Compressing code generation language models on CPUs

Using Group Lasso pruning and post-training quantization

Author: Dan Sochirca
Supervisors: Maliheh Izadi, Ali Al-Kaswan
Responsible Professor: Arie van Deursen

1. Introduction

- Large Language Models (LLMs) require significant computational resources and memory for training and deployment.
- The CO2 emission of training GPT-3 model (175B parameters) amounts to three times that of a whole jet plane for San Francisco ↔ New York [1].
- Previous research only demonstrated compression of BERT models, up to 60% in size & 40% in runtime.
- Limited application of compression techniques on LLMs for the GPT model and the code generation task.

Compression techniques:

- **Knowledge distillation** - training a smaller student model from outputs of a larger teacher model.
- **Pruning** - dropping unnecessary connections and neurons.
- **Quantization** - lower the parameter precision (INT8 instead of FP32).

CodeGPT-small [2] - Microsoft's model, fine-tuned for one epoch on the code completion PY150 dataset [3].

2. Research Questions

1. How does the performance of the CodeGPT model change after applying *Group Lasso pruning*?

2. How does the performance of the **pruned** model change after applying *post training quantization*?

3. Methodology

Group lasso pruning - uses Group lasso regularization to prune entire rows, columns, or blocks of parameters that result in a smaller dense network.

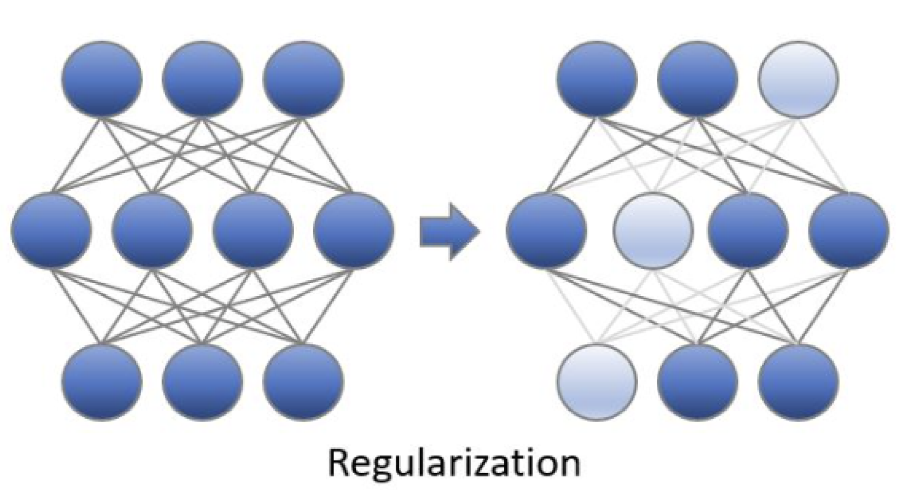

Fig. 1: Weight pruning visualized. The light blue circles denote that entire blocks (matrices) were pruned (set to 0). Source: [4]

Post training dynamic quantization - The weights of the neural network get quantized into int8 format from float32, where the clipping range is determined dynamically:

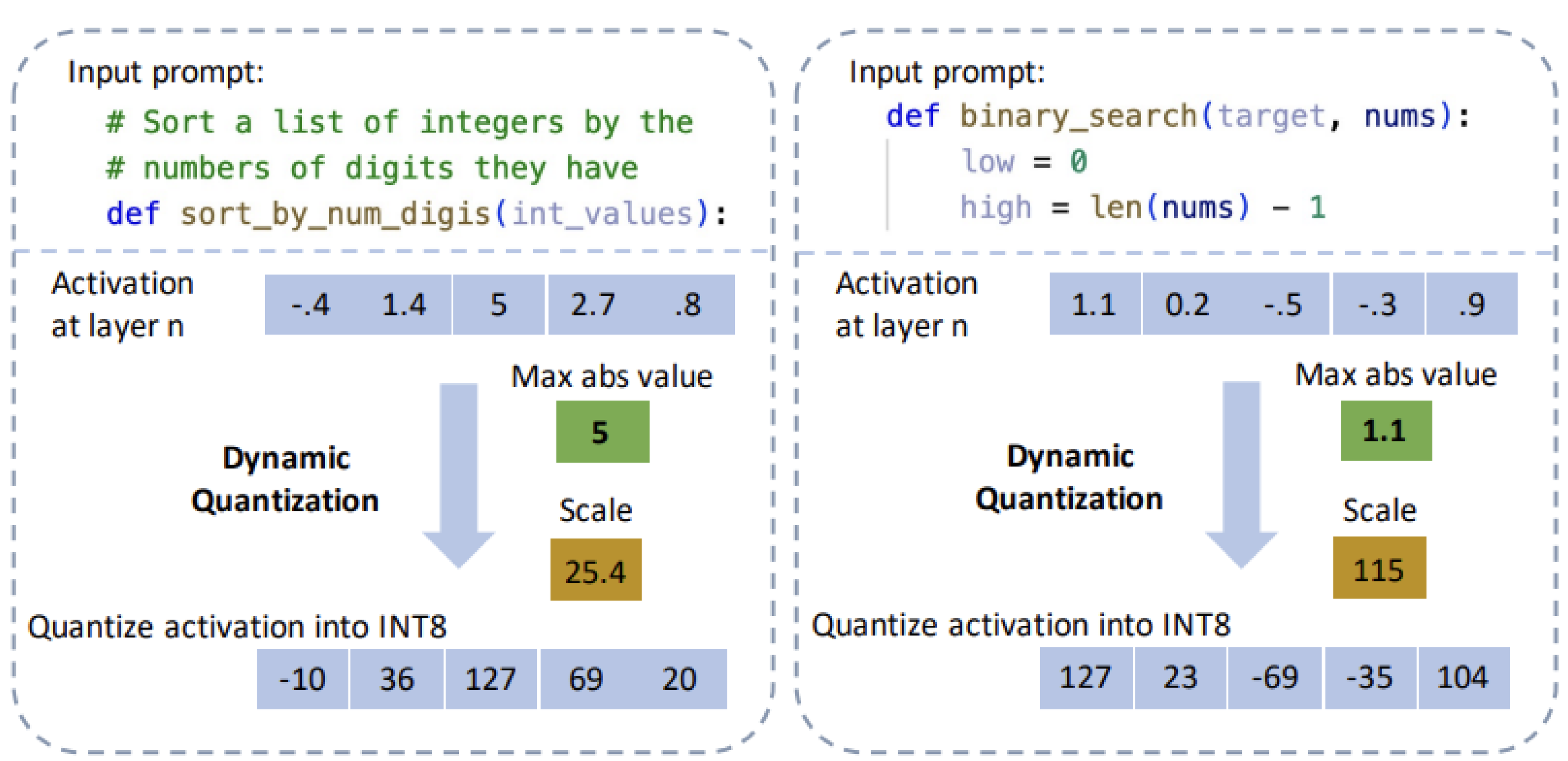

Fig. 2: PTQ visualized, where the weight clipping range is determined dinamically. Each input prompt has a different max abs value and therefore gets a different scale. Source: [5]

4. Results

Hardware: *Delft-Blue* HPC's Intel XEON E5-6248R 24C 3.0GHz CPUs (8 units, 4GB of memory). *Local setup* with Intel i9 11900H.
Compression library: intel-extension-for-transformers, introduced in [6]
Evaluation dataset: PY150, 1000 samples.
Evaluation metrics: Exact Match (**EM**) and Edit Similarity (**ES**).

Delft-Blue results

Model name	Disk size (MB)	Mem usage (MB)	CPU inf (samples/sec)	Edit sim (%)	EM (%)	Params (Mil.)
original	462.26	2976	2.66	39.05	14.5	124.2
pruned	**240.52**	3175	2.79	30.54	9	**49.77**
quantized	260.39	3293	2.68	20.76	0.5	49.77

Laptop results on pruning

Model name	Disk size (MB)	Mem usage (MB)	CPU inf (samples/sec)	Edit sim (%)	EM (%)	Params (Mil.)
original	462.26	2184.49	0.80	39.05	14.5	124.2
compressed	**240.52**	2189.63	0.80	30.54	9.0	**49.77**
onnx inference	311.75	4166.37	**1.59**	30.54	9.0	**49.77**

5. Discussion

- With *pruning at 60% sparsity* achieved **48% reduction in model size** with minimal drop in accuracy.
- **2x inference speedup** using ONNX runtime optimizations.
- Limited quantization results.

Comparable performance to other concurrent compression methods:

- **CodeGPT on XTC [6]** - highest reduction in disk size (15x) and the most impressive CPU and GPU inference results.
- **Distill-CodeGPT [7]** - low GPU model size, and considerable memory usage and inference speed improvements.
- **MP and PEG PTQ on CodeGPT [8]** - 4x reduction and highest scores in the accuracy metrics but lacking results for other measures.
- **Our solution** - 2x reduction in model size and lowest number of parameters.

Considerations:

- CodeGPT - a small model with limited accuracy.
- Variations/noise in memory usage and inference measurements.
- Limited library support for compressing GPT models & missing Inference Engine optimized for low-precision computations.
- Uncertain generalizability of results to a wider range of models.

6. Conclusion

- Group Lasso pruning at 60% sparsity achieved 48% reduction in model size with 8.5% absolute drop in ES and a 5.5% in EM.
- 2x inference speedup using ONNX runtime optimizations.
- Quantization did not provide any speedups.
- Enabled more efficient and eco-friendly use of GPT-based language models.

7. Future Work

- More complex models.
- Longer fine-tuning periods (>1 epoch).
- Using more mature compression libraries.
- More advanced compression techniques.

References

[1] Patterson, D. et al. Carbon emissions and large neural network training, 2021.
[2] Microsoft. microsoft/codegpt-small-py at main. https://huggingface.co/microsoft/CodeGPT-small-py/tree/main, 2023. Accessed: 2023-06-26.
[3] 0n1xus. 0n1xus/codexglue. https://huggingface.co/datasets/0n1xus/codexglue, 2023. Accessed: 2023-06-26.
[4] Intel(r) Neural Compressor. Pruning. Github, May 2023. https://github.com/intel/neural-compressor/blob/master/docs/source/pruning.md (Accessed on 16 May 2023).
[5] Wei, X. et al. Greener yet Powerful: Taming Large Code Generation Models with Quantization, March 2023. arXiv:2303.05378 [cs].
[6] Shen, H. et al. Fast DistilBERT on CPUs, December 2022. arXiv:2211.07715 [cs]
[7] de Moor, A. et al. CodeGPT on XTC: Compressing a CodeGPT Model Using Hybrid Layer Reduction and Extreme Quantisation through Knowledge Distillation. Delft University of Technology, 2023. http://resolver.tudelft.nl/uuid:f37924fc-ecac-4bd4-b923-7d4c73f74a72.
[8] Malmsten, E. et al. Distil-CodeGPT, Distilling Code-Generation Models for Local Use. Delft University of Technology, 2023. http://resolver.tudelft.nl/uuid:22217e2b-0db8-4c56-8808-9713dd678425.
[9] Storti, M. et al. Leveraging Efficient Transformer Quantization for CodeGPT: A Post-Training Analysis. Delft University of Technology, 2023. http://resolver.tudelft.nl/uuid:b1f0ef47-9c85-41ce-9b0f-fb092ba333db.