Assignment4 - Himanshu Pal 2023701003

Part 1: Quantization from Scratch

```
python Part1.py
```

Full Quantization

Model Stats before quantization:

Footprint of the model in MBs: 15284.324352 MB

Time taken to generate text before quantization: 27.68s

> Detailed Model Analysis:

Layer Types:

- Phi3ForCausalLM: 1
- Phi3Model: 1
- Embedding: 1
- Dropout: 65
- ModuleList: 1
- Phi3DecoderLayer: 32
- Phi3Attention: 32
- Linear: 129
- Phi3RotaryEmbedding: 32
- Phi3MLP: 32
- SiLU: 32
- Phi3RMSNorm: 65

Memory Usage by dtype (MB): - float32: 14576.27 MB - float16: 0.00 MB - int8: 0.00 MB - other: 0.00 MB

Model Stats after quantization:

Footprint of the model in MBs: 4416.751616 MB

Memory Usage by dtype (MB):

- float32: 756.14 MB
- float16: 0.00 MB
- int8: 3456.00 MB
- other: 0.00 MB

W8A16LinearLayer Properties:

- int8_weights dtype: torch.int8
- scales dtype: torch.float32

Time taken to generate text after quantization: 221.07s

> Results:

Perplexity before quantization: 3.049601

Perplexity after quantization: 3.050290

Perplexity difference: 0.000689

Relative perplexity change: 0.022594%

Footprint of the model in MBs: 15284.324352 MB

Footprint of the model in MBs: 4416.751616 MB

Time taken to generate text before quantization: 27.68s

Time taken to generate text after quantization: 221.07s

Selective Quantization

Model Stats before quantization:

Footprint of the model in MBs: 15284.324352

Detailed Model Analysis:

Layer Types:

- Phi3ForCausalLM: 1
- Phi3Model: 1
- Embedding: 1
- Dropout: 65
- ModuleList: 1
- Phi3DecoderLayer: 32
- Phi3Attention: 32
- Linear: 129
- Phi3RotaryEmbedding: 32
- Phi3MLP: 32
- SiLU: 32
- Phi3RMSNorm: 65

Memory Usage by dtype (MB):

- float32: 14576.27 MB
- float16: 0.00 MB
- int8: 0.00 MB
- other: 0.00 MB

Time taken to generate text before quantization: 29.22s

Perplexity before quantization: 3.05

Model Stats after quantization:

Footprint of the model in MBs: 4416.751616

> Detailed Model Analysis:

Memory Usage by dtype (MB):

- float32: 756.14 MB
- float16: 0.00 MB
- int8: 3456.00 MB
- other: 0.00 MB

W8A16LinearLayer Properties:

- int8_weights dtype: torch.int8
- scales dtype: torch.float32 Time taken to generate text after quantization: 218.33207869529724

> Results:

Perplexity before quantization: 3.049601

Perplexity after quantization: 3.050290

Perplexity difference: 0.000689

Relative perplexity change: 0.022594%

Time taken to generate text before quantization: 29.22s

Time taken to generate text after quantization: 218.33s

Part 2: Bitsandbytes Integration and NF4 Quantization

Part 2.1: Bitsandbytes Quantization: Use Bitsandbytes to quantize a model from original precision (FP32/FP16) to both 8-bit and 4-bit formats.

```
python part2_1.pt
```

Results

ORIGINAL MODEL:

```
CPU Memory: 15.57 GB
GPU Memory: 0.00 GB
Inference Latency: 8.1272 seconds
Perplexity: 3.05
```

8BIT MODEL:

```
CPU Memory: 3.24 GB
GPU Memory: 3.78 GB
Inference Latency: 2.3598 seconds
Perplexity: 3.13
```

4BIT MODEL:

```
CPU Memory: 3.76 GB
GPU Memory: 2.28 GB
Inference Latency: 1.1650 seconds
Perplexity: 3.31
```

Part 2.2: Apply NF4 quantization and compare its effectiveness with linear quantization.

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

NF4 (NormalFloat 4-bit) quantization is an advanced quantization technique introduced in the QLoRA paper, which differs significantly from traditional linear quantization scales.

NF4 quantization is designed specifically for normally distributed data, which is common in neural network weights and activations. The key concept behind NF4 is that it divides the quantization range into bins where each bin has an equal area under a standard normal distribution N(0, 1), normalized to the range [-1, 1].

Key features of NF4:

1. **Non-linear bin spacing**: Unlike linear quantization, NF4 uses non-uniformly spaced bins. Bins are closer together near zero and farther apart at the tails of the distribution.
2. **Optimized for normal distributions**: It's particularly effective for data that follows a normal distribution, which is often the case for neural network parameters.
3. **Equal probability bins**: Each quantization level represents an equal probability mass under the normal curve, rather than equal intervals on the number line.
4. **4-bit precision**: NF4 uses 4 bits per value, allowing for 16 distinct quantization levels.

Differences from Linear Quantization

1. **Distribution-aware**:
    - NF4: Tailored to the statistical properties of normally distributed data.
    - Linear: Uses evenly spaced intervals regardless of the underlying data distribution.
2. **Bin spacing**:
    - NF4: Non-uniform spacing with more levels near zero and fewer at the extremes.
    - Linear: Uniform spacing across the entire range.
3. **Precision in different ranges**:
    - NF4: Higher precision for values near the mean of the distribution, where most data points typically lie in neural networks.
    - Linear: Equal precision across the entire range, which may waste quantization levels on rarely occurring extreme values.
4. **Quantization error**:
    - NF4: Tends to have lower overall quantization error for normally distributed data.
    - Linear: May have higher error, especially for values near the center of the distribution.
5. **Implementation complexity**:
    - NF4: More complex to implement, requiring precomputed bin boundaries based on the normal distribution.
    - Linear: Simpler to implement with straightforward scaling and rounding operations.
6. **Adaptability**:
    - NF4: Fixed scheme optimized for normal distributions, may not adapt well to other distributions.
    - Linear: More general-purpose and can be applied to various types of data distributions.

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

Quantization techniques impact the efficiency and accuracy of neural networks in distinct ways. Here's a breakdown of how linear and nonlinear quantization compare:

Linear Quantization

- **Method**: Maps values to discrete levels with uniform spacing.
- **Efficiency**: Simple to implement, memory-efficient, and enables fast arithmetic operations.
- **Accuracy**: Works well for data with uniform distribution, but can lose accuracy when values are highly concentrated in certain regions.

Nonlinear Quantization

- **Method**: Uses non-uniform spacing, adapting to the data distribution (e.g., NF4 quantization).
- **Efficiency**: Often more complex but achieves high compression with minimal accuracy loss.
- **Accuracy**: Better for normally distributed data, especially in large language models, maintaining accuracy at low precisions like 4-bit.

Comparative Summary

- **Accuracy**: Nonlinear often outperforms linear quantization, especially at low bit widths, due to distribution adaptation.
- **Computation**: Linear is computationally simpler; nonlinear may require more processing power.
- **Memory**: Both reduce model size, but nonlinear can achieve higher compression rates.
- **Compatibility**: Linear has better hardware support, while nonlinear may need specialized hardware.
- **Adaptability**: Nonlinear techniques adapt well to various distributions and tasks, offering more versatility.

Practical Implications

- **Trade-offs**: Nonlinear quantization, like NF4, is advantageous for extreme compression needs (e.g., on edge devices) but may require tuning for best accuracy.
- **Deployment**: Linear quantization is better for environments with limited computational resources, while nonlinear methods excel in applications requiring high accuracy retention.

In conclusion, linear quantization is simple and broadly compatible, whereas nonlinear approaches like NF4 provide better accuracy preservation for data-heavy models. The choice depends on the specific needs of the deployment, balancing factors like accuracy, size, and hardware compatibility.

NF4 MODEL:

```
CPU Memory: 4.94 GB
GPU Memory: 0.78 GB
Inference Latency: 1.4796 seconds
Perplexity: 3.17
```