Task 0: Quantization Research

Introduction

Large language models such as **BERT** and **LLaMA** contain hundreds of millions or even billions of parameters. Running them in full precision (32-bit floats, FP32) requires huge memory and compute resources.

For example:

- BERT-base ($\approx$ 110M parameters) needs $\approx$ 440MB just for weights in FP32.
- LLaMA-13B ($\approx$ 13B parameters) needs $\approx$ 52GB in FP32.

This makes them difficult to deploy on resource-constrained devices (laptops, mobile, edge).

Quantization as a Solution

Quantization reduces model size and speeds up inference by storing parameters with lower precision datatypes:

- FP32 $\rightarrow$ FP16: $2\times$ smaller
- FP32 $\rightarrow$ INT8: $4\times$ smaller
- FP32 $\rightarrow$ INT4: $8\times$ smaller

Formula

If P is the number of parameters and b is the bit-width:

$$\text{Memory Size} = \frac{P \times b}{8} \text{ bytes}$$

For BERT-base ($P = 110 \times 10^6$):

$$\text{FP32: } 440\,\text{MB}, \quad \text{FP16: } 220\,\text{MB}, \quad \text{INT8: } 110\,\text{MB}, \quad \text{INT4: } 55\,\text{MB}$$

—

Graph Example

A simple matplotlib script to visualize model size vs. bit-width:

```
import matplotlib.pyplot as plt

params = 110_000_000
bit_widths = [32, 16, 8, 4]
sizes_mb = [params * b / 8 / (1024**2) for b in bit_widths]

plt.plot(bit_widths, sizes_mb, marker="o")
plt.title("Model␣Size␣vs␣Bit-Width␣(BERT-base␣~110M␣params)")
plt.xlabel("Precision␣(bits)")
plt.ylabel("Model␣Size␣(MB)")
plt.grid(True)
plt.show()
```

—

Code Examples

A) PyTorch Dynamic Quantization

```
import torch
from torch import nn
from torch.ao.quantization import quantize_dynamic

class SmallNet(nn.Module):
    def __init__(self):
        super().__init__()
        self.fc1 = nn.Linear(768, 768)
        self.fc2 = nn.Linear(768, 2)
    def forward(self, x):
        return self.fc2(torch.relu(self.fc1(x)))

fp32_model = SmallNet().eval()
quantized_model = quantize_dynamic(fp32_model, {nn.Linear},
   dtype=torch.qint8)

print("FP32␣size␣(MB):", sum(p.numel()*p.element_size() for p
    in fp32_model.parameters())/1024**2)
print("INT8␣size␣(MB):", sum(p.numel()*p.element_size() for p
    in quantized_model.parameters())/1024**2)
```

B) Hugging Face Transformers with bitsandbytes

```python
from transformers import AutoModelForSequenceClassification,
    AutoTokenizer

model_id = "distilbert-base-uncased"
tokenizer = AutoTokenizer.from_pretrained(model_id)

# Load model in 8-bit
model = AutoModelForSequenceClassification.from_pretrained(
    model_id,
    device_map="auto",
    load_in_8bit=True
)

text = "This is a simple test"
inputs = tokenizer(text, return_tensors="pt")
outputs = model(**inputs)
print(outputs.logits)
```

—

Conclusion

Quantization is an effective method to shrink large models like BERT and LLaMA, making them feasible to run on smaller devices. It reduces memory usage and speeds up inference while causing only a small loss in accuracy.

For more details, see the official Hugging Face Quantization Guide.