Nuevas arquitecturas hardware de procesamiento de alto rendimiento para aprendizaje profundo

Antonio J. Rivera, Francisco Charte, Macarena Espinilla, and María D. Pérez-Godoy

Departamento de Informática, Universidad de Jaén.
{arivera,fcharte,mestevez,lperez}@ujaen.es

Resumen El diseño y fabricación de hardware es costoso, tanto en tiempo como en inversión económica, razón por la que los circuitos integrados se fabrican siempre en gran volumen, para aprovechar la economía de escala. Por esa razón la mayoría de procesadores fabricados son de propósito general, ampliando así su campo de aplicaciones. En los últimos años, sin embargo, cada vez se fabrican más procesadores para aplicaciones específicas, entre ellos aquellos destinados a acelerar el trabajo con redes neuronales profundas. Este artículo introduce la necesidad de este tipo de hardware especializado, describiendo su finalidad, funcionamiento e implementaciones actuales.

Palabras clave: procesador, arquitectura de computadores, aprendizaje profundo, TPU.

Abstract The design and manufacture of hardware is costly, both in terms of time and economic investment, which is why integrated circuits are always manufactured in large volumes, to take advantage of economies of scale. For this reason, the majority of processors manufactured are general purpose, thus broadening their scope of application. In recent years, however, more and more processors have been manufactured for specific applications, including those designed to accelerate work with deep neural networks. This article introduces the need for this type of specialized hardware, describing its purpose, operation and current implementations.

Keywords: processor, computer architecture, deep learning, TPU.

1. Introducción

Durante las últimas décadas el microprocesador, un circuito integrado de propósito general programable por software que básicamente implementa una CPU (*Central Processing Unit*) [1], ha sido el centro de una gran parte de los desarrollos dirigidos a tareas de procesamiento de datos, indistintamente de aspectos como el volumen de información a tratar o la complejidad de los algoritmos a aplicar. El microprocesador, en sus diversas variantes, es asimismo el núcleo de microcontroladores [2], muy usados en sistemas empotrados de tipo industrial, y los SoC (*System-on-Chip*) [3], corazón de dispositivos IoT (*Internet-of-Things*) [4], teléfonos móviles y muchos otros equipos digitales.

La demanda de potencia de procesamiento se ha incrementado de forma muy notable a lo largo de los últimos diez años, siendo una de las razones principales la aparición de nuevos métodos de aprendizaje automático, especialmente los basados en técnicas de aprendizaje profundo o *deep learning* (DL) [5]. Son estos métodos los que hacen posible que nuestros actuales dispositivos móviles puedan procesar instrucciones expresadas en lenguaje natural, facilitando la interacción con la personas, o sean capaces de reconocer caras, simplificando el proceso de identificación de acceso al propio dispositivo.

Satisfacer dicha demanda de potencia no es posible sencillamente incrementando la frecuencia de reloj que rige el funcionamiento de una CPU clásica. El incremento en el nivel de paralelismo es un factor clave, así como la especialización de las unidades de ejecución a la hora de tratar ciertos tipos de datos. Los actuales microprocesadores de alta gama incorporan decenas de núcleos de ejecución, actuando cada uno de ellos como una CPU de propósito general. Además, dichos núcleos cuentan con instrucciones específicas de tipo SIMD (*Simple Instruction Multiple Data*) [6], las distintas versiones de MMX (*MultiMedia eXtensions*), SSE (*Streaming SIMD Extensions*) y AVX (*Advanced Vector eXtensions*), que aceleran la computación con tipos de datos específicos.

A pesar de los avances en la arquitectura de los microprocesadores, son muchos los escenarios en que la potencia de procesamiento que ofrecen resulta insuficiente. Es la razón principal de que las GPU (*Graphics Processing Units*) [7], inicialmente diseñadas para el tratamiento de texturas, cálculo de transformaciones sobre los vértices de geometrías y procesamiento de millones de píxeles, lleven años usándose para acelerar la creación de los modelos de DL. A diferencia de las CPU, con unas pocas decenas de núcleos de procesamiento [8] en la gama más alta (familia *Intel Xeon*[1]), las GPU cuentan con varios miles de núcleos. Estos no son de propósito general como los de una CPU, sino especializados en realizar unas pocas operaciones con tipos de datos fijos. No obstante, esto es lo que se necesita al trabajar con DL, de ahí que el incremento de rendimiento que se experimenta al implementarlos sobre una GPU es muy considerable respecto a una CPU.

El uso de GPU para generar modelos de DL ha influido de manera importante en las estrategias de los propios fabricantes de hardware. Quizá el mayor exponente sea Nvidia, cuyos últimos productos, como las familias Pascal y Volta [9], y su máquina de marketing van específicamente dirigidos a potenciar su uso en el campo del aprendizaje profundo. Paralelamente varios de los más importantes protagonistas de dicho campo, incluyendo a Google, Intel o Microsoft, exploran el desarrollo de nuevas arquitecturas, alternativas a las de las CPU y GPU, diseñadas a medida para incrementar el rendimiento de sus soluciones de reconocimiento de voz, búsquedas, identificación de objetos en imágenes, etc. Son esos nuevos desarrollos, que están dando como fruto circuitos integrados específicos para inteligencia artificial o *IA chips*, los que nos interesan en este artículo.

La estructura del presente artículo es la siguiente: en la sección 2 se introducen los fundamentos de las técnicas de aprendizaje profundo, a fin de definir el origen de las nuevas necesidades surgidas en los últimos años. La sección 3 describe cómo se

[1] La familia de coprocesadores Xeon Phi, con núcleos x86 pero basada en un diseño GPU de Intel, ofrece un mayor número de núcleos que las CPU clásicas y contempla el desarrollo de soluciones con un enfoque similar.

han cubierto dichas necesidades hasta ahora, antes de la aparición de las arquitecturas hardware enumeradas en la sección 4. Este trabajo termina con algunas reflexiones recogidas en la sección 5.

2. Aprendizaje profundo y cálculo tensorial

Actualmente los sistemas más avanzados para el tratamiento de lenguaje natural, identificación de patrones en imágenes, traducción automatizada y reconocimiento de personas, por mencionar algunas de las aplicaciones que estamos acostumbrados a usar en nuestros dispositivos, están principalmente basadas en redes neuronales. Para ser más específicos, se recurre a distintos modelos de lo que conoce genéricamente como aprendizaje profundo o DL, entre ellos las *Deep Belief Networks* o DBN [10], las redes neuronales convolucionales (CNN) [11], los *auto-encoders* [12] o las redes *Long Short-Term Memory* (LSTM) [13]. Los fundamentos de DL surgieron [14] hace más de 30 años, pero no fue hasta más de dos décadas después cuando se propusieron los primeros algoritmos de entrenamiento [10] que hicieron que su uso resultase práctico.

2.1. Cómo funciona una red neuronal profunda

En contraposición a las redes neuronales tradicionales (véase la Figura 1), en las que habitualmente existe una capa de entrada (unidades en negro), una capa oculta o interna (unidades en blanco) y una capa de salida[2] (unidades en gris), las redes neuronales profundas (Figura 2) suelen contar con múltiples capas ocultas. En ciertas tareas, como es el reconocimiento de patrones en imágenes, el número de capas suele ser de varias decenas, llegándose incluso a superar el centenar [15]. Como es obvio, a mayor número de capas más parámetros habrá que ajustar en la red neuronal para que su funcionamiento sea apropiado.

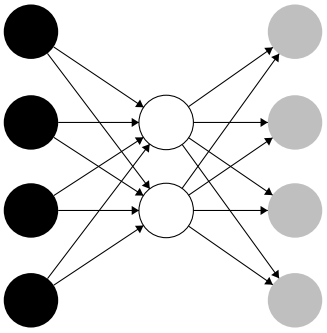

Figura 1. Red neuronal clásica, con capa de entrada, oculta y de salida.

[2] El número de unidades/neuronas en cada capa puede diferir según el tipo de red, no siendo obligatorio que el número de salidas y de entradas coincidan.

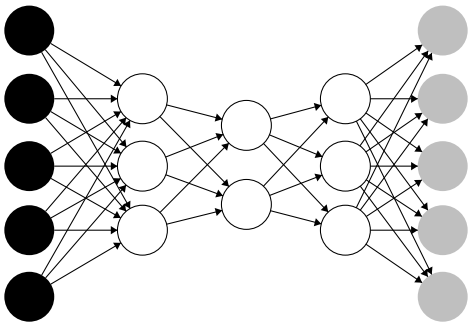

Figura 2. Red neuronal profunda con tres capas ocultas.

En la mayoría de los casos una red neuronal se emplea como herramienta predictiva, de forma que, a partir de unos valores de entrada conocidos, el vector de atributos, se obtienen uno o más resultados, el vector de salidas. Sin entrar en detalles que quedarían fuera del ámbito de este artículo, el funcionamiento básico esencial sería el descrito a continuación.

Para transformar las entradas en las salidas se efectúan operaciones sobre los valores introducidos en la primera capa, a medida que dichos valores van viajando por la red de izquierda a derecha:

1. Cada valor original de entrada es enviado a cada neurona en la capa oculta, previa multiplicación por un peso que está asociado a la conexión entre neuronas.
2. Todas las entradas que recibe cada neurona de la capa oculta se agregan.
3. Se aplica una función de activación al resultado de la agregación anterior, obteniéndose un valor de salida para cada neurona.
4. La salida de la neurona en la capa N de la red se envía a las neuronas de la capa N+1, hasta alcanzar la de salida.

Los pasos fundamentales de este algoritmo, conocido como *feed-forward propagation*[3], pueden ser expresados de forma matricial. Llamando X al vector de entradas, compuesto por valores $x_1, ..., x_f$ siendo f el número de atributos (neuronas en la capa de entrada); W a la matriz de pesos, con valores $w_{1,1}, w_{1,2}, ..., w_{1,f}, ..., w_{m,f}$ siendo m el número de neuronas en la capa oculta, y h la función de activación de las unidades en dicha capa, se obtendría la salida Y de esa capa según se muestra en la Eq. 1. La función h se aplicaría sobre la matriz resultante componente a componente. Esto es generalizable para una red con L capas, obteniéndose las salidas para la capa l a partir de aplicar iterativamente el proceso anterior a la capa $l-1$.

$$X = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}, W = \begin{bmatrix} w_{1,1}\ w_{2,1} \\ w_{1,2}\ w_{2,2} \\ w_{1,3}\ w_{2,3} \end{bmatrix}, Y = h(W^T X) \tag{1}$$

[3] Por simplicidad no se ha incluido aquí el vector de sesgos (*bias*) asociado a cada capa de la red a excepción de la entrada, ni se ha entrado tampoco a describir el algoritmo de propagación hacia atrás (*backpropagation*) encargado de calcular el error cometido y ajustar la matriz de pesos.

El cálculo anterior habría que realizarlo L veces, tantas como capas tenga la red, por cada una de las N muestras de datos de que se disponga. Además, esas N muestras habitualmente se entregarán al algoritmo múltiples veces (*epochs*) hasta que la red se estabilice. Esto implica realizar miles o millones de multiplicaciones de matrices. El tiempo necesario para ello dependerá lógicamente del tamaño de estas, pero de partida este es un tipo de operación muy exigente en cuanto a demanda de potencia computacional.

2.2. Redes neuronales y tipos de datos complejos

Las redes neuronales tradicionales, como el conocido MLP (*Multi-Layer Perceptron*), se han venido aplicando a conjuntos de datos en los que cada muestra está compuesta por unos pocos valores, habitualmente de tipo real. En contraposición, las redes neuronales profundas se están empleando para procesar tipos de datos de mayor complejidad, tales como imágenes, audio o vídeo. Este hecho tiene una influencia esencial en la potencia de procesamiento necesaria para ajustar dichas redes, ya que se opera con estructuras de datos de entrada mucho más grandes.

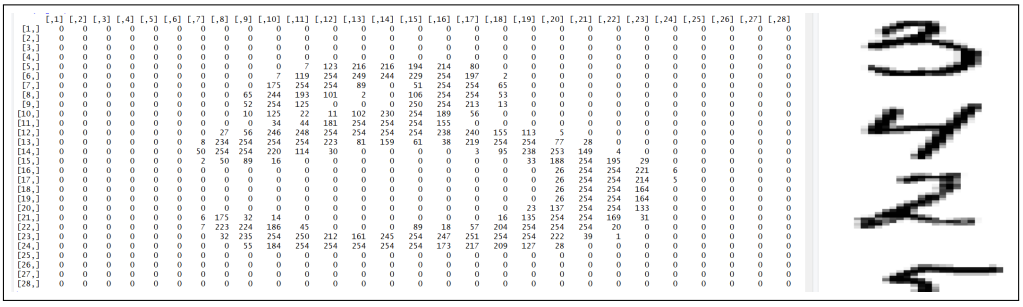

Figura 3. Algunos dígitos de MNIST y la matriz de 28x28 valores enteros que definen el primero de ellos.

Una imagen en blanco y negro puede codificarse como un vector de valores binarios. Análogamente, las imágenes en escalas de grises pueden representarse como vectores de valores en un cierto intervalo. El conocido conjunto de datos MNIST [16], compuesto de 70 000 imágenes de 28×28 píxeles (véase la Figura 3) correspondientes a dígitos manuscritos, es procesado habitualmente como un vector de 784 valores enteros entre 0 y 255 más el atributo de clase. Dependiendo del tipo de red neuronal con la que vaya a tratarse esta información, será posible tomarla como un vector de 784 valores (MLP, autoencoders, etc.) o bien como una matriz bidimensional de 28×28 elementos (redes tipo CNN).

La mayor parte de los conjuntos de datos no están compuestos de imágenes en blanco y negro o en escalas de grises, sino que se generan a partir de dispositivos de captura que emplean tres o más canales a fin de obtener imágenes en color y multi-

espectrales. Lo más habitual es que los canales sean tres[4]: R, G y B, tal y como se muestra en la Figura 4. Cada imagen, por tanto, se representa con tres dimensiones: dos permiten direccionar un píxel cualquiera y la tercera selecciona el canal.

Figura 4. Imagen en color y su descomposición en canales R, G y B.

El siguiente nivel, en cuanto a complejidad de los datos se refiere, sería el procesamiento automatizado de vídeo, por regla general con el objetivo de detectar la presencia de ciertos objetos. Se trata de una secuencia de imágenes, cada una de las cuales se compone de varios canales formados por píxeles en una distribución bidimensional. Se precisa, por tanto, una dimensión adicional, alcanzando un total de cuatro.

Las operaciones descritas en la sección previa, mediante las cuales la información de origen va procesándose en la red neuronal, han de adaptarse convenientemente para trabajar no ya con un vector más o menos grande de valores, sino con estructuras de datos considerablemente más complejas.

[4] Ciertos formatos de imagen, como el estándar PNG, suelen incluir un cuarto canal conocido como *alpha*. Su finalidad es facilitar la información de transparencia para cada píxel de la imagen.

2.3. Qué es un tensor y para qué sirve

Actualmente existen múltiples *frameworks* que ofrecen a desarrolladores e investigadores las herramientas necesarias para crear todo tipo de redes profundas. Seguramente el más conocido sea TensorFlow [17], creado por Google. En esta y otras librerías de servicios se emplea recurrentemente el término *tensor*, haciéndose referencia al cálculo tensorial como el procedimiento para aplicar a datos complejos, como los descritos en la sección previa, las operaciones necesarias para conseguir la propagación hacia adelante en la red neuronal y posterior vuelta atrás para el ajuste de pesos.

En física a los tensores suele otorgárseles una interpretación geométrica. Un vector, compuesto de una magnitud y una dirección, es un tensor de orden 1. En un espacio tridimensional dicho tensor podría tomar la forma (M_x, N_y, O_z), siendo x, y y z los ejes de coordenadas y M, N y O la magnitud en cada eje. En este caso tenemos $3^1 = 3$ componentes. Si dicho vector denota la fuerza que actúa sobre un plano, se precisará una componente adicional para establecer la normal a dicha superficie, obteniéndose un tensor de orden 2. Este tendría $3^2 = 9$ componentes (véase la Figura 5), en lugar de los tres previos. Las magnitudes se asociarían a las 9 combinaciones posibles de ejes: $xx', xy', xz', yx', yy', yz', zx', zy', zz'$, por lo que el resultado se dispone habitualmente en forma de matriz bidimensional. Existen situaciones de mayor complejidad, en las que el tensor sería de un orden superior a 2.

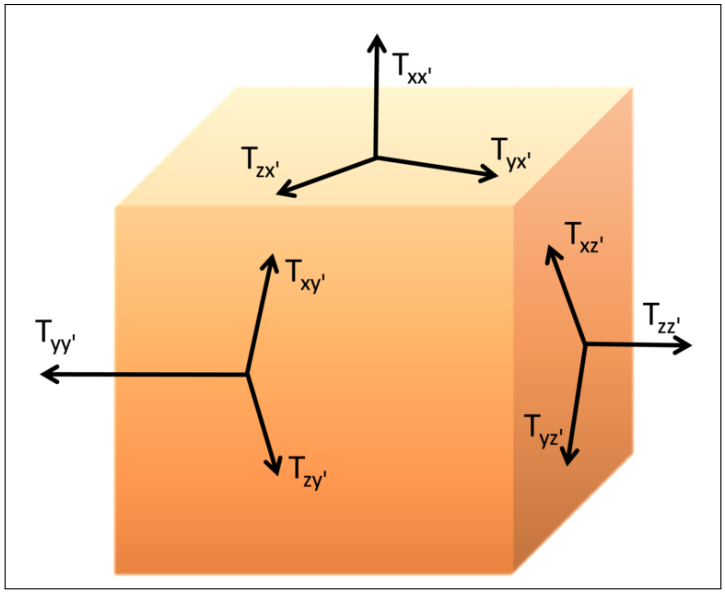

Figura 5. Representación geométrica de un tensor de orden 2.

La definición matemática de tensor no es trivial [18], estando implicados aspectos como los espacios vectoriales, espacios duales, aplicaciones multilineales, etc. El álgebra tensorial, aplicable a este tipo de objetos, básicamente extiende los conceptos y métodos

empleados en el álgebra lineal clásica. Esto permite actuar sobre tensores mediante operaciones como el producto, habitual entre matrices, pero extendiéndolas a un mayor número de dimensiones.

Para los desarrolladores de software un tensor es una estructura de datos, como lo son los vectores o las matrices. Habitualmente se habla de tensor cuando el número de dimensiones es superior a 2. En este contexto, el orden de un tensor haría referencia al número de índices que son necesarios para acceder a un componente de esa estructura de datos. Así, una referencia del tipo `logo[x][y][p]`, usando tres índices, servirían para acceder a cualquier píxel de una imagen mediante sus coordenadas y el plano de color. Lo interesante es que las descomposiciones habituales en álgebra tensorial permiten trabajar sobre estas estructuras de datos de una forma más eficiente, acelerando ciertas operaciones tal y como se describe en [19].

3. Paralelismo en CPU y GPU

Las operaciones sobre matrices mediante álgebra lineal, especialmente cuando son de gran tamaño, demandan una gran potencia de procesamiento. Suponiendo que tenemos una matriz $A_{m\times n}$ y otra $B_{n\times p}$, la multiplicación de ambas tendría una complejidad computacional de $\mathcal{O}(mnp)$ que, en el caso de $m = n = p$ sería $\mathcal{O}(n^3)$ o complejidad cúbica.

Suponiendo que tuviésemos como entrada una pequeña imagen monocroma de 100×100 píxeles, su multiplicación con un filtro de similar tamaño precisaría 1 000 000 de operaciones simples. Dicho trabajo hay que realizarlo multitud de veces, dependiendo del número de capas y unidades por capa de la red neuronal, por lo que estaríamos hablando de centenares o miles de millones de operaciones por cada muestra de datos. En cuanto incrementemos el tamaño de las imágenes y su complejidad, pasando a trabajar en color, con imágenes multi-esprectrales o con vídeo, estaríamos hablando de varios órdenes de magnitud por encima. Al recurrir al álgebra tensorial es posible acelerar tareas como la multiplicación de matrices, pero aun así estaríamos afrontando un cantidad enorme de operaciones.

Por regla general una red profunda precisa de muchos miles o incluso millones de muestras de datos para generar un modelo estable. Habitualmente cada muestra es facilitada a la red múltiples veces. Estos son factores multiplicativos que se agregan a las operaciones antes descritas. En conclusión, es fácil darse cuenta de que el entrenamiento de una de estas redes exige una gran potencia computacional y, en consecuencia, mucho tiempo de procesamiento. Esto es especialmente cierto si el algoritmo se ejecuta en una arquitectura hardware que no está específicamente diseñada para ofrecer una alto nivel de paralelismo, como es el caso de las CPU.

A pesar de que el nivel de paralelismo de las CPU se ha ido incrementando paulatinamente [8], primero con técnicas de segmentación del cauce y la incorporación de múltiples unidades funcionales [20], después mediante la incorporación de conjuntos de instrucciones SIMD [6] y, finalmente, introduciendo múltiples núcleos o *cores* en cada CPU, lo cierto es que la arquitectura clásica de una CPU, diseñada como un procesador de propósito general que busca una eficiencia homógenea en todo tipo de aplicaciones, no es la más adecuada para afrontar algoritmos de entrenamiento de redes profundas.

Por ejemplo, el tiempo necesario para procesar por completo una imagen de 224×224 píxeles, incluyendo la propagación hacia adelante y hacia atrás, con ResNET-200 [21], una red profunda con 200 capas, es de más de 22 segundos en una CPU Dual Xeon E5-2630[5]. El entrenamiento de un conjunto de datos completo, hasta obtener la red preparada para predecir, en una configuración así podría tardar semanas.

La arquitectura de una GPU moderna [7] es completamente diferente a la de una CPU. Un microprocesador dedica una pequeña fracción de su circuitería a componentes de cálculo, la ALU (*Arithmetic Logic Unit*). Esencialmente hay una ALU, con un reducido número de unidades funcionales, por cada núcleo o *core*. En contraposición, gran parte de la circuitería de una GPU la conforman los miles de núcleos de que consta (bloques verdes en la Figura 6). Cada uno de ellos es mucho más simple que un núcleo de CPU, al estar especializado en un reducido número de operaciones con tipos de datos fijos, pero en conjunto ofrecen un alto nivel de paralelismo. A esto se suma que el tamaño del circuito integrado, y por tanto número de componentes, de una GPU suele ser mucho mayor[6] que el de una CPU.

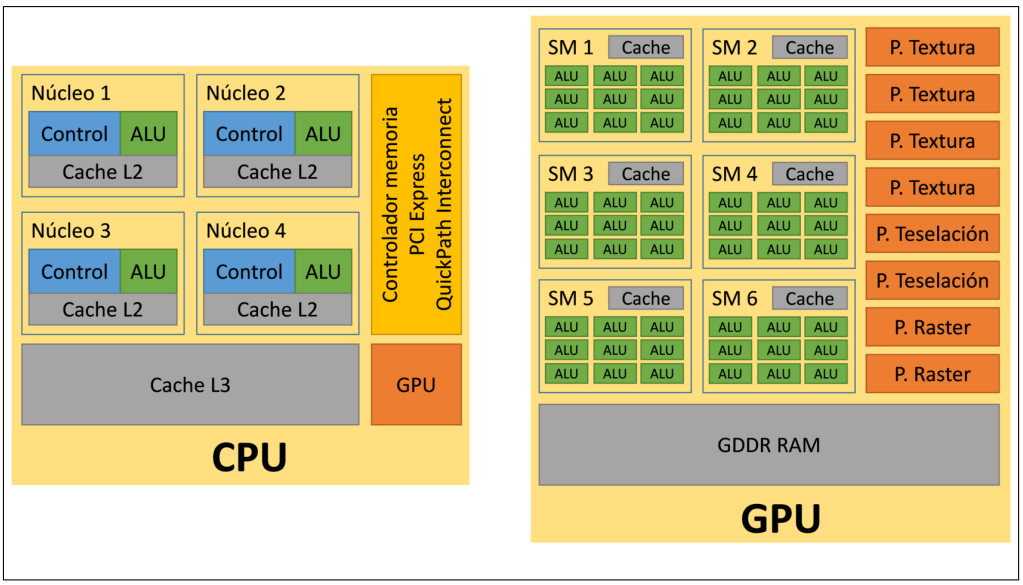

Figura 6. Esquema de bloques de una CPU y una GPU. En color verde puede apreciarse la cantidad de circuitería dedicada a la realización de cálculos, mucho mayor en la GPU.

[5] Una configuración de 2 procesadores, cada uno de los cuales cuenta con 8 núcleos capaces de ejecutar 2 hilos en paralelo, lo que hace un total de 32 hilos concurrentes.

[6] Una CPU de 2017 de alta gama cuenta típicamente con 8 núcleos, tiene un superficie de en torno a 200 mm^2 y unos 5 000 millones de transistores, mientras que una GPU del mismo año supera los 3 000 núcleos, 800 mm^2 y 20 000 millones de transistores.

El uso de GPU para ejecutar modelos de aprendizaje profundo se ha incrementado notablemente a lo largo de la última década y es, en parte, la técnica que ha hecho posible la existencia de muchos servicios que de otra forma habrían tardado mucho más en estar disponibles. La multiplicación de matrices es altamente paralelizable, permitiendo reducir los 22 segundos antes indicados para una CPU a entre 0.2 y 0.5 segundos, dependiendo de la GPU usada. Una sola GPU tiene capacidad, en este contexto, para realizar el trabajo de decenas o cientos de CPU, lo cual también contribuye a abaratar el hardware necesario para afrontar un cierto problema.

4. Tensor Processing Unit, la arquitectura para DL

A pesar de la impresionante mejora, en cuanto a rendimiento, que supone el uso de GPU en lugar de CPU para abordar el entrenamiento de redes profundas, una GPU no está específicamente diseñada para esta tarea. En la misma coexisten los núcleos básicos de cálculo, equivalentes a una ALU de CPU simplificada y que son los que usan los *frameworks* tipo TensorFlow, con otras unidades especializadas, pensadas para operar con texturas, vértices, etc. Estas ocupan espacio en el circuito integrado y consumen energía.

Una unidad de procesamiento de tensores (TPU) es un circuito integrado diseñado específicamente para llevar a cabo operaciones de álgebra tensorial, en especial multiplicaciones. Esta idea surgió en Google hace más de una década, aunque no fue hasta 2016 cuando dicha empresa anunció la primera versión de TPU. En los últimos dos años no solo Google ha perfeccionado su diseño, con dos nuevas versiones de TPU, sino que muchos otros fabricantes han diseñado sus propios *chips* específicos para Inteligencia artificial. La razón para ello es la sustancial ganancia en rendimiento [22] respecto al uso de GPU, al tiempo que se reduce el consumo energético.

4.1. ¿Qué es una TPU?

Como se ha indicado antes, una TPU es una unidad de cómputo especializada en operar con tensores. Se trata, por tanto, de un circuito tipo ASIC (*Application-Specific Integrated Circuit*). Reduciéndolo a una frase, podríamos decir que una TPU es un circuito con gran número de núcleos, muchos más que los ofrecidos por una GPU, especializados en realizar operaciones compuestas sobre tipos de datos simples. Esto les permite incrementar el rendimiento en varios órdenes de magnitud respecto a las CPU, multiplicando por entre 30 y 50 veces el ofrecido por las GPU. La mejora en cuanto a rendimiento/consumo es incluso superior.

Para entender mejor la anterior descripción, centrémonos en los aspectos clave de la arquitectura de la primera TPU, que son los siguientes:

- **Simplificación de los núcleos.** Una ALU de CPU/GPU permite operar con datos de distintos tamaños, tanto enteros como en punto flotante, y efectuar sobre ellos distintos cálculos. Aplicando un proceso de *cuantización* [23] se representa el rango de valores de entrada empleando únicamente 8 bits y operando con datos

de punto fijo[7]. Esto contribuye tanto a reducir el tamaño de los núcleos como el tiempo necesario para realizar las operaciones, al trabajar sobre un tipo de dato más compacto. En consecuencia, usando la misma superficie se pueden integrar muchos más núcleos, concretamente 65 536 en la primera versión de TPU.

- **Operaciones compuestas.** Desde hace años los procesadores se diseñan siguiendo una filosofía RISC (*Reduced Instruction Set Computer*) [20], ofreciendo un conjunto de instrucciones simples que pueden ser ejecutadas en pocos ciclos. En contraposición una TPU ofrece operaciones de mayor complejidad, con instrucciones compuestas que, por ejemplo, permiten realizar múltiples productos y sumas en un único paso. De esta manera se optimizan las tareas a las que está destinado un circuito de este tipo, fundamentalmente la convolución y la multiplicación de matrices/tensores.
- **Funcionamiento sistólico.** Las ALU tradicionales trabajan bajo el control de una máquina de estados, de forma que en cada ciclo se seleccionan operandos desde los registros del procesador, se ejecuta la operación requerida y se lleva el resultado a un registro de destino, volviéndose al inicio para ejecutar un nuevo ciclo. Parte del tiempo de procesamiento, en consecuencia, se dedica a la recuperación y almacenamiento de operandos. La TPU diseñada por Google tiene un comportamiento calificado como *sistólico*, mediante el cual los operandos se leen una sola vez y se reutilizan en múltiples operaciones consecutivas antes de generar un resultado de salida. La circuitería está diseñada de forma que las ALU adyacentes se transfieren las entradas/salidas entre sí en cada pulso (ciclo de reloj). Este diseño resulta ideal para la multiplicación de matrices y operaciones similares, en las que un mismo operando de entrada se usa en múltiples operaciones escalares simples hasta componer el resultado final.

Además de los núcleos de cálculo una TPU también incorpora una importante cantidad de memoria RAM, como en el caso de las GPU, lo cual les permite almacenar localmente la información que van a procesar. En palabras de los ingenieros de Google [24] la diferencia entre una CPU, una GPU y una TPU sería análoga a la existente entre una impresora que imprime carácter a carácter, otra que lo hace línea a línea y una tercera que imprime páginas completas.

4.2. Arquitectura de la TPU de Google

Simplificando el diseño hasta quedarnos exclusivamente con los componentes esenciales, en la Figura 7 se ha representado mediante un esquema de bloques la arquitectura de la TPU fabricada por Google. Como puede apreciarse, gran parte de la circuitería está dedicada a los 64K núcleos de cálculo, las ALU que operan sobre los datos efectuando operaciones como el producto-y-suma antes citada. Es lo que Google denomina MXU (*Matrix Multiplier Unit*).

Además de un grupo de acumuladores, estrechamente conectados a la MXU, la TPU cuenta con un gran conjunto de registros a los que se denomina genéricamente

[7] Esta representación lógicamente conlleva una pérdida de precisión en los cálculos, pero en general esa disminución es tolerable en la mayoría de escenarios.

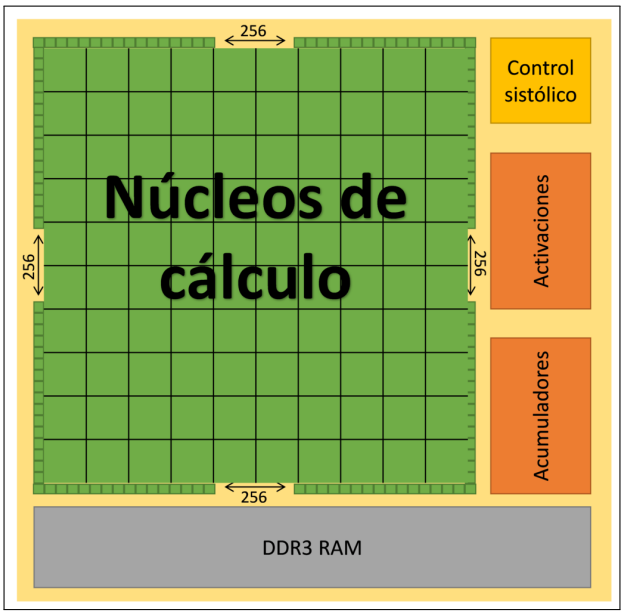

Figura 7. Esquema de bloques simplificado de la primera TPU diseñada por Google.

UB (*Unified Buffer*). Se trata de 24MB de memoria RAM estática (SRAM) usada para almacenar los datos sobre los que se está operando. Asimismo se dispone de una cierta cantidad de memoria tipo DDR, concretamente 8GB en la versión 1 de la TPU.

Al procesar las muestras de datos en una red neuronal, tras las operaciones de multiplicación de matrices, que aplican los pesos a las entradas y las agregan, se procede a convertir el valor resultante mediante una función de activación. Algunas de las funciones de activación más usuales son la sigmoidal, tangente hiperbólica o la conocida como ReLU (*Rectified Linear Unit*). Este tipo de funciones se implementan habitualmente mediante software, de forma que se convierten en un conjunto de operaciones más simple a ejecutar en la CPU/GPU. La TPU, sin embargo, cuenta con una AU (*Activation Unit*) que ofrece dichas funciones implementadas directamente en el hardware, de forma que se pueden calcular de forma mucho más rápida y sin emplear las unidades de cálculo que componen la MXU.

En comparación con las CPU, GPU y otros tipos de circuitos integrados, en la TPU la circuitería dedicada a operaciones de control representa un espacio mínimo, de en torno al 2 % en la primera versión según datos de Google. En contraposición, la MXU, UB y AU ocupan un 60 % del espacio. El resto[8] lo ocupan interfaces de comunicación con la memoria, con el bus PCIe al que se conectan los módulos que sirven de soporte a las TPU, etc. Parte de esos recursos sirven asimismo para conectar las TPU entre sí, formando mallas a las que se denomina comercialmente *TPU Pods*. Por ejemplo, una TPUv2 Pod se compone de 64 dispositivos TPUv2.

[8] En [24] se facilita información ampliada y múltiples esquemas, incluyendo animaciones, describiendo la arquitectura y funcionamiento de Google TPU.

4.3. Rendimiento de una TPU

Más allá de los datos que facilita Google en [24], artículo en el que se deduce que la TPUv1 a 700MHz ejecuta 92 Teraops ($9{,}2 \times 10^{13}$) por segundo, o del estudio de rendimiento aportado en [22], probablemente los mejores indicadores de la potencia que ofrece este tipo de ASIC a la hora de trabajar con redes profundas sean los obtenidos empíricamente, por ejemplo a partir de competiciones en las que se opera con hardware heterogéneo.

Si bien la mayoría de las competiciones se centran casi exclusivamente en el nivel de precisión de las predicciones, la inferencia llevada a cabo por la red neuronal, la Universidad de Stanford ha puesto en marcha DAWNBench[9], una competición cuyo objetivo es determinar el tiempo y recursos que se emplean entrenando y haciendo inferencia con técnicas DL para procesar un conjunto de datos predeterminado. Para ello se fija un nivel mínimo de precisión que es necesario alcanzar: el 93 %.

A fecha de junio de 2018 el primer puesto del ranking en esta competición lo ocupa una implementación de ResNet50 ejecutada usando la mitad de una TPUv2 Pod, consiguiéndose la precisión mínima en algo más de 30 minutos. La configuración más rápida usando GPU, también basada en ResNet50, tarda casi 3 horas. Para conseguir realizar el trabajo en ese tiempo se ha usado una instancia de AWS (*Amazon Web Services*) de tipo *p3.16xlarge*, compuesta de 8 GPU Tesla V100, 128GB de memoria en GPU y 488GB de RAM. Usando exclusivamente CPU es necesario usar 64 nodos, cada uno de ellos con Intel Xeon Platinum de 72 núcleos y 144GB de RAM, y se tardan algo más de 6 horas en completar la tarea.

Como es fácil apreciar a partir de los datos previos, en el caso de GPU y CPU se precisaría una configuración hardware con un coste muy alto y, aun así, no se llegaría a alcanzar el mismo rendimiento que el ofrecido por una TPU. En los casos citados se trata de configuraciones en la nube de pago por uso, su adquisición tendría un precio fuera del alcance de la mayoría de usuarios y de muchas empresas.

4.4. Procesadores para IA y aceleradores de redes neuronales

Habiendo conocido su finalidad y fundamentos de funcionamiento, cabe preguntarse qué productos existen actualmente disponibles para cualquiera interesado en trabajar con procesadores IA, tipo TPU, u otro tipo de aceleradores para redes neuronales.

En los anteriores apartados se ha tomado como referencia la TPU desarrollada por Google al ser el primer dispositivo de este tipo del que se hizo pública información al respecto, pero desde entonces han sido varios los fabricantes que no han querido perder su parte de este mercado, entre ellos algunos tan importantes como Intel o NVidia. En muchos casos este hardware especializado no se vende, sino que es ofrecido como un servicio por el que se paga según el tiempo de uso.

A continuación se facilita una enumeración no exhaustiva del hardware para DL desarrollado por distintas empresas, junto con un resumen de sus características públicas más destacables.

[9] `http://dawn.cs.stanford.edu/benchmark`

Google Cloud TPU `cloud.google.com/tpu`
Inicialmente las TPU fueron desarrolladas en Google para uso interno en sus propios centros de procesamiento de datos, pero desde la segunda versión esta empresa ofrece un servicio denominado *Google Cloud TPU*. Cada instancia de este servicio consta de de cuatro circuitos integrados que albergan dos núcleos TPU cada uno, haciendo un total de 8 dispositivos TPU. Cada uno de estos tiene una MXU con 16 384 núcleos de cálculo (128×128) que, a diferencia de los anteriormente descritos, operan con datos en un formato compacto de punto flotante de 16 bits, en lugar de enteros de 8 bits. Según Google, una Cloud TPU ofrece 180 teraflops y 64GB de memoria. Se ofrecen distintas herramientas para desarrolladores, que han de crear sus modelos usando TensorFlow, incluyendo algunas redes profundas muy conocidas ya preparadas para su uso.

Intel Nervana `ai.intel.com/intel-nervana-neural-network-processor`
Esta es la denominación comercial que Intel da a sus NNP (*Neural Network Processor*), cuyo primer producto se anunció a mitad del pasado 2017. Aunque la información pública sobre Nervana es aun escasa, se conocen dos de los aspectos clave que han influido en su diseño. Por una parte se utiliza Flexpoint [25], una representación alternativa al estándar IEEE FP16 para operar con tensores en punto flotante, en los núcleos de cálculo. Flexpoint es más compacto, permitiendo incorporar más núcleos en el mismo espacio, y ofrece mejor rendimiento que el cálculo en punto flotante estándar. Por otra parte, Intel ha trabajado en reducir el principal cuello de botella: el acceso a memoria. Nervana dispone de un controlador con capacidad para 1.2 terabit/s de ancho de banda bidireccional. Recientemente el fabricante ha presentado Nervana Neural Net L-1000, un servicio que permitirá a desarrolladores usar estos NNP si bien no se espera que esté disponible hasta 2019.

Nvidia Tensor Core Unit `www.nvidia.com/data-center/tesla-v100`
La última generación de hardware de Nvidia, denominada Volta y que da lugar a la familia de productos V100, se diferencia esencialmente de todas las anteriores por incorporar, además de los habituales núcleos CUDA, un nuevo tipo de núcleo: el TCU (*Tensor Core Unit*) [26]. Concretamente se cuenta con 640 núcleos de este tipo, siendo capaz cada uno de ejecutar 64 operaciones por ciclo usando precisión mixta: multiplicaciones en 16 bits y suma de agregación en 32 bits. Estos nuevos núcleos están diseñados para que los desarrolladores creen sus soluciones usando TensorFlow o una biblioteca similar. Además también se ha incrementado el ancho de banda entre la memoria y las unidades de cómputo, acercándose a 1 terabit/s. A diferencia de los dos productos previos, este puede adquirirse físicamente. A los TCU hay que añadir que la familia V100 cuenta con 5 120 núcleos CUDA FP32 y 2 560 núcleos FP64, lo cual permite construir soluciones mixtas en las que se explota el paralelismo clásico en GPU y el cálculo tensorial en TCU.

ARM Trillium `www.arm.com/products/processors/machine-learning`
La empresa ARM es conocida por sus diseños de procesador que, fabricados por terceros, hacen funcionar miles de millones de dispositivos móviles. La demanda para que

estos procesen vídeo, lenguaje natural o realicen traducciones ha llevado a la creación de Trillium por parte de ARM. Se trata de tres circuitos integrados diseñados para tareas específicas: ARM ML (*Machine Learning*), ARM OD (*Object Detection*) y ARM NN (*Neural Network*). No son productos comparables a los anteriores, ya que su objetivo es ofrecer el mejor rendimiento con el menor consumo posible, facilitando así su integración en tabletas, teléfonos móviles y dispositivos similares.

Microsoft Brainwave `aka.ms/aml-real-time-ai`
La fabricación de ASIC no es la única vía para implementar arquitecturas hardware a medida para la aceleración de los cálculos al trabajar con modelos DL, tal y como ha demostrado Microsoft con su reciente anuncio (en la *Build 2018 Conference*) del proyecto Brainwave. En este se ha recurrido a una FPGA [27] para implementar lo que denominan *soft* DPU (*DNN Processing Unit*), un tipo de procesador para redes neuronales profundas que se caracteriza por ser reconfigurable (*soft*), en contraposición a la arquitectura fija de las TPU y similares (*hard*). Esto permite, por ejemplo, ajustar la implementación de la DPU para el tipo de dato específico[10] que se precise en un determinado modelo, optimizando el espacio usado y mejorando el rendimiento. En realidad la DPU es uno de los tres componentes de Brainwave, proyecto que también incluye una infraestructura de comunicación de muy baja latencia y gran ancho de banda para comunicar las DPU, por una parte, y un compilador y entorno de ejecución que facilitará a los desarrolladores la creación y despliegue de los modelos DL. Se trata de un servicio que estará disponible a través de Microsoft Azure.

Además de productos diseñados para su venta a usuarios finales, son muchos los fabricantes que han desarrollado sus propios ASIC IA para consumo interno, generalmente como parte de SoC para dispositivos móviles. El SoC A11 de Apple, incorporado en el iPhone X, cuenta con un *Neural Engine*. El procesador Kirin 970, usado en algunos dispositivos de Huawei, incorpora una NPU (*Neural Processing Unit*). Los últimos SoC PowerVR tienen como novedad el NNA (*Neural Network Accelerator*). Microsoft ha desarrollado una HPU (*Holographic Processing Unit*) para sus HoloLens que cuenta con un coprocesador IA. La lista seguirá creciendo, ya que la tendencia a incorporar hardware para mejorar el rendimiento de tareas como las ya mencionadas no se detendrá en los dispositivos de más alta gama, como ocurre ahora.

5. Conclusiones

Las técnicas de DL, empleadas para entrenar y realizar inferencia con redes neuronales profundas, cada vez se aplican a más campos, incluyendo la traducción automatizada, el procesamiento de lenguaje natural que permite a los usuarios dar instrucciones a sus dispositivos, el etiquetado automático de personas en fotografías o la identificación biométrica, entre otros. Los modelos DL tienden a incrementar su complejidad, al tratar con tipos de datos cada vez más complejos o sencillamente para mejorar su

[10] Virtualmente es posible operar con datos de cualquier precisión y tamaño simplemente personalizando la descripción hardware que después se enviará a la FPGA.

nivel de acierto, lo cual se traduce en una mayor demanda computacional. Esta supera en muchos casos lo que pueden ofrecer las CPU y GPU, lo cual justifica el desarrollo de hardware a medida (ASIC) para satisfacer dicha demanda.

En este artículo se han explicado los fundamentos del funcionamiento de una red neuronal profunda, introduciendo el concepto de cálculo tensorial como una extensión del álgebra lineal. La multiplicación de matrices y tensores, operación parcialmente paralelizable en una CPU/GPU, se optimiza en las TPU, cuya arquitectura general se ha descrito antes de proceder a realizar una revisión de los distintos productos que varias empresas fabrican actualmente.

Referencias

1. D. López Talavera, C. Rus Casas, F. Charte Ojeda, Estructura y tecnología de computadores, Anaya, ISBN: 84-415-2606-8, 2009.
2. E. Martín Cuenca, J. M. Angulo Usategui, I. Angulo Martínez, Microcontroladores PIC. La solución en un chip, ITP-Paraninfo, ISBN: 84-283-2371-2, 1999.
3. T. L. Floyd, Digital Fundamentals, 10/e, Pearson Education, 2011.
4. Wikipedia, Internet de las cosas.
   URL `https://es.wikipedia.org/wiki/Internet_de_las_cosas`
5. I. Goodfellow, Y. Bengio, A. Courville, Deep Learning, MIT Press, 2016, `http://www.deeplearningbook.org`.
6. C. J. Hughes, Single-Instruction Multiple-Data Execution, 2015. `doi:10.2200/S00647ED1V01Y201505CAC032`.
7. F. Charte, A. J. Rueda, M. Espinilla, R. A. J., Evolución tecnológica del hardware de vídeo y las GPU en los ordenadores personales, Enseñanza y Aprendizaje de Ingeniería de Computadores (7) (2017) 111–128.
   URL `http://hdl.handle.net/10481/47376`
8. F. Charte, A. J. Rivera, F. J. Pulgar, M. J. d. Jesús, Explotación de la potencia de procesamiento mediante paralelismo: un recorrido histórico hasta la GPGPU, Enseñanza y Aprendizaje de Ingeniería de Computadores (6) (2016) 19–33.
   URL `http://hdl.handle.net/10481/41910`
9. Nvidia, The New GPU Architecture Designed to Bring AI to Every Industry.
   URL `https://www.nvidia.com/en-us/data-center/volta-gpu-architecture/`
10. G. E. Hinton, S. Osindero, Y.-W. Teh, A fast learning algorithm for deep belief nets, Neural computation 18 (7) (2006) 1527–1554.
11. W. Rawat, Z. Wang, Deep convolutional neural networks for image classification: A comprehensive review, Neural computation 29:9 (2017) 2352–2449.
12. D. Charte, F. Charte, S. García, M. J. del Jesús, F. Herrera, A practical tutorial on autoencoders for nonlinear feature fusion: Taxonomy, models, software and guidelines, Information Fusion 44 (2018) 78–96.
13. S. Hochreiter, J. Schmidhuber, Long short-term memory, Neural computation 9 8 (1997) 1735–80.
14. D. E. Rumelhart, G. E. Hinton, R. J. Williams, Learning internal representations by error propagation, Tech. rep., DTIC Document (1985).
15. C. Szegedy, S. Ioffe, V. Vanhoucke, A. A. Alemi, Inception-v4, inception-resnet and the impact of residual connections on learning, in: AAAI, 2017.
16. Y. LeCun, C. Cortes, C. J. Burges, The MNIST database of handwritten digits.
   URL `http://yann.lecun.com/exdb/mnist/`

17. M. Abadi, et al., TensorFlow: Large-scale machine learning on heterogeneous systems, software available from tensorflow.org (2015).
URL https://www.tensorflow.org/
18. S. Treil, Linear algebra done wrong, 2016.
URL https://www.math.brown.edu/~treil/papers/LADW/LADW.html
19. T. G. Kolda, B. W. Bader, Tensor decompositions and applications, SIAM review 51 (3) (2009) 455–500.
20. J. Ortega Lopera, M. Anguita López, A. Prieto Espinosa, Arquitectura de computadores, Thomson, 2005.
21. K. He, X. Zhang, S. Ren, J. Sun, Identity mappings in deep residual networks, in: European Conference on Computer Vision, Springer, 2016, pp. 630–645.
22. N. P. Jouppi, et al., In-datacenter performance analysis of a tensor processing unit, SIGARCH Comput. Archit. News 45 (2) (2017) 1–12. doi:10.1145/3140659.3080246.
23. D. Lin, S. Talathi, S. Annapureddy, Fixed point quantization of deep convolutional networks, in: International Conference on Machine Learning, 2016, pp. 2849–2858.
24. K. Sato, C. Young, D. Patterson, An in-depth look at Google's first Tensor Processing Unit (TPU) (2017).
URL https://cloud.google.com/blog/big-data/2017/05/an-in-depth-look-at-googles-first-tensor-processing-unit-tpu
25. U. Köster, T. Webb, X. Wang, M. Nassar, A. K. Bansal, W. Constable, O. Elibol, S. Gray, S. Hall, L. Hornof, et al., Flexpoint: An adaptive numerical format for efficient training of deep neural networks, in: Advances in Neural Information Processing Systems, 2017, pp. 1742–1752.
26. S. Markidis, S. W. Der Chien, E. Laure, I. B. Peng, J. S. Vetter, Nvidia tensor core programmability, performance & precision, arXiv preprint arXiv:1803.04014.
27. F. Charte, M. Espinilla, R. A. J., P. F. J., Uso de dispositivos FPGA como apoyo a la enseñanza de asignaturas de arquitectura de computadores, Enseñanza y Aprendizaje de Ingeniería de Computadores (7) (2017) 37–52.
URL http://hdl.handle.net/10481/47371