El aprendizaje automático en sistemas embebidos

Ignacio Latre Ayen
Escuela de Ingeniería y Arquitectura, Universidad de Zaragoza, Zaragoza, España
igna120598@hotmail.com

***Resumen*—Este documento recoge los diferentes métodos existentes en la implementación de técnicas de aprendizaje automático en sistemas embebidos y hace una comparación entre ellos.**

***Palabras clave*—*Machine learning*, *deep learning*, GPU, TPU, NPU, MCU, *edge*, nube, coma fija.**

I. Introducción

La continua evolución de los sensores y de los algoritmos de aprendizaje automático ha desembocado en la necesidad de la implementación de estos en sistemas embebidos donde los límites del propio *hardware* marcan la complejidad del propio algoritmo [1].
A su vez este problema ha generado una gran cantidad de diferentes opciones que van desde la optimización de los algoritmos para minimizar el número de parámetros, hasta el uso de *hardware* específico para su implementación o la inferencia en sistemas externos de forma que se exterioriza el peso de la ejecución.
Es por ello que este articulo recoge las diferentes posibilidades existentes haciendo hincapié en las diferencias y coincidencias existentes.
Este artículo se encuentra estructurado de forma que primero se hace una revisión de las opciones de ejecución externas al dispositivo en la sección II, posteriormente se analizan las técnicas de implementación en el propio *hardware* del sistema embebido en la sección III, y finalmente se analizan las técnicas de optimización de los algoritmos para reducir su tiempo de inferencia y la cantidad de recursos empleados en la sección IV. En la sección V se realiza una vista general de todo lo descrito anteriormente para así obtener las conclusiones pertinentes.

II. Técnicas de implementación de algoritmos de *machine learning*

Existen dos ramas diferentes a la hora de implementar este tipo de algoritmo que se caracteriza por el volumen de parámetros que estos tienen, lo cual se traduce en mucha memoria consumida y una velocidad de procesamiento reducida.
Para resolver esto se puede optar por una inferencia en el propio dispositivo o una inferencia externa al dispositivo, para la cual se necesita a su vez acceso a la red.

*Este trabajo no ha contado con el apoyo de ninguna organización

I. Latre es un Ingeniero Electrónico y Automático por la Escuela de Ingeniería y Arquitectura de la Universidad de Zaragoza, trabajando como investigador técnico en CIRCE, Zaragoza, España.

III. Técnicas de implementación fuera del dispositivo

En el caso de tener un algoritmo demasiado pesado se pueden aplicar métodos de inferencia fuera del propio dispositivo, de forma que la carga de la ejecución no recaiga en este. Para ello existen dos posibilidades:

- **Ejecución en la nube.**
- **Ejecución *edge*.**

Además, esto permite a su vez reducir costes, ya que no es necesario usar un *hardware* especialmente diseñado para la ejecución de este tipo de algoritmo, lo cual puede aumentar el coste de forma considerable.

III-A. Implementación edge

La informática ha experimentado una transformación significativa en las últimas dos décadas, pasando de un enfoque basado en máquinas a un servicio prácticamente invisible y centrado en el ser humano conocido como computación ubicua. Este cambio se ha logrado mediante la incorporación de pequeños dispositivos integrados en un sistema computacional más grande, conectados a través de redes y denominados dispositivos *edge* [2].
Esta metodología ha estado en continua evolución como muestra la Figura. 1 [3].

Figura 1: Evolución de la computación al borde (Fuente: NVIDIA)

Este método de implementación se basa derivar el procesamiento localmente colocando servidores u otro *hardware* cerca de la ubicación física de la fuente de datos para procesarlos externamente. Esto da como resultado un mayor ancho de banda a menores costes.
Además este método se puede ejecutar en varios servidores para acortar la distancia entre los lugares donde se recopilan y procesan los datos para reducir los cuellos de botella y acelerar las aplicaciones [3].

Esta metodología tiene ventajas en diferentes aspectos recogidos en la Figura. 2:

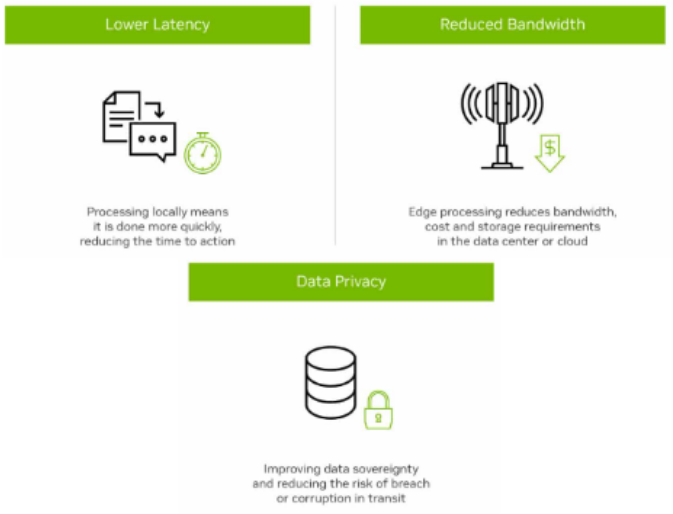

Figura 2: Ventajas de la computación al borde (Fuente: NVIDIA)

Dado que la computación *edge* procesa los datos localmente (en el borde de la red, en lugar de en la nube o en un centro de datos centralizado), minimiza la latencia y los costos de tránsito de datos, lo que permite la retroalimentación y la toma de decisiones en tiempo real. Esto es especialmente crítico para aplicaciones donde la seguridad humana es un factor como en los vehículos autónomos, donde ahorrar incluso milisegundos de procesamiento de datos y tiempos de respuesta puede ser clave para evitar accidentes [3] [4] [5].
En contrapartida la gestión en el *edge* puede llevar mucho tiempo y ser costosa cuando hay que instalar y mantener docenas o miles de dispositivos, lo cual la limita a pequeñas areas locales ya que sino se deben gestionar un elevado número de sitios dispersos por todo el territorio .
Otro reto es asegurar la seguridad, por lo que la aplicación empresarial que gestiona la propiedad intelectual y los datos confidenciales recogidos de los sensores debe estar siempre protegida [3] [5] .

III-B. Implementación en la nube

Otra forma de inferencia de los algoritmos de *machine learning* de forma externa al dispositivo es ejecutarlos en la nube. Esto presenta enormes ventajas ya que elimina la necesidad de comprar y mantener caros clústers de GPUs y da acceso instantáneo a capacidades de computación masiva bajo demanda. Ya sea que se necesite entrenar por lotes o hacer inferencias en tiempo real, la nube lo tiene cubierto [6]. El flujo de datos entre la nube y el procesador se puede visualizar en la Figura. 3:

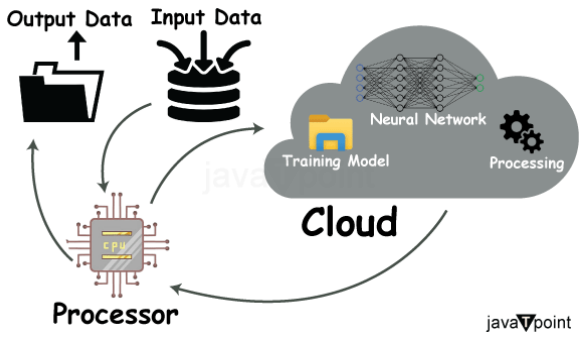

Figura 3: Esquema de ejecución en la nube (Fuente: javatpoint)

Las ventajas principales de este tipo de implementación son:

- **Modelos de precios bajo demanda.**
  Esto se traduce en una menor inversión.
- **Inversión innecesaria en *harware*.**
  Se obtiene la velocidad y el rendimiento de las GPU, TPU y FPGA sin invertir en la compra e integración de estas.
- **Escalabilidad.**
  Capacidad de crecer la potencia de procesamiento fácilmente.

Pero en cambio existen otras limitaciones:

- **Seguridad en los datos.**
  Está sujeto a los mismos problemas que cualquier plataforma de computación en la nube, es decir, están expuestos a redes públicas y pueden verse comprometidos por atacantes, que pueden manipular los resultados de ML (*machine learning*) o aumentar los costos de infraestructura.
- **Movilidad de los datos.**
  Cuando se ejecutan modelos de ML (*machine learning*) en la nube, puede ser difícil realizar la transición de los sistemas de una nube o servicio a otro. Esto requiere mover los datos de una manera que no afecte al rendimiento del modelo.

Gigantes en la nube como AWS [7], Google Cloud [8], IBM Cloud [9] y Microsoft Azure [10] ofrecen una amplia gama de servicios de aprendizaje automático listos para usar. Desde notebooks hasta contenedores de entrenamiento, estos proveedores facilitan la implementación de soluciones de IA (inteligencia artificial) escalables.

IV. Técnicas de implementación en el propio hardware

Otra forma de enfocar la inferencia es utilizar *hardware* especializado, de forma que la ejecución del algoritmo se realiza en el propio dispositivo con las ventajas y desventajas que esto acarrea. Entre las diversas posibilidades se encuentran:

- **FPGA**
- **ASIC**
- **GPU**
- **TPU**
- **NPU**

Si no se tienen dispositivos *hardware* especializado se pueden emplear técnicas de optimización de algoritmos entre los que se encuentra la implementación mediante coma fija, compresión de modelos o elegir arquitecturas con mayor eficiencia.

IV-A. Uso de hardware especializado

IV-A1. FPGA:

La FPGA (*Field Programmable Gate Array*) puede generar mapeos de *hardware* de alto rendimiento y bajo consumo en redes neuronales debido a su flexibilidad [1].
Entre las ventajas de su uso se encuentra el paralelismo que presentan este tipo de dispositivos al ser un circuito *hardware* y no *software*, lo cual aumenta la velocidad de procesamiento.
En contrapartida, estos dispositivos son menos potentes a la hora de operar con variables del tipo coma flotante (lo cual fuerza a implementar los algoritmos en como fija que requieren de un trabajo añadido). Además normalmente necesitan un ciclo de reloj para acceder y escribir los distintos datos en una memoria, por lo que produce retrasos en el procesamiento [11].

IV-A2. ASIC:

ASIC (Application Specific Integrated Circuit) es un tipo de circuito integrado de propósito especial que es altamente personalizado y menos programable. Por lo tanto, depende en gran medida del diseño de algoritmos. Para implementar algoritmos de *machine learning* en ASIC, es necesario diseñar aceleradores de hardware adecuados para la estructura específica. La optimización de los aceleradores de hardware implica la utilización de recursos paralelos y la reducción de la complejidad computacional.
La principal ventaja de los ASIC es que están diseñados específicamente para una tarea particular, lo cual se traduce en que son más rápidos y eficientes energéticamente que los microprocesadores generales. También significa que requieren menos espacio en la placa de circuito impreso y menos potencia para funcionar. Esto los hace ideales para su uso en sistemas embebidos en los que el espacio y la energía son limitados.

IV-A3. GPU:

La GPU (unidad de procesamiento gráfico) es un procesador especializado que se utiliza para acelerar la velocidad de procesamiento de imágenes en el hardware. Actualmente, las GPU's también se utilizan ampliamente para acelerar las redes neuronales en el aprendizaje automático debido al procesamiento en paralelo de estos dispositivos [1].
Las ventajas de este tipo de implementación son la paralización de los cálculos y la velocidad de procesamiento de operaciones matriciales y de convolución con distintos tipos de datos.
Entre sus desventajas se encuentran el consumo, lo cual imposibilita su implementación en sistemas embebidos, la cantidad de memoria de esta limita el tamaño de los modelos y la complejidad de su programación (en NVIDIA [3] se utiliza CUDA (*Compute Unified Device Architecture*) para la implementación de los algoritmos en las GPU's) [12].

IV-A4. TPU:

La Unidad de procesamiento tensorial (TPU) es un chip de procesamiento diseñado específicamente para acelerar el procesamiento de redes neuronales y otros algoritmos de aprendizaje automático creado por Google [8] [13].
Entre las ventajas de este tipo de dispositivo se encuentra el alto rendimiento y eficiencia, la eficiencia energética y la especialización en el manejo de operaciones tensoriales, las cuales son la base de los algoritmos de aprendizaje automático.
En contrapartida se encuentra el precio, ya que el suministro de estos dispositivos está muy limitado.

IV-A5. NPU:

Una NPU (Neural Processing Unit) es otro tipo de procesador diseñado específicamente para la aceleración de algoritmos de inteligencia artificial (IA) [14].
Están diseñadas para manejar operaciones matemáticas de precisión baja pero frecuentes, que son comunes en los algoritmos de IA [14].
Una característica clave de las NPUs es su capacidad para realizar cálculos en paralelo. Esto significa que pueden procesar múltiples elementos de datos simultáneamente, lo que las hace extremadamente rápidas. Además, tienen otras características importantes como la compresión de datos y la cuantización, que reducen la cantidad de memoria y ancho de banda necesarios.
Estas se suelen utilizar en dispositivos móviles y sistemas con restricciones de energía debido a su alta eficiencia, aunque se debe destacar simultáneamente su alto precio.

IV-B. Uso de hardware no especializado

En el caso de no tener disponible de un *hardware* optimizado para el *machine learning* como puede ser el caso de un microcontrolador simple (MCU) o una CPU, y no tener acceso a la red se pueden utilizar otras técnicas basadas en la reducción del peso del algoritmo en la memoria.

IV-B1. Coma fija:

Este método sugiere sustituir el formato de coma flotante de los pesos que componen la red en el formato de coma fija, de forma que se aumenta la eficiencia a la hora de realizar las operaciones aritmética. Esto es especialmente util en el caso de procesadores sin unidades especializadas en calculo con coma flotante como ocurre en las FPGA's.
Además, los números de coma fija ofrecen un grado constante de precisión pero están limitados por un rango fijo. Esta coherencia es ventajosa en aplicaciones donde la escala de los números es predecible y no varía mucho.

La Figura. 4 muestra la diferencia estructural entre ambos tipos de representación:

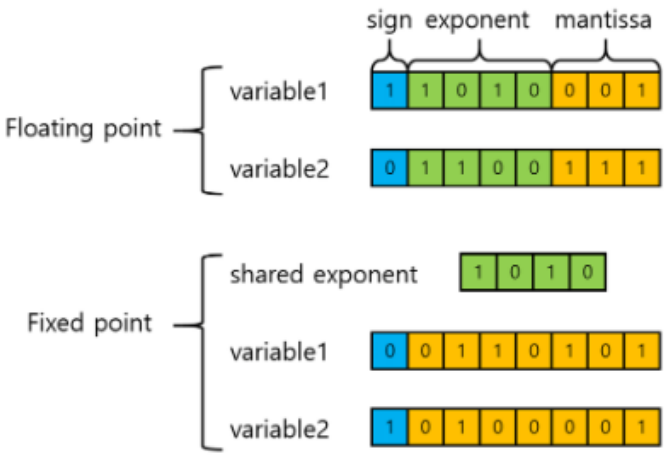

Figura 4: Coma fija vs coma flotante (Fuente: ResearchGate)

IV-B2. Reducción del número de parámetros:
Otra forma de resolver el problema es mediante el uso de algoritmos lo más eficientes posibles, es decir, tratar de reducir el número de parámetros necesarios mediante una elección adecuada de hiperparámetros.
Un ejemplo de ello sería emplear redes neuronales convolucionales para el procesamiento de imágenes en lugar de densamente conectadas o reducir el número de neuronas por capa mediante el aumento de la profundidad de la red para reconocer patrones más complejos.
Si los problemas son sencillos se podría tratar de evitar este tipo de algoritmos, ya que el uso de los algoritmos de *machine learning*, y más específicamente el campo de las rede neuronales es muy útil a la hora de resolver problemas de naturaleza no lineal o de una gran complejidad debido al número de variables que intervienen o la incapacidad de descubrir patrones de comportamiento. En otros casos sería conveniente tratar de resolver los problemas mediante métodos más sencillos basados en simples arboles de decisiones, estadística o análisis matemático.

IV-B3. Algoritmos de compresión:
Los algoritmos de compresión tratan de reducir el peso del algoritmo de *machine learning* mediante la eliminación de parámetros que no suponen ninguna influencia en el procesamiento de datos.
Entre las diferentes posibilidades se encuentran las técnicas de poda para eliminar conexiones innecesarias, los métodos de cuantización para reducir la precisión de los pesos, los métodos de destilación del conocimiento para generar una red más eficiente a partir de otra más grande y las técnicas de regularización.

V. Conclusiones

La metodología a seguir durante la implementación de algoritmos de *machine learning* depende en su mayoría de las limitaciones existentes.
De forma que en dispositivos embebidos con acceso a la red se podría optar por una inferencia externa al dispositivo mediante la ejecución en la nube o en el *edge*, ya que esto quitaría peso al dispositivo, y este se podría centrar en tareas más críticas.
En el caso de no tener acceso a la red y se quisiera reducir al máximo el precio, se debería tratar de evitar el uso de algoritmos demasiado pesados, y sino tratar de optimizarlos mediante la aplicación de métodos de regularización, de implementación en coma fija o de poda. Por otro lado, si el precio del dispositivo no fuera un problema se podría tratar de usar *hardware* especializado de pequeño consumo como pueden ser las NPU's o los ASIC.
En cambio, si se quiere ejecutar grandes modelos pero con un coste reducido, se debería optar por el uso de los servicios de la nube, ya que el precio se ajusta al *hardware* empleado. Si el precio no es ningún problema se podría optar por el uso de GPU's y TPU's.
Si la seguridad de los datos es un problema se debería intentar evitar la inferencia de forma externa al dispositivo.

Acknowledgment

Referencias

[1] Zhaoyun Zhang and ORCID yJingpeng Li, *A Review of Artificial Intelligence in Embedded Systems*, Micromachines 2023, Vol. 14, Page 897, accessed: 2024
[2] Anastasios Fanariotis and Theofanis Orphanoudakis, *Power Efficient Machine Learning Models Deployment on Edge IoT Devices*, Sensors 2023, Vol. 23, Page 1595, accessed: 2024
[3] Nvidia, https://www.nvidia.com/es-es/, accessed: 2024
[4] Chen, Z., Chen, J., Ding, *A lightweight CNN-based algorithm and implementation on embedded system for real-time face recognition*, 2022, accessed: 2024
[5] Ding XWang HCao Z,*An Edge Intelligent Method for Bearing Fault Diagnosis Based on a Parameter Transplantation Convolutional Neural Network*, 2023, accesed: 2024
[6] Ozlem Durmaz Incel, Sevda Özge Bursa, *On-Device Deep Learning for Mobile and Wearable Sensing Applications: A Review*, 2023, accessed: 2024
[7] Amazon, *Amazon Web Services*, 2006, accessed: 2024
[8] Google, *Google Cloud Platform*, 2008, accesed: 2024
[9] IBM, *IBM Cloud* , 2005, accessed: 2024
[10] Microsoft, *Microsoft Azure*, 2010, accessed: 2024
[11] Javier Moreno Segurado, *Implementación de red neuronal en FPGA*, 2019, accessed: 2024
[12] Jean-Sébastien Lerat, Sidi Ahmed Mahmoudi, *Single node deep learning frameworks: Comparative study and CPU/GPU performance analysis*, 2021,accessed: 2024
[13] Kiran Seshadri,Berkin Akin,*An Evaluation of Edge TPU Accelerators for Convolutional Neural Networks*, 2022, accessed: 2024
[14] Stylianos I. Venieris, Mario Almeida, *NAWQ-SR: A Hybrid-Precision NPU Engine for Efficient On-Device Super-Resolution*, 2024, accessed: 2024