AARHUS UNIVERSITY
DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

Bridging the Gap

Integrating Linear Algebra in the Development and Understanding of Large Language Models for Software Engineering Applications.

by

Asger Poulsen

202106630

Bachelor's Thesis

in

Computer Engineering

Supervisor: Hugo Daniel Macedo

Aarhus University, Department of Electrical and Computer Engineering

8 June 2024

Preface

This bachelor's thesis was written for the Department of Electrical and Computer Engineering, Aarhus University. It is part of the Computer Engineering study program, and was written in the spring of 2024.
All source files associated with this thesis are found at: `https://github.com/asgersong/BSc`

Asger Poulsen, May 5, 2024

Abstract

Acknowledgements

Contents

Chapter 1

Introduction

Large Language Models (LLMs) have become a cornerstone in the field of natural language processing (NLP) and artificial intelligence (AI), driving significant advancements and innovations. These models are designed to understand, generate, and interpret human language at a level that is increasingly indistinguishable from that of a human being. The development and evolution of LLMs mark a pivotal shift in how machines can learn from and interact with textual data, enabling a plethora of applications ranging from automated text generation to sophisticated conversational agents.

Chapter 2

Literature Review

2.1 Overview of Large Language Models

The evolution of LLMs can be traced back to earlier models of machine learning that attempted to process and understand language. However, it was the introduction of models like Google's BERT (Bidirectional Encoder Representations from Transformers) and OpenAI's GPT (Generative Pre-trained Transformer) series that marked a significant leap in the capabilities of language models. Each iteration of these models has brought about improvements in understanding context, generating text, and general language comprehension, culminating in state-of-the-art models that are capable of writing essays, composing poetry, and even generating code.

- A Survey on Model Compression for Large Language Models
- The complete guide to LLM compression

2.2 Previous Studies on LLM Compression and Optimization

Over the past few years, the increasing demand for Large Language Models (LLMs) across a vast spectrum of applications, from natural language processing to content generation, has underscored the critical need for optimization strategies tailored to these complex systems. Researchers have delved into a variety of optimization techniques aimed at refining LLMs, with a keen focus on enhancing model efficiency and reducing the computational burden without compromising the specialized performance that these models are known for. Key areas of investigation have included model compression, prompt engineering, and other innovative approaches, all geared towards optimizing resource utilization and facilitating wider accessibility and deployment of LLMs in real-world scenarios. This section aims to shed light on seminal contributions to the optimization of LLMs, detailing the inventive methodologies adopted and the significant advancements realized in making these models more efficient and adaptable to the ever-evolving demands of technology and society.

2.2.1 Low-Memory Optimization

In addressing the optimization of LLMs with constrained resources, Lv et al., 2023 presents a pioneering approach named **LO**w-**M**emory **O**ptimization (LOMO). This method innovates by fusing gradient computation and parameter update into a single operation, significantly reducing the memory requirements for full model fine-tuning. Notably, LOMO enables the comprehensive fine-tuning of models with up to 65 billion parameters on hardware as accessible as a single setup equipped with 8 RTX 3090 GPUs.

A critical innovation within LOMO is the application of Stochastic Gradient Descent (SGD) as the optimizer, distinguished by its lack of reliance on intermediate state storage, thereby contributing to reduced memory consumption. Specifically, LOMO simplifies the traditional two-step process of gradient descent ($\text{grad} = \frac{\partial \mathcal{L}}{\partial p}$, $p = p - lr * \text{grad}$) into a single-step process ($p = p - lr * \frac{\partial \mathcal{L}}{\partial p}$). This optimization not only minimizes memory usage but also maintains, or in some cases, enhances the fine-tuning efficacy for downstream tasks.

The empirical evaluation of LOMO, particularly on the SuperGLUE benchmark, showcases its effectiveness in lowering memory requirements while preserving or improving model performance on various NLP tasks. The advancements brought forth by LOMO significantly democratize the process of fine-tuning LLMs, reducing the computational barrier and facilitating broader research and application possibilities in the field of Natural Language Processing.

2.2.2 Optimization by prompting

Optimization by PROmpting (OPRO), as introduced by Yang et al., 2023, represents a novel methodology for leveraging Large Language Models (LLMs) as optimizers. The core concept involves using the generative capabilities of LLMs to iteratively generate candidate solutions to optimization problems described in natural language. This process is guided by a meta-prompt, which provides the LLM with a concise description of the optimization task, historical performance data of past solutions, and a directive to generate new solutions aimed at optimizing the given objective function.

The OPRO framework's approach to generating solutions is iterative and dynamic, enabling a detailed exploration and exploitation of the solution space. Each round of solution generation takes into account the performance trajectory of previous solutions, allowing the LLM to refine its approach continuously and hone in on more promising solution regions. This method allows for a sophisticated balance between exploring new solution avenues and exploiting known high-potential areas, a balance that is crucial for the success of any optimization process.

Despite its innovative approach and promising results across various optimization tasks, the study acknowledges several limitations of the OPRO framework. The effectiveness of the optimization process is highly dependent on the quality and detail of the meta-prompt and the relevance and robustness of the initial solutions provided. Moreover, the computational demands of the iterative solution generation process, particularly for complex problems requiring numerous iterations for convergence, present significant challenges. These factors underscore the need for further research and development to refine the OPRO framework, focusing on improving prompt design, solution evaluation mechanisms, and computational efficiency to broaden its applicability and effectiveness in solving a wide range of optimization problems.

2.2.3 Low-Rank Adaptation (LoRA)

Low-Rank Adaptation (LoRA) introduces an efficient technique for adapting large pre-trained models like GPT-3 to specific tasks without the need for extensive retraining of all model parameters. This approach leverages the observation that despite the high parameter count in large neural models, their effective operational space often exhibits a significantly lower intrinsic dimensionality.

Theoretical Foundation At the core of LoRA is the adaptation of weight matrices $W \in \mathbb{R}^{d\times k}$ through the introduction of low-rank matrices $A \in \mathbb{R}^{r\times k}$ and $B \in \mathbb{R}^{d\times r}$, where $r \ll \min(d, k)$. This results in the adapted weight matrix W' being represented as:

$$W' = W + BA = W + \Delta W \tag{2.1}$$

Here, r denotes the rank and serves as a crucial parameter that balances the efficiency and expressiveness of the adaptation. This formulation ensures that the pre-trained weights (W) remain unchanged, preserving the foundational knowledge acquired during pre-training, while A and B encapsulate the task-specific adjustments.

Advantages LoRA's methodology brings forth several advantages:

- **Parameter Efficiency:** By optimizing the low-rank matrices A and B, LoRA significantly reduces the number of trainable parameters, leading to efficient storage and faster adaptation processes. This is due to r being much smaller than both d and k, so the total number of parameters in A and B combined $(r \times (d + k))$ is much less than the number of parameters in W $(d \times k)$.
- **Preservation of Pre-trained Knowledge:** The approach ensures that the valuable knowledge captured in the pre-trained model is retained, making it particularly beneficial for adapting computationally expensive models like GPT-3.
- **Flexibility and Scalability:** LoRA's adaptive process is highly scalable and flexible, making it possible to adapt large models to various tasks with minimal computational overhead.

Practical Implementation Implementing LoRA involves selecting the transformer layers most relevant to the target task and applying the low-rank adaptations. The process requires initializing A and B, choosing an appropriate rank r, and training these matrices while keeping the rest of the model parameters fixed. This strategy allows for the efficient adaptation of large-scale models to specific tasks, significantly reducing the need for computational resources compared to traditional full model retraining.

2.3 Evaluating Summarization with ROUGE

The evaluation of automated summarization models is crucial for assessing their efficiency and effectiveness in capturing the essence of text data. ROUGE, which stands for Recall-Oriented Understudy for Gisting Evaluation, provides a set of metrics that are indispensable for this purpose. Introduced by Lin and Hovy (2004), ROUGE measures compare the overlap between computer-generated summaries and a set of reference summaries typically created by humans.

ROUGE Metrics ROUGE includes several specific metrics, such as ROUGE-N (n-gram overlap), ROUGE-L (longest common subsequence), and ROUGE-S (skip-bigram co-occurrence statistics), each suited for different aspects of summarization. For instance, ROUGE-N focuses on the exactness of content at various n-gram levels, providing insights into the precision of the summarization model in replicating key information. In contrast, ROUGE-L assesses the fluency and structure of the generated summaries by measuring sequence similarity, which is crucial for evaluating the narrative flow of the text.

Application and Relevance The application of ROUGE in evaluation has been extensively validated across various tasks, including the Document Understanding Conferences (DUC), where it has been used to measure the performance of summarization systems in a competitive environment. These metrics have proven to be reliable indicators of human judgment, making them a standard against which the summarization capabilities of LLMs are benchmarked. The relevance of ROUGE scores lies in their ability to provide quantifiable measures that correlate strongly with human evaluations, thus facilitating the improvement and development of more efficient summarization models.

Significance in LLM Research In the context of Large Language Models, understanding the effectiveness of different compression and optimization techniques often relies on the ability to evaluate how well the reduced models can summarize content. ROUGE metrics serve this purpose by quantifying the trade-offs between model complexity and performance retention, helping researchers and developers optimize LLMs without significant loss in functionality.

Chapter 3

Theoretical Foundations

3.1 Linear Algebra in Deep Learning

Linear algebra forms the cornerstone of deep learning, providing the necessary mathematical framework to model and understand complex relationships within data. It is instrumental in defining the operations and transformations that occur within deep neural networks, including those underlying LLMs.

3.1.1 Vectors, Matrices, and Tensors

Vectors and matrices are fundamental to representing data and parameters in neural networks. A vector $\mathbf{v} \in \mathbb{R}^n$ can represent a point in n-dimensional space or a single data instance with n features. Matrices $A \in \mathbb{R}^{m \times n}$ facilitate linear transformations from $\mathbb{R}^n$ to $\mathbb{R}^m$, and tensors generalize these concepts to higher dimensions, accommodating the multi-dimensional data structures processed by neural networks.

A typical case, representing a basic neural network operation, can be expressed as:

$$\mathbf{y} = A\mathbf{x} + \mathbf{b} \tag{3.1}$$

where A is the weight matrix, $\mathbf{x}$ is the input vector, $\mathbf{b}$ is the bias vector, and $\mathbf{y}$ is the output vector of the transformation.

3.1.2 Matrix Operations

Matrix operations such as addition, multiplication, and transposition are essential in neural network computations. Matrix multiplication, in particular, plays a crucial role in transforming data between layers, capturing the relationships between input and output features. The dot product of two matrices $A \in \mathbb{R}^{m \times n}$ and $B \in \mathbb{R}^{n \times p}$ is defined as:

$$C = AB = \sum_{i=1}^{n} A_{ij} B_{jk} \tag{3.2}$$

where $C \in \mathbb{R}^{m \times p}$ is the resulting matrix. Matrix multiplication is a key operation in neural networks, enabling the transformation of input data through multiple layers of weights and biases.

Flops

Floating-point operations (FLOPs) are a measure of the computational complexity of matrix operations. The number of FLOPs required for matrix multiplication is proportional to the product of the dimensions of the matrices involved. For example, the number of FLOPs for multiplying two matrices $A \in \mathbb{R}^{m \times n}$ and $B \in \mathbb{R}^{n \times p}$ is $2mnp$.

3.1.3 Singular Value Decomposition

Singular Value Decomposition (SVD) is a powerful technique for decomposing a matrix into singular vectors and singular values, providing insight into the structure of the data. For any matrix $A \in \mathbb{R}^{m \times n}$, SVD is given by:

$$A = U \Sigma V^T \tag{3.3}$$

where U and V are orthogonal matrices containing the left and right singular vectors, respectively, and Σ is a diagonal matrix with singular values. SVD is essential in many machine learning tasks, including noise reduction, data compression, and the analysis of neural network layers.

3.1.4 Neural Networks in Deep Learning

a Neural network consist of interconnected nodes or "neurons" arranged in layers, with each layer designed to perform specific transformations on its inputs to capture and transmit increasingly abstract features to subsequent layers.

Architecture of Neural Networks

The architecture of a neural network is defined by layers, each comprising a set of neurons connected by weights. These weights are adjusted during the training process to minimize the difference between the actual output of the network and the desired output. A typical feedforward neural network can be mathematically represented as:

$$\mathbf{h}^{(l)} = f(W^{(l)}\mathbf{h}^{(l-1)} + \mathbf{b}^{(l)}) \tag{3.4}$$

where $W^{(l)}$ and $\mathbf{b}^{(l)}$ are the weight matrix and bias vector for the l-th layer, $\mathbf{h}^{(l-1)}$ is the output from the previous layer, and f is a non-linear activation function such as ReLU or sigmoid.

Learning Process

The learning process in neural networks involves adjusting weights and biases to reduce a loss function, commonly through backpropagation. This method efficiently computes gradients using chain rule calculus:

$$W^{(l)} \leftarrow W^{(l)} - \eta \frac{\partial \mathcal{L}}{\partial W^{(l)}} \tag{3.5}$$

where η is the learning rate and $\mathcal{L}$ is the loss function.

Optimization and Regularization

Optimization algorithms like Stochastic Gradient Descent (SGD), Adam, and RMSprop are crucial for weight updates. Regularization techniques such as dropout and weight decay help prevent overfitting, ensuring the network generalizes well to new data.

3.2 Understanding LLMs

Large Language Models (LLMs) such as GPT (Generative Pre-trained Transformer) and BERT (Bidirectional Encoder Representations from Transformers) have revolutionized the field of natural language processing (NLP) by leveraging deep neural networks to understand and generate human-like text unlocking a whole host of new applications.

3.2.1 What are LLMs?

LLMs are deep neural networks trained on vast amounts of text data. They learn to predict the next word in a sentence, understand context, generate text, and perform various NLP tasks with minimal task-specific adjustments. The strength of LLMs lies in their ability to capture intricate patterns in language through extensive pre-training.

3.2.2 Architecture of LLMs

The architecture of most LLMs is based on the Transformer model, introduced by Vaswani et al.Vaswani et al., 2017, which relies on self-attention mechanisms to weigh the significance of different words in a sentence. The Transformer architecture is composed of two main components: an encoder and a decoder. The encoder takes the input text and produces a sequence of hidden states, which represent the meaning of the text. The decoder then takes the encoder's hidden states and generates the output text, on word at a time.

Encoder

The encoder consists of a stack of N identical layers, each containing two sub-layers: a multi-head self-attention mechanism and a feed-forward neural network. Furtherly, a residual connection is employed around each of the two sub-layers, followed by layer normalization. So the output of each sub-layer is $\text{LayerNorm}(x + \text{SubLayer}(x))$, where $\text{SubLayer}(x)$ represents the function implemented by the sub-layer.

The self-attention mechanism allows the model to weigh the importance of different words in the input sequence when generating the output. The feed-forward neural network processes the output of the self-attention mechanism to produce the final hidden states of the encoder. A key feature of the encoder is that it processes all words in the input sequence in parallel, which contributes to the efficiency of the Transformer model. The output of the encoder is a sequence of vectors, each representing an input word in a high-dimensional space.

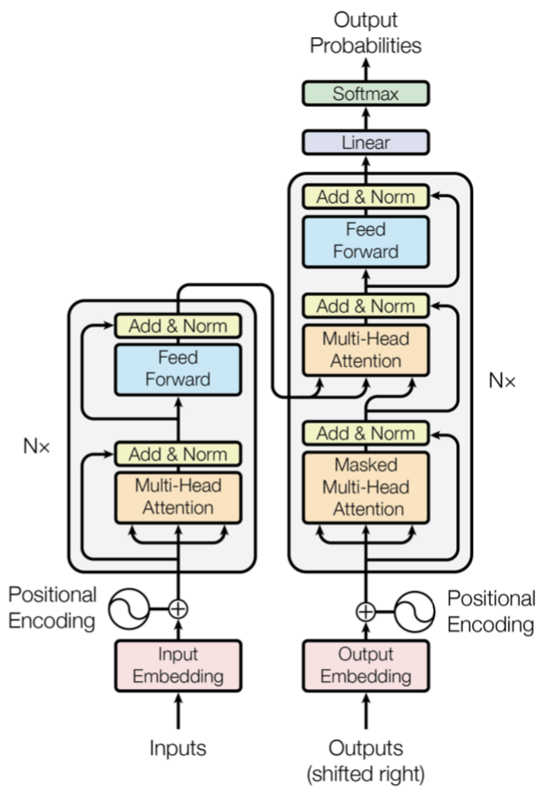

Figure 3.1: Transformer model architecture - Taken from “Attention Is All You Need“

Decoder

The decoder, on the other hand, also consists of a stack of N identical layers, but with an additional third sublayer in each decoder layer, which performs multi-head attention over the encoder’s output. This allows the decoder to focus on different parts of the encoder’s output for each word in the output sequence. In the first sublayer of the decoder, self-attention is used, but with a constraint (masking) to prevent positions from attending to subsequent positions. This ensures that the predictions for position i can depend only on the known outputs at positions less than i. The purpose of the decoder is to generate an output sequence one word at a time, using the encoder’s output and what it has produced so far as inputs.

Attention

The attention mechanism is a crucial component of the Transformer model, allowing it to weigh the importance of different words in the input sequence when generating the output. The attention mechanism is defined as follows:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{3.6}$$

where Q, K, and V represent the query, key, and value matrices, respectively, and d_k is the dimension of the key vector. This self-attention mechanism allows the model to focus on relevant parts of the input sequence when performing a task, enabling superior handling of long-range dependencies.

Multi-head attention

Multi-head attention is an extension of the attention mechanism that allows the model to focus on different parts of the input sequence simultaneously. It achieves this by, instead of performing a singel attention function with d_{model}-dimensional keys, values and queries, linearly projecting the query, key, and value matrices h times into multiple subspaces with dimensionality d_k, d_k, and d_v, respectively, before applying the attention function. The output of each of these h attention heads is then concatenated and linearly projected to produce the final output. This mechanism allows the model to capture different aspects of the input sequence in parallel, enhancing its ability to learn complex patterns in the data.

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \ldots, \text{head}_h)W^O \tag{3.7}$$

$$\textbf{where } \text{head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V) \tag{3.8}$$

Where $W_i^Q \in \mathbb{R}^{d_{\text{model}} \times d_k}$, $W_i^K \in \mathbb{R}^{d_{\text{model}} \times d_k}$, $W_i^V \in \mathbb{R}^{d_{\text{model}} \times d_v}$, and $W^O \in \mathbb{R}^{hd_v \times d_{\text{model}}}$ are learned linear projections, and d_{model} is the dimension of the model.

Why Transformers?

The Transformer model has several advantages over traditional RNNs and LSTMs, including the ability to capture long-range dependencies, parallelize computation, and scale to larger datasets. The self-attention mechanism allows the model to focus on relevant parts of the input sequence, enabling it to learn complex patterns in the data. The multi-head attention mechanism further enhances the model's ability to capture different aspects of the input sequence in parallel, improving its performance on a wide range of NLP tasks such as translation, summarization, and image captioning. This is why Transformers have become the architecture of choice for many state-of-the-art NLP models, including GPT and BERT.

3.2.3 Training and Fine-Tuning

The foundation of an LLM's understanding and generation of human language lies in its pre-training phase. During this stage, the model is exposed to a large corpus of text data, often encompassing a wide range of topics, genres, and styles. The primary objective of pre-training is to enable the model to learn a generalized representation of language.

The pre-training is typically conducted using unsupervised learning techniques, where the model is trained on tasks like Masked Language Modeling (MLM) or Next Sentence Prediction (NSP). In MLM, for example, a percentage of the input tokens are randomly masked, and the model's objective is to predict the original tokens at these masked positions. The MLM objective can be formally represented as:

$$\mathcal{L}_{\text{MLM}} = -\sum_{i \in \mathcal{M}} \log p(x_i | x_{\setminus \mathcal{M}}) \tag{3.9}$$

where $\mathcal{M}$ is the set of masked positions, x_i is the original token at position i, and $x_{\setminus \mathcal{M}}$ represents the input with masked tokens.

Following pre-training, LLMs undergo a fine-tuning phase, wherein the model is specialized to perform specific NLP tasks. This phase involves training the pre-trained model on a smaller, task-specific dataset, allowing the model to adjust its weights to better perform the target task.

The fine-tuning process can be represented as a continuation of the training process, optimizing the following objective:

$$\mathcal{L}_{\text{fine-tune}} = - \sum_{(x,y)\in\mathcal{D}_{\text{task}}} \log p(y|x;\theta_{\text{pre-train}} + \Delta\theta) \tag{3.10}$$

where $\mathcal{D}_{\text{task}}$ is the task-specific dataset, (x, y) are the input-output pairs, $\theta_{\text{pre-train}}$ are the parameters learned during pre-training, and $\Delta\theta$ represents the parameter updates during fine-tuning.

Fine-tuning LLMs presents challenges such as catastrophic forgetting and overfitting, particularly when the task-specific dataset is small. Various strategies, including careful learning rate selection, regularization techniques, and the use of adapters, are employed to mitigate these issues.

Chapter 4

Methodology

4.1 Educational Synergy Development

4.2 Software Engineering Application

4.3 LLM Optimization

Optimizing Large Language Models (LLMs) for efficiency and performance without compromising their effectiveness is a critical area of research in the field of artificial intelligence. Various techniques have been developed to address this challenge, each employing unique strategies to reduce computational resources, decrease model size, and maintain, if not improve, the model's performance. This section explores the general methodologies applied in the optimization of LLMs, focusing particularly on model compression techniques. Among these, Low-Rank Adaptation (LoRA) stands out as a significant innovation, offering a balance between model efficiency and task performance.

4.3.1 Optimization Techniques for Large Language Models

Optimization strategies for Large Language Models (LLMs) are crucial for improving computational efficiency and model efficacy. These strategies are generally divided into two primary types: model pruning and parameter sharing.

Model Pruning

Model pruning aims to reduce the model's complexity and size effectively without substantial loss in performance. It involves systematically eliminating parameters or connections within the model that are least consequential to the output, thereby enhancing operational efficiency and making the model more adaptable for use in environments with limited resources.

Parameter Sharing

Conversely, parameter sharing utilizes the model's existing parameters across various parts of the model or different tasks. This method optimizes the use of the model's capacity, enabling multifunctional performance without an increase in parameter count.

Focus on Low-Rank Approximation

This thesis will specifically focus on model pruning techniques, with a particular emphasis on low-rank approximation. This approach approximates large matrices or tensors with ones of a lower rank, reducing the number of parameters and computational demands while preserving the model's critical information processing capabilities.

4.3.2 Metrics for Evaluation

Evaluating the effectiveness of optimization techniques like LoRA involves several key metrics:

Model Size Reduction: A primary metric is the reduction in the total number of parameters, indicating the efficiency of the compression technique.

Inference Speed: The impact on the model's inference latency is critical, especially for applications requiring real-time responses.

Cosine Similarity: In the context of model tuning and adaptation, cosine similarity can serve as a measure of how well the optimized model maintains the semantic properties of the original. It is especially useful in assessing the quality of embeddings and the alignment between the compressed and original model outputs.

Performance Retention: The maintenance of task-specific performance, measured through metrics such as accuracy, F1 score, or ROUGE scores for language tasks, is essential to ensure the utility of the compressed model.

Computational Efficiency: The reduction in computational resources required for training and inference reflects the practical benefits of the optimization technique.

These metrics provide a comprehensive framework for assessing the trade-offs involved in LLM optimization, guiding the development and application of techniques like LoRA in enhancing the practical utility of state-of-the-art language models.

4.4 Case Study: Low-Rank Approximation

To illustrate the application of low-rank approximation in compressing LLMs, we consider Facebook's the BART-Base model as a case study. BART is a transformer-based LLM that has been widely used for various natural language processing tasks. By applying low-rank approximation to the attention weight matrices of the BART-base model, we aim to reduce its parameter size while maintaining its performance on a summarization task.

The choice to apply low-rank approximation specifically to the attention matrices stems from their pivotal role in the transformer architecture. These matrices, which help the model assess the relevance of different words within the input data, tend to be large and often encapsulate redundant information (Aghajanyan, Zettlemoyer, and Gupta, 2020). By reducing the dimensionality of these matrices, we preserve the model's ability to perform complex relational reasoning with less computational overhead, thus maintaining efficacy in summarization tasks while enhancing efficiency.

4.4.1 Low-Rank Approximation Technique

Low-rank approximation is a matrix factorization technique that decomposes a given matrix into the product of some lower-dimensional matrices. In the context of LLMs, low-rank approximation can be applied to the attention weight matrices to reduce the number of parameters while preserving the model's representational capacity.

4.4.2 Implementation Steps

The implementation of low-rank approximation for BART involves the following steps:

Singular Value Decomposition (SVD): The attention weight matrices (Key, Query, Value, and Output) of the model are decomposed using SVD to obtain the low-rank approximation.

Custom Layer Implementation: A custom layer is implemented to replace the original attention layers with the low-rank approximation, reducing the model's parameter size.

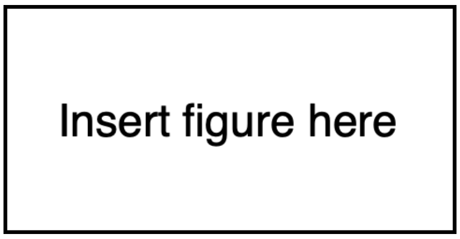

Figure 4.1: Attention layers replaced with low-rank approximation

Evaluation Metrics: The compressed model is evaluated based on ROUGE scores (Lin, 2004), cosine similarity, metrics such as model size reduction, computational efficiency, cosine similarity, and task-specific performance on a summarization task using the Samsum dataset (Gliwa et al., 2019).

Rank Selection: The rank of the approximation is chosen based on the observed spectrum decay of the ROUGE-scores, ensuring a balance between speed and task performance retention.

Chapter 5

Implementation

5.1 Curriculum Component Implementation

5.2 Chatbot Development

5.2.1 Tools and Libraries Used

5.2.2 Integration of LLM

Chapter 6

Evaluation and Results

6.1 Curriculum Effectiveness

6.2 Chatbot Performance and Usefulness

6.3 Compression and Optimization of LLM

6.3.1 Methodology Applied

6.3.2 Results and Analysis

The evaluation of the compressing the BART-Base model using low-rank approximation reveals the following insights:

Model Size Reduction: The low-rank approximation technique achieves a significant reduction in the total number of parameters, demonstrating its effectiveness in compressing the model.

Computational Efficiency: The compressing the model shows that the `eval_runtime` is almost linear with the rank r, indicating the potential for computational savings.

Cosine Similarity:

ROUGE Scores: The compressed model maintains competitive ROUGE scores compared to the baseline BART-Base model, indicating that the low-rank approximation does not significantly impact the model's summarization performance.

Chapter 7

Discussion

7.1 Interpretation of Results

7.2 Theoretical and Practical Implications

7.3 Limitations and Challenges

Chapter 8

Conclusion and Future Work

8.1 Summary of Key Findings

8.2 Contributions to the Field

8.3 Recommendations for Future Research

Bibliography

McCarthy, John (1960). "Recursive Functions of Symbolic Expressions and Their Computation by Machine, Part I". In: *Communications of the ACM.*

Hu, Edward J. et al. (2021). *LoRA: Low-Rank Adaptation of Large Language Models.* arXiv: `2106.09685 [cs.CL]`.

Vaswani, Ashish et al. (2017). *Attention Is All You Need.* arXiv: `1706.03762 [cs.CL]`.

Lv, Kai et al. (2023). *Full Parameter Fine-tuning for Large Language Models with Limited Resources.* arXiv: `2306.09782 [cs.CL]`.

Yang, Chengrun et al. (2023). *Large Language Models as Optimizers.* arXiv: `2309.03409 [cs.LG]`.

Zhu, Xunyu et al. (2023). *A Survey on Model Compression for Large Language Models.* arXiv: `2308.07633 [cs.CL]`.

Gliwa, Bogdan et al. (Nov. 2019). "SAMSum Corpus: A Human-annotated Dialogue Dataset for Abstractive Summarization". In: *Proceedings of the 2nd Workshop on New Frontiers in Summarization.* Hong Kong, China: Association for Computational Linguistics, pp. 70–79. DOI: `10.18653/v1/D19-5409`. URL: `https://www.aclweb.org/anthology/D19-5409`.

Lin, Chin-Yew (July 2004). "ROUGE: A Package for Automatic Evaluation of Summaries". In: *Text Summarization Branches Out.* Barcelona, Spain: Association for Computational Linguistics, pp. 74–81. URL: `https://aclanthology.org/W04-1013`.

Aghajanyan, Armen, Luke Zettlemoyer, and Sonal Gupta (2020). *Intrinsic Dimensionality Explains the Effectiveness of Language Model Fine-Tuning.* arXiv: `2012.13255 [cs.LG]`.