Paralelización y distribución en redes neuronales

UAM - Neurocomputación - 2020/2021

David Cabornero Pascual
david.cabornero@estudiante.uam.es
Sergio Galán Martín
sergio.galanm@estudiante.uam.es
Alejandro Santorum Varela
alejandro.santorum@estudiante.uam.es

Contenido

- Introducción
- Clasificación de métodos de paralelización y distribución
- Paralelización usando GPUs
  - Uso de GPUs en Google Colab
  - CPU vs GPU
- Entrenamiento distribuido
  - Paralelización del descenso del gradiente
  - Entrenamiento distribuido en Keras
- Conclusiones

David Cabornero Pascual
david.cabornero@estudiante.uam.es
Sergio Galán Martín
sergio.galanm@estudiante.uam.es
Alejandro Santorum Varela
alejandro.santorum@estudiante.uam.es

Introducción

- Las redes profundas tienen un **gran coste** computacional.
- Los procesos de una máquina suelen ser **secuenciales**.
- Aprovechamiento de:
  - Múltiples núcleos de la CPU
  - GPU
- Librería **TensorFlow** en Google Colab

David Cabornero Pascual
david.cabornero@estudiante.uam.es
Sergio Galán Martín
sergio.galanm@estudiante.uam.es
Alejandro Santorum Varela
alejandro.santorum@estudiante.uam.es

Clasificación de métodos

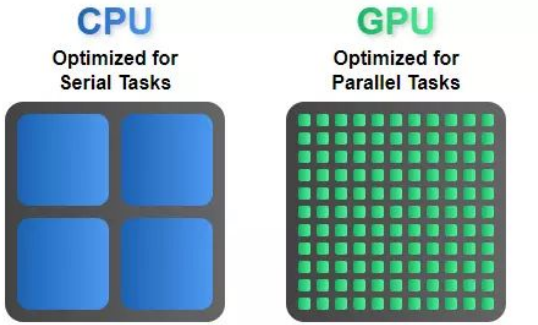
- **Paralelización**: en una única máquina
- Tres tipos:
  - Procesamiento *multi-core*: cada núcleo de la CPU trabaja con pequeños lotes.
  - Uso de la GPU para acciones computacionalmente caras.
  - Modelo híbrido.

- **Distribución**: en varias máquinas
- Dos tipos:
  - Paralelización de datos
  - Paralelización del modelo (solo si no queda otra opción)

David Cabornero Pascual
david.cabornero@estudiante.uam.es
Sergio Galán Martín
sergio.galanm@estudiante.uam.es
Alejandro Santorum Varela
alejandro.santorum@estudiante.uam.es

Paralelización usando GPUs

- Muchos algoritmos de aprendizaje de redes neuronales son muy propensos a ser paralelizados
- En este trabajo analizaremos la multiplicación de matrices y el entrenamiento de una red convolucional
- Tensorflow y Google Colab nos ofrecen una interfaz muy cómoda para trabajar con diferentes entornos (GPU, CPU)



David Cabornero Pascual
david.cabornero@estudiante.uam.es
Sergio Galán Martín
sergio.galanm@estudiante.uam.es
Alejandro Santorum Varela
alejandro.santorum@estudiante.uam.es

Uso de GPUs en Google Colab

- Para poder usar una GPU en Google Colab tan solo tenemos que cambiar el entorno de ejecución
- Podemos trabajar con CPU y GPU simultáneamente sin cambiar el entorno de ejecución con las siguientes instrucciones:
  - `with tf.device('/device:GPU:0')`
  - `with tf.device('/cpu:0')`

David Cabornero Pascual
david.cabornero@estudiante.uam.es

Sergio Galán Martín
sergio.galanm@estudiante.uam.es

Alejandro Santorum Varela
alejandro.santorum@estudiante.uam.es

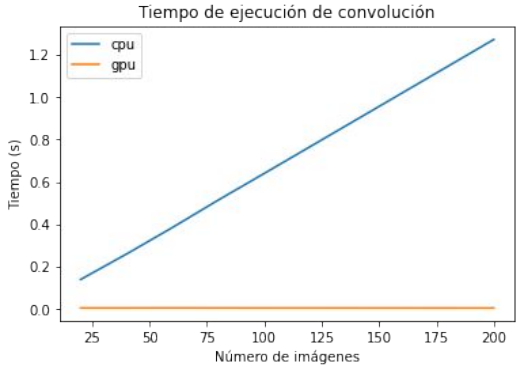
CPU vs GPU

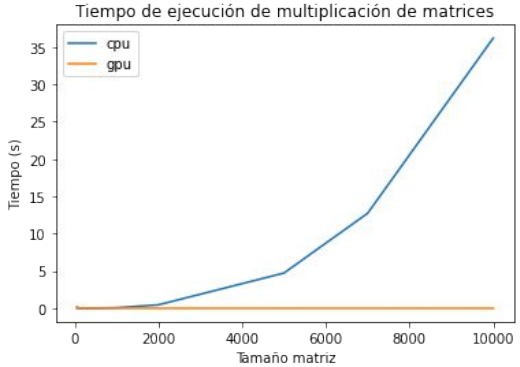
- Hemos implementado scripts para comparar de forma empírica los tiempos de ejecución
- Multiplicación de matrices cuadradas:
  - Tamaños: 50, 100, 500, 1000, 2000, 5000, 7000, 10000
- Entrenamiento de red convolucional:
  - Nº imágenes: 20, 40, 60, 80, 100, 200
  - Cada imagen es de 128 x 128 píxeles
  - Tamaño de salida: 32
  - Tamaño del kernel: 8
- Las matrices y las imágenes se generan de manera aleatoria
- Se repite cada caso 10 veces y se toma el tiempo medio para tener una mejor estimación del tiempo real

David Cabornero Pascual
david.cabornero@estudiante.uam.es
Sergio Galán Martín
sergio.galanm@estudiante.uam.es
Alejandro Santorum Varela
alejandro.santorum@estudiante.uam.es

Resultados

- El tiempo de ejecución en la GPU es marcadamente inferior
- A modo de comparativa cuantitativa:
  - Para n = 10000 en matrices, la CPU es 31500 más lenta
  - Para n = 200 imágenes, la CPU es 210 veces más lenta
- Esta paralelización no modifica las complejidades teóricas de ejecución ($O(n^3)$ para matrices y $O(n)$ para convolucionales)





Universidad Autónoma de Madrid

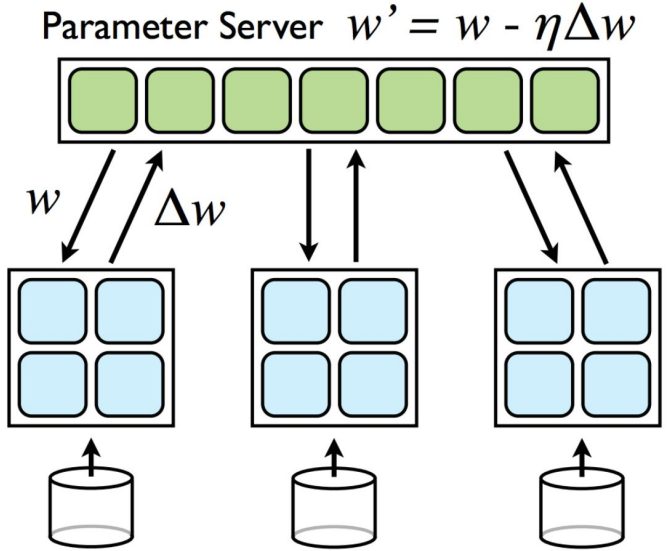
David Cabornero Pascual
david.cabornero@estudiante.uam.es

Sergio Galán Martín
sergio.galanm@estudiante.uam.es

Alejandro Santorum Varela
alejandro.santorum@estudiante.uam.es

Paralelización descenso del gradiente

1. **Barajar** conjunto de datos total
2. Para cada una de las **máquinas**:
   a. **Entrenar** con SGD una **porción** de los datos
   b. Calcular el **gradiente** en esa porción
   c. **Devolver** el gradiente calculado
3. Calcular **media aritmética** gradientes
4. **Actualizar los pesos** con el gradiente final



Universidad Autónoma de Madrid

David Cabornero Pascual
david.cabornero@estudiante.uam.es

Sergio Galán Martín
sergio.galanm@estudiante.uam.es

Alejandro Santorum Varela
alejandro.santorum@estudiante.uam.es

Entrenamiento distribuido en Keras

- **Una** máquina con **varias** GPUs
- Uso de la API de Tensorflow ***distributed.MirroredStrategy***
- Google Colab **no** permite usar varias GPUs

```python
import tensorflor as tf
from tensorflow import keras

# Create a MirroredStrategy
strat = tf.distribute.MirroredStrategy()

# Open a strategy scope
with strat.scope():
    # Everything that creates variables should be under the scope
    # It's usually only model construction and compilation
    model = Model(...)
    model.compile(...)

# Train the model on all available devices
model.fit(train_data, validation_data=val_data, ...)

# Test the model on all available devices
model.evaluate(test_data)
```

Universidad Autónoma de Madrid

David Cabornero Pascual
david.cabornero@estudiante.uam.es

Sergio Galán Martín
sergio.galanm@estudiante.uam.es

Alejandro Santorum Varela
alejandro.santorum@estudiante.uam.es

Conclusiones

- Las redes profundas tienen un **gran coste** computacional.
- Los procesos de una máquina suelen ser **secuenciales**.
- Aprovechamiento de:
  - Múltiples núcleos de la CPU
  - GPU
- Librería **TensorFlow** en Google Colab

David Cabornero Pascual
david.cabornero@estudiante.uam.es
Sergio Galán Martín
sergio.galanm@estudiante.uam.es
Alejandro Santorum Varela
alejandro.santorum@estudiante.uam.es

¡Gracias por su atención!

David Cabornero Pascual
david.cabornero@estudiante.uam.es

Sergio Galán Martín
sergio.galanm@estudiante.uam.es

Alejandro Santorum Varela
alejandro.santorum@estudiante.uam.es