

UNIVERSIDAD ALFONSO X EL SABIO

CUANTIZACIÓN LLM DE 4 BITS

Germán Llorente y Carlos Puigserver

October 24, 2023

October 24, 2023

1 Objetivo

El objetivo de este estudio es comprender el proceso de cuantización de 4-bits.

2 Introducción

La cuantización es un concepto fundamental en el ámbito de los modelos de lenguaje masivos. Es crucial entender los fundamentos y la importancia de la cuantificación en estos modelos. En este estudio, nos enfocaremos en el proceso específico de cuantización de 4-bits. Exploraremos cómo este proceso es fundamental para la representación precisa de datos en modelos de lenguaje avanzados y cómo influye en su rendimiento y precisión.

3 Revisión Teórica

En las últimas décadas, la investigación en el campo de la cuantización ha llevado al desarrollo de varios métodos avanzados para la optimización y aproximación de datos. Entre estos métodos se encuentran el Optimal Brain Quantizer (OBQ) y el algoritmo GPTQ, propuesto por Frantar et al. en 2023. En este paper, se revisarán detenidamente estos dos enfoques, comprendiendo su inspiración, operación y comparando sus diferencias y ventajas.

3.1 Optimal Brain Quantizer (OBQ)

El OBQ se inspira en el framework clásico de OBS, que ha sido ampliamente utilizado en técnicas de poda para DNNs. Analizamos cómo el OBQ extiende estas ideas al proceso de cuantización, proporcionando una visión integral de su inspiración teórica.

Este método se basa en una técnica de poda diseñada para eliminar con cuidado los pesos de una red neuronal densa ya completamente entrenada, conocida como Optimal Brain Surgeon (OBQ). Utiliza una técnica de aproximación y proporciona fórmulas específicas para determinar el peso óptimo que se debe eliminar y cómo ajustar de forma óptima el conjunto de pesos restantes no cuantificados para compensar esta eliminación:

Con OBQ, primero cuantificamos el peso más sencillo y luego ajustamos todos los demás pesos no cuantificados para compensar la pérdida de precisión. A continuación, elegimos el siguiente peso para cuantificar y así sucesivamente. Sin embargo, surge un problema cuando hay pesos atípicos, lo que puede llevar a un alto error de cuantificación. Normalmente, estos valores atípicos se cuantificarían en último lugar, cuando quedan pocos pesos no cuantificados que podrían ajustarse para compensar el gran error. Este problema se agrava cuando algunas ponderaciones se desplazan más allá del rango aceptable mediante actualizaciones intermedias. Para evitar esto, aplicamos una heurística simple: los valores atípicos se cuantifican tan pronto como aparecen.

Este proceso puede ser intensivo en términos computacionales, especialmente para los Modelos de Lenguaje Grandes (LLM). Para abordar este problema, OBQ utiliza un truco que evita recalcular todas las operaciones cada vez que se simplifica un peso. Después de cuantificar un peso, ajusta la matriz utilizada en los cálculos (la hessiana) eliminando la fila y la columna asociadas con ese peso mediante la eliminación gaussiana. Además, el método aprovecha la vectorización para procesar múltiples filas de la matriz de peso simultáneamente. A pesar de su eficiencia, el tiempo de cálculo de OBQ aumenta considerablemente a medida que crece el tamaño de la matriz de pesos. Este aumento cúbico dificulta la aplicación de OBQ en modelos extremadamente grandes con miles de millones de parámetros.

3.2 Operación del OBQ

Asumiendo la suposición estándar de que el gradiente en el punto actual w es despreciable, las fórmulas de OBS para el peso óptimo a podar w_p y la actualización correspondiente δ_p se pueden derivar escribiendo el problema localmente cuadrático bajo la restricción de que el elemento p de δ_p es igual a $-w_p$, lo que significa que w_p es cero después de aplicar la actualización a w . Este problema

tiene la siguiente Lagrangiana:

$$L(\delta_p, \lambda) = \delta_p^T H \delta_p + \lambda(e_p^T \delta_p - (-w_p)),$$

donde H denota el Hessiano en w y e_p es el vector canónico p -ésimo. La solución óptima se deriva encontrando primero la solución óptima para δ_p mediante el establecimiento de la derivada $\partial L / \partial \delta_p$ a cero y luego sustituyendo esta solución de vuelta en L y resolviendo para λ .

3.3 GPTQ: Una Innovación en Cuantización

El algoritmo GPTQ, desarrollado por Frantar et al. en 2023, presenta un enfoque innovador en la cuantización de datos. GPTQ representa un proyecto dinámico en constante evolución, perfeccionando continuamente sus funcionalidades y habilidades. Sus últimas mejoras incluyen la integración de bibliotecas de optimización de rendimiento, la adaptabilidad para trabajar con diversos tipos de modelos y la optimización de la velocidad del kernel CUDA.

Uno de los puntos fuertes más destacados de GPTQ es su impresionante velocidad de inferencia. Las comparaciones realizadas en GPU revelan cifras asombrosas en términos de tokens por segundo, donde el modelo cuantizado con AutoGPTQ supera a sus contrapartes. Por ejemplo, al utilizar un tamaño de lote de entrada de 1, emplear una estrategia de decodificación de búsqueda de rayo y obligar al modelo a generar 512 tokens, el modelo cuantizado Llama-7b supera al modelo original en términos de velocidad de inferencia (25,53 tokens por segundo frente a 18,87 tokens por segundo). Estos resultados resaltan el impacto significativo de las optimizaciones implementadas por GPTQ, marcando una diferencia palpable en la eficiencia del proceso de inferencia.

AutoGPTQ ofrece a los usuarios la posibilidad de expandir las capacidades del sistema para dar soporte a modelos personalizados de manera sencilla. Este proceso simplificado otorga a los usuarios un mayor control sobre sus tareas de aprendizaje automático. La capacidad de adaptación de AutoGPTQ se destaca al compararlo con otros paquetes de cuantización, lo que lo hace más versátil y capaz de ajustarse a una amplia gama de escenarios de aplicación. Esta flexibilidad y adaptabilidad son atributos fundamentales que diferencian a AutoGPTQ en el campo de la cuantización de modelos.

3.4 Comparación OBQ y GPTQ

Inspiración y Enfoque

- **OBQ:** OBQ se basa en el framework clásico de OBS, utilizado principalmente en técnicas de poda para DNNs. Se enfoca en la poda óptima de pesos basada en técnicas de optimización.
- **GPTQ:** GPTQ es un algoritmo innovador con un enfoque adaptativo y versátil. Puede integrarse con diversos tipos de modelos y se ha optimizado para la velocidad de inferencia.

Flexibilidad y Adaptabilidad

- **OBQ:** Aunque OBQ ofrece una técnica de cuantización sólida, puede ser menos adaptable a diferentes modelos y tipos de datos.
- **GPTQ:** GPTQ es altamente flexible y puede manejar una variedad de modelos y tipos de datos, desde imágenes hasta texto, con facilidad.

Velocidad de Inferencia y Escalabilidad

- **OBQ:** Mientras OBQ puede acelerar la inferencia en ciertas configuraciones, puede tener desafíos para manejar grandes volúmenes de datos debido a su enfoque específico de poda.
- **GPTQ:** GPTQ está diseñado para ser altamente escalable y puede manejar grandes volúmenes de datos mientras mantiene una alta velocidad de inferencia.

Integración y Desarrollo Continuo

- **OBQ:** Aunque OBQ tiene una base teórica sólida, su desarrollo continuo puede ser más limitado.
- **GPTQ:** GPTQ es un proyecto en constante evolución, con capacidad para integrarse con bibliotecas de optimización de rendimiento y beneficiarse de actualizaciones constantes.

En resumen, GPTQ se destaca por su versatilidad, adaptabilidad y capacidad para manejar grandes volúmenes de datos. Su enfoque innovador y su desarrollo continuo lo convierten en una opción preferida para aplicaciones que requieren una cuantización eficiente y rápida en entornos con requisitos de datos diversos y exigentes.

4 Importancia del Hardware en el Proceso de Cuantización

La importancia del hardware, especialmente de las tarjetas gráficas (GPUs), en el proceso de cuantización se ha vuelto cada vez más significativa con el aumento en la complejidad de los modelos de aprendizaje profundo y la necesidad de manejar grandes volúmenes de datos de manera eficiente. A continuación, se presentan algunas razones clave por las que el hardware, incluidas las tarjetas gráficas, es crucial en el proceso de cuantización:

4.1 Velocidad de Inferencia

Las tarjetas gráficas modernas están optimizadas para realizar operaciones matriciales y cálculos paralelos de manera eficiente. Algoritmos de cuantización, especialmente durante la inferencia en modelos cuantizados, implican operaciones matriciales y de punto flotante que pueden beneficiarse significativamente de la potencia de cálculo de las GPUs. La capacidad de las GPUs para procesar múltiples operaciones en paralelo permite una aceleración significativa en la velocidad de inferencia, lo que es esencial en aplicaciones en tiempo real.

4.2 Optimizaciones de Precisión de Punto Flotante

Las GPUs modernas a menudo tienen unidades de procesamiento de precisión reducida (por ejemplo, FP16) que permiten realizar cálculos de punto flotante con una precisión menor. En la cuantización, donde los valores de los pesos y activaciones se representan con menos bits, estas optimizaciones de precisión de punto flotante pueden mejorar la velocidad del proceso sin comprometer significativamente la precisión del modelo.

4.3 Entrenamiento Eficiente

Durante el entrenamiento de modelos de aprendizaje profundo, especialmente en tareas que implican grandes conjuntos de datos, las GPUs permiten un entrenamiento más rápido al paralelizar las operaciones en mini lotes. La cuantización a menudo implica ajustar los modelos para adaptarse a representaciones de datos de menor precisión. Las GPUs pueden facilitar este proceso permitiendo un entrenamiento eficiente de modelos cuantizados.

4.4 Implementación de Algoritmos de Cuantización

Algunos algoritmos de cuantización, como la cuantización por espaciado de clústeres (k-means clustering) y la cuantización por búsqueda de código (codebook search), implican cálculos intensivos que se pueden acelerar significativamente mediante el uso de GPUs. La implementación eficiente de estos algoritmos es esencial para garantizar que la cuantización se realice de manera precisa y rápida.

4.5 Manejo de Grandes Volúmenes de Datos

En aplicaciones que implican grandes conjuntos de datos, como el procesamiento de imágenes o videos de alta resolución, las GPUs pueden manejar eficientemente grandes volúmenes de datos durante la cuantización y la inferencia. La capacidad de memoria y el ancho de banda de memoria de las GPUs son particularmente útiles al trabajar con grandes lotes de datos durante la cuantización.

En resumen, las tarjetas gráficas y otros componentes de hardware optimizados para cálculos paralelos desempeñan un papel fundamental en el proceso de cuantización al mejorar la velocidad de inferencia, permitir entrenamiento eficiente y facilitar la implementación de algoritmos de cuantización complejos. Su contribución es esencial para garantizar que los modelos cuantizados sean eficientes tanto en tiempo como en recursos computacionales.

5 Otras técnicas de cuantificación

Exploraremos dos de estas técnicas avanzadas de cuantización: GGML (Generative Gaussian Mixture Layer) y NF4 (Neural Factorization 4). Estos métodos no solo permiten una reducción significativa en el tamaño de los modelos, sino que también preservan la precisión y la capacidad de generalización. A medida que avanzamos en este documento, profundizaremos en los detalles de GGML y NF4, explorando sus principios fundamentales y su aplicación en el contexto del aprendizaje automático moderno.

6 GGML: Biblioteca de C para Aprendizaje Automático y Cuantización de Modelos de Lenguaje Grandes

GGML es una biblioteca de C para aprendizaje automático (ML), donde las iniciales "GG" hacen referencia a su creador (Georgi Gerganov). Además de definir primitivas de aprendizaje automático de bajo nivel (como un tipo tensor), GGML define un formato binario para distribuir modelos de lenguaje grandes (LLMs). Esta biblioteca proporciona enlaces en Rust hacia la implementación de referencia de GGML, así como una colección de utilidades nativas de Rust para proporcionar acceso seguro y idiomático a estos enlaces. GGML utiliza una técnica llamada "cuantización" que permite que los modelos de lenguaje grandes se ejecuten en hardware de consumo. Este documento describe los conceptos básicos del formato GGML, incluido cómo se utiliza la cuantización para democratizar el acceso a los LLMs.

6.1 Formato

Los archivos GGML consisten en datos codificados en binario que siguen un formato específico. Este formato especifica qué tipo de datos está presente en el archivo, cómo se representa y el orden en el que aparece. La primera pieza de información presente en un archivo GGML válido es un número de versión de GGML, seguido por tres componentes que definen un modelo de lenguaje grande: los hiperparámetros del modelo, su vocabulario y sus pesos. Continúa leyendo para obtener más información sobre las versiones de GGML y los componentes de un modelo GGML.

6.2 Versiones de GGML

GGML es una tecnología de vanguardia y está en constante evolución. Para respaldar el desarrollo rápido sin sacrificar la compatibilidad con versiones anteriores, GGML utiliza el sistema de versiones para introducir mejoras que pueden cambiar el formato de codificación. Por ejemplo, las versiones más recientes de GGML utilizan la puntuación del vocabulario, lo que introduce información adicional en la codificación, así como mmap, que mejora el rendimiento mediante la asignación de memoria. El primer valor presente en un archivo GGML válido es un "número mágico" que indica la versión de GGML que se utilizó para

codificar el modelo.

6.3 Hiperparámetros

El término "hiperparámetro" describe un valor que se utiliza para configurar el comportamiento de un modelo de lenguaje grande; esto contrasta con los parámetros del modelo, que son los pesos derivados en el proceso de entrenamiento que se utilizó para crear el modelo. Cada modelo define su propia estructura de hiperparámetros que define los valores de los hiperparámetros aceptados por ese modelo. Los archivos GGML válidos deben enumerar estos valores en el orden correcto, y cada valor debe estar representado utilizando el tipo de datos correcto. Aunque los hiperparámetros son diferentes entre modelos, algunos atributos aparecen en los hiperparámetros para la mayoría de los modelos:

- **n_vocab**: el tamaño del vocabulario del modelo.
- **n_embd**: el tamaño de la "capa de incrustación" del modelo, que se utiliza durante la ingestión de comandos.
- **n_layer**: el número de capas en el modelo; cada capa representa un conjunto de pesos.

6.4 Vocabulario

Como su nombre indica, el vocabulario de un modelo comprende componentes que utiliza el modelo para generar lenguaje (texto). Sin embargo, a diferencia del vocabulario de un humano, que está compuesto por palabras, el vocabulario de un modelo de lenguaje grande está formado por "tokens". Un token puede ser una palabra completa, pero a menudo son fragmentos de palabras. Al igual que los humanos pueden componer millones de palabras a partir de una docena o dos de letras, los modelos de lenguaje grandes usan tokens para expresar un gran número de palabras a partir de un número relativamente menor de componentes. Considera un vocabulario con los siguientes tokens: `whi`, `ch` `le`, `who` y `a`; este vocabulario se puede usar para crear las palabras en inglés "which", "while", "who", "a" y "leach". ¿Cómo cambiaría el comportamiento si el modelo contuviera los siguientes tokens: `wh`, `ich`, `ile`, `o` y `leach`? Elecciones como estas permiten a los creadores de modelos ajustar el comportamiento y el rendimiento de sus modelos.

Como se describe anteriormente, los hiperparámetros del modelo generalmente contienen un valor que especifica el número de tokens en el vocabulario. El vocabulario se codifica como una lista de tokens, cada uno de los cuales incluye un entero de 32 bits que especifica la longitud del token. Dependiendo de la versión de GGML, el token también puede incluir un número flotante de 32 bits, que representa la frecuencia de ese token en los datos de entrenamiento del modelo.

6.5 Pesos

El componente final y más grande de un archivo GGML son los pesos del LLM que representa el archivo. Abstractamente, un modelo de lenguaje grande es un software que se utiliza para generar lenguaje, de la misma manera que el software utilizado para generar imágenes se puede mejorar aumentando el número de colores con los que se pueden renderizar las imágenes, los modelos de lenguaje grande pueden mejorarse aumentando el número de pesos en el modelo. El número total de pesos en un modelo se denomina "tamaño" de ese modelo. Por ejemplo, la implementación StableLM de la arquitectura del modelo de lenguaje GPT-NeoX está disponible en varios tamaños, como 3B y 7B, que significan 3 mil millones y 7 mil millones, respectivamente. Estos números se refieren al número total de pesos en ese modelo. Como se describe en la sección de hiperparámetros, los pesos se agrupan en conjuntos llamados "capas", que, al igual que los hiperparámetros, tienen estructuras definidas de manera única por la arquitectura del modelo; dentro de una capa, los pesos se agrupan en estructuras llamadas "tensores". Por ejemplo, tanto StableLM 3B como StableLM 7B utilizan capas que comprenden los mismos tensores, pero StableLM 3B tiene relativamente menos capas en comparación con StableLM 7B.

En GGML, un tensor consta de varios componentes, incluido un nombre, una lista de 4 elementos que representa el número de dimensiones en el tensor y sus longitudes, y una lista de los pesos en ese tensor. Por ejemplo, considera el siguiente tensor de 2 2 llamado `tensor_a0`:

$$tensor_a0 = 1.00.00.11.1$$

Una simplificación de la representación de GGML de `tensor_a0` es {"tensor_a0", [2, 2, 1, 1], [1.0, 0.0, 0.1, 1.1]}. Ten en cuenta que la lista de 4 elementos de dimensiones usa 1 como marcador de posición para dimensiones no utilizadas,

esto se debe a que el producto de las dimensiones no debe ser igual a cero.

Los pesos en un archivo GGML se codifican como una lista de capas, cuya longitud se especifica típicamente en los hiperparámetros del modelo; cada capa se codifica como un conjunto ordenado de tensores.

6.6 Cuantización

Los pesos de LLM son números de punto flotante (decimales). Al igual que se requiere más espacio para representar un número entero grande (por ejemplo, 1000) en comparación con un número entero pequeño (por ejemplo, 1), se requiere más espacio para representar un número de punto flotante de alta precisión (por ejemplo, 0.0001) en comparación con un número de punto flotante de baja precisión (por ejemplo, 0.1). El proceso de "cuantización" de un modelo de lenguaje grande implica reducir la precisión con la que se representan los pesos para reducir los recursos necesarios para usar el modelo. GGML admite varias estrategias de cuantización diferentes (por ejemplo, cuantización de 4 bits, 5 bits y 8 bits), cada una de las cuales ofrece diferentes compensaciones entre eficiencia y rendimiento. Puedes encontrar más información sobre estas compensaciones en la documentación de llama.cpp, que es otro proyecto del desarrollador de GGML. Los detalles técnicos sobre la cuantización se describen en este video de @Aemon-Algiz.

7 Comparación de Métodos de Cuantización

En esta sección, se presenta una comparación detallada entre varios métodos de cuantización utilizados en el procesamiento de señales digitales. Se analizan dos enfoques de cuantización de 8 bits, a saber, el método **absmax** y la cuantización de punto cero, junto con dos técnicas de cuantización de 4 bits, GGML (*Gradual Gray Mixed Level*) y NF4 (*Near-Far 4*). A continuación se presenta un resumen de cada método:

7.1 Cuantización de 8 bits

7.1.1 **absmax** (Valor Absoluto Máximo)

En el método **absmax**, se determina el valor absoluto máximo en el conjunto de datos y se utiliza este valor como referencia para la cuantización. Los valores se escalan y cuantizan en un rango que va desde el negativo del valor máximo

hasta el valor máximo. Esto proporciona una alta resolución y es especialmente útil cuando se necesita preservar la información detallada de los datos.

7.1.2 Cuantización de Punto Cero

En la cuantización de punto cero, se encuentra el valor medio de los datos y se utiliza como punto de referencia para la cuantización. Los valores por encima de este punto se consideran positivos y los valores por debajo se consideran negativos. Este enfoque es efectivo cuando los datos tienen una tendencia alrededor de cero y se desea mantener la información sobre la polaridad de los valores.

7.2 Cuantización de 4 bits

7.2.1 GGML (*Gradual Gray Mixed Level*)

GGML fue diseñado para utilizarse en conjunto con la biblioteca `llama.cpp`, también creada por Georgi Gerganov. Esta biblioteca está escrita en C/C++ para una inferencia eficiente de modelos Llama. Puede cargar modelos GGML y ejecutarlos en una CPU. Originalmente, esta era la principal diferencia con los modelos GPTQ, que se cargan y ejecutan en una GPU. Sin embargo, ahora puedes trasladar algunas capas de tu LLM a la GPU con `llama.cpp`. Por ejemplo, hay 35 capas para un modelo de 7 mil millones de parámetros. Esto acelera drásticamente la inferencia y te permite ejecutar LLMs que no caben en una VRAM.

Si se prefieren las herramientas de línea de comandos, `llama.cpp` y el soporte GGUF se han integrado en muchas interfaces gráficas de usuario, como `text-generation-web-ui` de oobabooga, `koboldcpp`, `LM Studio` o `ctransformers`. Simplemente se pueden cargar los modelos GGML con estas herramientas e interactuar con ellos de manera similar a ChatGPT. Afortunadamente, muchos modelos cuantizados están disponibles directamente en Hugging Face Hub. La mayoría de ellos están cuantizados por TheBloke, una figura popular en la comunidad de LLM.

En cuanto a los archivos dentro del repositorio `TheBloke/Llama-2-13B-chat-GGML`, podemos ver 14 modelos GGML diferentes, correspondientes a diferentes tipos de cuantización. Siguen una convención de nomenclatura específica: "q" + el número de bits utilizados para almacenar los pesos (precisión) + una variante específica. A continuación se expone una lista de todos los posibles métodos de cuantización y sus casos de uso correspondientes, basados en las tarjetas de

modelo creadas por TheBloke:

- `q2_k`: Utiliza `Q4_K` para los tensores `attention.vw` y `feed_forward.w2`, `Q2_K` para los otros tensores.
- `q3_k_l`: Utiliza `Q5_K` para los tensores `attention.wv`, `attention.wo` y `feed_forward.w2`, en caso contrario `Q3_K`.
- `q3_k_m`: Utiliza `Q4_K` para los tensores `attention.wv`, `attention.wo` y `feed_forward.w2`, en caso contrario `Q3_K`.
- `q3_k_s`: Utiliza `Q3_K` para todos los tensores.
- `q4_0`: Método de cuantización original, de 4 bits.
- `q4_1`: Mayor precisión que `q4_0` pero no tan alta como `q5_0`. Sin embargo, tiene una inferencia más rápida que los modelos `q5`.
- `q4_k_m`: Utiliza `Q6_K` para la mitad de los tensores `attention.wv` y `feed_forward.w2`, en caso contrario `Q4_K`.
- `q4_k_s`: Utiliza `Q4_K` para todos los tensores.
- `q5_0`: Mayor precisión, mayor uso de recursos y una inferencia más lenta.
- `q5_1`: Incluso mayor precisión, mayor uso de recursos y una inferencia más lenta.
- `q5_k_m`: Utiliza `Q6_K` para la mitad de los tensores `attention.wv` y `feed_forward.w2`, en caso contrario `Q5_K`.
- `q5_k_s`: Utiliza `Q5_K` para todos los tensores.
- `q6_k`: Utiliza `Q8_K` para todos los tensores.
- `q8_0`: Casi indistinguible de `float16`. Uso intensivo de recursos y lento. No recomendado para la mayoría de los usuarios.

Como regla general, se recomienda usar `Q5_K_M` ya que conserva la mayoría del rendimiento del modelo. Alternativamente, se puede usar `Q4_K_M` si se quiere ahorrar memoria. En general, las versiones `K_M` son mejores que las versiones `K_S`. Las versiones `Q2` o `Q3` disminuyen drásticamente el rendimiento del modelo.



7.2.2 NF4 (*Near-Far 4*)

NF4 divide el rango de valores en cuatro regiones: cerca de cero, lejos de cero en el lado positivo, lejos de cero en el lado negativo y valores extremos. Este enfoque es beneficioso cuando los datos tienen una distribución asimétrica o están sesgados hacia ciertos valores. NF4 proporciona una representación eficiente y equitativa de estas regiones del rango de valores.

7.2.3 NF4 vs. GGML vs. GPTQ

¿Qué técnica es mejor para la cuantización de 4 bits? Para responder a esta pregunta, necesitamos introducir los diferentes backends que ejecutan estos LLM cuantizados. Para los modelos GGML, `llama.cpp` con modelos `Q4_K_M` es la mejor opción. Para los modelos GPTQ, tenemos dos opciones: `AutoGPTQ` o `ExLlama`. Finalmente, los modelos NF4 pueden ejecutarse directamente en `transformers` con la bandera `--load-in-4bit`.

Oobabooga realizó múltiples experimentos en una excelente publicación de blog que compara diferentes modelos en términos de perplejidad (menor es mejor):

		
	Llama-13b Perplexity	Llama-30b Perplexity
NF4	5.73047	5.24609
GPTQ (AutoGPTQ)	5.72656	5.30078
GPTQ (ExLlama)	5.72581	5.25923
GGML (Q4_K_M)	5.71705	5.21557

Basándonos en estos resultados, podemos decir que los modelos GGML tienen una ligera ventaja en términos de perplejidad. La diferencia no es particularmente significativa, por lo que es mejor enfocarse en la velocidad de generación en términos de tokens por segundo. La mejor técnica depende de tu GPU: si tienes suficiente VRAM para ajustar todo el modelo cuantizado, GPTQ con

ExLlama será el más rápido. Si ese no es el caso, puedes trasladar algunas capas y usar modelos GGML con `llama.cpp` para ejecutar tu LLM.

7.3 Comparación y Conclusiones

En términos de resolución, la cuantización de 8 bits ofrece una representación más detallada en comparación con la cuantización de 4 bits. Sin embargo, los métodos GGML y NF4 permiten una gestión más flexible de diferentes regiones del rango de valores, lo que ayuda a preservar los detalles importantes en los datos. La elección del método de cuantización dependerá de la naturaleza específica de los datos y de los requisitos de la aplicación en cuestión.

Esta sección proporciona una visión general de los métodos de cuantización considerados, lo que facilita la elección del enfoque más adecuado según el contexto y los objetivos del procesamiento de señales digitales.

8 Implicaciones de los Avances en la Cuantización para el Futuro de la Inteligencia Artificial y el Procesamiento del Lenguaje Natural

En los últimos años, los avances en técnicas de cuantización han tenido un impacto significativo en el campo de la inteligencia artificial (IA) y el procesamiento del lenguaje natural (PLN). Estas innovaciones no solo han permitido una mejor eficiencia en el almacenamiento y procesamiento de datos, sino que también han abierto nuevas posibilidades para el desarrollo de modelos más rápidos y precisos. Al reflexionar sobre estas implicaciones y mirar hacia el futuro, podemos vislumbrar varias tendencias y desafíos que podrían dar forma al panorama de la IA y el PLN.

8.1 Cierre de Brechas de Eficiencia

Uno de los principales logros de los avances en cuantización es la capacidad para ejecutar modelos de IA y PLN de alta complejidad en dispositivos con recursos limitados. Esto incluye desde dispositivos móviles hasta sistemas embebidos en el Internet de las cosas (IoT). La cuantización eficiente permite que los modelos funcionen más rápido y con un consumo de energía reducido, lo que es crucial para aplicaciones en tiempo real y dispositivos alimentados por batería.

8.2 Desarrollo de Modelos Más Complejos

Con la capacidad de cuantizar modelos complejos, los investigadores y desarrolladores pueden enfocarse en la creación de arquitecturas más profundas y sofisticadas. Esto incluye modelos de IA que pueden comprender y generar lenguaje humano de manera más natural, lo que abre la puerta a aplicaciones como traducción automática avanzada, generación de texto creativo y conversaciones más contextuales y significativas en chatbots y asistentes virtuales.

8.3 Desafíos en la Preservación del Significado

A medida que los modelos se vuelven más complejos y se aplican técnicas de cuantización agresiva para hacerlos más eficientes, surge el desafío de preservar el significado y la coherencia semántica. La cuantización excesiva puede llevar a la pérdida de información crucial, lo que afecta la calidad de las respuestas generadas por modelos de PLN. La investigación futura deberá abordar cómo mitigar este problema para garantizar que la calidad del lenguaje generado no se vea comprometida.

8.4 Integración de la Cuantización en Plataformas de Desarrollo

A medida que la cuantización se convierte en una parte integral del desarrollo de modelos de IA y PLN, las plataformas y frameworks de desarrollo deberán proporcionar herramientas más robustas para facilitar este proceso. Las bibliotecas de cuantización y las API que simplifican la implementación de técnicas de cuantización se volverán cada vez más importantes para los desarrolladores, permitiéndoles centrarse en la creación y mejora de modelos en lugar

9 Bibliografía

frantar2021optimal; Optimal Brain Compression: A Framework for Accurate Post-Training Quantization and Pruning; Frantar, Elias and Singh, Sidak Pal and Alistarh, Dan; 2023; <https://arxiv.org/pdf/2208.11580.pdf>

AutoGPTQ: un paquete de cuantización de LLMs fácil de usar – EcoAGI. (2023, 6 abril). <https://ecoagi.ai/es/topics/ChatGPT/auto-gptq>

Repositoria, Repositoria. (2023). Local AI Playground. RepositorIA. <https://repositoria.com/experimentos/local-ai-playground/>

Rustformers. (s. f.). LLM/crates/ggml/README.md at main · Rustformers/llm. GitHub. <https://github.com/rustformers/llm/blob/main/crates/ggml/README.md>

Eightify. (2023, 19 septiembre). Tipo de datos flotante normal de 4 bits para ajustar modelos de lenguaje en GPU. Resúmenes de Youtube de Eightify. <https://eightify.app/es/summary/computer-science-and-technology/4-bit-normal-float-data-type>

Labonne, M. (2023, 5 septiembre). Quantize Llama models with GGML and Llama.CPP — Towards Data science. Medium. <https://towardsdatascience.com/quantize-llama-models-with-ggml-and-llama-cpp-3612dfbcc172>