Quantisation

Advanced NLP : Assignment 4

Varun Edachali (2022101029)

November 16, 2024

1 Quantisation from Scratch

Symmetric Quantisation was used for this purpose, for simplicity.

1.1 Whole Model Quantisation

This medium article was used as a guide in building custom Linear, Embedding and LayerNorm layers (which covers the spectrum of layers found in the model of choice) to quantise the model as a whole. The implementation spreads far beyond the article.

1.2 Selective Quantisation

Here, I chose to quantise only the layers found in the Self-Attention block, the Q_{Proj}, K_{Proj}, V_{Proj} and Out_{Proj} to analyse the efficiency of losing some precision in this region.

1.3 Analysis

All of the models were run on the first 3000 sentences of the PennTreeBank dataset. The perplexity was calculated as $2^{\sum cross-entropy-losses}$ on referring here and here. The model of choice was gpt-neo-125m.

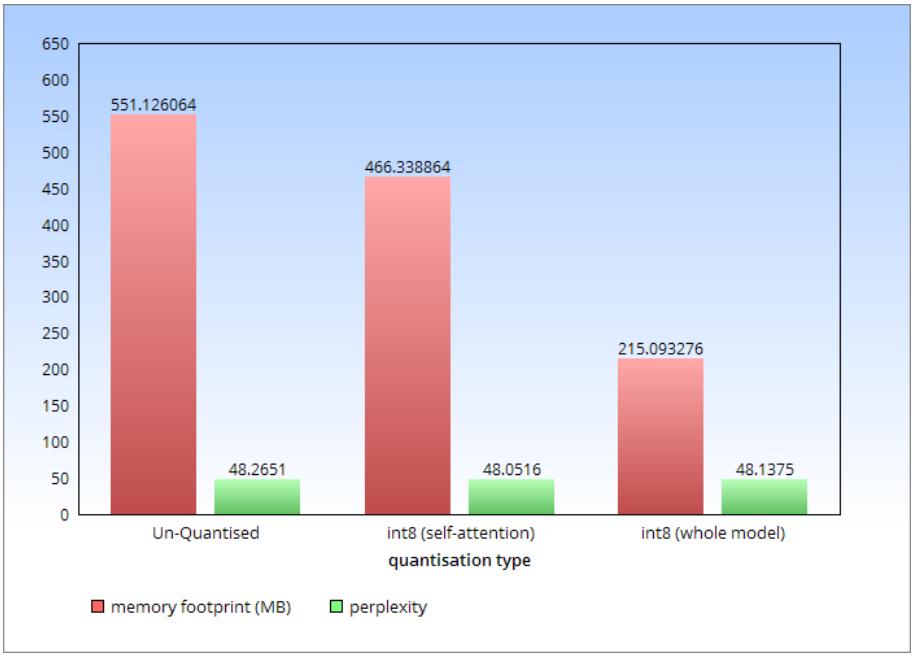

Figure 1: quantisation type against memory consumption and perplexity

We see a clear decrease in memory consumption without any significant effects on the

perplexity of the models in consideration. This validates quantisation as an approach to increase efficiency for the current task. Even on quantising the whole model, the perplexity effectively does not change while the memory footprint more than halves.

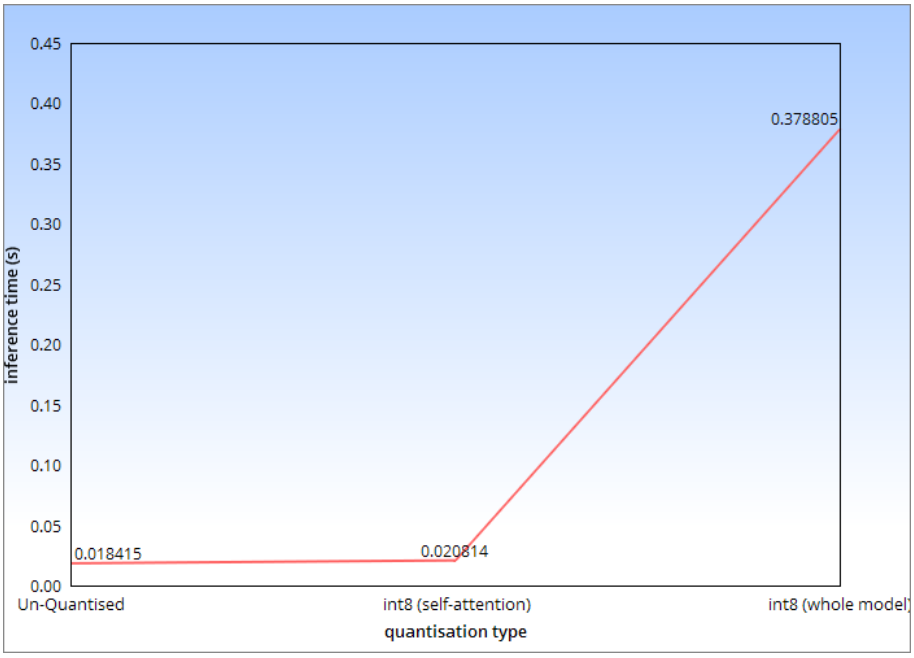

Figure 2: quantisation type against time taken

Note:

- The increase in inference time for quantised models may be expected, due to the extra work done in de-quantising during the forward passes.
- Here, the whole model quantisation consumed too much memory during inference (in the custom embedding layer I had to multiply a large scales matrix with a large weights matrix) forcing me to run it on CPU instead of GPU. This is the reason for the large time taken. Despite this, we can infer that whole model quantisation is not very-efficient (because of the substantially larger space it had to occupy on inference, forcing a less efficient computation mechanism).
- Only the custom embedding layer was this inefficient. When I did not quantise the embedding layer, only the linear and LayerNorm layers, the inference time was comparable to the selective quantisation - so this could likely be an implementation flaw too.

2 BitsAndBytes

2.1 Theory

2.1.1 Explain the concept of NF4 quantization and how it differs from linear quantization scales.

NF4 Quantisation uses a normal (gaussian) distribution to determine spacing between quantized values, with values being closer (more granular) towards zero. Since pre-trained neural network weights usually have a zero-centered normal distribution with standard deviation σ, the paper proposes transforming all weights to a single fixed distribution σ such that the distribution fits exactly into the range of our data-type.

We can set an arbitrary range for our data-type (so long as it can fit into 4-bits), but usually the range of $[-1, 1]$ is taken. The paper further goes into how exactly the quantisation is performed on this basis.

Fundamentally, the difference from linear-quantisation scales is that they adopt uniform spacing between quantised values.

2.1.2 Discuss the impact of linear vs. nonlinear quantization on model accuracy and efficiency.

The same paper as before proposes that the NF4 data type is information-theoretically optimal. They evaluated its performance on language modelling and some zero-shot tasks. The Pile Common Crawl mean perplexity for various models was about 34.34 for the linear Int4 model, while it was 27.41 for NF4 with double-quantisation. In addition, mean zero-shot accuracy over a number of challenges showed that NF4 improves performance significantly over Int4.

Purely introducing NF4 may not increase model efficiency much (as a matter of fact, the quantisation may be slower due to more involved calculations), but the introduction of double-quantisation leads to significant decrease in memory footprint without a large decrease in performance. This couple becomes both more accurate and more efficient than linear methods due to increased performance and decreased memory footprint.

2.2 Analysis

2.2.1 BnB-4 vs BnB-8

Here, I refer to the 4 bit and 8 bit representations in the BitsAndBytes config. The rest of the parameters remained default.

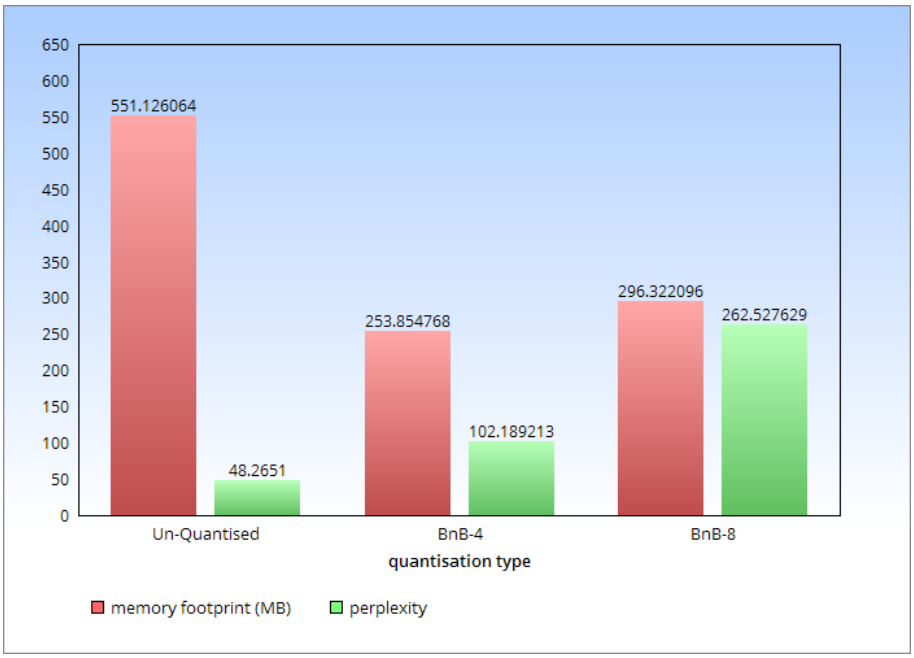

Figure 3: BitsAndBytes quantisation against memory and perplexity

We notice that both methodologies obtain substantial decreases in memory consumption, likely due to highly efficient implementations of quantisaton. The reason for the bad performance of BitsAndBytes' 8-bit representation can be attributed to the following:

- I could not run the 8-bit quantisation code locally, because some gpu's do not seem to support it.
- Consequently, I ran it on Kaggle, upon which, I saw the following warning during the run:

```
calculating perplexity...:   0%|                    | 0/3000 [00:00<?, ?it/s]/opt/conda/lib/python3.10/site-packages/bitsa
ndbytes/autograd/_functions.py:316: UserWarning: MatMul8bitLt: inputs will be cast from torch.float32 to float16 during quant
ization
  warnings.warn(f"MatMul8bitLt: inputs will be cast from {A.dtype} to float16 during quantization")
```

Figure 4: Kaggle Warning during BnB-8

  This warning may suggest that inputs are losing a very large amount of their precision as a result of their quantisation, potentially leading to highly inaccurate results.

It is difficult to comment on the decrease in performance in comparison to the ground-truth without knowing the internals of the implementation, however.

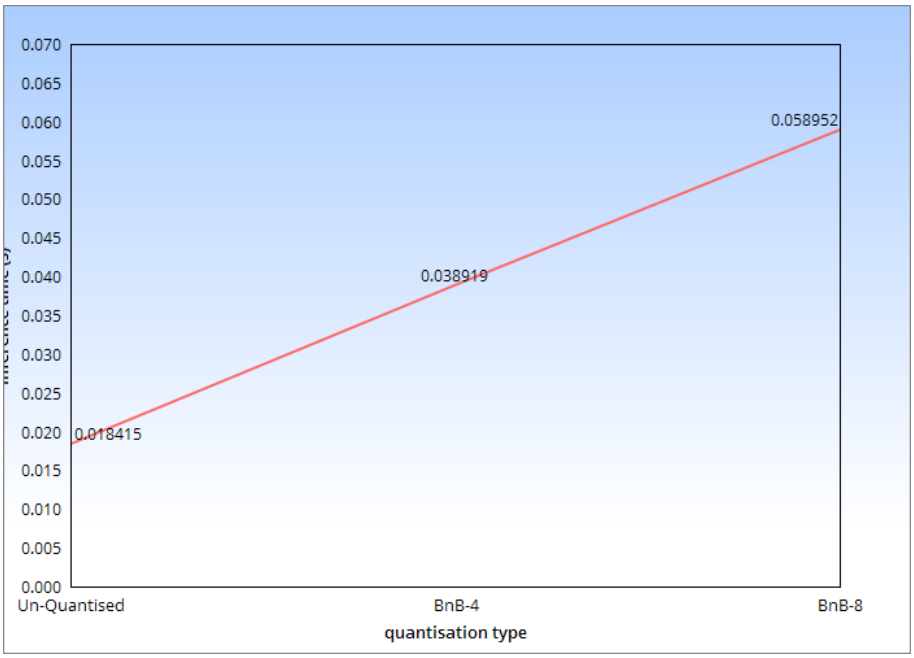

Figure 5: BitsAndBytes quantisation against time

The increase in inference time for quantised models may be expected, due to the extra work done in de-quantising during the forward passes. It is possible that BnB-8 is more or less efficient than suggested above due to the fact that it was run on GPU-P100 on Kaggle, as suggested above.

2.2.2 BnB-4 vs BnB-NF4

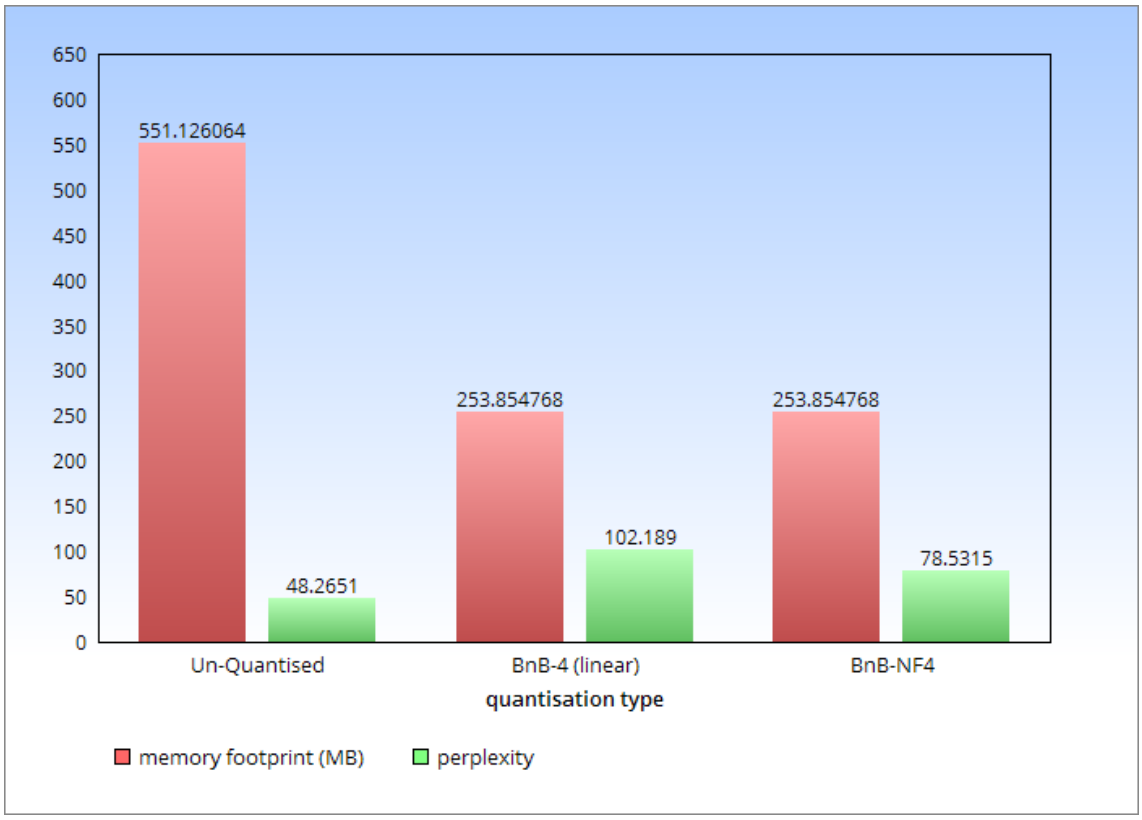

Figure 6: linear vs non linear memory and perplexity

As we may expect, both 4-bit representations occupy the same exact memory footprint. As suggested from the report above, non-linear representations tend to be more accurate, however. This conforms to our results.

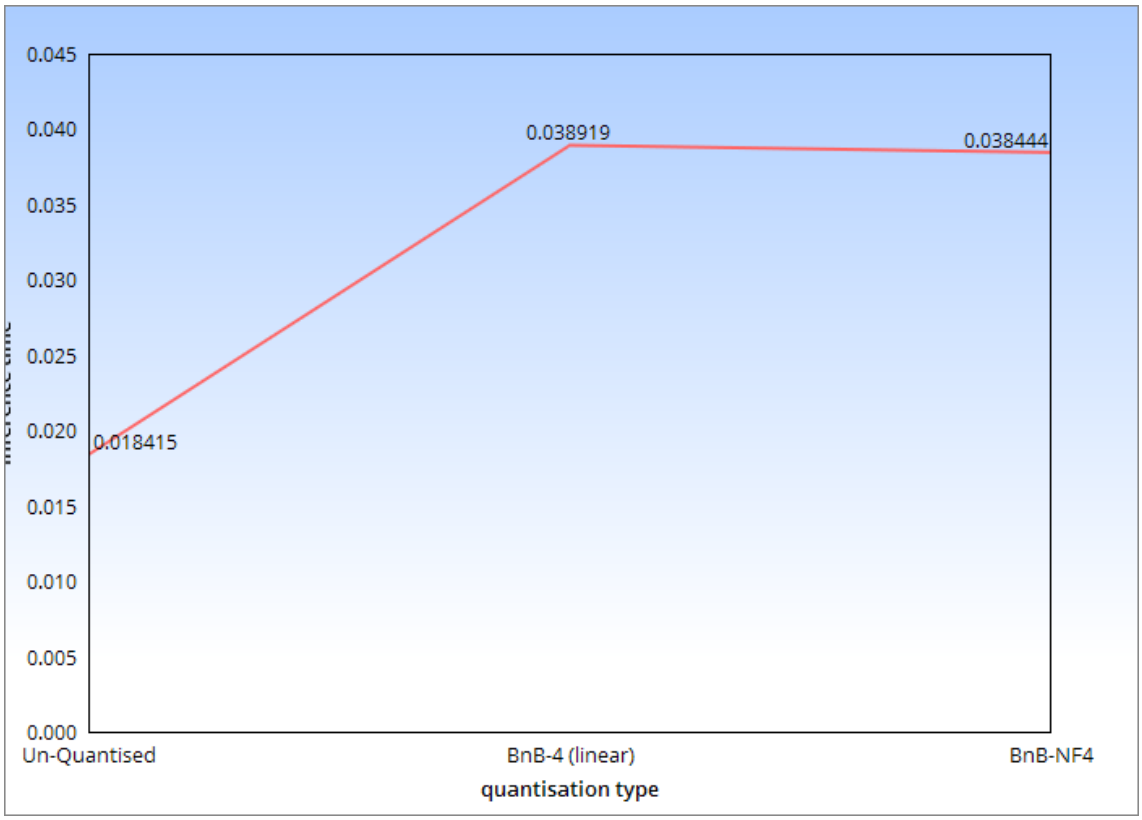

Figure 7: linear vs non linear time taken

Again, the increase in inference time of quantised models may be expected due to the extra work they do during the forward pass. Due to the similar fundamental bit-representations of the two methodologies, their inference time also seems similar. Theoretically, I expected NF4 to be slightly slower - this shift from expectations could be due to a number of external factors, but does not seem significant.

3 Lightweight Model Types for Local Deployment

I followed this official tutorial to download SmolLM-135M, convert it to a GGUF format, run it locally, and deploy it to HuggingFace-Hub.

GGML (GPT-Generated Model Language) is a tensor library for machine learning, facilitating large models and high performance. It enabled sharing models in a single file, enhancing usability. However, it had limited flexibility and could not add too much information about the model itself, often leading to compatibility issues with older models.

GGUF (GPT-Generated Unified Format) is a successor to GGML that is meant to be

more flexible and robust. It allows for the addition of new features while maintaining compatibility with older models.

Source: Phillip Gimmi.