Title

Neural Compression Coprocessor: Enabling Efficient Inference of Large Language Models on Edge Devices

Problem Statement

Large language models (LLMs) require significant memory and computational resources, limiting their deployment on edge devices and in resource-constrained environments. This research aims to develop a novel method for compressing and decompressing LLM weights and activations on-the-fly, enabling efficient inference without permanent model modifications.

Motivation

Current approaches to improve LLM inference efficiency, such as quantization, pruning, and knowledge distillation, often trade off model performance for efficiency. These methods typically involve permanent modifications to the model, which can lead to irreversible loss of information and degraded performance. By leveraging neuromorphic computing principles, we can design a specialized coprocessor that dynamically compresses and decompresses LLM components during inference, potentially achieving better efficiency-performance trade-offs than existing methods.

Proposed Method

We introduce a Neural Compression Coprocessor (NCC) that sits between the main processor and memory. The NCC uses a specialized neural network architecture inspired by autoencoders to learn compact representations of LLM weights and activations. During inference, the NCC dynamically compresses and decompresses model components as they are transferred between memory and the main processor. The compression network is trained using a novel loss function that balances compression ratio, computational complexity, and fidelity to the original model outputs. Additionally, the NCC incorporates adaptive compression levels based on the current inference task and available resources.

Step-by-Step Experiment Plan

Step 1: Implement NCC Architecture

Design and implement the Neural Compression Coprocessor architecture using PyTorch or TensorFlow. The NCC should consist of an encoder network for compression and a decoder network for decompression. Use a bottleneck layer to achieve the desired compression ratio.

Step 2: Develop Training Pipeline

Create a training pipeline for the NCC. The loss function should include terms for reconstruction error, compression ratio, and computational complexity. Implement gradient-based optimization to train the NCC on LLM weight matrices and activation tensors.

Step 3: Integrate NCC with LLM

Modify an existing LLM implementation (e.g., HuggingFace Transformers) to incorporate the NCC. Implement hooks to intercept weight and activation transfers, routing them through the NCC for compression and decompression.

Step 4: Implement Adaptive Compression

Develop a mechanism for dynamically adjusting compression levels based on available resources and task requirements. This could involve training multiple compression levels and selecting the appropriate one at runtime.

Step 5: Prepare Evaluation Datasets

Select benchmark datasets for language modeling (e.g., WikiText-103), question answering (e.g., SQuAD), and text classification (e.g., GLUE benchmark). Prepare data loaders and evaluation scripts for each task.

Step 6: Baseline Experiments

Run inference experiments using uncompressed models, quantized models (e.g., 8-bit quantization), and pruned models (e.g., magnitude pruning) on the selected datasets. Measure inference speed, memory usage, and task-specific metrics (e.g., perplexity, F1 score) for each method.

Step 7: NCC Experiments

Conduct inference experiments using the LLM with integrated NCC. Evaluate performance across different compression ratios and adaptive compression settings. Measure the same metrics as in the baseline experiments.

Step 8: Edge Device Deployment

Deploy the NCC-enabled LLM on representative edge devices (e.g., Raspberry Pi, NVIDIA Jetson). Measure real-world performance, including inference latency, power consumption, and memory usage.

Step 9: Ablation Studies

Perform ablation studies to analyze the impact of different components of the NCC architecture and training process. This may include varying the compression network architecture, adjusting loss function components, and testing different adaptive compression strategies.

Step 10: Analysis and Visualization

Analyze the results, comparing the NCC approach to baselines across different metrics and tasks. Create visualizations to illustrate the trade-offs between compression ratio, inference speed, and model performance. Examine the learned compression representations to gain insights into what information is preserved or discarded.

Test Case Examples

Baseline Prompt Input (Uncompressed Model)

What is the capital of France?

Baseline Prompt Expected Output (Uncompressed Model)

The capital of France is Paris.

Proposed Prompt Input (NCC-Enabled Model)

What is the capital of France?

Proposed Prompt Expected Output (NCC-Enabled Model)

The capital of France is Paris.

Explanation

In this example, both the uncompressed baseline model and the NCC-enabled model should produce the same correct answer. However, the NCC-enabled model would achieve this with reduced memory usage and potentially faster inference time, especially on resource-constrained devices. The key is to demonstrate that the NCC can maintain accuracy while improving efficiency.

Fallback Plan

If the proposed NCC method fails to achieve satisfactory performance improvements over baselines, we can pivot the project in several directions. First, we could conduct a detailed analysis of where and why the compression fails, potentially revealing insights into the structure of LLM weights and activations. This could lead to a paper on the compressibility of different LLM components. Second, we could explore hybrid approaches that combine the NCC with existing methods like quantization or pruning, potentially achieving better results than either approach alone. Finally, we could investigate the use of the NCC for other tasks beyond inference, such as efficient fine-tuning or continual learning, where dynamic compression might offer unique advantages.

Ranking Score: 5