Implementación de redes neuronales en FPGAs usando tipos de datos de punto fijo

Daniel Enériz, Nicolás Medrano, Belén Calvo

Grupo de Diseño Electrónico (GDE)
Instituto de Investigación en Ingeniería de Aragón (I3A)
Universidad de Zaragoza, Mariano Esquillor s/n, 50018, Zaragoza, Spain.
Tel. +34-976762707, e-mail: eneriz@unizar.es

Resumen

La capacidad de estimar funciones no lineales hace que las redes neuronales sean una de las herramientas más usadas para aplicar fusión sensorial, permitiendo combinar la salida de diferentes sensores para obtener información de la que a priori no se dispone. Por otra parte, la capacidad de procesamiento paralelo de las FPGAs (*Field-Programmable Gate Array*) las hace idóneas para implementar redes neuronales ubicuas, permitiendo inferir resultados más rápido que una CPU (*Central Processing Unit*) sin necesidad de una conexión activa a internet. De esta forma, en este artículo se propone un flujo de trabajo para diseñar, entrenar e implementar una red neuronal en una FPGA Xilinx PYNQ Z2 que use tipos de dato de punto fijo para hacer fusión sensorial. Dicho flujo de trabajo es probado mediante el desarrollo de una red neuronal que combine las salidas de una nariz artificial de 16 sensores para obtener una estimación de las concentraciones de CH_4 y C_2H_4.

Introducción

Hoy en día las redes neuronales son una herramienta que se extiende en infinidad de campos y, sin duda, la fusión sensorial es uno de ellos [1]-[3]. Normalmente, las operaciones que tienen lugar en las redes neuronales se computan en la nube, precisando una conexión activa a internet. Aun así, ya sea por motivos de privacidad, ancho de banda disponible o simplemente porque no es viable, se está tratando de evitar dicha conexión activa, acercando el dispositivo de cómputo de las redes neuronales a la aplicación o dispositivo que las precisan, empleando el paradigma de la computación ubicua.

Debido a su alto grado de paralelización y a su reprogramabilidad, las FPGAs se están usando para implementar redes neuronales que, entre otras cosas, permiten hacer fusión sensorial en tiempo real [4]-[6]. Una de las claves para poder implementar redes neuronales en FPGAs de bajo coste es la cuantización de los pesos. Debido a la limitación de recursos disponibles, es habitual representar los valores internos de la red neuronal con punto fijo o enteros, reduciendo así el número de recursos de registros que se necesitan para implementar la red a costa de un incremento en el error de los valores de salida.

Metodología

El flujo de trabajo para poder desarrollar e implementar una red neuronal se divide en dos grandes bloques: la virtualización, donde se elegirá la arquitectura de red, se realizará el entrenamiento y se evaluará la cuantización de los parámetros; y la implementación, donde será necesario traducir la red virtualizada a un leguaje de descripción de hardware y se tendrá que desarrollar un *driver* que permita inferir resultados en la red implementada en la FPGA.

Virtualización

El *dataset* con el que vamos a trabajar es una serie temporal de las salidas de 16 sensores de gas químicos de una nariz artificial en base a diferentes combinaciones de las concentraciones de CH_4 y C_2H_4 en la atmósfera [7], las cuales trataremos de estimar usando las salidas de los sensores mediante la red neuronal. Resultados preliminares sugieren el empleo de una arquitectura de red *fully-conected*, donde todas las neuronas de una capa están conectadas con todas de la capa anterior y posterior, de dos capas ocultas de 32 y 12 neuronas con función de activación sigmoide.

Realizamos un primer entrenamiento de la red neuronal obteniendo los resultados de la Figura 1. Por otra parte, en la Figura 2 aparece la evolución de los costes del *training set* y del *validation set* durante el primer entrenamiento. Este último, el coste del *validation set*, comienza a dar signos de estabilización cuando interrumpimos el entrenamiento, signo de que el modelo está cercano al punto de aprendizaje óptimo. La interrupción del entrenamiento en este momento nos permite simular

el modelo neuronal implementado en la FPGA usando diferentes tipos de dato de punto fijo para representar los parámetros. Tras estas simulaciones se elige usar el tipo de dato 12-3 (12 bits en total, de los cuales 3 están dedicados a la parte entera), ya que es el que mejor compromiso alcanza entre el error promedio para la serie temporal y el número de bits que usa.

Una vez elegido el tipo de dato de punto fijo con el que vamos a representar la red neuronal, se reanuda el entrenamiento. Esta vez se va a tener en cuenta la representatividad de los parámetros en el tipo de dato de punto fijo 12-3, de forma que se consigue ajustar el modelo a las salidas esperadas a la vez que se obtienen parámetros representables. En la Tabla 1 se muestra el descenso del error promedio de los resultados del segundo entrenamiento.

Implementación

Para poder programar una FPGA se debe usar un leguaje de descripción de hardware o HDL, por sus siglas en inglés. Existe una herramienta, HLS (*High-Level Synthesis*), que se encarga de traducir un lenguaje de alto nivel, como puede ser C/C++ [8], a un HDL, de forma que con hacer una descripción algorítmica del modelo neuronal en C/C++ bastará para obtener dicho modelo en un HDL. Además, HLS, permite introducir ciertas directrices a la hora de sintetizar el diseño que permiten reducir la latencia de este a costa de hacer uso de un mayor número de recursos de la FPGA. En nuestro caso hemos conseguido reducir el tiempo de inferencia a 2,75 µs por cada lectura, por debajo de los 5,6 µs que tardaba una CPU i5-3470.

Una vez sintetizado y validado el diseño en HLS, se obtiene el código HDL que permite generar un archivo que se va a encargar de configurar la matriz de conexionado de la FPGA para dar lugar al sistema lógico que se comporta como describía el código en C/C++, es decir, nuestra red neuronal.

Como se ha comentado en líneas anteriores, en este proyecto se está trabajando con la plataforma PYNQ Z2 de Xilinx, la cual incorpora un SoC (*System on Chip*) ZYNQ XC7Z020-1CLG400C que incluye un procesador Cortex-A9 de doble núcleo y una FPGA de 13300 CLBs (*Configurable Logic Blocks*) y 220 bloques DSP (*Digital Signal Processing*). Dicha plataforma es accesible mediante Python a través de cuadernos de Jupyter y el fabricante provee de un paquete para poder controlar la FPGA, permitiendo cargar archivos de configuración y hacer escritura y lectura de los puertos entrada y salida de los diseños.

En nuestro caso, además, para poder hacer uso de la red neuronal implementada en la FPGA que está representada por valores de punto fijo hemos desarrollado un *driver* específico en Python que automatiza el proceso de carga de parámetros, así como la inferencia de resultados. De esta forma, hemos sido capaces de inferir a través de la FPGA los resultados de la Figura 3, los cuales son coincidentes con las simulaciones previas del sistema que obteníamos en HLS.

Conclusiones

Con la intención de encontrar una solución al cómputo ubicuo de la red neuronal, se presenta un flujo de trabajo para implementar redes neuronales en FPGAs usando representación en punto fijo, concretamente en la plataforma PYNQ-Z2 de Xilinx. Este proceso tiene dos etapas marcadas: la virtualización, donde se realiza un entrenamiento interrumpido, permitiendo seleccionar el tipo de dato óptimo para representar el modelo neuronal y obteniendo valores de los parámetros representables en el tipo de dato elegido; y la implementación de dicho modelo en la FPGA usando HLS, evitando así hacer descripciones de bajo nivel.

Además, como se comprueba en la Tabla 1, el incremento del error promedio al implementar la red neuronal en la FPGA no es mayor que el error promedio que presentaba el modelo tras finalizar la segunda fase del entrenamiento, lo que es signo de que el paso a punto fijo no implica un error mayor que el asociado a la propia red.

REFERENCIAS

[1]. KOTSIOPOULOS, T., LEONTARIS, L., DIMITRIOU, N., IOANNIDIS, D., OLIVEIRA, F., SACRAMENTO, J., AMANATIADIS, S., KARAGIANNIS, G., VOTIS, K., TZOVARAS, D. y SARIGIANNIDIS, P., 2020. Deep multi-sensorial data analysis for production monitoring in hard metal industry. *The International Journal of Advanced Manufacturing Technology* [en línea], [Consulta: 6 noviembre 2020]. ISSN 0268-3768, 1433-3015. DOI 10.1007/s00170-020-06173-1. Disponible en: http://link.springer.com/10.1007/s00170-020-06173-1.

[2]. DUMITRESCU, C., MINEA, M. y CIOTIRNAE, P., 2020. UAV Detection Employing Sensor Data Fusion and Artificial Intelligence. En: L. BORZEMSKI, J. ŚWIĄTEK y Z. WILIMOWSKA (eds.), *Information Systems Architecture and Technology: Proceedings of 40th Anniversary International Conference on Information Systems Architecture and Technology – ISAT 2019* [en línea]. Cham: Springer International Publishing, pp. 129-139. [Consulta: 9 noviembre 2020]. ISBN 9783030304393. Disponible en:

http://link.springer.com/10.1007/978-3-030-30440-9_13.

[3]. ADAK, M. y YUMUSAK, N., 2016. Classification of E-Nose Aroma Data of Four Fruit Types by ABC-Based Neural Network. *Sensors*, vol. 16, no. 3, pp. 304. ISSN 1424-8220. DOI 10.3390/s16030304.

[4]. LEE, M., HWANG, K., PARK, J., CHOI, S., SHIN, S. y SUNG, W., 2016. FPGA-Based Low-Power Speech Recognition with Recurrent Neural Networks. *2016 IEEE International Workshop on Signal Processing Systems (SiPS)*. S.l.: s.n., pp. 230-235. DOI 10.1109/SiPS.2016.48.

[5]. SOLTANI, S., SAGDUYU, Y.E., HASAN, R., DAVASLIOGLU, K., DENG, H. y ERPEK, T., 2019. Real-Time and Embedded Deep Learning on FPGA for RF Signal Classification. *MILCOM 2019 - 2019 IEEE Military Communications Conference (MILCOM)*. S.l.: s.n., pp. 1-6. DOI 10.1109/MILCOM47813.2019.9021098.

[6]. JIA, T., GUO, T., WANG, X., ZHAO, D., WANG, C., ZHANG, Z., LEI, S., LIU, W., LIU, H. y LI, X., 2019. Mixed Natural Gas Online Recognition Device Based on a Neural Network Algorithm Implemented by an FPGA. *Sensors*, vol. 19, no. 9, pp. 2090. DOI 10.3390/s19092090.

[7]. FONOLLOSA, J., SHEIK, S., HUERTA, R. y MARCO, S., 2015. Reservoir computing compensates slow response of chemosensor arrays exposed to fast varying gas concentrations in continuous monitoring. *Sensors and Actuators B: Chemical*, vol. 215, pp. 618-629. ISSN 09254005. DOI 10.1016/j.snb.2015.03.028.

[8]. XILINX, 2020. *Vivado Design Suite User Guide: High-Level Synthesis (UG902)* [en línea]. 2020. [Consulta: 11 noviembre 2020]. Disponible en: https://www.xilinx.com/support/documentation/sw_manuals/xilinx2019_2/ug902-vivado-high-level-synthesis.pdf.

Tabla 1. Comparación de los errores promedio para las diferentes etapas del modelo neuronal

Fase \ Error	Error promedio CH_4 (%)	Error promedio C_2H_4 (%)
Primer entrenamiento	2,4	3,1
Segundo entrenamiento	1,7	2,1
Simulación/ FPGA	2,5	2,5

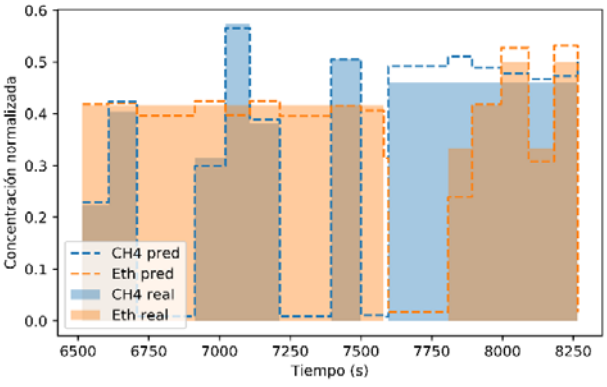

Figura 1: Ventana temporal de la serie con los resultados del primer entrenamiento.

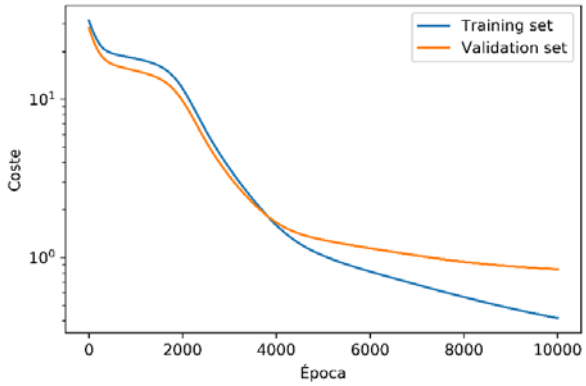

Figura 2: Evolución del coste del *training set* y del *validation set* durante el primer entrenamiento.

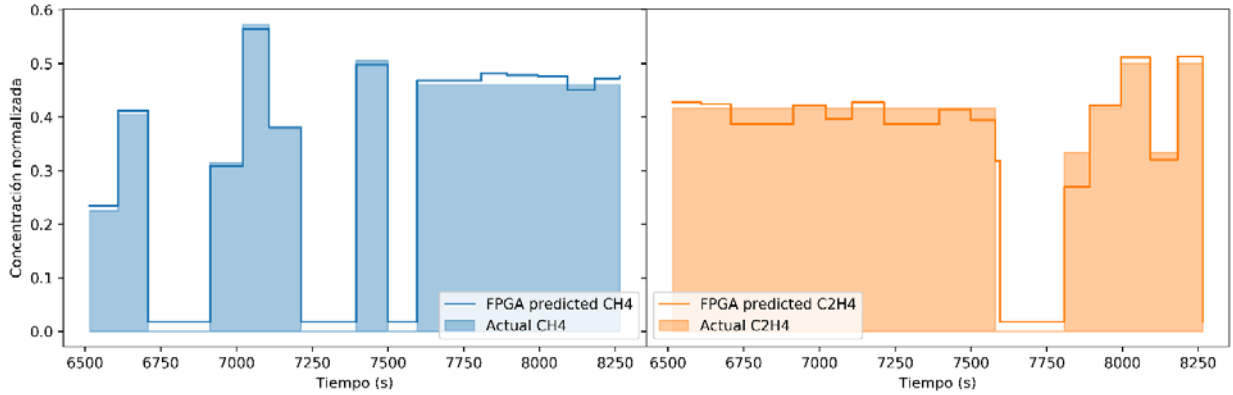

Figura 3: Ventana temporal de la serie con los resultados obtenidos con el modelo neuronal implementado en la FPGA.