Advanced Natural Language Processing

Assignment 4 Report

Mihika Sanghi

2021113014

Quantization From Scratch

In this assignment, we implemented model quantization using the bitsandbytes library. The task settings were as follows:

1. **Type of Task**: Language Modelling
2. **Dataset**: Penn Tree Bank Dataset
3. **Base Model**: GPT2 LM Head Model

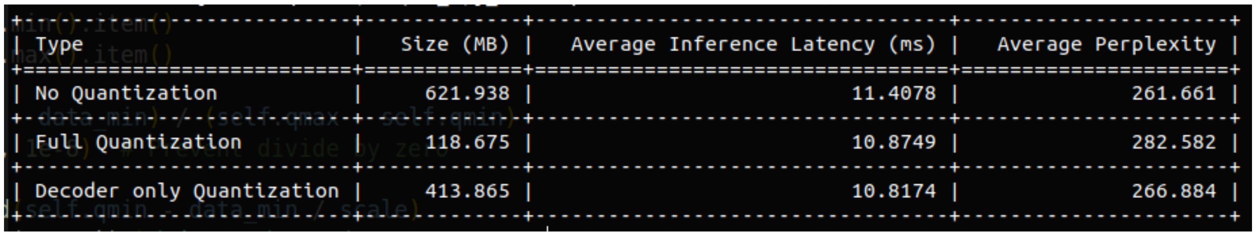
We performed inference on the test split of the dataset (3761 samples) using the base model and its quantized variants. The quantized variants were:

1. Fully Quantized Model : The entire model was quantized to int8. Note that only weights and biases were quantized and not the activations of the layers. Quantization done asymmetrically.

2. Decoder Only Quantized Model : Random Decoder Blocks were quantized (in our case, block 2, 4 and 6). Other parameters remain unchanged. Quantization done asymmetrically.

We performed inference on the test split of the dataset using the base model and its quantized variants. The quantized variants were:

Results:



```
+--------------------------+------------+-----------------------------------+----------------------+
| Type                     |  Size (MB) |  Average Inference Latency (ms)   |  Average Perplexity  |
+==========================+============+===================================+======================+
| No Quantization          |    621.938 |                           11.4078 |              261.661 |
+--------------------------+------------+-----------------------------------+----------------------+
| Full Quantization        |    118.675 |                           10.8749 |              282.582 |
+--------------------------+------------+-----------------------------------+----------------------+
| Decoder only Quantization |   413.865 |                           10.8174 |              266.884 |
+--------------------------+------------+-----------------------------------+----------------------+
```

Model Size Analysis

The size comparison follows an intuitive pattern: the base model occupies the most storage, with the decoder-quantized version taking up an intermediate amount, and the fully quantized model requiring the least space. This pattern directly results from the quantization process - when we convert parameters from higher precision (e.g., FP32/FP16) to lower precision (int8), each parameter requires fewer bits to store. Consequently, as more parameters undergo quantization, the model's overall memory footprint decreases proportionally. This is why the fully quantized model, with all its parameters in int8 format, achieves the most significant size reduction.

Latency Analysis

Interestingly, our testing revealed comparable inference times across all model variants, which might seem counterintuitive at first glance. While quantization typically aims to reduce latency, the similar performance can be attributed to GPU architecture optimization. Modern GPUs are specifically designed to excel at FP32/FP16 operations, which means the base model already runs efficiently on these devices. Additionally, while some quantization implementations need to dequantize parameters during inference (potentially increasing latency), our approach differentiates itself by performing dequantization during initialization. This means we measure model size post-quantization but pre-dequantization, providing a more accurate representation of the model's memory efficiency.

Perplexity Analysis

A clear performance hierarchy emerges in our perplexity measurements: the base model achieves the best (lowest) perplexity, followed by the decoder-quantized model, with the fully quantized model showing the highest perplexity. This degradation in performance directly correlates with the degree of quantization applied. As we reduce numerical precision through quantization, we inevitably sacrifice some information fidelity. This loss of precision can impact the model's ability to capture subtle patterns in the data, leading to less accurate predictions and, consequently, higher perplexity scores. The results demonstrate that while quantization offers valuable memory savings, it comes with a measurable trade-off in model performance.

This part demonstrates that with increasing quantization, the model size decreases but perplexity increases (the model performs worse).

Bitsandbytes Integration and NF4 Quantization

Experimental Framework

For our investigation into model compression techniques, we leveraged the bitsandbytes library to implement various quantization strategies. Our experimental setup consisted of:

Core Components

- **Task Domain**: Language Modeling
- **Data Source**: Penn Tree Bank Dataset
- **Model Architecture**: GPT2 LM Head Model
- **Test Sample Size**: 3,761 instances

Quantization Approaches

We evaluated three distinct quantization methods:

1. **Linear 8-bit Quantization**
   - Full model conversion to 8-bit precision
   - Uniform linear scaling implementation
   - Utilizing bitsandbytes framework
2. **Linear 4-bit Quantization**
   - Complete model transformation to 4-bit precision
   - Linear weight distribution
   - Implementation via bitsandbytes library
3. **NF4 (Non-linear 4-bit) Quantization**
   - 4-bit precision with non-linear scaling
   - Logarithmic weight distribution
   - Advanced implementation through bitsandbytes

Quantization Methodology Comparison

The key distinction between linear and NF4 quantization lies in their weight distribution approaches:

Linear Quantization:

- Employs uniform spacing across the quantization range
- Equal representation for all value magnitudes
- Straightforward weight scaling implementation

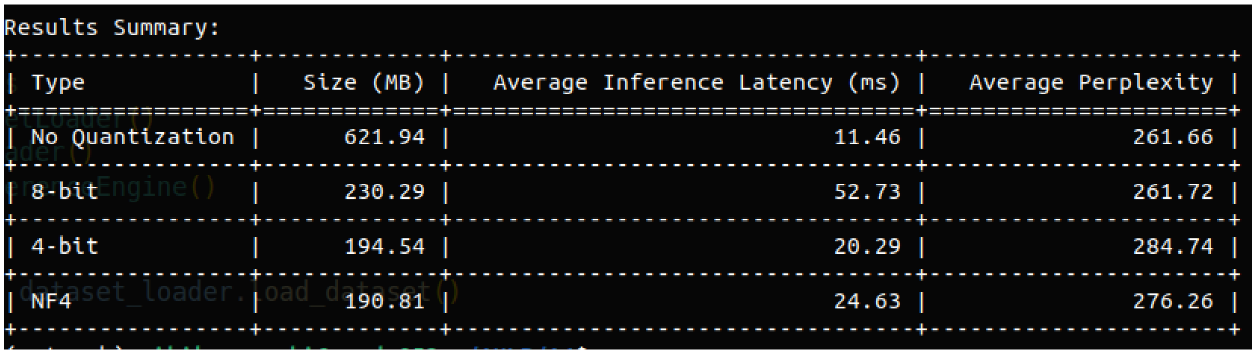
NF4 Quantization:

- Implements logarithmic-style value distribution
- Concentrates quantization levels near zero
- Optimized for neural network weight distributions
- Provides enhanced precision for smaller values
- Reduces precision for larger, less frequent values

This technical setup allowed us to comprehensively evaluate the impact of different quantization strategies on model performance and efficiency.

1. **8-bit Quantization**
    - Full model quantization to 8 bits using linear quantization
    - Implemented using bitsandbytes library
2. **4-bit FP4 Quantization**
    - Full model quantization to 4 bits using linear quantization
    - Implemented using bitsandbytes library with FP4 format
3. **4-bit NF4 Quantization**
    - Full model quantization to 4 bits using non-linear quantization
    - Implemented using bitsandbytes library with NF4 format

Our results for these models are as follows:



```
Results Summary:
+-----------------+------------+----------------------------------+---------------------+
| Type            |  Size (MB) |  Average Inference Latency (ms) |  Average Perplexity |
+=================+============+==================================+=====================+
| No Quantization |     621.94 |                            11.46 |              261.66 |
+-----------------+------------+----------------------------------+---------------------+
| 8-bit           |     230.29 |                            52.73 |              261.72 |
+-----------------+------------+----------------------------------+---------------------+
| 4-bit           |     194.54 |                            20.29 |              284.74 |
+-----------------+------------+----------------------------------+---------------------+
| NF4             |     190.81 |                            24.63 |              276.26 |
+-----------------+------------+----------------------------------+---------------------+
```

Model Size Analysis

The relationship between quantization and model size demonstrates a clear inverse correlation - as we increase quantization (decrease bit precision), the model's memory requirements diminish proportionally. Our measurements confirm this pattern, with the base model consuming the most memory, followed by a significant reduction in the 8-bit variant, and further compression achieved in the 4-bit implementations. This reduction occurs because each parameter requires fewer bits for storage under higher quantization levels.

Latency Analysis

Contrary to what might be expected, our latency measurements revealed some surprising patterns. While one might anticipate faster inference times with smaller quantized models, we observed that:

1. The base model actually demonstrated the fastest inference times
2. 4-bit quantized versions showed intermediate latency
3. 8-bit quantization exhibited the slowest performance

This counter-intuitive behavior can be attributed to several technical factors:

1. **Runtime Dequantization Impact**
    - The bitsandbytes library performs dequantization during inference

    - This additional computational step introduces overhead during runtime
2. **Precision Conversion Differences**
    - 8-bit models required additional float16 conversion steps
    - 4-bit variants came pre-configured with float16, avoiding this extra conversion
    - This disparity particularly affected 8-bit model performance
3. **Algorithmic Complexity Variations**
    - NF4 shows slightly higher latency compared to standard 4-bit
    - Additional computational overhead from non-linear weight distribution
    - Trade-off between precision and processing speed

Perplexity Analysis

Our perplexity measurements reveal a clear quality-compression trade-off. The pattern shows:

1. **Base Model**: Best perplexity scores
2. **8-bit Quantization**: Moderate increase in perplexity
3. **4-bit Variants**: Highest perplexity, with an important distinction:
    - NF4 outperforms standard 4-bit linear quantization
    - This superiority stems from NF4's intelligent weight distribution
    - Better preservation of small magnitude parameters
    - More efficient information retention in commonly occurring value ranges

The superior performance of NF4 in the 4-bit category demonstrates how sophisticated quantization strategies can help mitigate the accuracy loss typically associated with aggressive compression.