Introducción a la Cuantización

Miguel González

23 de octubre de 2023

1. Investigación previa

La cuantización se refiere al proceso de convertir valores continuos, como números de punto flotante en redes neuronales, en valores discretos. Este proceso es esencial para implementar modelos de inteligencia artificial en hardware con recursos limitados, como dispositivos móviles o sistemas embebidos. La cuantización se utiliza comúnmente para comprimir modelos y mejorar su eficiencia computacional sin una pérdida significativa de precisión. Hay dos enfoques principales para la cuantización en el contexto de la inteligencia artificial: Cuantización Posterior al Entrenamiento (PTQ) y Entrenamiento Consciente de la Cuantización (QAT). Aquí se describen ambos métodos y sus diferencias:

- **Cuantización Posterior al Entrenamiento (PTQ):**
    1. **Después del Entrenamiento:** PTQ se realiza después de que un modelo de inteligencia artificial ya ha sido entrenado. La idea es tomar un modelo previamente entrenado, que generalmente funciona en números de punto flotante de 32 bits, y cuantizarlo en números de bits más bajos para su implementación en hardware con limitaciones de recursos.
    2. **Uso de Datos de Validación:** En PTQ, se utiliza un conjunto de datos de validación que no se utilizó en el entrenamiento original. Este conjunto de datos se usa para evaluar el impacto de la cuantización en el rendimiento del modelo.
    3. **Optimización Posterior:** Durante el proceso de cuantización, se optimizan los parámetros del modelo para minimizar la pérdida de precisión. Se pueden aplicar diferentes técnicas de redondeo, como redondeo simétrico o redondeo estocástico, para minimizar la pérdida de información en la conversión de números de punto flotante a números enteros.
    4. **Ajuste del Modelo:** El modelo cuantizado resultante puede requerir ajustes en la estructura, como la reducción del número de unidades de neuronas o capas. Esto es necesario para garantizar que el modelo cuantizado se ajuste a la memoria y la potencia disponibles en el hardware de destino.
- **Entrenamiento Consciente de la Cuantización (QAT):**
    1. **Durante el Entrenamiento:** En el enfoque de QAT, la cuantización se realiza durante el entrenamiento del modelo, en lugar de aplicarse después. Esto significa que el modelo se entrena directamente en representaciones cuantizadas en lugar de números de punto flotante de 32 bits.
    2. **Simulación de Cuantización:** Durante el entrenamiento, se simula la cuantización aplicando las operaciones de cuantización al modelo, pero sin forzar una representación cuantizada en el hardware subyacente. Esto permite que el modelo aprenda a lidiar con la pérdida de precisión desde el principio.

3. **Uso de Funciones de Pérdida Personalizadas:** En QAT, es común utilizar funciones de pérdida personalizadas que penalizan la pérdida de precisión, lo que ayuda a guiar al modelo a cuantizar de manera óptima sin perder mucho rendimiento.
4. **Reducción de la Pérdida de Precisión:** QAT tiende a producir modelos cuantizados con una pérdida de precisión menor en comparación con PTQ, ya que el modelo se entrena desde el principio para operar en representaciones cuantizadas.

En resumen, PTQ cuantiza un modelo preentrenado después del entrenamiento, mientras que QAT entrena un modelo para cuantizarlo desde el principio. QAT suele ofrecer un mejor rendimiento y menor pérdida de precisión, pero puede requerir más recursos de entrenamiento. La elección entre estos dos enfoques depende de las restricciones de recursos y los requisitos de rendimiento de la aplicación en particular.

2. Configuración inicial

2.1. Obtenga un modelo preentrenado GPT-2

Estos han sido los resultados obtenidos usando un modelo gpt2 preentrenado

modelo-gpt2.py > ...

```python
from transformers import GPT2LMHeadModel, GPT2Tokenizer

model_name = "gpt2"  # Nombre del modelo preentrenado (puedes cambiarlo según tus
model = GPT2LMHeadModel.from_pretrained(model_name)
tokenizer = GPT2Tokenizer.from_pretrained(model_name)

input_text = input("Ingrese el texto:")
input_ids = tokenizer.encode(input_text, return_tensors="pt")
output = model.generate(input_ids, max_length=50, num_return_sequences=1, no_repea

generated_text = tokenizer.decode(output[0], skip_special_tokens=True)
print(generated_text)

```

PROBLEMS OUTPUT TERMINAL PORTS COMMENTS DEBUG CONSOLE

```
order to see activate developer mode, see this article: https://docs.microsoft.com/en-us/w
able-your-device-for-development
  warnings.warn(message)
Downloading (…)neration_config.json: 100%|                                        | 124/1
Downloading (…)olve/main/vocab.json: 100%|                                        | 1.04M/1.04
Downloading (…)olve/main/merges.txt: 100%|                                        | 456k/456
Downloading (…)/main/tokenizer.json: 100%|                                        | 1.36M/1.36
Ingrese el texto:Hola modelo llm gpt2 como estas
The attention mask and the pad token id were not set. As a consequence, you may observe un
pass your input's `attention_mask` to obtain reliable results.
Setting `pad_token_id` to `eos_token_id`:50256 for open-end generation.
Hola modelo llm gpt2 como estas de la mensa de los más.

The monsieur de monde de las mondos de lugar de una mondo de sua.
```

2.2. Establezca una métrica base del rendimiento del modelo sin cuantizar

modelo-gpt2.py > ...

```python
from transformers import GPT2LMHeadModel, GPT2Tokenizer

# Cargar el modelo preentrenado GPT-2
model_name = "gpt2"
model = GPT2LMHeadModel.from_pretrained(model_name)
tokenizer = GPT2Tokenizer.from_pretrained(model_name)
#Limpiar terminal
print("\033c")

# Generar texto de muestra
prompt = "En un mundo lejano"
input_ids = tokenizer.encode(prompt, return_tensors="pt")
output = model.generate(input_ids, max_length=50, num_return_sequences=1, no_rep

for sequence in output:
    generated_text = tokenizer.decode(sequence, skip_special_tokens=True)
    print("Texto Generado:", generated_text)

```

PROBLEMS OUTPUT TERMINAL PORTS COMMENTS DEBUG CONSOLE

```
The attention mask and the pad token id were not set. As a consequence, you may observe
pass your input's `attention_mask` to obtain reliable results.
Setting `pad_token_id` to `eos_token_id`:50256 for open-end generation.
Texto Generado: En un mundo lejano, una uno uni, de unia unio, en unicu, por unico, con
quando unas, se un
```

3. Representación de Coma Flotante

3.1. Investigar sobre la representación de números de coma flotante, centrándose en FP32, FP16 y BF16

La representación de números de coma flotante es un formato binario utilizado para representar números reales (números con decimales) en sistemas digitales, como computadoras. Se utilizan varios estándares para la representación de números de coma flotante, incluyendo FP32 (formato de 32 bits), FP16 (formato de 16 bits) y BF16 (formato de 16 bits).

1. **FP32 (32 bits - IEEE 754):**
   - **Precisión:** FP32 utiliza 32 bits para representar números de coma flotante. Esto proporciona una precisión razonable para una amplia gama de aplicaciones científicas y de ingeniería.
   - **Rango:** Permite representar números en un amplio rango, desde valores muy pequeños (en el orden de 1e-38) hasta valores muy grandes (en el orden de 1e38).
   - **Formato:** Utiliza 1 bit para el signo, 8 bits para el exponente y 23 bits para la mantisa.

- **Uso común:** FP32 es ampliamente utilizado en aplicaciones que requieren una precisión moderada, como cálculos científicos, gráficos 3D y procesamiento de señales.

2. **FP16 (16 bits - IEEE 754 mitad precisión):**

   - **Precisión:** FP16 utiliza 16 bits para representar números de coma flotante, lo que resulta en una menor precisión en comparación con FP32. La mantisa se almacena en 10 bits.
   - **Rango:** El rango es menor que FP32, lo que significa que no puede representar números tan pequeños ni tan grandes.
   - **Formato:** Al igual que FP32, utiliza 1 bit para el signo, 5 bits para el exponente y 10 bits para la mantisa.
   - **Uso común:** FP16 se utiliza en aplicaciones donde la memoria y la velocidad de cálculo son críticas, pero la precisión puede sacrificarse. Ejemplos incluyen el aprendizaje profundo (donde se utiliza para acelerar cálculos en redes neuronales) y aplicaciones de gráficos en tiempo real.

3. **BF16 (16 bits - Brain Floating-Point):**

   - **Precisión:** BF16 también utiliza 16 bits para representar números de coma flotante, pero a diferencia de FP16, BF16 destina más bits al exponente y menos a la mantisa, lo que proporciona una precisión intermedia entre FP16 y FP32.
   - **Rango:** El rango es similar al de FP32, lo que permite representar una amplia gama de valores.
   - **Formato:** BF16 utiliza 1 bit para el signo, 8 bits para el exponente y 7 bits para la mantisa.
   - **Uso común:** BF16 se utiliza en aplicaciones que requieren un equilibrio entre precisión y eficiencia de cálculo. Es común en el aprendizaje profundo, ya que ofrece un mejor rendimiento en comparación con FP16 en algunas tareas.

En resumen, FP32 ofrece la mayor precisión pero a costa de más memoria y cálculos, mientras que FP16 y BF16 son formatos de coma flotante de menor precisión que se utilizan en aplicaciones donde la eficiencia de cálculo y el uso de memoria son factores críticos. La elección entre estos formatos dependerá de las necesidades específicas de la aplicación y las limitaciones de hardware.

3.2. Documente las ventajas y desventajas de cada tipo en términos de precisión, rango y uso de memoria

- **Precisión:**

  1. **FP32 (Floating-Point 32-bit):**
     - **Ventajas:** Ofrece alta precisión y puede representar números con una gran cantidad de dígitos decimales. Apropiado para aplicaciones donde se requiere alta precisión numérica, como cálculos científicos y de ingeniería.
     - **Desventajas:** Requiere más bits, lo que aumenta el uso de memoria y reduce el rendimiento en comparación con formatos de menor precisión.
  2. **FP16 (Half-Precision Floating-Point):**

- **Ventajas:** Es más eficiente en términos de memoria y velocidad de procesamiento debido a su menor número de bits. Apropiado para aplicaciones donde la precisión numérica puede ser sacrificada por un mejor rendimiento, como gráficos 3D y algunas tareas de aprendizaje profundo.
- **Desventajas:** Ofrece una precisión significativamente menor que FP32 y no es adecuado para todas las aplicaciones, especialmente aquellas que requieren alta precisión.

3. **BF16 (Bfloat16 o Brain Floating-Point):**
   - **Ventajas:** Proporciona una precisión intermedia, que es más alta que FP16 pero aún más eficiente en términos de memoria y velocidad en comparación con FP32. Es especialmente útil en aplicaciones de aprendizaje profundo donde la pérdida de precisión es tolerable y se prioriza la eficiencia.
   - **Desventajas:** Aunque es más preciso que FP16, todavía es menos preciso que FP32 y puede no ser adecuado para aplicaciones que requieren alta precisión numérica.

- **Rango:**

1. **FP32 (Floating-Point 32-bit):** Cubre un amplio rango de valores numéricos sin desbordamiento o pérdida de precisión para la mayoría de las aplicaciones.
2. **FP16 (Half-Precision Floating-Point):** Tiene un rango limitado en comparación con FP32, lo que significa que no puede representar números extremadamente grandes o pequeños sin pérdida de precisión.
3. **BF16 (Bfloat16 o Brain Floating-Point):** Ofrece un rango similar al de FP16, que es más limitado en comparación con FP32.

- **Uso de Memoria:**

1. **FP32 (Floating-Point 32-bit):** Requiere más memoria en comparación con FP16 y BF16 debido a su mayor cantidad de bits por número.
2. **FP16 (Half-Precision Floating-Point):** Es más eficiente en términos de uso de memoria, ya que utiliza menos bits por número.
3. **BF16 (Bfloat16 o Brain Floating-Point):** También es eficiente en términos de memoria, ya que utiliza la misma cantidad de bits que FP16, pero ofrece una mayor precisión.

En resumen, la elección entre FP32, FP16 y BF16 dependerá de las necesidades específicas de la aplicación. FP32 es adecuado cuando se requiere alta precisión, mientras que FP16 y BF16 son útiles cuando la eficiencia en términos de memoria y velocidad es prioritaria, y se puede tolerar cierta pérdida de precisión. La elección debe equilibrar las ventajas y desventajas de cada formato en función de los requisitos de la aplicación.

4. Implementación PTQ

4.1. Aplique la técnica PTQ al modelo GPT-2

Este código primero carga el modelo GPT-2 preentrenado y su tokenizer. Luego, crea un modelo cuantizado utilizando la configuración de cuantización de PTQ **(QuantizationConfig.from-float(model))** y carga los pesos del modelo original. A continuación, el código coloca el modelo en

modo de evaluación **(quantized-model.eval())** y cuantiza el input **(input-ids)** antes de realizar la generación de texto.

ptq.py > ...

```python
import torch
from transformers import GPT2LMHeadModel, GPT2Tokenizer
from transformers import QuantizationConfig, QuantizedGPT2LMHeadModel

# Cargar el modelo preentrenado GPT-2
model_name = "gpt2"
model = GPT2LMHeadModel.from_pretrained(model_name)
tokenizer = GPT2Tokenizer.from_pretrained(model_name)

# Cuantización del modelo
quantization_config = QuantizationConfig.from_float(model)
quantized_model = QuantizedGPT2LMHeadModel(quantization_config)
quantized_model.load_state_dict(model.state_dict())

# Generar texto de muestra
prompt = "En un mundo lejano"
input_ids = tokenizer.encode(prompt, return_tensors="pt")

# Es importante cambiar el modelo para que esté en modo evaluación
quantized_model.eval()

# Cuantización del input
input_ids = input_ids.to(dtype=torch.int8)

# Realizar la generación de texto
output = quantized_model.generate(input_ids, max_length=50, num_return_sequences=1, no_repeat_ngram_s

for sequence in output:
    generated_text = tokenizer.decode(sequence, skip_special_tokens=True)
    print("Texto Generado:", generated_text)

```

Ln 31, Col 1 Spaces: 4 UTF-8 CRLF {}

4.2. Evalúe el rendimiento del modelo después de la cuantización

He tenido problema con la ejecución del código, es solo un esbozo básico de cómo cuantizar un modelo GPT-2 en PyTorch. Hay que tener en cuenta que los detalles específicos y las configuraciones de cuantización pueden variar según tus necesidades y recursos. La cuantización es un proceso delicado que a menudo requiere experimentación y ajustes finos para obtener el mejor equilibrio entre precisión y rendimiento. Además, puede ser necesario modificar el procesamiento de entrada y salida para que coincida con el formato cuantizado.

5. Análisis y Conclusiones:

5.1. Compare el rendimiento y el tamaño del modelo original con sus versiones cuantizadas

1. **Tamaño del Modelo:**
   - **Modelo Original (GPT-2):** El modelo GPT-2 original es significativamente más grande en términos de tamaño de almacenamiento. Puede tener cientos de megabytes o incluso varios gigabytes, dependiendo de la variante del modelo.
   - **Modelo Cuantizado por PTQ:** El modelo cuantizado es más pequeño en tamaño de almacenamiento en comparación con el modelo original. La cuantización reduce el tamaño del modelo al reducir la precisión de los parámetros.
2. **Rendimiento:**
   - **Modelo Original (GPT-2):** El modelo original suele ser más rápido en la generación de texto debido a su alta precisión. Sin embargo, esto puede requerir hardware más potente.
   - **Modelo Cuantizado por PTQ:** El modelo cuantizado tiende a ser más rápido en la inferencia debido a la representación más compacta de los parámetros. Puede funcionar eficientemente en hardware menos potente.

5.2. Reflexione sobre los trade-offs entre tamaño, rendimiento y precisión en modelos de lenguaje

Al comparar el rendimiento y el tamaño del modelo original GPT-2 con su versión cuantizada por PTQ, hay varios trade-offs notables entre tamaño, rendimiento y precisión:

1. **Precisión:**
   - **Modelo Original (GPT-2):** El modelo GPT-2 original es más preciso en términos de la representación de los números de punto flotante. Esto significa que puede generar texto de mayor calidad y coherencia.
   - **Modelo Cuantizado por PTQ:** El modelo cuantizado sacrifica la precisión al representar números de punto flotante con menos bits (generalmente 8 bits). Esto puede llevar a una ligera degradación en la calidad de la generación de texto.
2. **Aplicación:**
   - **Modelo Original (GPT-2):** Si la calidad de la generación de texto es fundamental y el tamaño del modelo no es una restricción, el modelo original es preferible. Esto podría ser relevante en aplicaciones de generación de contenido de alta calidad.
   - **Modelo Cuantizado por PTQ:** Si el tamaño del modelo y la eficiencia en la inferencia son más importantes que la calidad exacta de la generación, el modelo cuantizado puede ser la elección correcta. Esto podría ser útil en aplicaciones de dispositivos con recursos limitados o en sistemas que requieren una alta velocidad de generación.

En resumen, la elección entre el modelo GPT-2 original y su versión cuantizada por PTQ depende de las necesidades específicas de la aplicación. Si la precisión y la calidad del texto son fundamentales, el

modelo original es preferible. Si se trata de aplicaciones donde el tamaño y la eficiencia en la inferencia son esenciales, la versión cuantizada puede ser la mejor opción, a pesar de una pequeña pérdida en la precisión. La cuantización es una técnica valiosa para equilibrar el rendimiento y el tamaño del modelo en situaciones donde el costo computacional y el almacenamiento son factores críticos.