidad de variables lo suficiente para que sean todas almacenables en la memoria RAM del microcontrolador. Esto es una enorme ventaja ya que significa que el microcontrolador tiene acceso a todos los pesos sin necesitar estar leyéndolos de la SD constantemente. Aun así, la SD fue una importante adición al dispositivo ya que se utiliza para hacerle llegar las variables pre-entrenadas en una computadora al microcontrolador durante el arranque del programa, evitando la necesidad de realizar el entrenamiento de la red neuronal en el mismo dispositivo.

Aun así, el agregado de memoria adicional puede ser una herramienta útil en otras aplicaciones. En casos donde predicciones en tiempo real no son necesarias, y el sistema cuenta con más tiempo para realizar predicciones, redes neuronales más grandes utilizando la técnica de lecturas consecutivas y sobreescritura, podrían ser una solución viable.

4. Conclusión

Se puede afirmar que la implementación de redes neuronales de capas completamente conectadas en sistemas embebidos de bajos recursos es

absolutamente posible. Podemos confirmar que las limitaciones de estos sistemas, si bien pueden ser muy restrictivas según la plataforma utilizada, pueden ser subsanadas con algunas técnicas de optimización. Por supuesto, también se debe moderar las expectativas según la aplicación deseada ya que ciertas aplicaciones siguen dependiendo plenamente de los recursos del sistema y el tipo de red especifica que se pueda utilizar, sin importar las optimizaciones.

En cuanto a las técnicas investigadas e implementadas, destacan la aproximación de la función de activación y el uso de punto fijo. La aproximación de la función de activación demostró ser una técnica extremadamente útil, ahorrando una enorme cantidad de tiempo de procesamiento sin una pérdida observable de precisión en los resultados.

La utilización de punto fijo también resulta vital en sistemas que no posean unidad de punto flotante, ya que, sin su uso, no habría sido posible obtener una detección de caracteres en tiempo real.

Por último, es importante destacar el valor de los prototipos realizados. Aunque su desarrollo no se abordó en este artículo, fueron fundamentales para verificar todos los avances, identificar inconvenientes y encontrar soluciones.

6. Bibliografía y Referencias

[1] Cotton, Nicholas J., Wilamovski, B. Compensation of Nonlinearities Using Neural Networks Implemented on Inexpensive Microcontrollers, 2011.

[2] Hoffman, Mychal, Bauer, Paul, Hemrnelman, Brian, Hasan, Abul. Hardware synthesis of artificial neural networks using field programmable gate arrays and fixed-point numbers, 2006.

[3] Jing, Yuang, Youssefi, Bahar, Mirhassani, M., Muscedere, R. An efficient FPGA implementation of Optical Character Recognition for License Plate Recognition, 2017.

[4] Liu, Wei, Anguelov, Dragomir, Erhan, Dumitru, Szegedy, Christian, Reed, Scott, FU, Cheng-Yang, BERG, Alexander. SSD: Single Shot MultiBox Detector, 2016.

[5] Moosmann, Julian, Giordano, Marco, Vogt, Christian, Magno, Michele. TinyissimoYOLO: A Quantized, Low-Memory Footprint, TinyML Object Detection Network for Low Power Microcontrollers, 2023.

[6] Nielsen, Michael. Neural Networks and Deep Learning. [libro en linea] 2019. <neuralnetworksanddeeplearning.com>.

[7] Redmon, Joseph, Divvala, Santosha, Girshick, Ross, Farhadi, Ali. You Only Look Once: Unified, Real-Time Object Detection, 2016.

[8] Timmons, G. Nicholas, Rice, Andrew. Approximating Activation Functions, [paper en linea] 2020. https://arxiv.org/pdf/2001.06370.pdf.