Despliegue local de grandes modelos de lenguaje de pesos libres y su consumo

Fabián Villena

Introducción

El tamaño de los grandes modelos requiere un alto volumen de memoria disponible en el dispositivo de aceleración.

La necesidad de poder desplegar y entrenar estos grandes modelos de lenguaje ha llevado al desarrollo de métodos tanto para comprimir los modelos como para ajustarlos de manera eficiente.

Tamaños de los modelos

Los parámetros de los modelos están almacenados en FP16, lo que significa que cada parámetro pesa 2 Bytes.

Modelo	Cantidad de parámetros	Requerimiento de memoria
Llama 3	70e9	140 GB
Llama 3	8e9	16 GB
Phi-3	3.8e9	8 GB
Phi-3	14e9	28 GB

Cuantización

En el contexto de los LLMs, la cuantización se refiere al proceso de reducir la precisión de los valores numéricos usados para representar los parámetros del modelo.

El proceso de cuantización genera una disminución en el tamaño del modelo al usar menor bits para representar cada parámetro y aumentar la velocidad de inferencia del modelo debido a que los cálculos que se realizan para procesar la entrada y producir la salida pueden ser realizados más rápidos.

Cuantización posentrenamiento

La cuantización posentrenamiento es la conversión de los parámetros de un modelo entrenado hacia una menor precisión sin ningún preentrenamiento.

$$x\_q = round(x/S + Z)$$

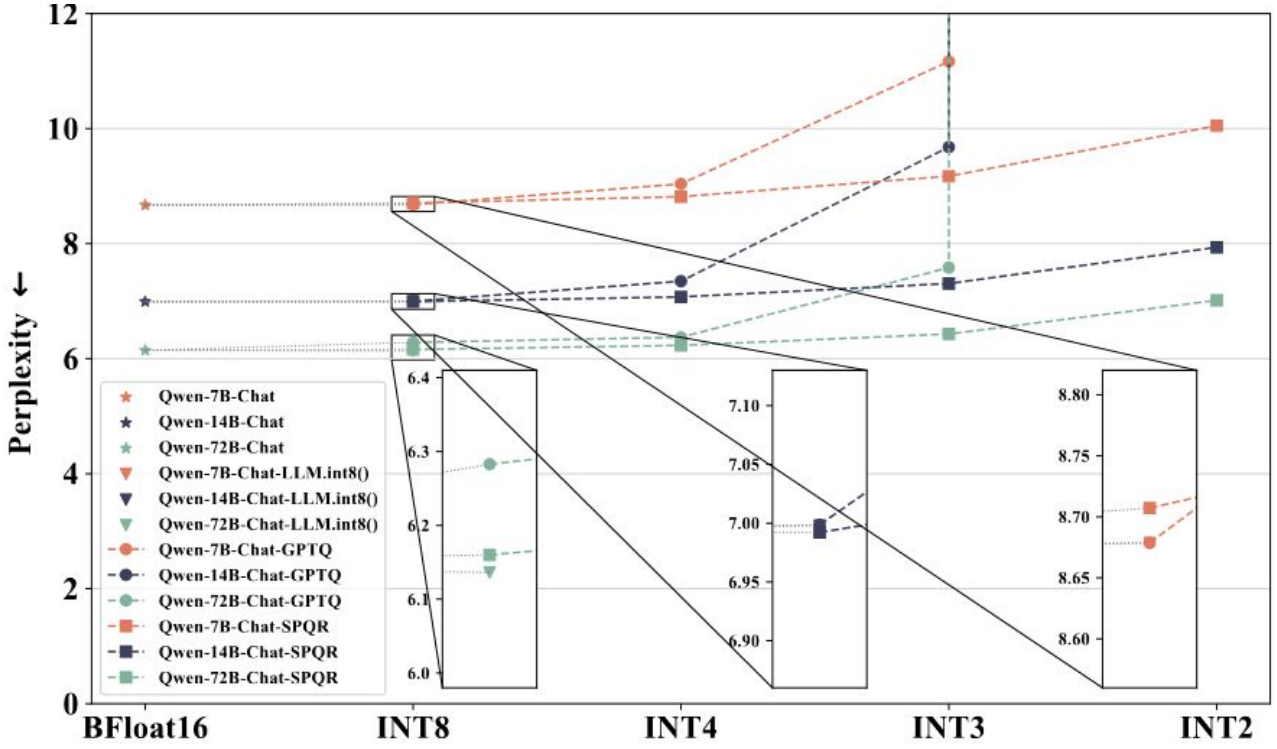
x_q: El valor de x cuantizado al INT más cercano
S: Un factor de escalamiento en FP32
Z: El punto cero, el valor en INT que corresponde al 0 en el espacio en FP

Entrenamiento consciente de la cuantización

Al contrario de aplicar la técnica de cuantización como un paso separado, el entrenamiento consciente de la cuantización incorpora cuantización durante el mismo proceso de entrenamiento. Este tipo de cuantización optimiza los parámetros del modelo con tal de mitigar la potencial pérdida de rendimiento asociada con la cuantización.

Perplejidad vs. cuantización

En general, los modelos pierden rendimiento de manera significativa desde INT4.



Tamaño de los modelos cuantizados

Modelo	Cantidad de parámetros	Tamaño en FP16	Tamaño en INT8	Tamaño en INT4
Llama 3	70e9	140 GB	70 GB	35 GB
Llama 3	8e9	16 GB	8 GB	4 GB
Phi-3	3.8e9	8 GB	4 GB	2 GB
Phi-3	14e9	28 GB	14 GB	14 GB

Despliegue de modelos

Para poner en producción sistemas basados en grandes modelos de lenguaje normalmente se decide por desplegar un servicio web que disponibiliza el modelo de lenguaje a través de consultas HTTP.

Para poder desplegar estos modelos de lenguaje debemos tener grandes recursos computacionales o utilizar alguna técnica de compresión de modelos.

Computación distribuida para Deep Learning

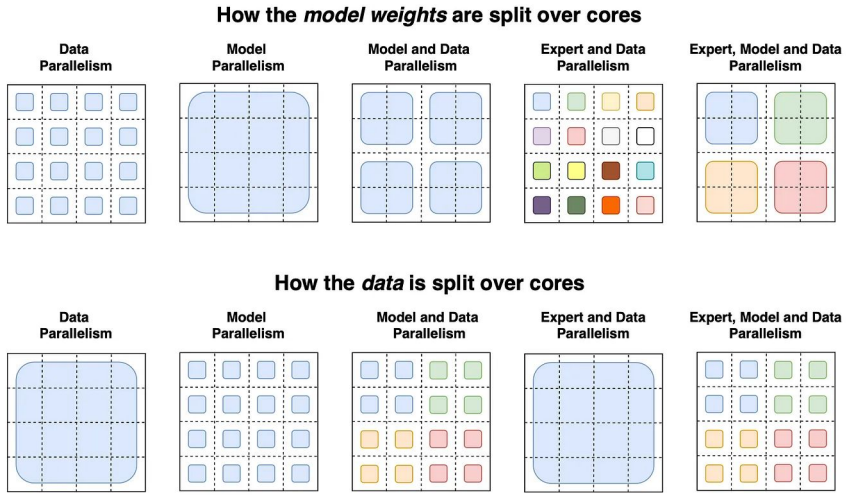
Las GPU son dispositivos especializados que pueden realizar múltiples cálculos matemáticos simultáneamente. Las operaciones realizadas en Deep Learning pueden ser divididas en una serie de multiplicaciones de matrices, por lo que las GPU se comportan bien.

El deep learning distribuido se utiliza cuando queremos aumentar la velocidad de los cálculos al utilizar múltiples GPUs.

Paralelismo de los datos

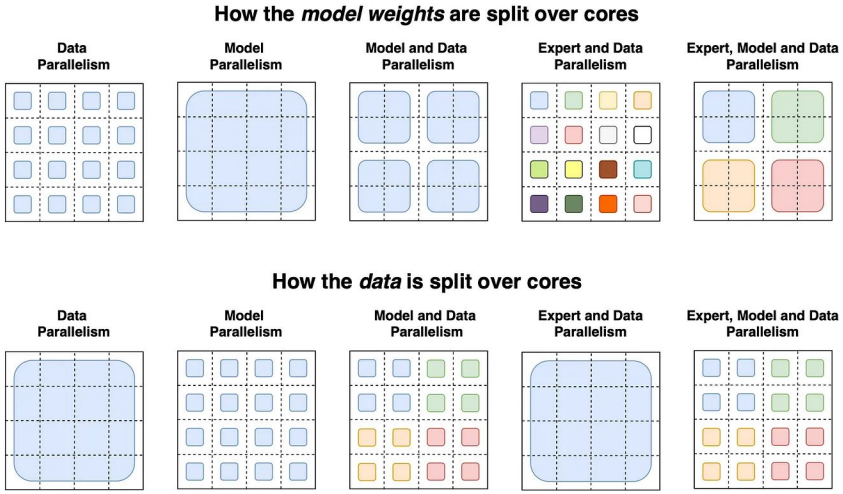
En el paralelismo de datos, pedazos de los datos son distribuidos a través de cada dispositivo en tres etapas:

1. Se distribuyen copias del modelo en cada dispositivo
2. Se dividen los datos y se distribuyen por los dispositivos
3. Se agregan los resutados



Paralelismo del modelo

En el paralelismo de modelo, pedazos del modelo (sus capas o vectores) se distribuyen a través de cada dispositivo, al contrario que en el paralelismo de datos en donde se tiene una copia completa del modelo en cada dispositivo.



Despliegue de LLMs en producción

Para utilizar grandes modelos de lenguaje en producción podemos consumirlos desde proveedores externos como OpenAI o Deep Infra como también podemos desplegarlos en un servidor propio. Este enfoque puede disminuir de manera significativa el costo, la latencia y los problemas de privacidad asociados con la transferencia de datos hacia proveedores externos.

Instalación de ollama

1. **Verificar dónde hay espacio suficiente en la máquina**

```
df -h
```

2. **Descargar la última versión de ollama desde https://github.com/ollama/ollama/releases**

```
wget https://github.com/ollama/ollama/releases/download/v0.2.1/ollama-linux-amd64
```

3. **Brindar permisos de ejecución al binario**

```
chmod +x ollama-linux-amd64
```

Levantamiento del servidor de ollama

- **Sin ningún requerimiento específico**

```
./ollama-linux-amd64 serve
```

- **Si los modelos deben ser almacenados en un directorio específico**

```
OLLAMA_MODELS=/data/fvillena/ollama_models/ ./ollama-linux-amd64 serve
```

- **Si se quiere sólo utilizar un subconjunto de GPUs**

```
CUDA_VISIBLE_DEVICES=1 ./ollama-linux-amd64 serve
```

- **Si se acaba el tiempo de espera al cargar un modelo**

```
OLLAMA_LOAD_TIMEOUT=10m ./ollama-linux-amd64 serve
```

Ejecución de un modelo para interactuar en la terminal

1. Verificar que hay suficiente memoria para ejecutar el modelo

```
nvtop
```

2. Ejecutar el modelo: Si el modelo no ha sido usado antes se descargará primero.
    a. Phi-3

```
./ollama-linux-amd64 run phi3
```

    b. Llama 3 8b a 4 bits

```
./ollama-linux-amd64 run llama3
```

    c. Llama 3 70 b a 4 bits

```
./ollama-linux-amd64 run llama3:70b
```

Consumir el modelo a través de la API de ollama

- **Verificar que ollama está siendo ejecutado y ver la lista de modelos disponibles**

```
curl http://localhost:11434
curl http://localhost:11434/api/tags
```

- **Generar texto**

```
curl --location 'http://localhost:11434/api/generate' \
--header 'Content-Type: application/json' \
--data '{
  "model": "phi3",
  "prompt": "¿Por qué los gatos son los mejores animales?",
  "stream": false
}'
```

Consumir la API de ollama desde Python

```
from ollama import Client
client = Client(host='http://192.168.7.28:11434')
response = client.chat(model='phi3', messages=[
    {
        'role': 'user',
        'content': '¿Cómo hacen los gatos?',
    },
])
print(response['message']['content'])
```

Consumir un modelo con la biblioteca de OpenAI

```
from openai import OpenAI
client = OpenAI(api_key="SUPER-API-KEY",
             base_url="http://192.168.7.28:11434/v1")

chat_completion = client.chat.completions.create(
    messages=[
    {
        "role": "user",
        "content": "Di que los gatos van a dominar el mundo",
    }
    ],
    model="phi3",
)

print(chat_completion.choices[0].message.content)
```

Clasificar un texto con zero-shot learning

```
chat_completion = client.chat.completions.create(
    messages=[
        {
            "role": "user",
            "content": "Clasifica el siguiente texto en una de las
siguientes categorías: deportes, política, tecnología, ciencia,
entretenimiento, salud, economía, educación, cultura, religión,
historia, medio ambiente, otros. El texto es: 'El Real Madrid ganó la
liga de campeones de fútbol'",
        }
    ],
    model="phi3",
)
print(chat_completion.choices[0].message.content)
```

Solicitar la respuesta en formato JSON

```
chat_completion = client.chat.completions.create(
    messages=[
    {
        "role": "user",
        "content": "Clasifica el siguiente texto en una de las siguientes
categorías: deportes, política, tecnología, ciencia, entretenimiento, salud,
economía, educación, cultura, religión, historia, medio ambiente, otros.
Responde en formato JSON con un objeto que sólo contenga la clave `label`. El
texto es: 'El Real Madrid ganó la liga de campeones de fútbol'",
    }
    ],
    model="phi3",
)
print(chat_completion.choices[0].message.content)
```

Clasificar un texto con one-shot learning

```
chat_completion = client.chat.completions.create(
      messages=[
      {
            "role": "user",
            "content": "Clasifica los siguientes textos en una de las siguientes categorías: deportes,
política, tecnología, ciencia, entretenimiento, salud, economía, educación, cultura, religión, historia, medio
ambiente, otros. Responde en formato JSON con un objeto que sólo contenga la clave `label`.",
      },
      {
            "role": "user",
            "content": "El Real Madrid ganó la liga de campeones de fútbol",
      },
      {"role": "assistant", "content": '{ "label": "deportes" }'},
      {
            "role": "user",
            "content": "El presidente de Estados Unidos es Joe Biden",
      },
      ],
      model="phi3",
)
print(chat_completion.choices[0].message.content)
```

Consumir un modelo desde un servicio externo

```
from openai import OpenAI
client = OpenAI(api_key="SUPER-API-KEY",
             base_url="https://api.groq.com/openai/v1")

chat_completion = client.chat.completions.create(
    messages=[
    {
        "role": "user",
        "content": "Di que los gatos van a dominar el mundo",
    }
    ],
    model="llama3-8b-8192",
)

print(chat_completion.choices[0].message.content)
```

Clasificar textos utilizando scikit-llm

```
from skllm.datasets import get_classification_dataset
from skllm.config import SKLLMConfig
from skllm.models.gpt.classification.zero_shot import
ZeroShotGPTClassifier

SKLLMConfig.set_openai_key("<YOUR_KEY>")
SKLLMConfig.set_gpt_url("http://192.168.7.28:11434/v1")

X, y = get_classification_dataset()
clf = ZeroShotGPTClassifier(model="phi3")
clf.fit(X, y)
print(clf.predict(X))
```

Desplegar OpenWebUI con docker para usarlo con ollama

```
docker run -d -p 3000:8080 -e
OLLAMA_BASE_URL=http://192.168.7.28:11434 -v
open-webui:/app/backend/data --name open-webui --restart
always ghcr.io/open-webui/open-webui:main
```

Fabián Villena

fabian@villena.cl