Skoltech

Skolkovo Institute of Science and Technology

Computationally Efficient Natural Language Processing Methods using Tensor Representations

Doctoral Thesis
by

Viktoriia Chekalina

Doctoral Program is Computational and Data Science and Engineering

Supervisor
Associate Professor, Alexander Panchenko

Moscow — 2023

I hereby declare that the work presented in this thesis was carried out by myself at Skolkovo Institute of Science and Technology, Moscow, except where due acknowledgement is made, and has not been submitted for any other degree.

Candidate (Viktoriia Chekalina)
Supervisor (Prof. Alexander Panchenko)

Abstract

Nowadays, achieving high quality in Natural Language Processing (NLP) tasks often requires processing large amounts of data, such as WikiData with 12 billion objects, or utilizing massive models, such as the GPT-2, GPT-3, GPT-3.5 models from the Transformer family. However, the use of these models requires significant computing resources, resulting in substantial energy and financial costs.

This thesis focusses on analysing computational algorithms and models for various NLP tasks. More specifically, in this thesis, we explore the use of multilinear algebraic representations for various NLP tasks: Link Prediction, Language Modelling, Natural Language Understanding, and Text Summarization. Namely, we use tensor representation in various ways: first, by presenting input data in Knowledge Graph in the form of tensor decomposition, and second, by presenting layers of neural models in the form of tensor decomposition products.

Efficient implementations with low-level optimisation of the proposed techniques are developed. Proposed optimizations lead to significant memory savings for a Link Prediction task with a slight quality improvement. Furthermore, an implementation of tensor representation-based layer structures is offered. These structures can be integrated into the computational graph of a Large Language Model (LLM). These kinds of models are validated in a large variety of tasks from the Natural Language Understanding and Language Modelling groups. A deep study of the comparative Question Answering problem is also provided and the efficient version of Large Language Model is validated on it. Experiments demonstrate that models with tensor-based layers outperform others when training architectures from scratch. In experiments with compressing of pre-trained weights, tensor-based approaches exhibit superior performance with larger compression rates, as opposed to smaller ones.

Publications

Main author

1. **Chekalina, V.**, Novikov G., Gusak G., Oseledets I., Panchenko A. (2023): Efficient GPT Model Pre-training using Tensor Train Matrix Representation.

   Proceedings of the 37th Pacific Asia Conference on Language, Information and Computation (PACLIC 37).

   Submitted to conference proceedings: CORE B

2. **Chekalina, V.**, Pletenev, S., Moskovskiy D., Seleznev M., Panchenko, A. (2023): A Computational Study of Matrix Decomposition Methods for Compression of Pre-trained Transformers.

   In Proceedings of the 11-th International Conference on Analysis of Images, Social Networks, and Texts (AIST-2023). Springer Lecture Notes in Computer Science (LNCS). Yerevan, Armenia

   Conference proceedings: Scopus, Web of Science

3. **Chekalina, V.**, Razzhigaev, A., Sayapin, A., Frolov, E., and Panchenko, A. (2022): MEKER: Memory Efficient Knowledge Embedding Representation for Link Prediction and Question Answering.

   In Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics: Student Research Workshop, pages 355–365, Dublin, Ireland. Association for Computational Linguistics.

   https://10.18653/v1/2022.acl-srw.27

   Conference Proceedings: CORE A*

4. **Chekalina, V.**, Bondarenko, A., Biemann, C., Beloucif, M., Logacheva, V., Panchenko, A. (2021): Which is better for Deep Learning: Python or MATLAB? Answering Comparative Questions in Natural Language.

   In Proceedings of the 2021 Conference of the European Chapter of the Association for Computational Linguistics – System Demonstrations. Kyiv, Ukraine

(Online).

https://aclanthology.org/2021.eacl-demos.36/

Conference proceedings: CORE A

5. **Chekalina, V.** and Panchenko, A. (2022): Retrieving Comparative Arguments using Deep Language Models.

   In proceedings of the Working Notes of CLEF 2022 - Conference and Labs of the Evaluation Forum. Bologna, Italy. CEUR-WS Vol-3180.

   https://ceur-ws.org/Vol-3180/paper-255.pdf

   Workshop proceedings: Scopus.

6. **Chekalina, V.** and Panchenko (2021): Retrieving Comparative Arguments using Ensemble Methods and Neural Information Retrieval.

   In proceedings of the Working Notes of CLEF 2021 – Conference and Labs of the Evaluation Forum. Bucharest, Romania. CEUR-WS Vol-2936.

   http://ceur-ws.org/Vol-2936/paper-211.pdf

   Workshop proceedings: Scopus.

7. **Chekalina, V.** and Panchenko, A. (2020): Retrieving Comparative Arguments using Deep Pre-trained Language Models and NLU.

   In proceedings of the working Notes of CLEF 2020 – Conference and Labs of the Evaluation Forum. Thessaloniki, Greece. CEUR-WS

   http://ceur-ws.org/Vol-2696/paper_210.pdf

   Workshop proceedings: Scopus.

Co-Author

1. Nikishina, I., Tikhonova M., **Chekalina, V.**, Zaytsev, A, Panchenko, A. (2023): Industry vs Academia: Running a Course on Transformers in Two Setups.

Proceedings of the Konferenz zur Verarbeitung natürlicher Sprache/Conference on Natural Language Processing (KONVENS 2023).

Dedicated to my parents, Olga Chekalina and Alexander Chekalin,
as well as my grandparents, Valentina Orlova and Oleg Orlov

Acknowledgments

I am enormously grateful to my supervisor Professor Dr. Alexander Panchenko for trusting me and giving me complete freedom everywhere, especially in choosing research topics and projects that helped to achieve them. Under his supervision I have gained a wealth of experience in life wisdom.

My special thanks are to Professor Dr. Ivan Oseledets, who supported me with my ideas (even if they looked fanciful), for immutable positive attitude. My sincere thanks are to Dr. Salman Ahmadi-Asl for the meticulous attention to detail endless patience in checking mathematical formulas.

I am deeply thankful to Dr. Gulia Gusak, previously a postdoc in Skoltech, who provided me with a role model of a person in the Postdoc position through her composure and trustworthiness.

I also express my gratitude to Georgiy Novikov, Anna Rudenko and Albert Sayapin, who are shared my research and inspired me to continue it.

Then, I must acknowledge to brave souls who proofread the text of this thesis: Mikhail Diskin, Dr. Salman Ahmadi-Asl, Dr. Evgeny Frolov, Dr. Irina Nikishina, Daniil Moskovskiy, Vadim Liventsev, Sergey Pletenev, Mikhail Salnikov, Nikolay Babakov and Aleksei Kalinichenko.

I should send special gratitude to my friends — Maria Sheyanova, Vadim Liventsev, Sergei Kozlukov, Lyubov Kartavtseva, Anastasia Belozerova, Mikhail Diskin, Daniil Udimov — who steered me from diving too deep into abstract research and provided as an alternative many attractive adventures in real life.

Contents

List of Figures

List of Tables

Glossary

CPD Canonical Polyadic Decomposition. 12, 20, 21, 26, 39, 41, 42

FC Fully-Connected. 21, 22, 80, 117

KG Knowledge Graphs. 20, 34, 39, 40

ML Machine Learning. 35

NLP Natural Language Processing. 18, 39, 40

NN Neural Network. 20

SVD Singular Value Decomposition. 21, 24, 80, 113, 115, 118, 120

TT Tensor Train. 29, 31, 72

TTM Tensor Train Matrix. 20, 21, 31, 59, 72, 80, 81, 115, 118, 120

Chapter 1

Introduction

1.1 Motivation

In recent years, there have been significant advances in Natural Language Processing (NLP) methods, leading to higher quality solutions. These advances can be attributed to two interconnected factors. First, there has been a remarkable improvement in the hardware capabilities for processing and storing information. This has enabled the storage of vast amounts of data for training and processing, resulting in more efficient and faster operations. For example, the DBpedia knowledge dataset [Auer et al., 2007] now contains 400 million facts, while the data set for the Bloom [Scao et al., 2022] model encompasses 350 billion tokens. Additionally, models like Bloom or Megatron [Shoeybi et al., 2019] offer distributed training in multiple graphic cards or clusters.

Second, the practicality of these advanced techniques has significantly enhanced language methods. One key development was the incorporation of the attention mechanism [Vaswani et al., 2017a] into Language Modelling. This technique enables the identification of the most relevant parts within large textual data. By applying the attention mechanism, models can effectively analyse semantic, grammatical, and other implicit patterns in language, even in texts comprising several thousand tokens.

This progress has resulted in a remarkable performance boost of NLP systems. Presently, transformer models, trained on a large corpus of texts, can be employed in

nearly any task with minimal modifications. Moreover, they can work in experiments with the few-shot setup - with a few prompts and zero-shot - without any prompting.

The study [Kaplan et al., 2020] presents an empirical scaling law that demonstrates how the performance of language models is influenced by their size and the size of the training data. While the law discussed in that study specifically relates to cross-entropy, works such as [Radford et al., 2019b], [Radford et al., 2019a], [Brown et al., 2020], and [Shoeybi et al., 2019] indicate that other quality metrics also improve with larger model sizes and training datasets. However, the rapid growth of these models has revealed redundancies in both the full modules within NLP architectures [Michel et al., 2019] and the parameters within different fully-connected layers [Denil et al., 2013]. Increasing the size of models presents several challenges. The computational cost associated with such models necessitates significant capital investment to reproduce them. While it is possible to interact with models via APIs, full access to large architectures and in-depth study generally belongs only to industrial companies. Consequently, research laboratories often face resource constraints that make working with state-of-the-art (SOTA) models and relevant datasets impractical or excessively time-consuming.

Another concern is related to the fact that when training and operating large models, the issue of energy consumption and the accompanying CO2 emission becomes significant. For models comparable to GPT-3, the training carbon footprint is several hundred tons [Rae et al., 2021], [Zhang et al., 2022]. The values of these parameters depend largely on the training settings and server architecture, the type of cards, and even the location of data centers [Patterson et al., 2021]. They are proportional to the number of FLOPS (Floating Point Operations) needed either to train the model or to forward the signal in inference. In this case, the effective reduction of language models due to low-ranking representations of their parts, described in Chapters 6.4 and 5, ceteris paribus, leads to a decrease in energy consumption and CO2 emissions.

To address these issues, our work focusses on utilising limited resources more efficiently. We achieve this by employing tensor representations [Cichocki et al., 2016] to obtain compressed versions of either the dataset itself, as explored in Chapter 3, or

the model architectures (as discussed in Chapters 6.4 and 5). Tensor representations represent data in factor matrices or in a set of interconnected low-order core tensors. In other words, we take tensors - multidimensional generalisations of matrices - and represent structured datasets or part of Neural Network models in this form. We use Canonical Polyadic Decomposition (CPD) to represent large dataset based on Knowledge Graphs (KG) and the Tensor Train Matrix (TTM) form to represent the weights of the Transformers layers. We then apply a decomposition technique appropriate for the selected tensor structure, to obtain a compressed representation of it. The compressed representation has fewer parameters with the same level of stored information, reducing the resources required for employment data or model.

It should be noted that the request for vast resources in training and employing processes often occurs due to the specifics of the training tools and framework settings utilised. Namely, the default algorithms of Pytorch, the most popular framework for building computation graphs, can require more memory than is necessary for the gradient computation task. The proposed work, in particular, shows optimisation options in related situations. For example, Chapter 4 describes the optimisation of signal backpropagation in structures, a sequence of multidimensional objects; Chapter 3 offers an example of a mixed type of gradient calculation for Neural Network (NN) architecture. In this type, the computational graph is created for tiny layers. On the contrary, for an embedding layer with considerable size, we count gradient analytically, which reduces the overall memory consumed by the model.

1.2 Thesis Objectives

The purpose of the presented thesis is to study and apply the methods of low-rank tensor approximations to language processing problems of various types. In more detail, this work investigates the following research questions:

1. Can we make the Knowledge Graphs (KG) embedding computation more efficient using tensor structure?

2. Can we reduce the size of the Transformer-based language model by replacing

some layers with Tensor Train Matrix (TTM) structures and how does it affect the model performance?

3. How can we improve the quality of the compressed model? More precisely, can we select layers more compatible with compression and can we integrate information about the downstream model task into the compression algorithm?

1.3 Contribution, Novelty and Impact

This study highlights that certain language models and techniques may require excessive resources for certain tasks, emphasising the importance of efficient compression. In our compression methods, we employ low-rank techniques and tensor decomposition approaches. We harness Canonical Polyadic Decomposition (CPD) to represent large datasets based on Knowledge graph data. We use the Tensor Train Matrix (TTM) decomposition to compress Neural Network layers and finally acquire a smaller version of Transformer-based language models. We achieve a smaller model in two ways: by training the complete architecture from scratch and by compressing the pretrained model. For the Transformers, we performed experiments on the encoder (BERT and BERT-based architectures) and the decoder (GPT-2).

1. The slight modifications to the CPD decomposition technique are made, which improves the efficiency of representing Knowledge Base data.

2. An implementation of TTM-based Neural Network (NN) layers is provided, which made them compatible for fine-tuning and training from scratch. These layers have been incorporated into the Transformer model, replacing the traditional Fully-Connected (FC) Layer with Tensor Train Matrix (TTM)-based and Singular Value Decomposition (SVD)-based layers.

3. A compressed version of GPT-2 is developed using TTM and SVD layers; its performance is evaluated on both language modelling and classification tasks.

4. The most suitable modules for TTM and SVD compression are identified in the Transformer model.

5. A comprehensive study of abstractive and extractive approaches to question-answering in a comparative case is conducted. For this task, FC layers in the corresponding Transformer model are replaced with its TTM and SVD compressed versions.

6. SVD-based and TTM-based layers for the BERT model are adjusted in a language understanding task.

1.4 Thesis Outline

This thesis has the following structure, which is outlined below.

In Chapter 1, "Introduction", a brief overview of the primary objectives of the proposed work is provided, as well as the potential impact of the research on the scientific community.

Chapter 2 "Background" introduces the notation and various tensor decomposition methods, focussing on the low-rank and full-rank representation formats used in efficient NLP methods or relied upon by comparison baselines.

Chapter 3 "Efficient Knowledge Embedding Using Canonical Polyadic decomposition" details how we used tensor methods to obtain a compact representation of large datasets that describe real-world facts in the form of a 3D structure. To achieve this, we employ a distribution-aware version of CP decomposition, compressing the data into embeddings.

In Chapter 4 "SVD and TTM Representations of Fully-Connected (FC) Layers", we discuss the implementation details of representing Fully-Connected (FC) layers in the TTM format. We focus on creating a Pytorch compatible layer based on a sequence of TTM cores for time and memory-efficient fine-tuning and training operations. We further use the proposed structures to replace FC layers in different Transformer architectures. This choice of harnessing FC layers is motivated by the fact that these layers are often the most significant bottleneck in Transformers.

We use these results in Chapter 5 "Efficient GPT Model using TTM Decomposition", where we replace layers in the Transformer architecture GPT-2 with a TTM object, training this kind of model from scratch, validating its performance

inside and outside the domain of the Language Modelling task, as well as on some downstream tasks.

Chapter 6, titled "Efficient Question Answering using TTM decomposition", explores the task of answering comparative questions. The chapter provides an overview of generative and abstractive methods for solving this problem, with a focus on the latter - finding relevant answers in the corpus. One promising solution is ColBERT, a model based on the BERT architecture that encodes queries and texts simultaneously in the search space. However, as with any transformer-based model, it is computationally intensive and heavy. In this chapter, we employ TTM decomposition to compress the fully-connected layers within ColBERT. We search for optimal TTM ranks that offer the best approximation. Additionally, we conduct a comparative analysis of models employing various compression techniques to accomplish QA tasks.

We will also investigate whether all modules in the Transformer architecture exhibit the same compression behaviour. In particular, we are investigating whether choosing certain layers for compression results in a smaller performance hit. Specifically, we investigate whether the selection of certain layers for compression results in a less severe degradation in performance.

Chapter 7 "Transformer-based Encoders compression using TTM decomposition" delves deeper into this topic, exploring the compression of BERT, a transformer-based architecture used for Language Modelling. In addition, we investigate the fine-tuning options available to improve the compressed model's performance.

Chapter 2

Background

In this chapter, we will discuss the matrix and tensor representations that we have utilised in our work to enhance the efficiency of our linguistic models in terms of speed and memory requirements. We will only introduce the representations that were employed in our experiments, to ensure that this manuscript is self-contained for the reader's convenience.

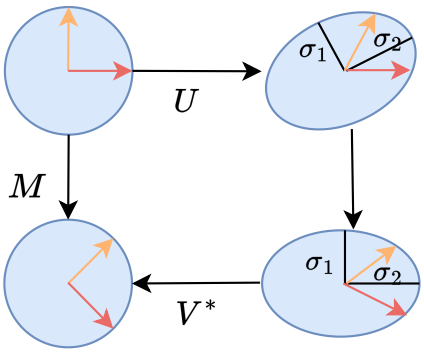
To enhance convenience, Table 2.1 incorporates notation that corresponds to algebraic objects mentioned in the text and the operations performed on them.

It is worth mentioning that there are numerous other methods for decomposition, such as Tensor Networks [Biamonte and Bergholm, 2017] and TensorRing [Zhao et al., 2016], along with a wide variety of algorithms [Kolda and Bader, 2009b], which we do not cover in this chapter. A complete overview of methods is provided in [Ji et al., 2019].

2.1 Matrix Decompositions

2.1.1 Singular Value Decomposition

The Singular Value Decomposition (SVD) of a complex matrix $\mathbf{M} \in \mathbb{R}^{m \times n}$ is a factorization of the form $\mathbf{M} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^*$ where $\mathbf{U} \in \mathbb{R}^{m \times m}$ complex unitary matrix, $\mathbf{\Sigma} \in \mathbb{R}^{m \times n}$ is a rectangular diagonal matrix with non-negative real numbers on the diagonal, $\mathbf{V} \in \mathbb{R}^{n \times n}$ is a complex unitary matrix, $\mathbf{V}^\star$ is the conjugate transpose of



$$M = U\Sigma V^*$$

Figure 2-1: The interpretation of SVD. The operator of transformation $\mathbf{M}$ is decomposed into rotation $\mathbf{U}$, scaling $\mathbf{\Sigma}$ and rotation back $\mathbf{V^T}$.

$\mathbf{V}$. If $\mathbf{M}$ is real, $\mathbf{V}^\star$ is equivalent to $\mathbf{V^T}$.

Matrix $\tilde{\mathbf{M}}$ with a rank r is said to be **a truncated approximation** of the matrix $\mathbf{M}$ if $\tilde{\mathbf{M}} = \mathbf{U}\tilde{\Sigma}\mathbf{V}^*$, where $\tilde{\mathbf{\Sigma}}$ is the same matrix as $\mathbf{\Sigma}$ except that it contains only the r largest singular values (the other singular values are replaced by zero).

As depicted in Fig. 2-1, three-factor matrices in the Singular Value Decomposition can be treated as rotation, scaling and rotation back operations. In other words, the object $\mathbf{M}$ is translated into a space where it can be represented as a sum of basis functions with weights equal to singular values. In this space, we can drop the part with minor weights (truncate it) and then transfer the object $\mathbf{M}$ back to the regular world.

2.2 Tensor Decompositions

Tensor decomposition is a powerful technique for representing multidimensional objects by breaking them down into elementary operations on smaller objects. This approach is particularly useful for addressing the curse of dimensionality, where the number of elements in an Nth-order tensor grows exponentially with the order. The sheer volume of data in high-dimensional objects can pose significant computational and memory challenges.

To tackle this issue, reducing the size and dimensions of the original object can

Table 2.1: Tensor symbols and notations.

Notation/Symbol	Meaning
$\mathcal{X} \in \mathbb{R}^{I_1 \times I_2 \times \ldots I_N}$	N^{th} order tensor of size $I_1 \times I_2 \times \ldots I_N$
$x_{i_1,i_2,\ldots,i_n} = \mathcal{X}(i_1, i_2, \ldots, i_n)$	$(i_1, i_2, \ldots i_N)^{th}$ entry of the tensor $\mathcal{X}$
$\underline{\mathcal{G}}, \mathcal{G}^{(k)}, \mathcal{G}^k$	Core tensors in Tucker and Tensor Train decompositions
Λ	Diagonal core tensor in Canonical Polyadic Decomposition
$\mathbf{A}, \mathbf{b}$	Matrix and vector
$\odot$, $\otimes$, $\circ$	Khartri-Rao, Kronecker and outer Products
$\mathcal{X} \times_n \mathbf{A}$	Mode-n product of $\mathcal{X} \in \mathbb{R}^{I_1 \times I_2 \times \ldots I_N}$ and matrix $\mathbf{A} \in \mathbb{R}^{J \times I_n}$
$\mathbf{A}^T$, $\mathbf{A}^{-1}$, $\mathbf{A}^\dagger$	Transpose, inverse, and Moore-Penrose pseudo-inverse of a matrix $\mathbf{A}$
$\mathcal{X}_{[\mathrm{n}]}$	Mode-n matricization (unfolding). This operation represents a tensor $\mathcal{X} \in \mathbb{R}^{I_1 \times \cdots \times I_n \times \cdots \times I_N}$ in a matrix $\mathcal{X}_{[\mathrm{n}]} \in \mathbb{R}^{I_n \times I_1 \cdots I_N}$

lead to more efficient and faster applications. There are various options available to represent a tensor as a set of smaller-dimensional objects, each with its own limitations. The choice of decomposition method depends on the specific problem statement and the types of data being used.

2.2.1 Canonical Polyadic Decomposition

One of the most popular tensor approximations is the Canonical Polyadic Decomposition (CPD) [Hitchcock, 1927], an extension of the factorisation of the low-rank matrix to multidimensional objects. This decomposition introduses an N-order tensor $\mathcal{X} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$ as a sum of $R < N$ tensors of rank 1, also known as Kruskal tensors:

$$\mathcal{X} \approx \sum_{r=1}^{R} \lambda_r b_r^{(1)} \circ b_r^{(2)} \cdots \circ b_r^{(n)} = \Lambda \times_1 \mathbf{B}^{(1)} \times_2 \mathbf{B}^{(2)} \cdots \times_N \mathbf{B}^{(N)} \tag{2.1}$$

Here, $N >= 2$ is an integer that defines the number of dimentions in the initial tensor, $B^{(n)} = [b_1^{(n)}, b_2^{(n)}, b_R^{(n)}] \in \mathbb{R}^{I_n \times R}$ is one of the N factor matrices, Λ is a diagonal core tensor with elements $\lambda_1, \ldots, \lambda_N$.

In other words, an N-dimensional object is represented as R scalar weights and N two-dimensional matrices. These matrices are called factor matrices; they are

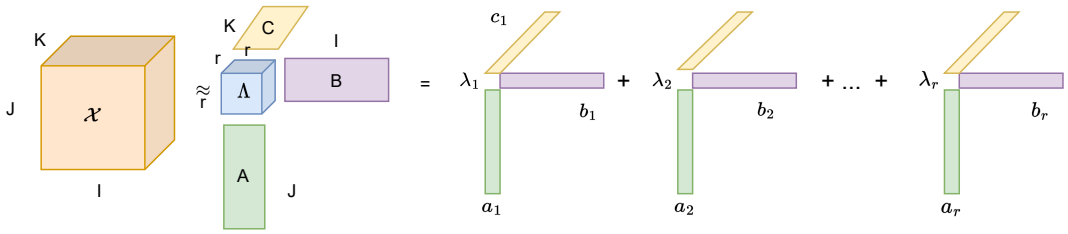


Figure 2-2: The CP decomposition scheme.

usually orthogonal and represent the principal component of each mode. Since the resulting decomposition objects are two-dimensional, CPD is good at addressing the curse of dimensionality. Such a representation gives good compression for tensors of many dimensions. Unfortunately, the widespread use of this decomposition is restricted by the fact that, for a number of problems, it does not give good convergence. The scheme of CPD decomposition in the 3-d tensor case is shown in Figure 2-2.

$$\begin{aligned}
\mathbf{A} &\leftarrow \underset{\mathbf{A}}{\operatorname{argmin}} ||\mathcal{X}_{[0]} - (\mathbf{B} \odot \mathbf{C})\mathbf{A}^T||_2 \\
\mathbf{B} &\leftarrow \underset{\mathbf{B}}{\operatorname{argmin}} ||\mathcal{X}_{[1]} - (\mathbf{A} \odot \mathbf{C})\mathbf{B}^T||_2 \\
\mathbf{C} &\leftarrow \underset{\mathbf{C}}{\operatorname{argmin}} ||\mathcal{X}_{[2]} - (\mathbf{A} \odot \mathbf{B})\mathbf{C}^T||_2
\end{aligned} \tag{2.2}$$

A popular approach for computing the CP decomposition of a tensor is the CP Alternating Least Squares (ALS) algorithm [Battaglino et al., 2018, Dunlavy et al., 2011]. This approach aims to minimise the norm $L2$ between the appropriate combination of factor matrices and the corresponding unfolding [Kolda and Bader, 2009b] of the initial tensor (see 2.2). Algorithm 1 presents this method to the 3-rd order case. The main idea is to fix all factor matrices except for one to optimise the non-fixed matrix. This step is then repeated for each matrix until a specific stopping criterion is reached.

In the proposed dissertation, we use Canonical Decomposition in the one of the Natural Language Processing task with some tips to improve convergence.

Algorithm 1 CP-ALS [Kolda and Bader, 2009a]

Input: : Tensor $\mathcal{X} \in \mathbb{R}^{I \times J \times K}$ ▷ Ground Truth Tensor
stopping criterion

Output: : $\mathbf{A}$, $\mathbf{B}$, $\mathbf{C}$ ▷ Factor matrices
Initialise factor matrices $\mathbf{A} \in \mathbb{R}^{I \times R}$, $\mathbf{B} \in \mathbb{R}^{J \times R}$, $\mathbf{C} \in \mathbb{R}^{K \times R}$

repeat
  $\mathbf{A} = \mathcal{X}_{[0]}[(\mathbf{B} \odot \mathbf{C})^T]^\dagger$
  $\mathbf{B} = \mathcal{X}_{[1]}[(\mathbf{A} \odot \mathbf{C})^T]^\dagger$
  $\mathbf{C} = \mathcal{X}_{[2]}[(\mathbf{A} \odot \mathbf{B})^T]^\dagger$
  for $i = 1 \dots N$ **do**
    normalise columns of $\mathcal{X}_{[i]}$
    define λ_i as the norm
  end for
until stopping criterion is met.

2.2.2 Tucker Decomposition

Another tensor decomposition, called the Tucker decomposition proposed by [Tucker, 1966c] — is an extension of CPD. Since the CPD considers only outer products between factor elements with the same indexes, the Tucker representation adds products of vectors with all possible indexes to the approximation sum. In Figures 2-2,2-3, it is seen that the Tucker decomposition has a dense core tensor $\underline{\mathcal{G}}$, while the CPD assumes only a diagonal part of it. This difference allows Tucker to more accurately approximate the object due to the more extensive set of possible elements in sum. However, the Tucker representation has a core tensor with the same dimension as the original compressing object. If the desired rank needs to be larger, it can raise the curse of the dimensionality problem.

$$\mathcal{X} \approx \sum_{r_1=1}^{R_1} \cdots \sum_{r_N=1}^{R_N} g_{r_1, r_2 .. r_N} b_{r_1}^{(1)} \circ b_{r_2}^{(2)} \cdots \circ b_{r_n}^{(n)} = \underline{\mathcal{G}} \times_1 \mathbf{B}^{(1)} \times_2 \mathbf{B}^{(2)} \cdots \times_N \mathbf{B}^{(N)} \quad (2.3)$$

Here, $N >= 2$ is an integer that defines the number of dimentions in the initial tensor, $\mathbf{B}^{(n)}, n = \overline{1, N}$ is one of the N factor matrices, $\underline{\mathcal{G}}$ is a non-diagonal core tensor.

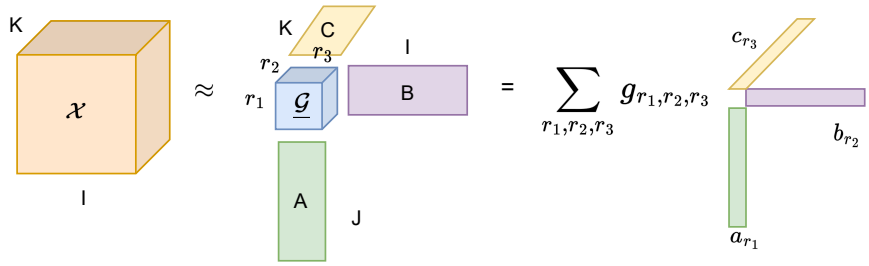


Figure 2-3: The scheme of Tucker decomposition in a case of 3-d tensor. The source of the image: Cichocki et al. [2016].

2.2.3 Tensor Train Format

Tensor Train (TT) decomposition [Oseledets, 2011a] represents a N order object as a sequence of third-order tensors $G_1, \ldots, G_N$, adjacent to each other along one of the axes 2-4. The TT format generalises the principle of low-rank approximation to objects of higher dimensions. A tensor $\mathcal{X} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$ is represented in the Tensor Train (TT) format if each element of $\mathcal{X}$ can be computed as:

$$\mathcal{X}(i_1, \ldots, i_N) \approx \sum_{\alpha_1=1}^{R_1} \cdots \sum_{\alpha_{N-1}=1}^{R_{N-1}} \mathcal{G}^{(1)}(\alpha_0, i_1, \alpha_1)\mathcal{G}^{(2)}(\alpha_1, i_2, \alpha_2) \ldots \mathcal{G}^{(N)}(\alpha_{N-1}, i_N, \alpha_N) \tag{2.4}$$

Here $\mathcal{G}^{(k)} \in \mathbb{R}^{R_k \times I_k \times R_{k+1}}$, $k = \overline{1, N}$, are 3-dimentional *core tensors (cores)* of TT decomposition, $N >= 2$ is a number of dimentions in the initial tensor $\mathcal{X}$. The $R_1, \ldots, R_k$ are called *TT-ranks* of the decomposition, given that $R_1 = R_k = 1$, $R = \max_k(R_1, \ldots, R_k)$. The index α_k goes from 1 to the corresponding rank R_k and is related to the corresponding dimension of the kernel, α_0 and α_N are always equal to 1.

In particular, the element $\mathcal{X}(i_1, i_2 \ldots i_N)$ is effectively the product of 2 vectors and $N - 2$ matrices as depicted in Fig. 2-4:

$$\mathcal{X}(i_1 \ldots i_N) = \underbrace{\mathcal{G}^{(1)}[i_1, :]}_{1 \times R_1} \underbrace{\mathcal{G}^{(2)}[:, i_2, :]}_{R_1 \times R_2} \ldots \underbrace{\mathcal{G}^{(N-1)}[:, i_{N-1}, :]}_{R_{N-2} \times R_{N-1}} \underbrace{\mathcal{G}^{(N)}[:, i_N]}_{R_{N-1} \times 1}, \tag{2.5}$$

, where R_k - the $k - th$ rank of the TT decomposition, $\mathcal{G}^{(k)}[:, i_l, :]$ - a slice of k-th core tensor along the second axis, when all elements of the tensor are considered, whose second index is equal to i_l.

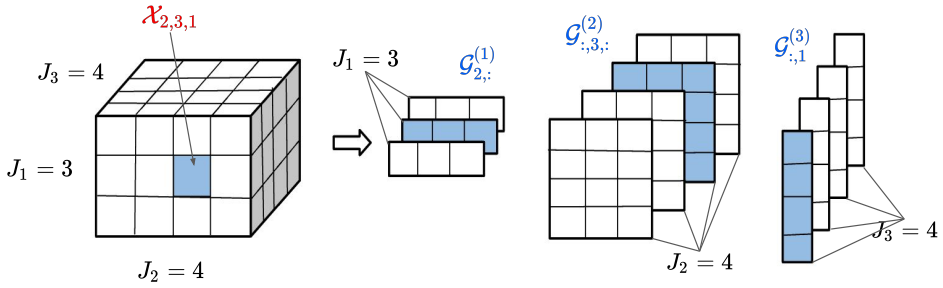


Figure 2-4: The example of computation tensor elements via it's TT representation. Source of the image: [Xu et al., 2022].

Regarding the enquiry about the existence of a TT-decomposition, the following statement remains valid [Oseledets, 2011b]:

Theorem 1. *If for each unfolding $\mathcal{X}_{[k]}$ of a d-dimensional tensor $\mathcal{X}$, where $k = 1, \ldots, d$, rank $\mathcal{X}_{[k]} = R_k$, then there exists a decomposition of the form 2.4 with TT-ranks not higher than R_k.*

In other words, this statement, first, means the possibility of obtaining a TT decomposition with ranks 1 for any data. Second, it determines the specific values of the cores' ranks.

In the proposed expansion, the cores store $R(I_1 + I_M) + R^2 \sum_{k=2}^{N-1} I_k$ parameters, while the original tensor contains $\prod_{m=1}^{M} I_m$. This can be cost-effective at low ranks but not so beneficial at high ranks due to quadratic complexity.

The most common algorithms for obtaining the TT decomposition are called **TT-SVD** [Oseledets, 2011b] and are generally based on the operation of the gaining k-th core. In this operation, we take an unfolding of a d- shaped tensor $\mathcal{C}$ along the dimension k and apply truncated SVD to it, where the truncation rank r is r_k, the k-th rank of the desired TT decomposition. The matrix $\mathbf{U}$ forms the k core and multiplication $\mathbf{\Sigma V^T}$ acts as object A for further step. The entire procedure is described in Algorithm 2.

Algorithm 2 TT-SVD [Oseledets, 2011b]

Input: d-dimentional tensor $\mathcal{A}$;

Output: Cores $\mathcal{G}^1, \ldots, \mathcal{G}^d$ of TT decomposition of $\mathcal{A}$

Temporary tensor $\mathcal{C} = \mathcal{A}, r_0 = 1$

for $k = 1$ to $d - 1$ **do**

$\mathbf{C} :=$ reshape($\mathcal{C}$, $[r_{k-1}n_k, \frac{\text{prod(C.shape)}}{r_{k-1}n_k}]$

r_k-truncated SVD: $\mathbf{C} \approx \mathbf{U}\mathbf{\Sigma}_{\mathbf{r_k}}\mathbf{V}^{\mathbf{T}}$

$\mathcal{G}^k :=$ reshape($\mathbf{U}$, $[r_{k-1}, n_k, r_k]$)

$\mathbf{C} := \Sigma_{r_k} V^T$

end for

$\mathcal{G}^k = \mathbf{C}$

Thus, the TT format potentially captures all the advantages of the previous methods:

- We can represent every tensor in a TT form, since SVD always exists for complex matrices.
- It can have a large compression ratio.
- It doesn't suffer from the course of the dimensionality since the maximum dimension of cores is 3.

The disadvantages of this method include the fact that, in practice, it is rarely possible to obtain a decomposition with minor ranks, and the memory occupied by the TT representation grows quadratically with R.

2.2.4 Tensor Train Matrix Format

The method is based on the representation of a matrix as a d-dimensional tensor and applying the TT-decomposition to given object is called Tensor Train Matrix (TTM) format. Tha main idea of this representation comes from the fact presented in [Loan and Pitsianis, 1992]. This work shows that for matrix the Kronecker minimization problem $min||\mathbf{A} - \mathbf{B} \otimes \mathbf{C}||_F$ may be transferred to a 1-rank approximation problem

$min||\tilde{\mathbf{A}} - vec(\mathbf{B})vec(\mathbf{C})^T||_F$ by rearranging blocks of A by vectorization operation vec:

$$\mathbf{X} \in \mathbb{R}^{p\times q} \rightarrow vec(\mathbf{X}) = \begin{bmatrix} X(1:p,1) \\ X(1:p,2) \\ \vdots \\ X(1:p,q) \end{bmatrix} \in \mathbb{R}^{pq} \tag{2.6}$$

The *vec* opearation can be used to express $min||\mathbf{A} - \mathbf{B} \otimes \mathbf{C}||_F$ the to rank-1 approximation problem with the amount of disperancy stayed remain. More precisely,

$$\left\| \left[\begin{array}{c|c} \mathbf{A_{1,1}} & \mathbf{A_{2,1}} \\ \hline \mathbf{A_{1,2}} & \mathbf{A_{2,2}} \end{array}\right] - \mathbf{B} \otimes \mathbf{C} \right\|_F = \left\| \left[\begin{array}{cc|cc} a_{1,1} & a_{2,1} & a_{3,1} & a_{4,1} \\ a_{1,2} & a_{2,2} & a_{3,2} & a_{4,2} \\ \hline a_{1,3} & a_{2,3} & a_{3,3} & a_{4,3} \\ a_{1,4} & a_{2,4} & a_{3,4} & a_{4,4} \end{array}\right] - \begin{bmatrix} b_{1,1} & b_{2,1} \\ b_{1,2} & b_{2,2} \end{bmatrix} \otimes \begin{bmatrix} c_{1,1} & c_{2,1} \\ c_{1,2} & c_{2,2} \end{bmatrix} \right\|_F \tag{2.7}$$

is equivalent to

$$\left\| \begin{bmatrix} vec(\mathbf{A_{11}})^T \\ vec(\mathbf{A_{21}})^T \\ vec(\mathbf{A_{31}})^T \\ vec(\mathbf{A_{41}})^T \end{bmatrix} - vec(\mathbf{B})vec(\mathbf{C})^T \right\|_F = = \left\| \left[\begin{array}{cc|cc} a_{1,1} & a_{2,1} & a_{1,2} & a_{2,2} \\ a_{3,1} & a_{4,1} & a_{3,2} & a_{4,2} \\ \hline a_{1,3} & a_{2,3} & a_{1,4} & a_{2,4} \\ a_{3,3} & a_{4,3} & a_{3,4} & a_{4,4} \end{array}\right] - \begin{bmatrix} b_{11} \\ b_{21} \\ b_{12} \\ b_{22} \end{bmatrix} [b_{11}, b_{21}, b_{12}, b_{22}] \right\|_F , \tag{2.8}$$

where $||\ ||_F$ is Frobenius norm and the right parts of equations obviously represent a product and give an object of rank 1.

This fact can also be interpreted as representing a 2-dimensional Kronecker matrix $\mathbf{A}$ as a 4-dimensional array $\mathcal{A}$. The axes in this array have the following meanings: row of an element in a matrix block, column of an element in a block, row of a block in a "big" block matrix, column of a block in this matrix. Vectorization operation changes the order element indexing, and thus, changes the order of these axes:

$$\tilde{\mathcal{A}} = permute(\mathcal{A}, [1, 3, 2, 4]) \tag{2.9}$$

Oseledets [2010] generalises it to the d-dimensional case - in fact, a block matrix was created d times from the structure obtained in the previous step. At the first step, this structure was a two-dimensional 2×2 matrix; at the second step, it was a 4×4 tensor with 4 dimensions. The resulting object $\mathcal{B}$ has dimensions $2d \times 2d$ and d axes that determine the position of the element in superstructured blocks. Vectorisation of elements inside the "initial" block matrix is equivalent to the following permutation:

$$\mathcal{C} = permute(\mathcal{B}, [1, \frac{d}{2} + 1, 2, \frac{d}{2} + 2, \ldots, \frac{d}{2}, d]) \tag{2.10}$$

It is interesting to note that the compression ranks depend on the ordering of dimensions; that is, the tensor can be "compressible" for one permutation of dimensions and not compressible for another. The rank r_k of decomposition is determined as the rank of the k-unfolding matrix $A_k = \mathcal{A}(i_1, i_2, \ldots, i_k, i_{k+1}, \ldots, i_d)$ as described in Section 2.2.3.

Having $2d$-dimention tensor $\mathcal{C}$, where the adjacent dimensions are pairwise linked (they determine the position of the pseudo-element in the matrix of level $k \in 1, , \ldots d$). For element $c(i_1, j_1, \ldots i_d, j_d)$ pair (i_k, j_k) defines the multiindex, and $\mathcal{C}$ has efficient dimentionality d and can be decomposed into TT format (see Section 2.2.3)

with d cores:

$$\mathcal{X}(i_1, j_1, \ldots, i_d, j_d) \approx \sum_{\alpha_1=1}^{R_1} \cdots \sum_{\alpha_{d-1}=1}^{R_{d-1}} G^{(1)}(\alpha_0, i_1, j_1, \alpha_1) G^{(2)}(\alpha_1, i_2, j_2, \alpha_2) \ldots G^{(N)}(\alpha_{d-1}, i_d, j_d, \alpha_d). \tag{2.11}$$

Note that the cores turn out to be 4-dimensional instead of 3-dimensional and are indexed as $G^{(k)}(\alpha_{k-1}, i_k, j_k, \alpha_k)$.

2.3 Knowledge Graphs

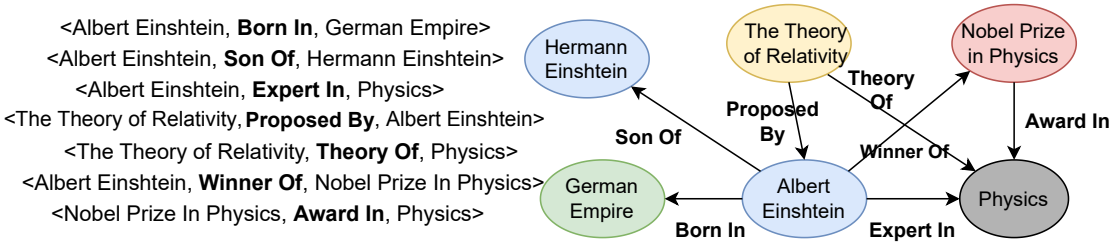


Figure 2-5: Examples of triples in a Knowledge Graph. Source of the image: Ji et al. [2020].

A Knowledge Graphs (KG) is a well-organised framework to represent factual information, including entities, relationships, and semantic descriptions. Entities refer to both tangible objects and abstract concepts, while relationships depict the connections between entities. Furthermore, entities and their relationships are characterised by types and properties that possess clearly defined meanings. Although the terms "knowledge graph" and "knowledge base" are essentially interchangeable, there exists a subtle distinction between them. The knowledge graph involves a structural representation of information, while a knowledge base is a technology used to store complex structured and unstructured information. Each unit of knowledge graph can be succinctly expressed as a factual triple in the form of (subject, relation, object). As an example, we can consider (Albert Einstein, WinnerOf, Nobel Prize), as depicted in Figure 2-5.

Numerous open knowledge bases and ontologies have been published, such as **WordNet** [Miller, 1992], **DBpedia** [Auer et al., 2007], **YAGO** [Suchanek et al.,

2007], and **Freebase** [Bollacker et al., 2008b]. Additionally, several related datasets specifically designed for Machine Learning (ML) tasks are available, including WordNet, DBpedia, YAGO, and Freebase.

The information contained in knowledge graphs can be very useful for NLP applications. A knowledge graph embedding (KGE) is a representation of KG elements in a continuous vector space. The objective of learning this representation is to simplify the manipulation of graph elements (entities, relations) for downstream tasks, such as question-answering and logical reasoning problems.

Knowledge-aware models gain significant advantages by incorporating diverse sources of information and comprehensive semantics; common sense knowledge allows us to get more complete representations of language elements.

2.4 Transformer-based Language Models

In a general sense, modelling something means predicting probability of some action in a given context. Language model (LM) estimates the probability of a given linguistic unit - which is called *token* and can be word, parts of word or char symbol - relative to surrounding or preceding text. LM generates a probability distribution of the appearance of each possible token in the current context. To estimate the language model, the *perplexity* metric is usually used. Perplexity is related to cross-entropy and is defined as follows:

$$ppl = e^{-\frac{1}{n}\sum_{i=1}^{n}\log p(y_i|context)} \tag{2.12}$$

,where n - number of token in text, $p(y_i|context)$ - probability of the given token under its context.

There are various types of language models, including statistical-based models, recurrent neural network models, and models that incorporate attention mechanisms.

Figures presented below 2-7, 2-8 illustrate the attention mechanism of Equation 2.13. In this representation, the text is treated as a sequence of tokens, with

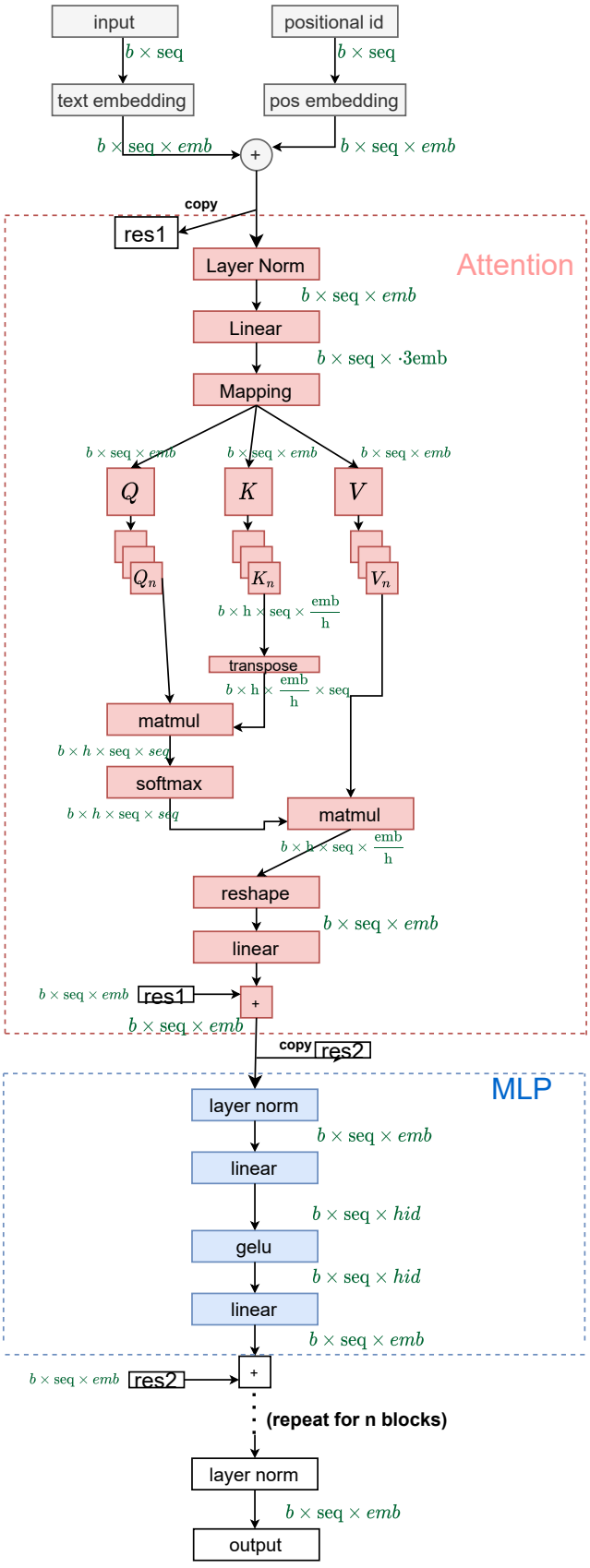


Figure 2-6: The scheme of the GPT-2 model.

each token being transformed into a vector representation. These vectors are then projected onto the space of queries (**Q**), keys (**K**), and values (**V**) through trainable matrices. A similarity function *sim* is applied to estimate the proximity of the query of the current token with the keys of all other elements. The similarity scores define weights that determine the importance of each part when combined. The resulting sum is then concatenated with the values of the current token.

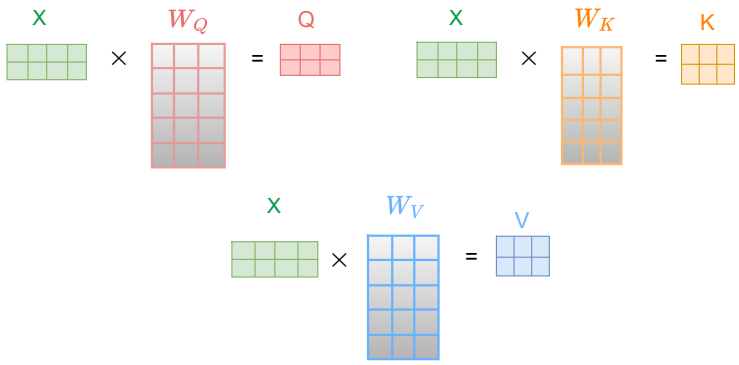


Figure 2-7: Visualization of Query, Key and Value claculation inside the transformer block.

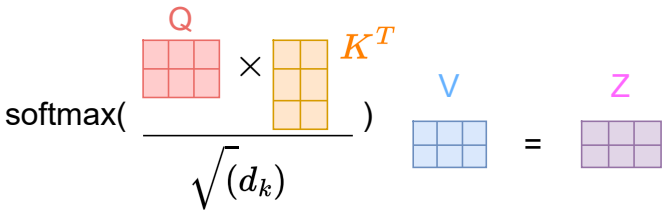


Figure 2-8: Visualization of attention calculation inside the transformer block.

$$
\begin{aligned}
attn(\mathbf{Q}, \mathbf{K}, \mathbf{V}) &= softmax(\frac{\mathbf{Q}, \mathbf{K^T}}{\sqrt{(d)}})\mathbf{V}, \\
attn(Q, K_i, V_i) &= \frac{\sum_{j=1}^{p} sim(Q_i, K_j)V_j}{\sum_{j=1}^{p} sim(Q_i, K_j)},
\end{aligned}
\tag{2.13}
$$

Transformer models possess an architecture composed of distinct modules, which is the reason behind their name "Transformers". Each module typically comprises an Attention layer, followed by a pair of consecutive Fully Connected (FC) layers. These layers first reduce the size of the input vector and then restore them to their

original dimensions with nonlinearities incorporated between them. The GPT-2 decoder transformer scheme with the corresponding sizes of the layer weights and input activations is shown in Figure 2-6.

The variances in different model architectures often lie in the number of modules they possess. For example, the BERT [Devlin et al., 2019] model consists of 16 blocks, while the GPT-2 [Radford et al., 2019a] model has 12, 24, or 36 blocks, depending on the specific variant (small, medium, or large). These models also differ in their training procedures. Encoder-based models such as BERT aim to preserve language information and are trained to understand context by masked language modelling and predicting subsequent sentences. Enhanced versions of the BERT model, such as RoBERTa [Liu et al., 2019] and others, feature improved data and learning processes. On the other hand, decoder-based models (GPT) are utilised for sequence generation tasks and trained using language modelling objectives, specifically generating the next token.

Chapter 3

Efficient Knowledge Embedding using Canonical Polyadic Decomposition

This Chapter is based on the paper "MEKER: Memory Efficient Knowledge Embedding Representation for Link Prediction and Question Answering" (cf. Section "Publications" at page 4).

In this chapter, we present MEKER: Memory Efficient Knowledge Embedding Representation model for Knowledge Graphs (KG) embeddings. Knowledge graphs are symbolically structured storages of facts. The KG embedding contains concise data used in NLP tasks that require implicit information about the real world. Furthermore, the size of KG that may be useful in real Natural Language Processing (NLP) applications is enormous, and the creation of embedding over it has memory cost issues. We represent a KG as a 3rd-order binary tensor and move beyond the standard Canonical Polyadic Decomposition (CPD) [Hitchcock, 1927] by using a generalised version specific to the data [Hong et al., 2020]. The generalisation of the standard CP-ALS algorithm allows obtaining optimization gradients without a Pytorch computational graph. It reduces the memory needed for training while providing computational benefits. MEKER is a memory-efficient KG embedding model that yields SOTA-comparable performance on link prediction tasks and KG-based question answering.

3.1 Introduction

NLP models have made great strides in recent years. For instance, language models can easily generate fluent human-like text. However, some applications like question answering and recommendation systems need correct and trustworthy answers. Large language models like GPT-3 [Brown et al., 2020] or ChatGPT [1] are trained to maximize human preferences in a text and often hallucinate in situations that need trustworthy answers [Yang et al., 2023]. The presence of this limitation renders the models unsuitable for implementation in the mentioned systems.

To avoid hallucinations, it is appropriate to leverage a KG [Bollacker et al., 2008a, Rebele et al., 2016] as a structured repository of essential facts about the real world. For convenience, the knowledge graph can be represented as a set of triples. A triple is created by two entities, subject e_s and object e_o — and a relation r between them. As a whole, the triple describes the fact.

For efficient use of information from KG, there is a need for low-dimensional embedding of graph entities and relations. KG embedding models usually use a standard Neural Networks (NN) backward mechanism for parameter tuning, duplicating its memory consumption. Hence, existing approaches to embedding learning have substantial memory requirements and can be deployed only on small datasets under a single GPU card. Processing large KGs appropriate for the custom downstream task is a challenge.

There are several libraries designed to solve this problem. Framework LibKGE [Ruffinelli et al., 2020] allows the processing of large datasets by using sparse embedding layers. Despite the memory saving, sparse embedding has several limitations, for example, in the PyTorch library, they are not compatible with several optimizers. PyTorch-BigGraph [Lerer et al., 2019] operates with large knowledge graphs by dividing them into partitions - distributed subgraphs. Subgraphs need a place for storing, embedding models need modifications to work with partitions and perform poorly.

One of contributions of thesis is a memory-efficient approach to learning Knowl-

[1] https://openai.com/blog/chatgpt

edge Graph embeddings MEKER (Memory Efficient Knowledge Embedding Representation). It allows more efficient KG embedding learning, maintaining performance comparable to state-of-the-art models. MEKER leverages generalized CPD [Hong et al., 2020], which allows for a better approximation of given data and analytical computation of the parameter gradient. MEKER is evaluated on a link prediction task using several standard datasets and large datasets based on Wikidata. Experiments show that MEKER achieves highly competitive results on these two tasks. To demonstrate downstream usability, we create a Knowledge Base Question Answering system Text2Graph and use embeddings in it. The system with MEKER embeddings performs better as compared to other KG embeddings, such as PTBG [Lerer et al., 2019].

3.2 Related Work

There are three types of approaches for learning KG embedding: distance-based, tensor-based, and deep learning-based models. The first group is based on the assumption of translation invariance in the embedding vector space. In model **TransE** [Bordes et al., 2013] relations are represented as connection vectors between entity representations. **TransH** [Wang et al., 2014] implies a relation as a hyperplane onto which entities are projected. **QuatE** [Zhang et al., 2019] extends the idea with hypercomplex space and represents entities as embeddings with four imaginary components and relations as rotations in the space.

Tensor-based models usually represent triples as a binary tensor and look for embedding matrices as factorization products. **RESCAL** [Nickel et al., 2011] employs tensor factorization as DEDICOM [Harshman et al., 1982], which decomposes each tensor slice along the relationship axis. **DistMult** [Yang et al., 2015] adapts this approach by restricting the matrix of relational embedding to the diagonal. On the one hand, it reduces the number of relation parameters, on the other hand, it losses the possibility of describing asymmetric relations. The **ComplEX** [Trouillon et al., 2016] represents the object and subject variants of a single entity as complex conjugate vectors. It combines tensor-based and translation-based approaches and

solves the asymmetric problem. **TuckER** [Balazevic et al., 2019] uses Tucker decomposition [Tucker, 1966c] to find representation of elements of knowledge graphs. This work can also be considered a generalization of several previous link prediction methods.

Standart CPD [Hitchcock, 1927] decomposition in the link prediction task does not show outstanding performance [Trouillon et al., 2017]. Several papers address this problem by improving the CPD approach. **SimplIE** [Kazemi and Poole, 2018] states that low performance is due to different representations of subject and object entity and deploys CPD with dependent learning of subjects and objects matrices. **CP-N3** [Lacroix et al., 2018] highlights the statement that the Frobenius norm regularizing is not fit for tensors of order more than 3 [Cheng et al., 2016] and proposes a Nuclear p-norm instead of it. Our approach also uses CPD with enhancement. We consider remark from **SimplIE** and set the object and subject representations of one entity to be equals. At the same time, within the local step of the CPD algorithm, the matrices of subjects and objects consist of different elements and are different (see subsection 3.4.1). In contradiction to **CP-N3**, we do not employ a regularizer to improve training, but change the objective. Instead of squared error, we use logistic loss, which is appropriate for one-hot data. We abandon the gradient calculation through the computational graph and count the gradient analytically, which makes the training process less resource-demanding.

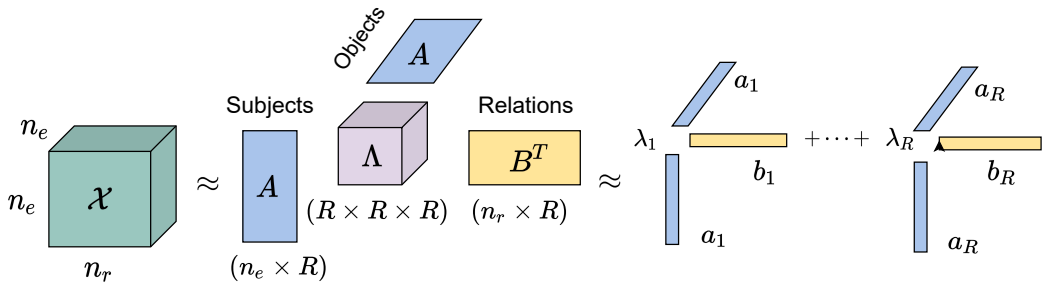


Figure 3-1: The scheme of CPD in case of informational embedding.

Approaches based on Deep Learning convolutions and attention mechanisms **ConvE, GAT, GAAT** [Dettmers et al., 2017, Nathani et al., 2019, Wang et al., 2020] achieve high performance in link prediction. Besides, they have their disadvantages - it necessitate more time and memory resources than other types of models

and usually needs pre-training.

3.3 MEKER: Memory Efficient Knowledge Embedding Representation

Our approach to entity embeddings relies on generalized CPD [Hitchcock, 1927]. Namely, R-rank CPD approximates an N-dimensional tensor as a sum of R outer products of N vectors. Every product can also be viewed as a rank-1 tensor. This approximation is described by the following formula: $\mathcal{X} \approx \mathcal{M} = [|\mathbf{A}, \mathbf{B}, \mathbf{C}|]$, where $\mathcal{X} \in \mathbb{R}^{I \times J \times K}$ is original data and $\mathcal{M} \in \mathbb{R}^{I \times J \times K}$ is its approximation. Factors have the following shape $\mathbf{A} \in \mathbb{R}^{I \times R}$, $\mathbf{B} \in \mathbb{R}^{J \times R}$, $\mathbf{C} \in \mathbb{R}^{K \times R}$. The scheme of CPD applied to the KG elements representation task is in Figure 3-1. We set matrix $\mathbf{A}$ equal to matrix C and simultaneously corresponding to subject and object entities.

3.3.1 Generalization of CPD

Following the determination of the approximation type, the next task is to find the parameters of the factor matrices that best match the ground truth data. [Battaglino et al., 2018, Dunlavy et al., 2011] describe the most widely used CPD algorithm, CP-ALS. The update rules for the factor matrices are derived by alternating minimizing squared error (MSE) loss between factor matrix, multiply by Khatri–Rao product of rest matrices and the corresponding unfolding of the tensor. [Hong et al., 2020] demonstrates that MSE corresponds to Gaussian data and is a particular case of a more general solution for an exponential family of distributions. In general, the construction of optimal factors originates from the minimisation problem:

$$\begin{aligned} \min F(\mathcal{M}; \mathcal{X}) &\equiv \sum_{i \in \Omega} f(x_i, m_i), \\ f(x, m) &\equiv \log p(x | l^{-1}(m)), \end{aligned} \tag{3.1}$$

where F - tensor consists of values of elementwise loss f between elements in intial tensor $\mathcal{X}$ and restored tensor $\mathcal{M}$, Ω - set of indices of known elements of $\mathcal{X}$, x_i and

m_i - the i-th elements of $\mathcal{X}$ and $\mathcal{M}$, respectively. The l is the link function, the artefact of the canonical representation of the exponential family of distributions, which relates to the expected value of the study variable. We also introduce $\mathcal{Y}$ - the tensor of derivatives of the elementwise loss with the same size as $\mathcal{X}$ and filled by zeros for $i \notin \Omega$. The data in the sparse one-hot triple tensor has a Bernoulli distribution. The link function for Bernoulli is $l(\rho) = log(\rho/(1-\rho))$ and associated probability is $\rho = \exp(m)(1 - \exp(m))$ so the loss function and elements of $\mathcal{Y}$ are defined as follows:

$$\begin{aligned} f(x_i, m_i) &= \log(1 + \exp m_i) - x_i m_i, \\ y(x_i, m_i) &= \frac{\partial f(x_i, m_i)}{\partial m_i} = \frac{\exp m_i}{1 + \exp m_i} - x_i. \end{aligned} \tag{3.2}$$

Hong et al. [2020] derives partial derivatives of F w.r.t. factor matrices and presents gradients $\mathbf{G}$ of it in a form similar to standard CPD matrix update formulas:

$$\begin{aligned} \mathbf{G_A} &= \mathcal{Y}_{[0]}(\mathbf{B} \odot \mathbf{C})^{T\dagger}, \\ \mathbf{G_B} &= \mathcal{Y}_{[1]}(\mathbf{A} \odot \mathbf{C})^{T\dagger}, \\ \mathbf{G_C} &= \mathcal{Y}_{[2]}(\mathbf{A} \odot \mathbf{B})^{T\dagger}, \end{aligned} \tag{3.3}$$

where $\dagger$, $\odot$ and $\mathcal{X}_{[n]}$ are defined in a Chapter 2. The importance of representation (3.1) is that we can calculate the gradients via an essential tensor operation called the matricized tensor times Khatri-Rao product (MTTKRP), implemented and optimized in most programming languages. Algorithm 3 describes the procedure for computing factor matrices gradients (3.3) in a Bernoulli distribution case (3.2).

3.3.2 Implementation Details

We use PyTorch [Paszke et al., 2019a] to implement the MEKER model. We set the object and subject factors equal and correspond to matrix A for the decomposition of the one-hot KG triplet tensor. Sparse natural and reconstructed tensors are stored in Coordinate Format as a set of triplets (COO). We combine actual triples and sampled negative examples in batches, and process them. The corresponding pieces of the ground truth tensor and current factor matrices are cut out for each

batch. Then the pieces are sent to Algorithm 3 for calculation of the gradients of the matrix elements with the appropriate indexes. Algorithm 4 is an extension of Algorithm 1 in Chapter 2 and describes the pseudocode of the factorisation KG tensor using GCP gradients.

We train the MEKER model using Bayesian search optimisation to obtain the optimal training parameters. We use the Wandb.ai tool Biewald [2020] for experiment tracking and visualisation. Table 3.3 shows the best combinations of training parameters for the proposed datasets.

Baselines As a comparison, we deploy related link prediction approaches that meet the following criteria: 1) it should learn the KG embedding from scratch, 2) it should report high performance. We use the Tucker, Hyper, ConvKB, and QuatE implementations from their respective repositories. For TransE, DistMult, ComplEx, and ConvE, we use LibKGE Ruffinelli et al. [2020] library with the best parameter setting to reproduce every model. We run each model five times for each observed value and provide means and sample standard deviation.

Algorithm 3 GCP GRAD Bernuilli

Input: $\mathcal{X}$ ▷ Ground Truth Tensor
$\mathbf{A}$, $\mathbf{B}$, $\mathbf{C}$ ▷ Factor matrices

Output: $\mathcal{F}$, $\mathbf{G_A}$, $\mathbf{G_B}$, $\mathbf{G_C}$

$\mathcal{M} = \{\mathbf{A}, \mathbf{B}, \mathbf{C}\}$ ▷ Model Restored tensor

$F = \sum_i f(x_i, m_i) = \sum log(1 + e_i^m) - x_i m_i$ ▷ Loss

$\mathcal{Y} = \sum_i \frac{\delta f(x_i, m_i)}{\delta m_i} =$ ▷ Derivative tensor
$= \sum \frac{1}{1+e^{(-m_i)}} - x_i$

$\mathbf{G_A} = \mathcal{Y}_{[0]}(\mathbf{B} \odot \mathbf{C})^{T\dagger}$. ▷ Element-wise gradient for $\mathbf{A}$
$\mathbf{G_B} = \mathcal{Y}_{[1]}(\mathbf{A} \odot \mathbf{C})^{T\dagger}$ ▷ Element-wise gradient for $\mathbf{B}$
$\mathbf{G_C} = \mathcal{Y}_{[2]}(\mathbf{A} \odot \mathbf{B})^{T\dagger}$ ▷ Element-wise gradient for $\mathbf{C}$

Algorithm 4 Factorization of the KG tensor using GCP gradients

Input: $\mathcal{X}$ ▷ Ground Truth Tensor
Triplets ▷ List of triplets
LR ▷ learning rate
R ▷ Desired size of embeddings
N ▷ Number of epoch

Output: **A**, **B** ▷ Updated factor matrices

Initialize factor matrices $\mathbf{A} \in \mathbb{R}^{R \times n_e}$, $\mathbf{B} \in \mathbb{R}^{R \times n_r}$

for $i = 1 \ldots N$ **do**
  for $[\text{inds}_a, \text{inds}_b, \text{inds}_c]$ in **Triplets do**
    $\mathcal{X}_{batch} = \mathcal{X}[\text{inds}_a, \text{inds}_b, \text{inds}_c]$
    g_a, g_b, g_c, $loss$ = GCP_GRAD($\mathcal{X}_{batch}$, $\mathbf{A}[\text{inds}_a]$, $\mathbf{B}[\text{inds}_b]$, $\mathbf{A}[\text{inds}_c]$)
    $\mathbf{A}[\text{inds}_a].\text{grad} = g_a$
    $\mathbf{B}[\text{inds}_b].\text{grad} = g_b$
    $\mathbf{A}[\text{inds}_c].\text{grad} = g_c$
    UPDATE($\mathbf{A}$, $\mathbf{B}$, LR)
  end for
end for

3.4 Experiments on Standard Link Prediction Datasets

3.4.1 Experimental Settings

The link prediction task estimates the quality of KG embedding. Link prediction is a classification that predicts whether a triple over graph element is true or not. The scoring function $\Phi(e_s, rel, e_o)$ returns the probability of constructing a true triple. We test our model on this task using standard link prediction datasets.

FB15k237 [Toutanova and Chen, 2015] is the dataset based on the FB15k adapted Freebase subset, which contains triples with the most mentioned entities. FB15k237 devised the method to select the most frequent relations and then filter inversions from the test and the valid parts. The **WN18RR** [Bordes et al., 2013] version of WN18 is devoid of inverse relations. WN18 is a WordNet database that contains the senses of words, as well as the lexical relationships between them. All of these data are presented in English. Table 3.4 shows the number of entities, relations, and train-valid-test partitions for each dataset used in the proposed work. For evaluation, we obtain complementary candidates from the entity set for each entity-relation pair from each test triple and estimate the probability score of the input triple being true. The presence of an entity that really completes a triple, among the candidates with the highest probability signifies a hit. The candidate

Table 3.1: Link Prediction scores for various models on the FB15k237 and WN18RR datasets. The embedding size is 200. The winner scores are highlighted in bold font, and the second results are underlined. The group of models, marked by ⋆ is employed using LibKGE framework.

Dataset	FB15k237				WNRR18			
Model	MRR	Hit@10	Hit@3	Hit@1	MRR	Hit@10	Hit@3	Hit@1
ConvKB Nguyen et al. [2018]	0.299	0.479	0.327	0.229	0.222	0.507	0.378	0.035
HypER Balazevic et al. [2018]	0.342	0.523	0.377	0.253	0.465	0.522	0.477	0.436
TuckER Balazevic et al. [2019]	0.345	<u>0.541</u>	0.389	0.261	0.465	0.526	0.478	0.436
QuatE Zhang et al. [2019]	**0.361**	**0.554**	**0.401**	**0.271**	**0.482**	**0.572**	**0.496**	**0.436**
CP-N3 Lacroix et al. [2018]	0.351	0.529	0.388	0.265	0.440	0.486	0.449	0.421
ConvE⋆ Dettmers et al. [2017]	0.337	0.521	0.368	0.238	0.428	0.505	0.449	0.393
TransE⋆ Bordes et al. [2013]	0.312	0.496	0.317	0.219	0.227	0.519	0.367	0.052
DistMult⋆ Yang et al. [2015]	0.333	0.518	0.367	0.241	0.451	0.522	0.463	0.416
ComplEx⋆ Trouillon et al. [2016]	0.339	0.526	0.372	0.247	0.475	<u>0.547</u>	0.481	0.436
MEKER	<u>0.359</u>	0.539	<u>0.392</u>	<u>0.268</u>	<u>0.477</u>	0.5447	<u>0.488</u>	<u>0.437</u>

Table 3.2: Sample Standard Deviation for Link Prediction scores on FB15k237 and WN18RR datasets for different models. The embedding size is 200. These deviations correspond to Table 3.1.

Dataset	FB15k237				WNRR18			
Model	MRR	Hit@10	Hit@3	Hit@1	MRR	Hit@10	Hit@3	Hit@1
ConvE	0.0058	0.0018	0.0016	0.0011	0.0018	0.0016	0.0016	0.0026
HypER	0.0004	0.0015	0.0007	0.0011	0.0019	0.0021	0.0030	0.0012
DistMult	0.0024	0.0027	0.0026	0.0023	0.0011	0.0014	0.0015	0.0010
ComplEx	0.0015	0.0014	0.0025	0.0013	0.0023	0.0031	0.0024	0.0027
TuckER	0.0012	0.0019	0.0009	0.0023	0.0012	0.0010	0.0011	0.0019
MEKER	0.0010	0.0014	0.0009	0.0015	0.0018	0.0026	0.0027	0.0026

ranking is performed with a filtered setting, which was first used in [Bordes et al., 2013]. In a filtered setting, all candidates who completed a true triple in the current step are removed from the set, except for the expected entity. We use Hit@1, Hit@3, Hit@10 as evaluation metrics. We also use mean reciprocal rank (MRR) to ensure that true complementary elements are ranked correctly.

Assumption of the Linearity In general, CPD assumes linearity in matrices **A**, **B**, and **C** parameters. We must ensure that each of the three input matrices is distinct. This requirement is met for the datasets considered in this thesis and the proposed calculation conditions. Despite the fact that objects and subjects are represented in the same matrix, in every batch step objects and subjects sets do not intersect at any point, and matrices pieces $\mathbf{A}[inds_{obj}]$, $\mathbf{A}[inds_{subj}]$ are different. Fig-

Table 3.3: The best hyperparameters of the MEKER.

Dataset	FB15k237	WN18RR
Optimizer	AdamW	AdamW
LR	0.01	0.009
Batch Size	156	128
L2 reg	0.001	0.0
Number of negative	6	8
Step of decay LR	3	15
Gamma of decay LR	0.8	0.6

Table 3.4: Statistics of link prediction datasets (all for English language).

Dataset	#ents	#rels	Number of Triplets		
			Train	Valid	Test
Fb15k237	14,541	237	$27.2{\cdot}10^4$	17,535	20,466
WN18RR	40,943	11	$8.6{\cdot}10^4$	30,034	3,134
Wiki4M	$4{,}316{\cdot}10^4$	1,245	$1{,}367{\cdot}10^4$	30,000	35,815
Wikidata5m	$4{,}594{\cdot}10^4$	822	$2{,}061{\cdot}10^4$	5,163	5,133

ure 3-2 shows a histogram of the number of intersections in the bathes of Fb15k237. It indicates how frequently a given entity appears in one batch as both an object and a subject role. From Figure 3-2 we can see that there are almost no intersections in the majority of cases. At the same time, datasets with inverted relations (for example, relations of the form *award/nomenee*, *nomenee/award* [Toutanova and Chen, 2015]) may not meet the linearity assumption in general. As a result, the proposed method based on the GCP algorithm should be used with caution for datasets with inversions.

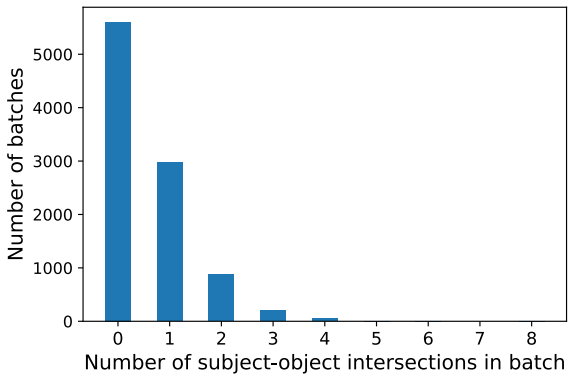


Figure 3-2: Intersections of objects and subjects entities in FB15k237 training batches.

3.4.2 Link Prediction

Table 3.1 shows the mean value of the experiment on small datasets for the embedding of size 200. The Hit@10 standard deviation for MEKER is 0.0034 for the FB15k237 dataset and 0.0026 for the WNRR18 dataset. The table with deviations from all experiments, comparable to Table 3.1, is in Table 3.2.

The best score belongs to the QuatE [Zhang et al., 2019] model due to its highly expressive 4-dimensional representations. Among the remaining approaches, MEKER outperforms its contestants' over all metrics except for the Hit@10 - Tucker model surpasses MEKER for Fb15k237, ComplEX by LibKGE for WNRR18. In general, MEKER shows decent results comparable to strong baselines of Zhang et al. [2019], Balazevic et al. [2019]. It is also worth noting that MEKER significantly improves the MRR and Hit@1 metrics on Freebase datasets, while on the word sense, according to the data, it has been enhanced in Hit@10.

Model efficiency in case of parameter size increasing Given a strong memory assumption, we can reduce the size of pre-trained MEKER embeddings by tenfold while losing only a few percent of performance.

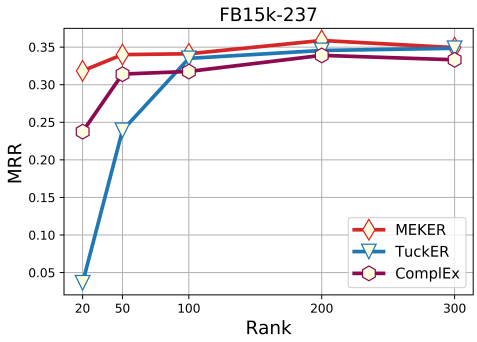


Figure 3-3: MRR scores in dependence of embedding ranks.

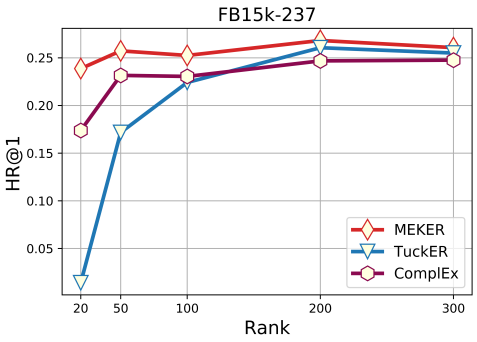


Figure 3-4: Hit@1 scores in dependence of embedding ranks.

Figures 3-3 and 3-4 show MRR and Hit@1 scores for MEKER, TuckER, and ComplEX models at various embedding sizes. Each model achieves a constant value on both metrics around rank 100. For ranks 200 and 300, difference in performance between the three models is approximately consistent for both metrics, with MEKER scoring the highest on rank 20. It means that the number of MEKER parameters can

be reduced while maintaining or improving quality. The quality loss is significant for other presented models.

3.4.3 Memory Complexity Analysis

We show the space complexity of models mentioned in the current work in the right column of Table 3.5. In the context of the Link Prediction task, all approaches have asymptotic memory complexity $\mathcal{O}((n_e + n_r)d)$, which is proportional to the size of the full dictionary of KG elements, i.e. the embedding layer or look-up table. Other detailes of the proposed models are less significant: the convolutional layers are not very extensive. The implementation determines the amount of real memory used by the model during the training process. The Neural Network backpropagation mechanism is used to tune parameters in the most number of related works. Backpropagation in Figure 3-5 creates a computational graph in which all model parameters are duplicated. It results in a multiplicative constant 2, insignificant for a small dictionary but critical in a large one. To summarize, the following factors should be taken into account to decrease MEKER's required memory:

1. In the MEKER algorithm gradients are computed analytically.

2. MEKER does not have additional neural network layers (linear, convolutional, or attention).

To measure GPU RAM usage, we run each considered embedding model on FB15k-237 on a single GPU and print peak GPU memory usage within the created process. The left column of a Table 3.5 demonstrates that MEKER has objective memory complexity that is at least twice lower than that of other linear approaches. This property reveals the possibility of obtaining representations of specific large databases using a single GPU card.

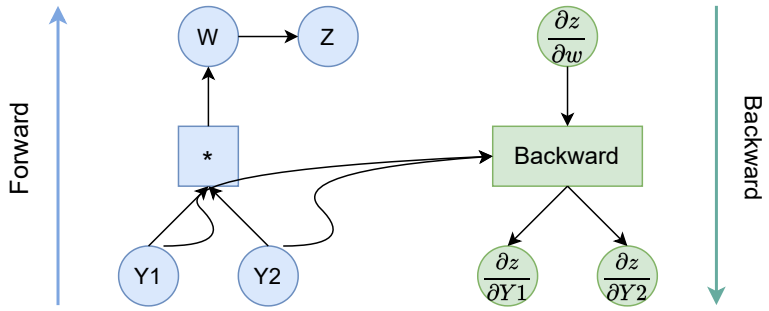


Figure 3-5: The scheme of the augmented computational graph of the Neural Network.

Table 3.5: Memory, reserved in the PyTorch Framework during the training process and theoretical approximation of given implementations' complexity. On the FB15k237 dataset, we train 200-size representations with a batch size of 128. *Lin* denotes the number of output features in a linear layer, *conv* denotes the size of convolutional layer parameters. The constant c represents the number of different layers.

Model	GPU Memory Usage, MB	Theoretical Approximation of Space Complexity
TuckER	357	$2 \cdot ((n_e + n_r + c \cdot lin) \cdot d)$
HypER	208	$2 \cdot ((n_e + n_r + c \cdot lin) \cdot d)$
ConvKB	3 563	$2 \cdot ((n_e + n_r) \cdot d + c \cdot conv)$
ConvE	229	$2 \cdot ((n_e + n_r) \cdot d + c \cdot conv)$
ComplEX	252	$2 \cdot (n_e + n_r) \cdot d$
DistMult	174	$2 \cdot (n_e + n_r) \cdot d$
QuatE	2 367	$2 \cdot 4 \cdot (n_e + n_d + c \cdot lin)$
CP (N3)	138	$2 \cdot (n_e + n_r) \cdot d$
MEKER	79	$((n_e + n_r) \cdot d)$

3.5 Experiments on Large-Scale KG Datasets

3.5.1 Experimental Settings

To test the model on large KGs, we employ two WikiData-based datasets. The first dataset in English, **Wikidata5m** [Wang et al., 2021][2], is selected due to the presence of related works and reproducible baseline Ruffinelli et al. [2020]. This dataset is created over the 2019 WikiData dump and contains elements with links to informative Wikipedia pages. Our experiments use the transductive setting of Wikidata5m — triplet sets to disjoint across training, validation, and test.

[2] https://deepgraphlearning.github.io/project/wikidata5m

The second English-Russian dataset is formed since its suitability for the NLP downstream task. We leverage KG-based fact retrieval over Russian Knowledge Base Questions (RuBQ) Rybin et al. [2021] benchmark. This benchmark is a subset of Wikidata entities with Russian labels. Some elements in RuBQ are not covered with Wikidata5m, so we created a link-prediction **Wiki4M** dataset over RuBQ. We select triples without literal objects and obtain approximately 13M triples across 4M entities (see Table 3.4). Wiki4M also fits the concept of multilingualism and could be used in a cross — lingual transfer or few-shot learning.

Table 3.6: Unfiltered link prediction scores for MEKER and PyTorch-BigGraph approaches for Wiki4M and Wikidata5m datasets and memory needed in leveraging every model. Storage means additional memory demanded for auxiliary structures. Batch size 256. Here "RAM" is GPU RAM or main memory RAM if GPU limit of 24 GB is reached. *Sparse* means sparse embeddings. Models without *sparse* mark employ dense embeddings matrix.

Model	MRR	Hit@1	Hit@3	Hit@10	Memory, GB	Storage, GB
English: Wikidata5m dataset						
PTBG (ComplEX)	0.184	0.131	0.210	0.287	45.15	9.25
PTBG (TransE)	0.150	0.091	0.176	0.263	43.64	9.25
LibKGE sparse (TransE)	0.142	0.153	0.211	0.252	33.29	0.00
LibKGE sparse (ComplEX)	0.202	**0.160**	0.233	0.316	**21.42**	0.00
MEKER (ours)	**0.211**	0.149	**0.238**	**0.325**	22.27	0.00
Russian: Wiki4M dataset						
PTBG (ComplEX)	0.194	0.141	0.212	0.293	42.83	9.25
LibKGE sparse (TransE)	0.183	0.126	0.191	0.275	26.75	0.00
LibKGE sparse (ComplEX)	0.247	0.196	0.275	0.345	**20.22**	0.00
MEKER (ours)	**0.269**	**0.199**	**0.303**	**0.410**	21.04	0.00

3.5.2 Link Prediction

We embed the datasets for ten epochs on a 24.268 Gb GPU card with the following model settings: LR $2.5 \cdot 10^{-4}$, increasing in 0.5 steps every 10 epoch, batch size 256, number of negative samples 4 for Wiki4M and 2 for Wikidata5m.

As a comparison, we use the PyTorch-BigGraph large-scale embedding system [Lerer et al., 2019]. PyTorch-BigGraph modifies several traditional embedding systems to focus on the effective representation of KG in memory. We select ComplEX and TransE and train graphs for these embedding models, dividing large datasets into four partitions. With a batch size of 256, the training process takes 50 epochs.

We also deploy LibKGE [Ruffinelli et al., 2020] to evaluate TransE and ComplEX approaches. For ComplEX model training, we use the best parameter configuration from the repository, for TransE, we obtain a set of training parameters by grid search.

Hyperparameters iterate over the following set: optimizer type {Adam, AdamW, SGD}, learning rate {0.00001, 0.00005, 0.0001, 0.0005, 0.001, 0.005, 0.01, 0.05}, batch size {64, 128, 256, 512}, number of negative samples per positive {2, 4, 5, 6}, L2 regularizer {0.0, 0.001, 0.01, 0.05, 0.1}. For SGD, we switch Nesterov momentum between {True, False} and set momentum value to {0.8, 0.9, 0.99}. We also vary learning rate decay in {0.2, 0.5, 0.8} in every {2, 8, 10} steps.

The final learning rate for TransE is 0.5, decaying in factor 0.45 every 5 steps and train model in 100 epochs. In both cases, we use sparse embedding in the corresponding model setting and batch size of 256. Models from both wrappers that did not fit in 24 GB, we train on the CPU.

Embedding sets, yielded by these experiments, we then test on the link prediction task. We provide scoring without filters because the partition-based setup of PyTorch-BigGraph does not support filtering evaluation. Tables 3.6 shows that MEKER significantly improves the results of PyTorch-BigGraph models across all proposed metrics. The ComplEX model with sparse embedding, fine-tuned by LibKGE, gives results almost approaching the MEKER and exceeding the Hit@1 in Wiki4M. The right part of Tables 3.6 shows that the baseline approaches consume twice as much memory as MEKER, but sparse ComplEX slightly improves memory consumption. TransE does not give such significant results as ComplEX.

3.5.3 Knowledge Base Question Answering (KBQA)

In this section, to further evaluate the proposed MEKER embeddings we test them extrinsically on a KBQA task on two datasets for English and Russian.

Experimental Setting

We perform experiments with two datasets: for English, we use the common dataset SimpleQuestions [Bordes et al., 2015] aligned with Wiki4M KG[3] (cf. Table 3.4), and for Russian, we use the RuBQ 2.0 dataset [Rybin et al., 2021] which comes with the mentioned above Wiki4M KG (cf. Table 3.4). RuBQ 2.0 is the Russian language QA benchmark with multiple types of questions aligned with Wikidata. For both SimpleQuestions and RuBQ, an answer is represented by a KG triple for each question.

For training, we use a SimpleQuestions training set. For verification, we use a SimpleQuestions and RuBQ 2.0 dataset test set for English and Russian, respectively. These Q&A pairs provide ground truth answers linked to this exact version of KG elements.

More specifically, in these experiments, we test answers to 1-hop questions, which are questions corresponding to one subject and one relation in the knowledge graph and take their object as an answer.

We want to leverage the KBQA model, which can process questions in English and Russian. To measure the performance of a KBQA system, we measure the accuracy of the answer / entity retrieved. This metric was used in previously reported results on SimpleQuestions and RuBQ. If the subject of the answer triple matches the reference by ID or name, it is considered correct.

KBQA methods

The key idea of the KBQA approaches is mapping questions in natural language to the low-dimensional space and comparing them to graph elements' given representation. In **KEQA** [Huang et al., 2019], LSTM models detect the entity and predicates from the question text and project it further into the entity and predicate embedding spaces. We select the closest subject in terms of similarity to the entity and predicate embeddings as the answer.

We created a simple approach **Text2Graph** which stems from the **KEQA** and differs from the original work in improved question encoder, entity extractor, addi-

[3] https://github.com/askplatypus/wikidata-simplequestions

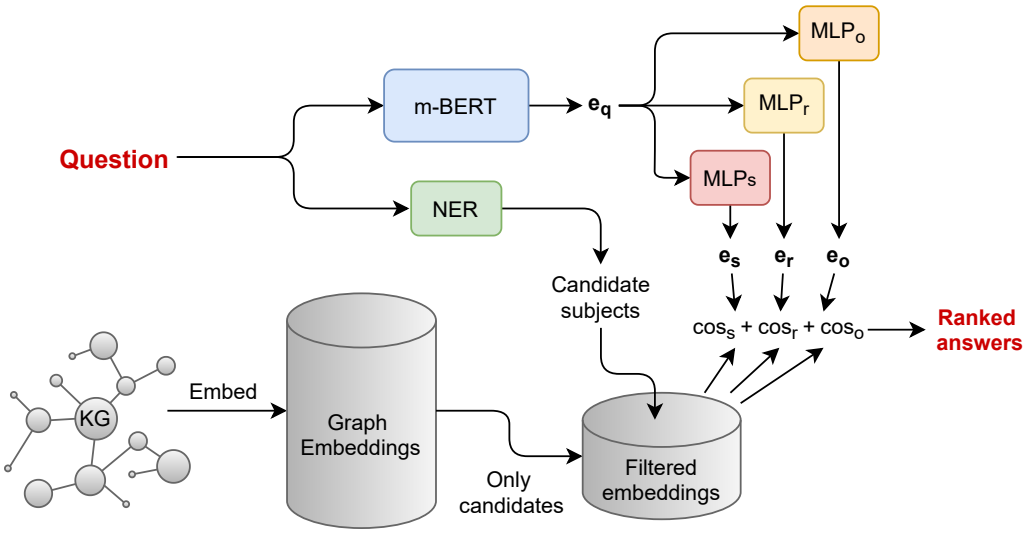


Figure 3-6: Text2Graph method used in our experiments: 1-Hop QA pipeline. First, we take an original entity and relation embeddings. The question is embedded using m-BERT. This embedding is then processed by MLP, yielding candidate representations of an object, relation, and subject. The sum of the subject, relation, and object cosines is the final score of the triple candidates.

tional subject embedding space and simplified retrieval pipeline. The Algorithm 5 describes projection of the input question to graph elements. The multilingual-BERT [Devlin et al., 2019] model encodes the input question, and all word vectors are averaged into a single deep contextualized representation e_q. This representation then goes through three MLPs jointly learning candidate embeddings of an object, relation, and subject. We minimize MSE between predicted embeddings and the corresponding KGE model's embeddings. The appropriateness score of every fact in KG is a sum of cosine similarity between MLP outputs and ground truth model representation for every element in the triple. We consider the triple with the highest score as an answer. We train the pipeline using an AdamW optimizer with default parameters for 10 epochs.

Baselines We evaluate our method on datasets with English and Russian questions. On every benchmark, we provide several baselines, which are described below and are, in fact, a combination of the embedding model with a QA system.

RuBQ 2.0

Our method MEKER is combined with several QA approaches compatible with questions from this benchmark.

Algorithm 5 Text2Graph question projection algorithm

Input: $\mathcal{Q}$, $\mathcal{G}$, $\mathbf{E}$, text encoder M_{enc}, projection modules: M_s, M_r, M_o, Subject Candidates Extractor: NER

Output: answer $\langle o_a, r_a, s_a \rangle$

```
e_q = M_enc(Q)
Initialize answers A=[], scores S=[], candidates C=[]
for entity in G do
  if entity.name in NER(Q) then
    C.append(entity)
  end if
end for
for entity in C do
  for relation in entity.relations do
    s = entity.id, r = relation.id, o = entity[r]
    triple = ⟨s, r, o⟩
    e_s = E[s], e_r = E[r], e_o = E[o]
    y_s = M_s(e_q), y_r = M_r(e_q), y_o = M_o(e_q)
    score = cos(e_o, y_o) + cos(e_r, y_r) + cos(e_s, y_s)
    A.append(triple), S.append(score)
  end for
end for
ind = argmax(S)
⟨s_a, r_a, o_a⟩ = A[ind]
return ⟨s_a, r_a, o_a⟩
```

QAnswer[4] is a rule-based system addressing questions in several languages, including Russian. **SimBa** is a baseline presented by RuBQ 2.0 authors. It is a SPARQL query generator based on an entity linker and a rule-based relation extractor. KBQA module of **DeepPavlov Dialogue System Library** [Burtsev et al., 2018] is also based on query processing.

[4] https://www.qanswer.eu

Table 3.7: Comparison of the Text2Graph system with the various KG embeddings with the existing solutions (QA-Ru, QA-En, SimBa) on RuBQ 2.0 benchmark.

KBQA Model	Embedding Model	Accuracy 1-Hop
DeepPavlov	-	30.5 ± 0.04
SimBa	-	32.3 ± 0.05
QA-En	-	32.3 ± 0.08
QA-Ru	-	30.8 ± 0.03
Text2Graph	PTBG (ComplEX) Wiki4M	48.16 ± 0.05
Text2Graph	PTBG (TransE) Wiki4M	48.84 ± 0.06
Text2Graph	MEKER Wiki4M	49.06 ± 0.06

Table 3.8: Comparison of the Text2Graph system with the various KG embeddings with the existing embedding-based solution on the SimpleQuestions benchmark.

KBQA Model	Embedding Model	Accuracy 1-Hop
KEQA	TransE FB5M	40.48 ± 0.10
Text2Graph	PTBG (TransE) Wikidata5m	59.97 ± 0.15
Text2Graph	MEKER Wikidata5m	61.81 ± 0.13

SimpleQuestions

Simple Question is an English language benchmark aligned with FB5M KG - the subset of Freebase KG. Training and validation parts consist of 100k and 20k questions, respectively. As a baseline solution, we employ **KEQA** [Huang et al., 2019]. We realign answers from this benchmark to our system, which is compatible with Wikidata5m. Not all of the questions from FB5M have answers among Wiki4M, that is why we test both systems on a subset of questions whose answers are present in both knowledge graphs.

Experimental Results

We compare the results of the Text2Graph with PTBG embeddings versus MEKER embedding and baseline KBQA models. Results on the RuBQ 2.0 dataset are shown in Table 3.7. Text2Graph outperforms baselines. Using MEKER embeddings instead of the PTBG version of ComplEX and TransE demonstrates slightly better accuracy.

Table 3.8 presents results on the SimpleQuestions dataset. As a Huang et al. [2019] model uses FB5M KG and Text2Graph uses Wikidata5m KG we test both

models on the subset of questions, which answers are present in both knowledge graphs for a fair comparison. Our model demonstrates superior performance and regarding the comparison within different embeddings in a fixed system, MEKER provides better accuracy of answers than TransE embeddings on the SimpleQuestions benchmark.

3.6 Conclusion

In this Chapter, we apply Canonycal Polyadic representation for Knowledge graph data, representing as a 3-dimensional tensor. This representation allows to perform KG data in compressed way, consisting of entity and relation embeddings.

To obtain the corresponding decomposition of the original tensor, we propose MEKER, a linear knowledge embedding model based on generalised CPD. This method allows an analytical calculation of the gradient, simplifying the training process under memory restriction. Compared to previous linear KG embedding models Balazevic et al. [2019], our approach achieves high efficiency while using less memory during training. On the standard link prediction datasets WN18RR and FB15k-237, MEKER shows competitive results.

In addition, we created a Text2Graph KBQA system based on the learnt KB embeddings to demonstrate the model's effectiveness in NLP tasks. We obtained the required representations using MEKER on the Wikipedia-based Wiki4M dataset for questions in Russian and Wikidata5m for questions in English. Text2Graph outperforms baselines for English and Russian, while using MEKER's embeddings provides additional performance gain compared to PTBG embeddings. The link prediction scores of our model on Wiki4M and Wikidata5m outperform the baseline results. MEKER can be helpful in question-answering systems over specific KG, in other words, in systems that need to embed large sets of facts with acceptable quality.

All codes for reproducing our experiments are available online[5]

[5] https://github.com/s-nlp/meker

Chapter 4

SVD and TTM Representations of the Fully-Connected (FC) Layers

This Chapter is based on the paper "Efficient GPT Model Pre-training using Tensor Train Matrix Representation" (cf. Section "Publications" at page 4).

4.1 Introduction

In this Chapter, we present a methodological tool used for efficient representation of linear layers in NLP deep neural networks.

Algebraic structures are commonly employed to represent linear layers, such as TTM decomposition [Usvyatsov et al., 2022] or the Kronecker decomposition [Edalati et al., 2021]. Since low-rank approaches decrease the expressivity, in Neural Network an adaptive version is usually deployed [Thakker et al., 2020], [Chen et al., 2019]. This replacement occurs to reduce the number of parameters in the layer and the overall size of the model. To achieve this, a layer class that is compatible with the chosen representation structure must be provided, and further can be effectively integrated within the neural network model for training and fine-tuning. Existing works, such as Usvyatsov et al. [2022] and Novikov et al. [2015], have addressed this for the TTM structure. However, these implementations sometimes suffer from high memory consumption during training due to their internal structure [Novikov et al., 2015]. To overcome this challenge, we have developed a TTM layer class that

incorporates time- and memory-aware signal propagation function implementations, ensuring efficient training performance.

We develop a custom TTM-layer that first has fewer parameters than regular Fully connected layer and, second, uses less memory during forward and backward passes.

4.2 Method SVD

Singular Value Decomposition (SVD) is defined as follows: $\mathbf{W} = \mathbf{U\Sigma V^T}$. To compress a linear layer with SVD, we use truncated products of it with rank r: $\mathbf{U_r} = \mathbf{U}[:, :\mathbf{r}], \mathbf{\Sigma_r} = \mathbf{\Sigma}[:\mathbf{r}, :\mathbf{r}], \mathbf{V_r} = \mathbf{V}[:, :\mathbf{r}]$. In order to avoid double multiplication of the input activation matrix to a diagonal matrix $\mathbf{\Sigma_r}$, we carry it inside $\mathbf{U_r}$ and $\mathbf{V_r}$:

$$\mathbf{W_2} = \mathbf{U}[:, :\mathbf{r}]\sqrt{\mathbf{\Sigma_r}}, \mathbf{W_1} = \sqrt{\mathbf{\Sigma_r}}\mathbf{U}[:, :\mathbf{r}]^\mathbf{T} \tag{4.1}$$

As a result, we get approximation of linear matrix $\mathbf{W} \approx \mathbf{W_2W_1}$ and approximation of the initial layer $\mathbf{Y} \approx \mathbf{XW_1^TW_2^T} + \mathbf{b}$.

This is equivalent to two sequential linear layers - first with weight $\mathbf{W_1}$ and zero bias and the second one with weight $\mathbf{W_2}$ and bias $\mathbf{b}$. If $\mathbf{W}$ have n_{in}, n_{out} shape, the number of parameters in layer before compression is $n_{in} \times n_{out}$, after representation by truncated SVD is $r \times (n_{in} + n_{out})$.

Based on this, the parameter compression rate for the SVD representation of a linear layer can be determined as follows:

$$c\_rate = \frac{r \times (n_{in} + n_{out})}{n_{in} \times n_{out}} \tag{4.2}$$

4.3 Method TTM

As described in section 2.2.4, Tensor Train Matrix (TTM) decomposition is efficient for matrices that can be represented as the Kronecker product of objects. In other words, such matrices are expected to be noninvariant to the permutation of

columns (features). For a pretrained matrix of a fully connected layer of neural network architectures, this is generally not true - in such a matrix, there is no well-defined structure and swapping rows and columns doesn't yield any significant changes.

However, it is possible to construct a container that utilizes the TTM structure to store weights. The formula 2.11 defines elements of the initial tensor in the TTM form through core tensors and can be employed to compute the elements of the matrix of the corresponding linear layer with the aid of these containers. By training this structure from scratch, we can attain a compressed representation of a "virtual" FC layer, where the weight matrix is already expressed in the TTM decomposition form.

As described in Chapter 2, the TTM form is, in fact, a multidimensional tensor over 2-dimensional objects with sizes 2×2. Consequently, the decomposed matrix should be expressed as a mosaic of elementary blocks from $\mathbb{R}^{2\times 2}$ space and has dimensions equal to the power of 2. Since matrices in FC layers are not engaged in this, we will deviate from this rule in the experimental implementation. Therefore, the elementary blocks have the dimensions of the factors of the initial matrix's sizes.

According to the Subection 2.2.4, tensor $\mathcal{T} \in \mathbb{R}^{I_1\times J_1\times\cdots\times I_M\times J_M}$ is represented in *TTM format* with ranks $(R_0, R_1, \ldots, R_M)$ if each element is computed as:

$$\mathcal{T}(i_1, j_1, \ldots, i_M, j_M) \approx \sum_{r_1=1}^{R_1} \cdots \sum_{r_{M-1}=1}^{R_{M-1}} G^{(1)}(i_1, j_1, r_1) G^{(2)}(r_1, i_2, j_2, r_2) \ldots G^{(M)}(r_{M-1}, i_M, j_M), \tag{4.3}$$

where $\mathcal{G}^{(m)} \in \mathbb{R}^{R_{m-1}\times I_m\times J_m\times R_m}$, $m = 1, 2, \ldots, M-1$ are *core tensors (cores)* of TTM decomposition, $R_0 = R_M = 1$.

Estimating the number of parameters in the original and compressed version of the matrix $\mathbf{W} \in \mathbb{R}^{D_{in}\times D_{out}}$, where $D_{in} = \prod_{m=1}^{M} I_m$ and $D_{out} = \prod_{m=1}^{M} J_m$, in the same way as in 2.2.3, we obtain that the complete representation stores $\prod_{m=1}^{M} I_m J_m$ elements, while the TTM core format contains only $R(I_1 J_1 + I_M J_M) + R^2 \sum_{m=2}^{M-1} I_m J_m$ parameters.

Then, the parameter compression rate for the TTM representation is:

$$c\_rate = \frac{R(I_1 J_1 + I_M J_M) + R^2 \sum_{m=2}^{M-1} I_m J_m}{\prod_{m=1}^{M} I_m J_m} \tag{4.4}$$

The substantial case in Neural Network (NN) modules is to define the proper way of signal transmission. In the TTM-based layers, forward signal propagation means sequentially *tensor contraction* of the input activation tensor $\mathcal{X}$ with the core tensors. Representing FC in this way, we should attend to memory and time-stable forward and backward methods.

TTM Tensor Contraction

Given two tensors $\mathcal{T}^1 \in \mathbb{R}^{I_1 \times \cdots \times I_M \times S_1 \times \cdots \times S_K}$ and $\mathcal{T}^2 \in \mathbb{R}^{S_1 \times \cdots \times S_K \times J_1 \times \cdots \times J_N}$ the result of *tensor contraction* along axis $s_1, \ldots, s_K$ is a tensor $\mathcal{T} \in \mathbb{R}^{I_1 \times \cdots \times I_M \times J_1 \times \cdots \times J_N}$, where one element is computed using formula

$$\begin{aligned} &\mathcal{T}(i_1, \ldots, i_M, j_1, \ldots, j_N) = \\ &= \sum_{s_1=1}^{S_1} \cdots \sum_{s_k=1}^{S_k} \mathcal{T}^1(i_1, \ldots, i_M, s_1, \ldots, s_K) \mathcal{T}^2(s_1, \ldots, s_K, j_1, \ldots, j_N) \end{aligned} \tag{4.5}$$

and requires $O(S_1 S_2 \ldots S_K) = O\left(\prod_{k=1}^{K} S_k\right)$ floating point operations (FLOPS). Thus, the number of FLOPs to compute tensor $\mathcal{T}$ is $O\left(\prod_{m=1}^{M} I_m \prod_{n=1}^{N} J_n \prod_{k=1}^{K} S_k\right)$. For example, a multiplication of two matrices of shapes (I, S) and (S, J) can be calculated for $O(IJS)$ operations.

Signal Transmission

If we have input $\mathcal{X}$ with batch size B, and sequentially contract it with the TTM cores, then after contracting with $\mathcal{G}^M, \ldots, \mathcal{G}^{k+1}$ we get a tensor of shape

$$(B, I_1, \ldots, I_K, J_{K+1}, \ldots, J_M, R_K) \tag{4.6}$$

Table 4.1: Peak memory footprints for signal propagation in full GPT-2 model with TTM layers with different ranks. At the rank 16 we have an increment in memory consumption.

Layer Type	TTM-16	TTM-32	TTM-64	Fully-Connected
Memory, GB	75.07	48.7	48.31	48.37

Table 4.2: Memory footprints for signal propagation in TTM wiht rank 16 and Fully-Connected Layers. PyTorch strategy leads to memory costs for TTM.

Layer	TTM-16	TTM-16	Fully-Connected
Backprop Strategy	PyTorch Autodiff	Einsum Full Matrix	PyTorch Autodiff
Single Layer, Batch 16	1100 MB	294 MB	395 MB

, its contraction with the next core $\mathcal{G}^k \in \mathbb{R}^{R_{k-1} \times I_k \times J_k \times R_k}$ requires

$$BI_1 \dots I_k J_k \dots J_M R_{k-1} R_k \tag{4.7}$$

steps. Thus the computational complexity of the TTM layer is

$$O(BM \max\{D_{in}, D_{out}\} \max_k\{I_k, J_k\} (\max_k R_k)^2) \tag{4.8}$$

and depends on the schedule in which we contract cores.

We measure peak memory during one training iteration in the GPT-2 model with TTM layers on different ranks. Experiments depicted in 4.2 show that on a rank 16 the TTM layer can be more memory-consuming than the regular FC layer. The memory footprint for FC and TTM layers for custom-defined and PyTorch signal propagation strategies (Table 4.2) confirms this claim.

For a tensor contraction, the PyTorch framework uses Einstein summation notation. Optimized Einsum library Smith and Gray [2018] optimizes the expression's contraction order by looking for an optimal *path* - a set of strings of the form "ikl,lkj->ij". By default optimization, the obtained path are time-stable, not memory-stable. We extend the existing research by proposing memory-efficient techniques to compute forward and backwards through the TTM layer for a more comprehensive

description of the proposed methods.

4.3.1 Forward Pass

Fully-connected layer. Given an input batch $\mathbf{X} \in \mathbb{R}^{B \times D_{in}}$ a forward pass through a fully-connected layer with weight matrix $\mathbf{W} \in \mathbb{R}^{D_{in} \times D_{out}}$ and bias vector $\mathbf{b} \in \mathbb{R}^{D_{out}}$ results in the output $\mathbf{Y} = \mathbf{XW} + \mathbf{b} \in \mathbb{R}^{B \times D_{out}}$ and requires $O(BD_{in}D_{out})$ operations.

TTM layer. In the TTM layer the weight $\mathbf{W}$ is a matrix $D_{in} \times D_{out}$ represented in the TTM format with M cores $\mathcal{G}^m \in \mathbb{R}^{R_{m-1} \times I_m \times J_m \times R_m}$, $m = \overline{1, M}$, where I_m and J_m are such that $D_{in} = \prod_{m=1}^{M} I_m$ and $D_{out} = \prod_{m=1}^{M} J_m$. More significantly, first we reshape the 2-dimensional matrix $\mathbf{W}$ into a $2M$ dimensional array $\mathcal{W}_{\mathcal{M}} = \mathbf{W}.reshape(I_1, I_2, \ldots, I_M, J_M)$.

Input matrix $\mathbf{X}$ is reshaped into a tensor $\mathcal{X} \in \mathbb{R}^{B \times I_1 \times \cdots \times I_M}$ before performing a forward pass through the layer, which outputs a tensor $\mathcal{Y} \in \mathbb{R}^{B \times J_1 \times \cdots \times J_M}$, such that

$$\mathcal{Y}(j_1, \ldots, j_M) = \sum_{i_1, \ldots, i_M} \mathcal{W}(i_1, j_1, \ldots, i_M, j_M) \mathcal{X}(i_1, \ldots, i_M), \tag{4.9}$$

where

$$\mathcal{W}(i_1, j_1, \ldots, i_M, j_M) = \sum_{r_1, \ldots, r_{M-1}} \mathcal{G}^1(r_0, i_1, j_1, r_1) \ldots \mathcal{G}^M(r_{M-1}, i_M, j_M, r_M). \tag{4.10}$$

4.3.2 Forward Pass as a Contraction Process

We represent $\mathcal{W}_M$ as a set of M cores $\mathcal{G}$. So, in equation 4.10 we should contract $\mathcal{X}$ with sequence ($\mathcal{G}^M$, ..., $\mathcal{G}^1$) sequentially. Please note that we can start with the first core ($\mathcal{G}^1, ...\mathcal{G}^M$) or with the last ($\mathcal{G}^M$, ..., $\mathcal{G}^1$), in general it doesn't matter.

We contract $\mathcal{X}$ with size (B, D_{in}) to $\mathcal{G}^m$ with size $(R_{m-1}, I_m, J_m, 1)$. As $D_{in} = \prod(I_1 \ldots I_m)$, we contract over I_m and have a tensor of shapes $(B, R_{m-1}, J_m, I_{m-1}, \ldots, I_1)$ as a result. We should contract this tensor to the core $\mathcal{G}_{m-1}$ with shapes

$$(R_{m-2}, I_{m-1}, J_{m-1}, R_{m-1}) \tag{4.11}$$

in dimensions $I_{m-1}R_{m-1}$. This operation produces the object of shapes

$$(B, R_{m-2}, J_m, J_{m-1}, I_{m-2}, \ldots, I_1). \tag{4.12}$$

Repetition of this operation K times results in the product with shapes

$$(B, I_1, \ldots, I_K, J_{K+1}, \ldots, J_M, R_K). \tag{4.13}$$

In the end, we gain the output of sizes $(B, J_1, \ldots J_M) = (B, D_{out})$.

The computational complexity of this operation is estimated above.

TTM layer: Einsum The contraction schedule computed during the forward pass is optimized via the *opt_einsum* function [Smith and Gray, 2018]. This function optimizes the time of expression's contraction in the BLAS library for common linear algebra operations. The default optimization strategy provides a recursive depth-first search over all possible paths by pruning candidates that exceed the best time.

Thus, due to some shared intermediate results, memory for saved activations might be optimized.

TTM layer: Fixed Schedule The order of cores to contract with is fixed in advance, we don't optimize it with *opt_einsum*. In this case, saved activations usually occupy the same amount of memory.

Algorithm 6 Forward pass (FC layer). Number of layer parameters is $O(BD_{in}D_{out})$. Computational complexity is $O(BD_{in}D_{out})$. SavedActivations is $O(BD_{in})$.

Input: data $\mathbf{X} \in \mathbb{R}^{B \times D_{in}}$; parameters $\mathbf{W} \in \mathbb{R}^{D_{in} \times D_{out}}$, $\mathbf{b} \in \mathbb{R}^{D_{out}}$;

Output: $\mathbf{Y} \in \mathbb{R}^{B \times D_{out}}$

$\mathbf{Y} = \mathbf{XW} + \mathbf{b}$

Algorithm 7 Forward pass (TTM layer, Fixed Scheduler). Number of layer parameters is $O(\sum_{m=1}^{M} R_{m-1} I_m J_m R_m)$.

Input: data $\mathbf{X} \in \mathbb{R}^{B \times D_{in}}; D_{in} = \prod_{m=1}^{M} I_m, D_{out} = \prod_{m=1}^{M} J_m;$
parameters $\mathcal{G}^m \in \mathbb{R}^{R_{m-1} \times I_m \times J_m \times R_m}, m = \overline{1, M}, R_0 = R_M = 1;$

Output: $\mathcal{Y} \in \mathbb{R}^{B \times J_1 \times \cdots \times J_M}$

$\mathcal{X} = Reshape(\mathbf{X}) \in \mathbb{R}^{B \times I_1 \times \cdots \times I_M}$

$\mathcal{Y}_0 := \mathcal{X}$

$ContractionSchedule := (1, 2, \ldots, M)$

for k in $ContractionSchedule$ **do**

$\mathcal{Y}_k := einsum(\mathcal{G}^k, \mathcal{Y}_{k-1})$ ▷ $FLOP_{\mathcal{Y}} = O(B \prod_{m=1}^{k+1} J_m \prod_{m=k+1}^{M} I_m R_k R_{k+1})$

$\mathcal{Y} = \mathcal{Y}_k$ ▷ $Memory_{\mathcal{Y}} = O(B \prod_{m=1}^{k} J_m \prod_{m=k+1}^{M} I_m R_k)$

end for

The forward path with a Fixed Scheduler approach is equivalent to a sequential forward pass through M linear layers. The order of contraction in the schedule might be either $(M, M-1, \ldots, 1)$ or $(1, 2, \ldots, M-1)$.

Algorithm 8 Forward pass (TTM layer, Einsum). Number of layer parameters is $O(\sum_{m=1}^{M} R_{m-1} I_m J_m R_m)$.

Input: data $\mathbf{X} \in \mathbb{R}^{B \times D_{in}}; D_{in} = \prod_{m=1}^{M} I_m, D_{out} = \prod_{m=1}^{M} J_m;$
parameters $\mathcal{G}^m \in \mathbb{R}^{R_{m-1} \times I_m \times J_m \times R_m}, m = \overline{1, M}, R_0 = R_M = 1;$

Output: $\mathcal{Y} \in \mathbb{R}^{B \times J_1 \times \cdots \times J_M}$

$\mathcal{X} = Reshape(\mathbf{X}) \in \mathbb{R}^{B \times I_1 \times \cdots \times I_M}$

$\mathcal{Y} := einsum(\mathcal{G}^1, \ldots \mathcal{G}^M, \mathcal{Y})$

4.3.3 Backward Pass

While training neural networks, intermediate activations are saved during the forward pass to compute gradients during the backward pass.

Fully-connected layer. For the layer $\mathbf{y} = \mathbf{W}\mathbf{x} + \mathbf{b}$ derivatives of loss $\mathcal{L}$ w.r.t. weight is computed as

$$\frac{\partial \mathcal{L}}{\partial \mathbf{W}} = \frac{\partial \mathcal{L}}{\partial \mathbf{y}} \mathbf{x}^T \tag{4.14}$$

TTM layer: Automatic Pytorch differentiation (Autodiff). Automatic Pytorch differentiation during backpropagation through the TTM layer results in storing many intermediate activations, as the TTM layer is considered as a sequence of linear layers (where the number of layers corresponds to the number of core tensors).

We propose several ways to perform a backward pass that require smaller memory consumption.

TTM layer: Full Einsum. In the first approach for each core tensor $\mathcal{G}_m$ we compute the gradient of the loss with respect to its parameters:

$$\frac{\partial \mathcal{L}}{\partial \mathcal{G}_m} = \frac{\partial \mathcal{L}}{\partial \mathbf{W}} \frac{\partial \mathbf{W}}{\partial \mathcal{G}_m} = \mathbf{X}^T \frac{\partial \mathcal{L}}{\partial \mathcal{Y}} \frac{\partial \mathbf{W}}{\partial \mathcal{G}_m}. \tag{4.15}$$

As a gradient computation might be considered as a tensor contraction along a specified axis, the process includes three main steps.

Firstly, we generate a string-type expression, which specifies the shapes of input and resulting tensors (e.g. "$ikl, lkj \to ij$" for performing tensor contraction along two axes). Second, the schedule of contraction is defined (e.g., firstly along axis 'l' and then along axis 'k'). And thirdly, einsum computation is performed.

In the Full Einsum approach first two steps (expression generation, contraction scheduling) are performed independently for all $\frac{\partial \mathcal{L}}{\partial \mathcal{G}_m}$. The third step, in turn, tracks simultaneously what contractions are computed for different derivatives and allows the sharing of intermediate results. Due to this sharing we get memory savings compared to the Autodiff approach.

TTM layer: Full Matrix. In the Full Matrix approach, we perform the same three steps as in the Full Einsum approach. The difference is that, as a first contraction, we usually convolve tensors $\mathcal{X}$ and $\frac{\partial \mathcal{L}}{\partial \mathcal{Y}}$ along a batch axis, and the schedule of the other contractions is further optimised. This provides complexity improvements when the batch size is large (which is the case in Transformer models,

where the batch dimension is the product of batch size by sequence length).

Algorithm 9 Backward pass (TTM layer, Autodiff).

Input: $\frac{\partial L}{\partial \mathcal{Y}}$; saved activations from forward $\mathcal{Y}_1, \dots \mathcal{Y}_M$

Output: $\frac{\partial L}{\partial \mathcal{X}}, \frac{\partial L}{\partial \mathcal{G}^1}, \dots, \frac{\partial L}{\partial \mathcal{G}^M}$

$\frac{\partial L}{\partial \mathcal{Y}_M} = \frac{\partial L}{\partial \mathcal{Y}}$

for k in $\{M, \dots, 1\}$ **do**

$\frac{\partial L}{\partial \mathcal{G}^k} = einsum(\mathcal{Y}_{k-1}, \frac{\partial L}{\partial \mathcal{Y}^k})$

$\frac{\partial L}{\partial \mathcal{Y}_{k-1}} = einsum(\frac{\partial L}{\partial \mathcal{Y}_k}, \mathcal{G}^k)$

end for

$\frac{\partial L}{\partial \mathcal{X}} = \frac{\partial L}{\partial \mathcal{Y}_0}$

Algorithm 10 Backward pass (TTM layer, Full Einsum).

Input: $\frac{\partial L}{\partial \mathcal{Y}}$; $\mathcal{X}$

Output: $\frac{\partial L}{\partial \mathcal{X}}, \frac{\partial L}{\partial \mathcal{G}^1}, \dots, \frac{\partial L}{\partial \mathcal{G}^M}$

▷ Results in the below for-cycle are computed only for the first batch during training and reused for others.

for k in $\{1, \dots, M\}$ **do**

Compose $einsum_k$ expression for $\frac{\partial L}{\partial \mathcal{G}^k}$

Optimize contraction schedule for composed $einsum_k$

end for

for k in $\{1, \dots, M\}$ **do**

$\frac{\partial L}{\partial \mathcal{G}^k} = einsum_k(\frac{\partial L}{\partial \mathcal{Y}}, \mathcal{G}^1, \dots, \mathcal{G}^M)$

end for

Backward with Full Einsum approach in the worst case has the same complexity as backward Autodiff.

Algorithm 11 Backward pass (TTM layer, Full Matrix).

Input: $\frac{\partial L}{\partial \mathcal{Y}}$; $\mathcal{X}$

Output: $\frac{\partial L}{\partial \mathcal{X}}, \frac{\partial L}{\partial \mathcal{G}^1}, \ldots, \frac{\partial L}{\partial \mathcal{G}^M}$

$\frac{\partial L}{\partial \mathbf{W}} = einsum(\frac{\partial L}{\partial \mathcal{Y}}, \mathcal{X})$ ▷ *einsum* here contracts only along batch dimension

▷ $FLOP_{\frac{\partial L}{\partial \mathbf{W}}} = O(BD_{in}D_{out})$

▷ $Memory_{\frac{\partial L}{\partial \mathbf{W}}} = O(D_{in}D_{out})$

▷ Results in the below for-cycle are computed only for the first batch during training and reused for others.

for k in $\{1, \ldots, M\}$ **do**

Compose $einsum_k$ expression for $\frac{\partial L}{\partial \mathcal{G}^k}$

Optimize contraction schedule for composed $einsum_k$

end for

for k in $\{1, \ldots, M\}$ **do**

$\frac{\partial L}{\partial \mathcal{G}^k} = einsum_k(\frac{\partial L}{\partial \mathbf{W}}, \mathcal{G}^1, \ldots, \mathcal{G}^M)$

▷ $FLOP_{\frac{\partial L}{\partial \mathbf{G}^k}} = O(D_{in}D_{out} \max_m(I_m, J_m)(\max_m R^m)^2)$

end for

4.4 Experiments

Thus, FLOP for backward pass using Full Matrix strategy is:

$$FLOP = O(BD_{in}D_{out}) + O(MD_{in}D_{out} \max_m(I_m, J_m)(\max_m R^m)^2) \tag{4.16}$$

When B is large (as it happens usually for Transformers, where B is batch size multiplied by sequence length) memory complexity of Full Matrix is better than for Full Einsum strategy as it doesn't depend on batch size B.

The memory footprints of each of these methods for TTM layers of rank 16 are in the Table 4.3. We select the most optimal pair according to memory and time - **Einsum Forward**, **Full Matrix Backward** - and employ it in TTM layer implementation.

Table 4.3: Time and memory footprints for different forward and backward strategies for TTM-16 layer.

Forward	Backward	Memory, Mb	Time, ms
Einsum	PyTorch Autodiff	1008	23.6
Einsum	Full Einsum	192	55.7
Einsum	**Full Matrix**	**192**	**17.5**
Fixed Schedule	PyTorch Autodiff	2544	58.4
Fixed Schedule	Full Einsum	192	84
Fixed Schedule	Full Matrix	192	125

4.5 Conclusion

This part introduces the representation of FC layers in a custom TTM format with proved compressibility. Taking into account the properties of TTM container, we establish a customised signal propagation strategy that maintains forward speed without creation of redundant activations in the backward direction. In addition, TTM layers can replace the FC layer in every Neural Network architecture in resource-restricted environments; it can be used to reduce the effective size of any Transformer-based models.

Chapter 5

Efficient GPT Model using TTM Decomposition

This Chapter is based on the paper "Efficient GPT Model Pre-training using Tensor Train Matrix Representation" (cf. Section "Publications" at page 4).

5.1 Introduction

With a sufficient amount of training data, big models usually outperform smaller models. The paper Kaplan et al. [2020] derives an empirical law for the dependence of the final loss on the model parameters, subject to a sufficiently large dataset and sufficient computing resources.

Large language models such as GPT-2, GPT-3 [Radford et al., 2019a, Brown et al., 2020] have the property of generality and show outstanding results in all areas of natural language processing. However, models with several billion parameters have difficulties in custom use. Training such models is associated with significant time costs, using a large amount of electricity and a carbon footprint [Zhang et al., 2022, Patterson et al., 2021]. Common approaches to large model compression, such as distillation [Sanh et al., 2019b], usually preserve the quality of a particular task while ignoring the generalisation property.

There are several approaches to simplify the employ of large language models. Some approaches focus on reducing the size of the entire model itself by reducing

the number of parameters [Sanh et al., 2019b, Michel et al., 2019] or the size of the weight representation [Hubara et al., 2016]. Other approaches focus on reducing the memory used in the training iteration [Hu et al., 2022]. Our TTM-based layers belong to the first group and aim at representing a model in compressed form.

In turn, this subset is divided into several approaches. Distillation trains a smaller version of the given model by aligning its output. Pruning reduces the size of the model by cutting out its modules, methods based on linear algebra get a compressed version of the model by getting low-rank representations of its layers.

We chose an approach based on a compressed representation of weights because it does not require changes in the training pipeline regarding the training of the main model, as in distillation, and it does not require the analysis of the effect of compressed layers on the output, as in efficient pruning.

There are several performance bottlenecks in models based on the Transformer architecture: the embedding layer, the layer that implements the attention mechanism, and fully connected layers, which usually hold about half of the memory allocated to train the model.

We reduce the size of the GPT language model by representing its fully connected layers with a custom TTM structure and train from scratch two types of model: based on GPT-2 small and based on GPT-2 medium. To make the GPT-2-based models easier to deploy, we replaced fully connected layers with sequential Tensor Train Matrix [Oseledets, 2010] containers, based on Tensor Train (TT) [Oseledets, 2011a] representation. The weight matrix is generally full-rank and cannot be approximated with low-rank objects. Therefore, we trained the model with custom TTM layers from scratch: thus, we were looking for the weights of the linear layer not among all matrices but among those represented in the TTM format. Then we study the behaviour of the pre-trained custom model on in-domain and out-of-domain language modelling tasks and several downstream tasks.

In this chapter, we provide a GPT-based model with up to 40% fewer parameters that show performance close to the original GPT in language modelling in the domain and outside the domain, GLUE benchmark, and text summarization.

5.2 Related Work

Several approaches explore ways to reduce the size of language models. The distillation mechanism [Hinton et al., 2015a] was applied to BERT [Sanh et al., 2019b] and GPT-2[1]. Open pre-trained transformers (OPT) [Zhang et al., 2022] provide a smaller model that mimics the behaviour of GPT-3 [Brown et al., 2020]. They employ more efficient training and use particular datasets to improve generalisation ability.

The Tensor Train Matrix (TTM) decomposition is an effective way to obtain low-rank representations of inner layers and is also used to reduce the number of parameters. Khrulkov et al. [2019] and Yin et al. [2021] reduce the size of the embedding layer using TTM. Novikov et al. [2015] uses the TTM format of linear layers to compress the computer vision models; however, TTM representations have not been tested before for generative Transformers.

5.3 Present FC Layer in TTM Format

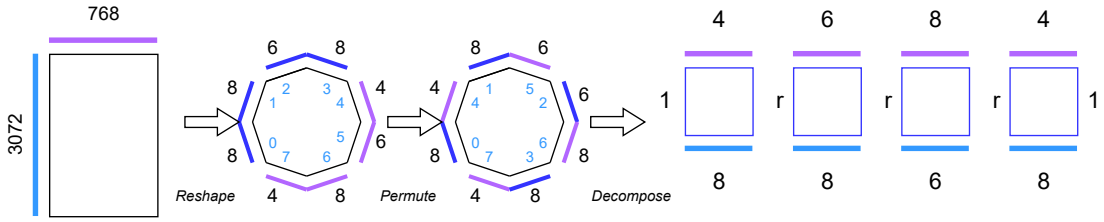


Figure 5-1: The scheme of 4-cores TTM representation of weight matrix in the GPT-2 small FC layer. The dimentions of the initial matrix are decomposed into 4 factors. The matrix is reshaped to these factors. Then the axis is permuted in such a way that the input and output dimensions are adjacent. The black digits indicate the size of the axes, and the light blue digits indicate their number.

To represent matrix $\mathbf{W}$ of shape $D_{in} \times D_{out}$ in TTM we should:

1. Reshape it to a tensor of shapes $I_1 \dots I_M, J_1 \dots J_M$ where $D_{in} = \prod_{m=1}^{M} I_m$ and $D_{out} = \prod_{m=1}^{M} J_m$.

[1] https://huggingface.co/distilgpt2

2. Permute the tensor axes so that the dimensions I_k, J_k became adjacents.

3. Represent the tensor in a TT (see section 2.2.3 for a detailed description of it) format.

Now the cores store only the parameters $R(I_1 J_1 + I_M J_M) + R^2 \sum_{m=2}^{M-1} I_m J_m$, the initial matrix $\prod_{m=1}^{M} I_m \prod_{n=1}^{M} J_n$.

Figure 5-1 presents the TTM-based layer scheme and the appropriate FC layer matrix. Purple and blue marks the dimensions of cores corresponding to the input and output sides of the initial weight matrix, respectively.

5.4 Experiments: End-to-end Training

We conducted experiments with a GPT-2 generative model on English-based datasets.

In standard GPT-2 architectures, we replaced fully connected layers with the TTM module, as it is described in Chapter 4, and trained the resulting models from scratch on the task of language modelling (LM). In this section, we examine the performance of the original model with TTM layers and also a GPT-2 compressed with SVD (with the same parameter budget as our model).

The general intuition of TTM layers superiority w.r.t. SVD is as follows: The TTM has been shown to be of full rank [Khrulkov et al., 2019], since the truncated SVD is a low rank method. Training the layers from scratch, we find a structure that defines weight matrices. The matrix $\mathbf{M} \in \mathcal{R}^{IJ}$ being restored from the TTM containers has rank $R_{TTM} = min(I, J)$, otherwise the matrix assembled from the SVD factors has a truncated rank $R_{SVD} < min(I, J)$

We can suggest that for matrices with a certain dimension:

- TTM is seeks a proper weight in a more comprehensive space by utilizing a set of full-rank matrices, which are more effective than a set of matrices with truncated ranks;

- A higher rank matrix can store more information than a matrix with the same dimensions but a lower rank.

5.4.1 Hyperparameter Selection

The proposed layer structure assumes two sets of hyperparameters: *TTM core shapes* and *TTM ranks*.

The matrix of the size (D_{in}, D_{out}) is represented in cores $\mathcal{G}^1 \in \mathbb{R}^{1,J_1,I_1,R_1}, \mathcal{G}^2 \in \mathbb{R}^{R_1,J_2,I_2,R_2}, \ldots, \mathcal{G}^M \in \mathbb{R}^{R_{M-1},j_M,i_M,1}$, where $D_{in} = \prod_{k=1}^{M} I_k, D_{out} = \prod_{k=1}^{M} J_k$, M - number of cores. The compression rate in a TTM layer is defined in the formula 4.4. Based on this equation, we state that for the maximum compression rate, the multiplication of shapes of $I_k * J_k$ should be as close to each other as possible among all M cores.

We choose $I_k * J_k$ in such a way that they are equal to each other and approximately equal to $(D_{in} * D_{out})^{1/M}$. Shapes selection is implemented with a custom algorithm, which will be presented in the source code. In our case, the dimension of GPT-2 small fully-connected layers [I, J] is [768, 3072]; $768 = 4 * 6 * 8 * 4$ and $3072 = 8 * 8 * 6 * 8$; $I_k * J_k$ are 8*4, 8*6, 6*8, 8*4.

As for the choice of ranks, we select them based on the desired compression of the entire model. For a small GPT, the compressions are from 50% to 90%. For a medium GPT, the reduction is 40%.

5.4.2 In-domain Language Modelling Task

To evaluate the in-domain performance on the LM task, we provide training and evaluation on the train and test partition of the same dataset, respectively. We replace the fully connected GPT-2-small layers with TTM of ranks 16, 32, 64, and 80. We train and validate the model with block size 512 on the Wikitext-103 dataset with approximately 100 mln tokens [Merity et al., 2016] for 40 epochs using the AdamW optimizer [Loshchilov and Hutter, 2019] and the Cosine warm-up scheduler [Goyal et al., 2017], increasing the training step from 0 to $2.5e^{-4}$. In this and subsequent experiments, we established the maximum learning rate point relative to the total number of training steps. Our goal was to ensure that the model reached its highest point and completed approximately 1/10 of the entire learning process. Table 5.1 shows that the resulting perplexity is comparable to the original model. However,

model compression has a negligible impact on quality within this domain. For example, a reduction in parameters greater than 30% only results in a half-percent decrease in perplexity, while a reduction greater than 40% leads to a drop in 3%. It should be noted that the Wikitext-103 dataset is not sufficiently large and can be used as a sandbox to evaluate the behaviour of the model.

Table 5.1: In-domain perplexities for GPT-2 small model, pre-training from scratch.

Model	Training	Validation	Number of parameters	% of classic GPT-2 size	Perplexity
GPT-2 small	Wikitext-103 train	Wikitext-103 test	124 439 808	100	17.55
GPT-2 small TTM-16	Wikitext-103 train	Wikitext-103 test	68 085 504	54	21.33
GPT-2 small TTM-32	Wikitext-103 train	Wikitext-103 test	71 756 544	57	21.06
GPT-2 small TTM-64	Wikitext-103 train	Wikitext-103 test	83 606 784	67	18.08
GPT-2 small TTM-80	Wikitext-103 train	Wikitext-103 test	107 698 944	86	17.61

5.4.3 Out-of-domain Language Modelling Task

In this setup, we perform validation on the Wikitext-103 test section while training the model on other datasets for the same language modelling task.

We train the GPT-TT model on a sufficiently large OpenWebText dataset [Gokaslan and Cohen, 2019], which contains approximately 8 million texts, imitates the WebText dataset and is publicly available. We train the model for 10 epochs with a similar optimiser and scheduler with a maximum learning rate $2.95 \exp -5$ and global batch size 340. When reaching the perplexity value of 50, we halved the batch size. We use an optimiser and scheduler as in the previous section, sequence length is 1024. The optimal parameters were chosen based on the perplexity in the validation part of the Wikitext-103 dataset of a small GPT-2 model with classical fully connected layers. After obtaining the optimal parameters for the classical model, the learning settings were fixed. The training process continued for approximately 20 days on 4 GPUs 3090ti. To receive a GPT-based model with a compatible size, we train from scratch under the same conditions the GPT-2 medium with linear layer replaced with SVD-structure layers with rank 50. As shown in Table 5.2, the best perplexity among the compressed models is related to OPT [Zhang et al., 2022] with 350 million parameters. Herewith, OPT saves 7% the full GPT-2, while TTM-72

Table 5.2: Out-domain perplexities for GPT-2 Medium and GPT TTM-72 models, pre-training from scratch.

Model	Training	Validation	Number of parameters	% of classic GPT-2 size	Perplexity
GPT-2 med	Webtext	Wikitext-103	354 823 168	100	20.56
GPT-2 TTM-72	Openwebtext	Wikitext-103	218 303 488	61	30.85
GPT-2 SVD-50	Openwebtext	Wikitext-103	220 920 832	62	55.46
Distill GPT-2	Openwebtext	Wikitext-103	81 912 576	23	51.45
OPT 350m	Openwebtext + BookCorpus + Pile [Gao et al., 2021]	Wikitext-103	331 196 416	93	24.75

saves 40%, and the perplexity decreases to 31. At the same time, an SVD-50 of a size similar to TTM-72 has perplexity 55, which is even worse than Distill GPT, the model with the smallest number of parameters.

Pre-trained models are available at Hugginface [2].

5.4.4 Natural Languge Understanding - GLUE

Table 5.3: Performance for GPT-2-based model on GLUE benchmark after one epoch fine-tining.

Model	STSB	CoLA	MNLI	MRCP	QNLI	QQP	RTE	SST2	WNLI	AVG
GPT-2 med	0.76	0.45	0.82	0.78	0.87	0.87	0.53	0.92	0.43	0.74
OPT 350m	0.73	0.32	0.81	0.78	0.88	0.86	0.56	0.92	0.39	0.69
GPT-2 TTM-72	0.77	0.23	0.79	0.80	0.61	0.86	0.47	0.82	0.56	0.66
GPT-2 SVD-50	0.73	0.08	0.78	0.68	0.84	0.84	0.57	0.89	0.43	0.64
DistilGPT	0.18	0.00	0.73	0.70	0.79	0.52	0.57	0.88	0.43	0.64

Table 5.4: Text Summarization Results.

Model	ROUGE-1	ROUGE-2	ROUGE-L
GPT-2 med	20.5	4.6	10.2
GPT-2 SVD-72	18.1	2.3	11.3
GPT-2 TTM-72	20.1	4.1	9.9

We take a pre-trained GPT TTM-72 model from the previous section (without fine-tuning) and validate it on a General Language Understanding Evaluation (GLUE) benchmark. It is a collection of nine natural language tasks, including language acceptability, sentiment analysis, paraphrasing and natural language inference. The evaluating script is based on the original Transformer repository [Wolf

[2] https://huggingface.co/s-nlp/

et al., 2020]. We add a top head compatible with the given task and run one training epoch. We choose just one epoch to avoid a situation where several models, all "large" concerning the number of tokens in the dataset but of different sizes relative to each other, converge to approximately the same loss during the entire training cycle [Kaplan et al., 2020]. We repeated these experiments 5 times with different random seeds, Table 5.3 shows the averaged obtained results with a standard deviation of no more than 0.0008. The classical models and models with TTM layers show approximately equal results, periodically overtaking each other. GPT-2 TTM-72 has a performance decrease in Acceptability and several Question-Answering datasets (QNLI, MNLI). The result of SVD-50 is close to TTM-72.

5.4.5 Text Summarization

We also compare the behaviour of the proposed models in the text summarization task when tuning on a small amount of data. Based on the pipeline from [Khandelwal et al., 2019], we trained both models on 3000 objects from the CNN/Daily Mail datasets [Hermann et al., 2015, Nallapati et al., 2016]. The obtained ROUGEs are not high (Table 5.4) but match the result from cited paper and highlight the similar behaviour of the classical GPT-2 and TTM-72. SVD-50 shows a bit worse outcome, except for the metric ROUGE-L.

5.5 Conclusion

We introduced an approach to obtain the compressed version of the GPT-2 model by representing its layer with its compressed analogue of a smaller number of parameters. We incorporate custom TTM layers as Fully Connected layers in a transformer-based GPT-2 architecture and evaluate it on Language modelling and Language Understanding tasks on English language. This modification results in a 40% reduction in model size, while maintaining performance on in-domain tasks without significant loss in quality. Furthermore, in out-of-domain tasks, our proposed model outperforms similar architectures that use SVD instead of Fully-Connected layers and training from scratch under the same conditions.

We compare our method with other approaches to compressing the effective model size: distillation (DistillGPT) and training cut version of the architechture from scratch (OPT model). Our model significantly surpasses DistillGPT, but the OPT model provides the best score. This trend continues in downstream tasks such as Language Understanding and Text Summarization, where the quality of our resulting model is lower than the original but superior to baseline compressed models. These results show that in case of training a compressed or cut version of the given model from scratch, the dataset and training setup play a more significant role than the choice of method. DistillGPT2 has a week setup; OPT has a strong setup and well-prepared dataset; our model employs a regular dataset and mimics the training setup of the original model.

Chapter 6

Efficient Question Answering using TTM Decomposition

This Chapter is based on the papers "Which is Better for Deep Learning: Python or MATLAB? Answering Comparative Questions in Natural Language", "Retrieving Comparative Arguments using Deep Language Models", "Retrieving Comparative Arguments using Ensemble Methods and Neural Information Retrieval" (cf. Section "Publications" at page 4).

6.1 Introduction

In this Chapter, we provide a detailed description of the comparative case of the Question-Answering problem. We have presented a demo version of the software that allows users to compare generative, retrieval-based and template-based approaches. Then we focus on the retrieval-based approach. More precisely, having a corpus with passages, we offer several options to extract the answer matching the particular question: methods based on statistics and techniques based on Ensembles of Trees. In addition, we employ strategies based on the Transformer architecture and aim to perform the information retrieval task (ColBERT). It provides a good result, but consumes a lot of memory. Finally, we compressed the FC layers in this model using SVD and TTM decomposition. For compression, we chose the layers with the sharpest SVD spectrum change within the SVD and TTM decomposition; the same

spectral analysis determines the rank for the TTM. This choice of layers allowed one to significantly improve the quality obtained by compressing the model, relative to the random selection of modules for compression.

6.2 Overview of Comparative Question-Answering Methods

In this section, we compare three approaches of a Comparative Question Answering(*Is X better than Y with respect to Z?*). Answering such questions in natural language is important for assisting humans in making informed decisions. These approaches are based on modern NLP methods: linguistic transformer type models and models for extracting relevant texts from corpora. For ease of comparison, we have created a system. The key component of our system is a natural language interface for comparative QA that can be used in personal assistants, chatbots, and similar NLP devices. Comparative QA is a challenging NLP task, since it requires collecting supporting evidence from many different sources, and direct comparisons of rare objects may not be available even on the entire Web.

Comparison of objects of a particular class (e.g. holiday destinations, mobile phones, programming languages) is an essential daily task that many individuals require every day. According to Bondarenko et al. [2020a], comparative questions comprise around 3% of search question queries submitted to major search engines—a non-negligible amount. Answering a comparative question (*What is better, X or Y?*) requires collecting and combining facts and opinions about compared objects from various sources. This challenges general-purpose question answering (QA) systems that rely on finding a direct answer in some existing datasets or extracting from web documents.

Nowadays, many websites (e.g. Diffen, WolphramAlpha, or Versus) provide users with a comparison functionality. Furthermore, the task of answering comparative questions has recently attracted the attention of the research community [Kessler and Kuhn, 2014, Arora et al., 2017, Yang et al., 2018b]. Most of the current researches suggest that an answer to a comparative question should not only indicate

the "winner" of a comparison, but also provide arguments in favour of this decision and arguments that support the alternative choice.

Therefore, we argue that a comparative QA system should be a combination of an argument mining engine and a dialogue system that mimics a human expert in the field. In this work, we make the first step towards the development of such technology. Namely, we develop a Comparative Question Answering System (CoQAS), an application that consists of a Natural Language Understanding (NLU) module that identifies comparative structures (objects, aspects, predicates) in input questions and a Natural Language Generation (NLG) module that constructs an answer. We tested various options for both NLU and NLG parts ranging from a simple template-based generation to Transformers-based language models.

The main contributions of our work are threefold: (i) we design an evaluation framework for comparative QA, featuring a dataset based on Yahoo! Answers; (ii) we test several strategies for identification of comparative structures and for answer generation; (iii) we develop an online demo using three answer generation approaches. A demo of the system is available online.[1] Besides, we release our code and data.

Text Generation. Most of the current text natural language generation tasks [Dušek and Jurčíček, 2016, Freitag and Roy, 2018] are based on the architecture of sequence-to-sequence models [Sutskever et al., 2014], these existing generation methods are developed employing the attention mechanism [Bahdanau et al., 2015] and the pointer-generator network [See et al., 2017]. More recent work on text generation focus on generating natural language using multitask learning from multi-document or multi-passage sources [Hsu et al., 2018, Nishida et al., 2019]. However, our generation task uses a list of arguments to build the final answer. This makes it similar to unsupervised summarization. There exist several approaches for tackling the latter task, e.g. graph-based [Litvak and Last, 2008] and neural models [Isonuma et al., 2019, Coavoux et al., 2019]. A common approach to the summarization task is based on TextRank graph algorithm [Mihalcea, 2004, Fan and Fang, 2017].

Comparative QA. According to Li and Roth [2002] questions can be divided into 6 coarse and 50 fine-grained categories, such as factoid questions, list ques-

[1] https://skoltech-nlp.github.io/coqas

tions or definition questions: we focus on comparative questions. Sun et al. [2006] proposed one of the first works on automatic comparative web search, where each object was submitted as a separate query, then system obtained an answer and compared the obtained results. Opinion mining of comparative sentences is discussed by Ganapathibhotla and Liu [2008] and Jindal and Liu [2006], yet with no connection to argumentation mining. Instead, comparative information needs are partially satisfied by several kinds of industrial systems mentioned above. Schildwächter et al. [2019] proposed Comparative Argumentative Machine (CAM)[2], a comparison system based on extracting and ranking arguments from the web. The authors have conducted a user study on 34 comparison topics, showing that the system is faster and more confident at finding arguments when answering comparative questions in contrast to a keyword-based search. Wachsmuth et al. [2017] presented args.me system for retrieval of pros and cons (arguments) given some input statement. The input of this system is not structured, but rather a query in a free textual form. The Touché shared task on argument retrieval at CLEF [Bondarenko et al., 2020b] featured a related track. The task was to retrieve from a large web corpus documents answering comparative question queries like "What IDE is better for Java: NetBeans or Eclipse?", which is similar to CAM and args.me.

6.2.1 System Design

Our system is designed to help the user make a proper choice by fully and reasonably describing the possible advantages and disadvantages of each of the matching options. For this purpose, we have defined structures that contain significant information about the desired comparison: compared *objects*, comparison *aspects*, and *predicates*.

In the example "Which is better for Deep Learning: Python or MATLAB?", the objects are entities that the user wants to compare (*Python, MATLAB*). The predicate is the entity that frames the comparison (*better*); it introduces a comparison relation between the objects and is often represented by a comparative adjective or adverb. Finally, the comparison aspects are shared properties along which the two

[2] https://ltdemos.informatik.uni-hamburg.de/cam

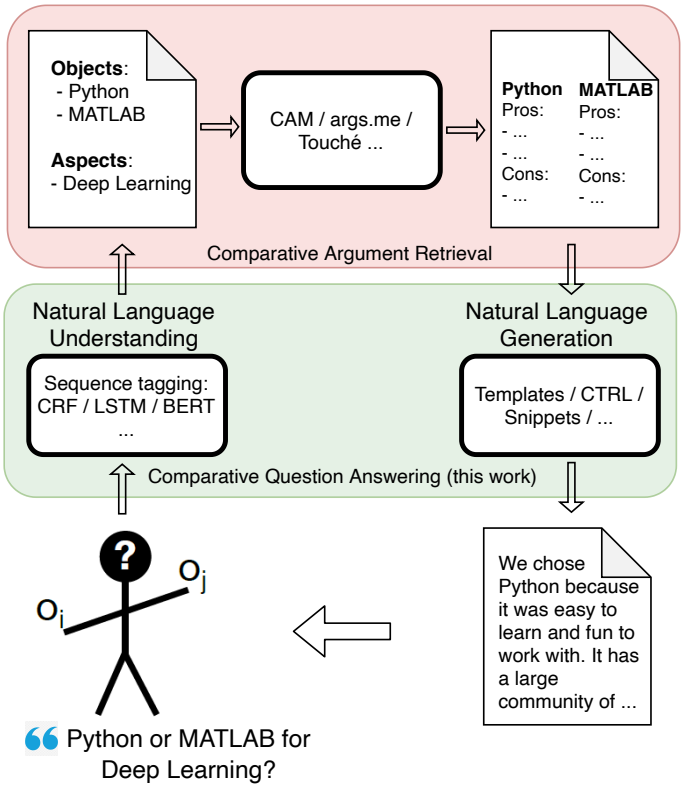


Figure 6-1: The comparative QA workflow. A user submits a comparative question, the NLU module identifies compared objects and aspects and transfers them to CAM to retrieves comparative arguments. Then, the NLG module represents the arguments in textual form.

objects are compared, e.g. *Deep Learning*.

Our comparative question answering system is based on CAM [Schildwächter et al., 2019], which retrieves pro/con arguments for a pair of compared objects. We extend CAM by enabling it to process natural language questions and generate coherent human-like answers as depicted in Figure 6-1.

CAM mines sentences in favour or against two compared objects with respect to an aspect specified by the user. First, using the Elasticsearch BM25, CAM retrieves sentences containing the two compared objects and the comparison aspect from the Common Crawl-based corpus featuring 14.3 billion sentences [Panchenko et al., 2018]. Then, CAM classifies the sentences as comparative or not and identifies the "winner" of the two compared objects in the sentence context. Besides, it extracts aspects and predicates from the retrieved comparative sentences [Panchenko et al., 2019]. Finally, CAM outputs a list of argumentative pro/con sentences and shows

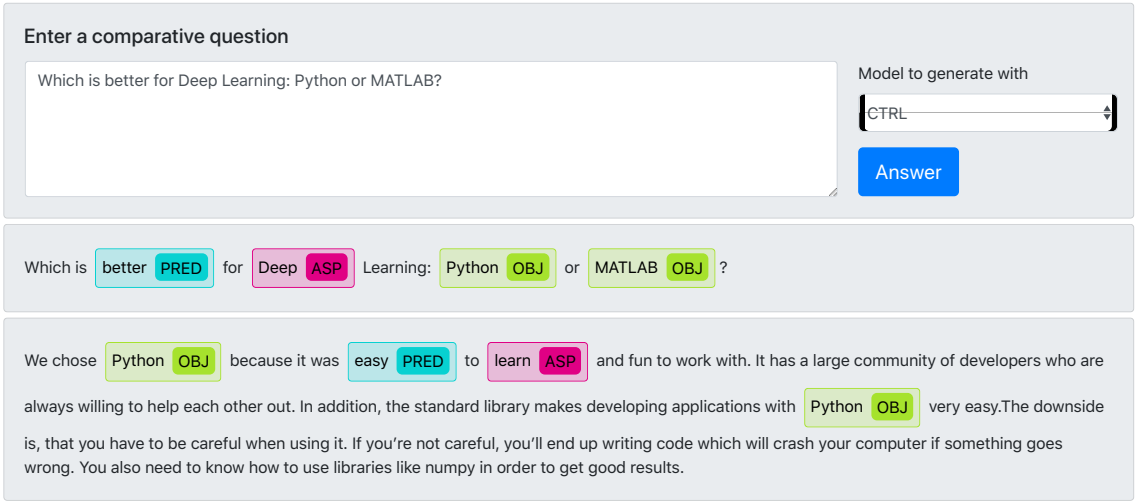


Figure 6-2: The interface of the Comparative Question Answering System (CoQAS).

the "winner" of the comparison along with the comparison aspects.

We extend CAM with natural language question understanding (described in Section 6.2.2) and natural language answer generation (described in Section 6.2.3) modules. The first module is developed to automatically identify the compared objects and the comparison aspect in a user-provided natural-language comparative question. This information is passed to CAM which queries DepCC for comparative sentences. The NLG module receives the output of CAM and transforms the retrieved argumentative sentences into a short text, the generated answer. The structure of our modular system is presented in Figure 6-1.

The user interface (Figure 6-2) contains an input form for submitting a comparative question, and an output box for a generated answer. To improve readability of the answer and help find the arguments in it, NLU module also labels the output with identified objects, aspects and predicates. In Figure 6-2, we present an example of the the input-output system's web interface in action.

In the NLG module we use several approaches to response generation: a retrieval-based approach and approach built upon pre-trained language models. These techniques provide different answers: the first is more structured, and the second one is based on experience and opinions. Therefore, we allow user to choose a generation model from different types: CAM, CTRL, and Snippets (cf. Figure 6-2).

Finally, for integration into NLP applications, e.g., personal assistants and chat-

Table 6.1: Statistics of the NLU dataset.

	Obj	Asp	Pred
# occurrences	7,555	2,593	3,990
# per sentence	2.51	1.35	1.34
Length (words)	1.04	1.37	1.16

bots, we also provide a RESTful API for our comparative QA.

6.2.2 Natural Language Understanding

The goal of NLU module is to identify the objects to compare and comparison structure aspects and predicates, if they were specified. We cast this as a sequence labelling task.

Training Dataset To train the NLU component, we created *Comparely*, a dataset with comparative sentences manually labeled with objects, aspects, and predicates. First, we extracted comparative sentences for 270 object pairs from the dataset of (not) comparative sentences by Panchenko et al. [2019]. We extracted them from DepCC corpus [Panchenko et al., 2018] using CAM. We then performed manual labelling (two annotators) using WebAnno tool [Yimam et al., 2013]. Some of the extracted sentences were not comparative, so the annotators were instructed to discard them. The majority of sentences were labelled once, but we also labelled 200 of them multiple times to compute the inter-annotator agreement. The Cohen's κ for the aspect labelling is 0.71 (substantial agreement). For predicates and objects the values are 0.90 and 0.93, respectively—perfect agreement. The dataset consists of 3,004 sentences, each of them has a comparison of two or more distinct objects and at least one aspect or predicate. The average length of sentence is 26.7 words (Table 6.1). The majority of sentences compare more than one pair of objects across multiple parameters (i.e. sentences often contain more than one aspect or predicate). As the NLU processed not statements but questions, for the further improvement of the dataset we could use comparative questions from [Bondarenko et al., 2020a].

This dataset is essentially similar to the ones by [Arora et al., 2017, Kessler and Kuhn, 2014]. They also contain comparative statements labelled with objects, as-

pects, and predicates. The primary difference of our dataset is the domain diversity. The mentioned datasets contain the sentences of only one domain, namely, the camera reviews. The information contained in such sentences is difficult to generalise. Thus, they demonstrate a proof of concept rather than a resource which can be used for real-world tasks. On the other hand, Comparely features objects of different domains. It was created based on real-world objects which are often compared. It contains data from three domains: brands, generic objects, and computer science. The two former domains are more numerous: 41% and 46% of sentences deal with objects of brands and generic domains, respectively. The rest 13% are devoted to objects of the computer science domain.

Method Identification of comparative question components (objects, aspects, predicates, or none) is a sequence-labelling task, where the classifier should tag respective tokens in an input question. We test several common baselines starting with simple one-layer bidirectional LSTM described by Arora et al. [2017] where the input is encoded with GloVe embeddings. For some further improvements, we add Conditional Random Field [Sutton and McCallum, 2012] to LSTM and use context-based ELMO [Peters et al., 2018] embeddings for token representations. We also experiment with Transformers [Vaswani et al., 2017c] using a pre-trained BERT model [Devlin et al., 2019] and RoBERTa [Liu et al., 2019].

For every classifier, during training, we tune hyperparameters by varying a batch size (from 16 to 100) and a learning rate (from 10^{-6} to 10^{-2}). To find a proper converge of training process, we apply two types of learning rate schedulers: Linear With Warmup and Slanted Triangular.

For the model with the highest achieved F1 (RoBERTa), we employ stochastic weight ensembling [Goodfellow and Vinyals, 2015, Garipov et al., 2018], i.e., we interpolate between the weights obtained by training a certain model with different random seeds. All models were trained on the *Comparely* dataset and tested on its manually re-labelled subset of 400 sentences. The overview of the classifiers' effectiveness is shown in Table 6.2.

Table 6.2: Evaluation in terms of F1 of the NLU tagger.

Model	Objects	Aspects	Predicates
RoBERTa	**0.925**	**0.685**	**0.894**
BERT	0.829	0.563	0.869
ELMO	0.654	0.487	0.825
BiLSTM-CRF	0.631	0.475	0.766
BiLSTM	0.582	0.328	0.730

Results and Discussion The evaluation shows that comparison aspect classification is the hardest task: the baseline one-layer BiLSTM achieves an F1 score equal to 0.33, and the most effective RoBERTa-based classifier achieves score 0.69. The most reliable classification was achieved for predicting the compared objects with an F1 0.58 for the baseline, and an F1 0.93 for RoBERTa. Adding Conditional Random Fields and deploying special ELMO embedding to the BiLSTM classifier slightly improved the results. Transformers demonstrated significant improvement in classification effectiveness over the baseline. Finally, we choose to deploy RoBERTa-based classifier in the NLU module of our system.

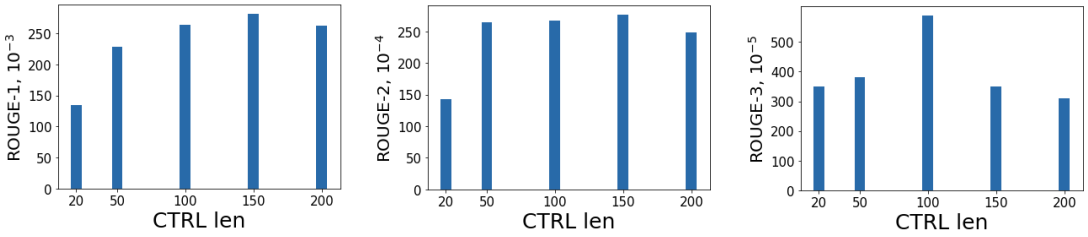


Figure 6-3: Dependence of ROUGE metrics on the maximum length of the generated sequence (CTRL model).

6.2.3 Comparative Answer Generation

Based on comparative sentences retrieved by CAM, we develop several generation approaches to construct a human-like concise answer: (1) generation with pretrained Transformers-based language models, (2) retrieval of argumentative sentences ranked by CAM or TextRank, (3) extracting context of sentence retrieved by CAM as support for the "winning" object, and (4) entering extracted comparative structures in templates.

Table 6.3: Evaluation of generation methods on the Yahoo! Answers. The best models of each type are highlighted.

Method	Type	ROUGE-1	ROUGE-2	ROUGE-3
CTRL: Question	Language Model	0.2423	**0.0226**	**0.0023**
CTRL: Which-better-x-y-for-z	Language Model	**0.2454**	0.0200	0.0021
CAM: First snippets	Doc.Retrieval	**0.2162**	**0.0167**	**0.0017**
CAM: Bullet points	Sent.Retrieval + Slots	**0.2298**	**0.0328**	**0.0040**
TextRank: Bullet points	Sent.Retrieval + Slots	0.2203	0.0238	0.0036
Templates	Object/Aspect Slots	0.1969	0.0195	0.0016

Generation Methods

Pre-trained Language Models Pre-trained language models have been shown to contain commonsense knowledge, so they can be successfully used for question answering [Andrews and Witteveen, 2019] and for generating sensible and coherent continuation of text. Therefore, we use Transformers-based CTRL [Keskar et al., 2019] models for answering comparative questions.

It should be noted that the research has been done back in the 2019 year and now with more powerful conditional generative models than CTRL (i.e. ChatGPT[3]), better results in Comparative QA can be obtained.

CTRL allows explicit control codes to vary the domain and the content of the text. We use the Links control code which forces the model to produce text similar to online news and reports. We feed into CTRL phrase "Links Which is better in respect to aspect: object_1 or object_2?" and a row question from the input.

We also vary a maximum number of tokens, generated by CTRL. We experiment with different length set, including: 20, 50, 100, 150, and 200 and generate answers to questions from the Yahoo! Answers dataset (cf. Section 6.2.4). For the evaluation part, we calculate ROUGE-1, ROUGE-2, ROUGE-3 scores between generated texts and corresponding Yahoo!'s "best answers". According to the results (cf. Figure 6-3), a model with maximum length of 100 tokens gives the highest ROUGE-3 score (we select this length parameter for our further experiments).

[3] https://openai.com/blog/chatgpt

Sentence-Retrieval-Based Methods The CAM output contains a list of the argumentative sentences which are ranked by the BM25 inverted index-based score. Every sentence is a supportive argument for the superiority of the respective compared object. Sentence-retrieval-based methods try to extract the most representative sentences and display it in the proper form. To create an answer, CAM: Bullet points mentions a "winner" defined by CAM with respect to aspects if they exist. It also takes the top-3 sentences supporting each of the objects and produces a list for highlighting the advantages and disadvantages of each object in comparison.

An alternative way of retrieving the most relevant sentences is clustering. This approach is used in TextRank: Bullet points. TextRank is a graph-based summarization algorithm. We use the version proposed by [Mallick et al., 2019]. We represent sentences with hidden states of a LSTM network pre-trained on Wikipedia. TextRank iteratively updates the weights of edges and sets the node weights to be proportional to the importance of adjacent edges. To make the graph sparse, we remove the edges with a score below average.

We create separate graphs for sentences supporting each of the objects. We apply TextRank to each of them and then cluster them. Clustering divides the nodes in graphs by semantic similarity and thus allows identifying groups of sentences supporting a particular idea. Then, we apply TextRank again to each of the clusters separately and select the three most characteristic sentences from each cluster as produced by Chinese Whispers [Biemann, 2006], an iterative clustering algorithm, which assigns vertices to the most common class among their neighbours. Argumentative sentences selected in this way are displayed as a bullet-list after declaring the "winner" object of comparison.

Document-Retrieval-Based Method To compose an answer, method CAM: First snippets takes the first sentence related to the "winner" object in CAM output. Then it finds a document corresponding to this sentence and extracts the surrounding context. The obtained context consists of 3 sentences and is considered to be a system answer.

Template-Based Answer Besides the argumentative sentences, CAM extracts aspects and predicates from them. The predicates are adjectives or adverbs, which allows using templates of the following form: "I would prefer to use $Object_1$ because it is $Predicate_1$ and $Predicate_2$. In addition, it is $Predicate_3$, ..., and $Predicate_i$. However, you should also take into account that $Object_2$ is $Predicate_{i+1}$, ..., and $Predicate_k$". Here $Object_1$ is the winner of comparison.

6.2.4 Experiments

Evaluation Dataset To evaluate answer generation module of our system, we use information extracted from Yahoo! Answers. Namely, we get a subset of L6– Yahoo! Answers Comprehensive Questions and Answers version 1.0 (multi-part) retrieved from Yahoo! Webscope. We take pairs of objects that we used for generating *Comparely* and extract a subset of questions from the Yahoo! Answers dataset which contains these objects, yielding 1,200 questions. Additionally, we extract the answers to these questions, which are labelled by users as "best answer", and use them to evaluate our NLG methods.

Evaluation Metric Generated and reference texts are usually compared by number of matched N-grams: BLUE (precision), ROUGE (recall), METEOR (F-score). For all-round representation of the texts similarity, we select F1 score from ROUGE-N outputs as evaluation metric. We evaluate our generation models on the Yahoo! Answers dataset using the "best answer" (defined by users) as the reference.

Discussion of Results Evaluation results are provided in Table 6.3. CTRL models receive the highest ROUGE-1 scores, that describe overlapping of single words, and CTRL's high performance relatively to it can be explained by the fact that the pre-trained language model stores information about a vast dictionary and, with some probability, yields the words that are placed in the standard answer. While the systems generate grammatically correct texts they may not necessarily satisfy the information need of the user. For example, the CTRL answers the question "What should I eat an orange or an apple?" with "It is simple: eat what you like

Table 6.4: User study results for answer completeness and fluency (30 questions, 3-point Likert scales).

	Answers a question (%)			Answer fluency (%)		
Method	Complete	Partial	Does not	Complete	Partial	Not fluent
Yahoo! Best Answer	62	28	10	86	6	8
CTRL: Question 100	30	37	33	80	12	8
CAM: Bullet points	28	58	14	22	48	30
CAM: First snippets	23	49	28	27	38	35

and don't worry about it."

Despite having low ROUGE-1, sentence retrieval-based approaches (Text Rank: Bullet points, CAM: Bullet points) have consistently higher ROUGE-2 and ROUGE-3. The generated answers are more structured and built on sentences marked by the system as comparative. They often contain special 2-gram and 3-gram sequences which are typical for an explanation.

Answers from CAM: First snippets, consisting of a single comparative sentences only, perform worse on all metrics. Interestingly, CAM: Bullet points has better performance than TextRank: Bullet points. It could indicate that modelling relevance by a standard index provides more accurate results than clustering. Meanwhile, template-based generation performs poorly. This indicates that the grammatical structure is essential for the answer generation task.

We choose 50 random sentences from the Yahoo! Answers dataset and calculate ROUGE-N scores for every generation method and Yahoo!'s "best answers". For each group of methods we select one providing the best result—CTRL: Question 100, CAM: First snippets and CAM: Bullet points—and add them to the system demonstration engine.

User Study

To additionally evaluate the proposed answer generation methods, we also collect human assessments in a small user study for the three models with the highest ROUGE scores (CTRL: Question 100, CAM: Bullet points, and CAM: First snippets).

Experimental Setup For our study, we randomly sampled 30 comparative questions from the Yahoo! Answers dataset and generated answers using three methods: CTRL: Question 100, CAM: Bullet points, and CAM: First snippets. Additionally, since we used Yahoo!'s "best answers" as ground truth for automatic evaluation, we asked our participants to also assess the quality of the human "best answers". For the user study, we internally recruited five (under-)graduate students. We focused on the two answer evaluation criteria: (1) Whether an answer is complete ("Does it answer the question?") and (2) how fluently it is written. The 120 question–answer pairs (3 generated answers and Yahoo!'s "best answer" for 30 questions) were randomly ordered and the participants had to rate the answer completeness and fluency on a three-point Likert scale (3: fully answers/fluent, 2: partially answers/fluent, and 1: does not answer/not fluent at all).

Results and Discussion The inter-annotator agreement shows a slight overall agreement between the five annotators (Fleiss' $\kappa = 0.20$ for answer completeness and $\kappa = 0.13$ for fluency) such that we decided to increase the reliability by calculating the κ-scores for all combinations of three or four annotators. We then decided to include only the three participants with the highest agreement ($\kappa = 0.32$ for answer completeness and 0.30 for fluency; both fair agreement) and to remove the two "outlier" participants from the study.

Table 6.4 summarizes the study results as the ratio of votes collected from the three annotators (we cannot use majority voting since about 60% of the question–answer pairs do not have a majority vote). Not surprisingly, the human-written answers are perceived as the most complete and fluent. The participants were almost equally satisfied with the answers generated by CTRL: Question 100 and CAM: Bullet points, however, they rated the CTRL answers as much more fluent. Interestingly, the relatively low inter-annotator agreement might indicate that humans have different perceptions of answer completeness and fluency (even some "best answers" were rated as incomplete and not fluent). For completeness, we calculated the statistical significance of the user study results using Bonferroni corrected p-values. For the pair CTRL: Question 100 (our best NLG model) and the Yahoo! Best

Answer: $p \ll 0.05$ for the answer completeness and $p \gg 0.05$ for the answer fluency. For the CTRL model, Pearson's $r = 0.121$ between the answer completeness and fluency (small correlation), and for the "best answers", $r = 0.407$ (medium correlation). The results show that our proposed system is almost as fluent as the human-written answers, but still needs some improvement in terms of adequacy.

Summarizing, we present a comparative question answering system targeted at answering comparative questions, such as "Is X better than Y with respect to Z?". Our system is based on the Comparative Argument Mining (CAM) system—a tool which retrieves from a large corpus textual comparative arguments for two to-be-compared objects. We extend CAM with an NLU module that identifies objects and aspects in a user textual query and highlights them in the answer, and a generation module that gives a concise and coherent answer based on the retrieved information. Evaluation of generation methods showed that a CTRL-based answer generation gives better performance with respect to ROUGE-1, and Sentence Retrieval Methods provide superior ROUGE-2 and ROUGE-3 scores.

6.3 Comparative Information Retrieval

6.3.1 Retrieving Comparative Arguments using Ensemble Methods and BERT

In this section, we present a submission to the Touché lab's Task 2 on Argument Retrieval for Comparative Questions [Bondarenko et al., 2021]. Our team Katana supplies several approaches based on decision tree ensembles algorithms to rank comparative documents in accordance with their relevance and argumentative support. We use PyTerrier [Macdonald et al., 2021] library to apply ensembles models to a ranking problem, considering statistical text features and features based on comparative structures. We also employ large contextualized language modelling techniques, such as BERT [Devlin et al., 2019], to solve the proposed ranking task. To merge this technique with ranking modelling, we leverage neural ranking library OpenNIR [MacAvaney, 2020].

Our systems substantially outperforming the proposed baseline and scored first in relevance and second in quality according to the official metrics of the competition (for measure NDCG@5 [Wang et al., 2013] score). Presented models could help to improve the performance of processing comparative queries in information retrieval and dialogue systems.

In this part of work, we use ensemble methods based on mixed statistical and comparative features to the document ranking; we are first to use neural information retrieval approach to the task of argument retrieval; we propose a model outperforming the baseline and yielding the first and the second-best result according to the relevance and quality metric, respectively.

The most relevant to this is the previous shared task Touche 2020 [Bondarenko et al., 2020c]. 17 participants took part in the competition and submitted 41 runs. Various approaches were tested by these participants, including methods based on extraction of structures corresponding to claims and premises, assessing argument quality, representation of documents by language models, expansion of the query by similar words. The ranking function from search engine ChatNoir [Potthast et al., 2012] based on BM25F [Robertson et al., 2004] approach was used as a baseline.

Only a few of the submitted solutions can slightly improve the baseline. The best overall approach in the previous competition was the method based on query extension and reranking documents by relevance, credibility, and supportive quality [Abye et al., 2020].

This work is based on our run submitted in the previous version of the Touche shared task [Chekalina and Panchenko, 2020]. In this work, we used a pre-trained language model to find relevance between the query and document. Extraction of comparative structures and counting the number of comparative sentences in a document help us to assess the quality of relevant arguments.

Therefore, the problem of argument retrieval arises in other scenarios. Comparative argumentation machine CAM [Schildwächter et al., 2019] retrieves comparative sentences with respect to accepted objects and comparison aspects. The paper [Fromm et al., 2019] explores the influence of context on an argument detecting system and proves the performance increasing related to it.

Table 6.5: Example of query and documents with different relevances in the Touche task dataset

Query	Document	Rank
What is better for the environment, a real or a fake Christmas tree?	Disease and condition content is reviewed by our medical review board real or artificial? There is so much confusing information out there about which is better for your health and the environment.	2
	You may think you're saving a tree, but the plastic alternative has problems too. Which is "greener" an artificial Christmas tree or a real one?	1
	This entry is part 25 of 103 in the series eco-friendly friday november 28th's tip christmas trees: stuck between choosing a real Christmas tree or a fake one?	0

Table 6.6: Example of query and documents with different relevances in the Antique dataset

Query	Document	Rank
Why do we put the letter k on the words knife and knob, knee?	They are saxon words. Knife would have been pronounced ker-niff.	4
	As a guess I would say that historically "kn" would have been pronounced differently to "n" and that time has altered the way the words are pronounced.	3
	Because English is a funny language.	2
	I don't really (k)now!	1

Datasets and Evaluation setup The organisers provided 50 comparative questions (topics), for which we should obtain documents containing convincing arguments for or against one or another option. The competition topics are available online. [4]

In addition, 50 topics and the corresponding relevance annotations of the previous year's competition [Bondarenko et al., 2020c] were given for supervised learning. These documents were also retrieved from ChatNoir and ranked manually to 0 (not relevant), 1 (relevant) or 2 (highly relevant) scores. We use this data to train and set up models based on the decision trees and fine-tune the BERT ranker. Also, last year's team submissions were available too.

Unfortunately, these data are not sufficient for fitting large supervised ranking

[4] https://webis.de/events/touche-21/shared-task-2.html

models, for example, based on the BERT technique. In this case, we use adjacent question-answering dataset called Antique [Macdonald and Tonellotto, 2020]. This dataset consists of the questions and answers of Yahoo! Webscope L6 and contains 2,626 open-domain non-factoid questions and 34,011 manual relevance annotations.

The example of query and ranked answers are in Table 6.5, Table 6.6 in Appendix A. It might be noticed that Antique dataset has a different set of ranking scores - 0, 1, 2 instead of 1, 2, 3, 4 - so we rewrite the Antique ranks according to the following mapping 1→0, 2→1, 3→1, 4→2.

We use every topic as a query in ChatNoir [Potthast et al., 2012] search engine and extract up to 100 unique documents from the ClueWeb12 corpus. We clean documents' bodies from HTML tags and markups and ranked them using one of the developed approaches described below.

As auxiliary data, the organizers provided the topics of the previous year's competition. For each proposed topic, a set of documents from ChatNoir was retrieved and labelled as described above. We use this data to train developing models and valid composing approaches. In the validation phase, we split the ranked data into 40 topics in train and 10 in validation.

In the run phase, we execute produced solutions on web evaluation platform Tira [Potthast et al., 2019]. In this stage to fit the model we use ranked data from the previous year entirely and predict rank for current proposed topics. The runs were evaluated using the NDCG [Wang et al., 2013] metrics based on the human judgements of the submitted runs. Retrieved documents were judged in accordance with two criteria: (i) document relevance, (ii) whether sufficient argumentative support is provided [Braunstain et al., 2016].

Document ranking using ensembles of trees In this section, we use ensembles of trees as a supervised machine learning technique to solve ranking problems. We choose either pointwise regression tree algorithms, like Random Forest, or boosted tree algorithms like XGBoost and LightGBM. In the cases of LightGBM model we employ LambdaMART [Wu et al., 2010] objective. It combines cost function derived from minimizing the number of inversions in ranking (LambdaRank [Burges et al.,

2007]) and objective for building gradient boosted decision trees (MART [Friedman, 2002]). We use PyTerrier platform for information retrieval.[5] It simplifies the extraction of the text features and allows expressing retrieval experiments [Macdonald and Tonellotto, 2020].

For our ranking ML methods, we use features that came from 3 origins described below: (i) ranking features extracted by PyTerrier, (ii) specific comparative features, (iii) score from ChatNoir system based on custom BM25 scoring function.[6]

PyTerrier provides a measure of the matching of query-document texts by several models. Among these models are statistical measures (TF-IDF), mesures based on language models (Heimstra, Diriclet), measures based on occurrence of a document depending on the fields that the term occurs in (BM25F, PLF). The list of all possible models are available at the cite [7]. Among these varieties we have chosen BM25, Heimstra, DFIC, DPH, TF-IDF, DiricletLM, PL2 for our exploration.

Table 6.7: Results on validation set for text features in PyTerrier models.

Method	**BM25**	Heimstra	**DFIC**	DPH	**TF-IDF**	DiricletLM	PL2
NDCG@5	**0.3637**	0.3616	**0.3642**	0.3110	**0.3637**	0.3307	0.3603

We applied each of the selected methods sequentially and independently to the training set, ranked documents by the obtained scores and evaluated the ranking on the validation set. The result of these tests is in Table 6.7. We have chosen 3 methods with the most promising results, and these 3 methods combine 3 features.

We focus not only on finding high relevant documents as on finding documents with a comparison of one object relative to another. The work [Chekalina et al., 2021] assumes that the comparative issue can be represented by comparative structures - objects for comparison, comparative aspects and predicates. We take the sequence-labelling model suggested in the cited paper and applied it to the query. It helps us to define objects for comparison for every topic. Then we apply the model to document and get a comparative feature set.

[5] https://pyterrier.readthedocs.io/en/latest/index.html

[6] https://www.elastic.co/guide/en/elasticsearch/reference/current/index-modules-similarity.html

[7] http://terrier.org/docs/current/javadoc/org/terrier/matching/models/package-summary.html

The feature `is_retrieved` describes are there any comparative structures in the document at all. Characteristic `objs_score` defines how many objects from query are found in document (0, 1 or 2). Feature `asp_pred_score` is counted in the following way: if at least one object from a query is in the document, every word in the document labelled as an aspect or predicate increases the score to 0.5. Finally, we combined defined features with scores obtained from the ChatNoir system, and a resulting feature vector for pair query-document is {`score_pl2`, `score_tf`, `score_bm`, `score_dfic`, `baseline_scores`, `is_retrieved`, `ap_score`, `objs_score`}.

Models

Random Forest We use the Random Forest model imported from Sklearn and wrapped by the PyTerrier pipeline. To find the best setup, we vary the number of estimators from 10 to 150, the value 20 gives the best valid score NDCG@5 of 0.408.

XGBoost We also wrapped gradient boosting library from Sklearn to PyTerrier class and tune hyperparameters by setting the learning rate from 1e−4 to 0.1 and max_depth from 4 to 16. The best setup is learning rate 0.01, max_depth 6 and gives NDCG@5 0.547.

LightGBM In the case of LightGBM, we vary the number of leaves from 8 to 20 and the learning rate from 0.001 to 0.1. The best configuration with num_leaves equal to 15 and a learning rate equal to 0.1 gives 0.579 score.

Table 6.8: Feature importance in the proposed LightGBM model

Feature	Pl2	TF-IDF	BM25	Dfic	ChatNoir	has comp	objs_score	asp_pred
Importance	1.76	1.19	1.51	2.3	20.8	0	1.66	1.51

The feature importance of the resulting model is in Table 6.8. It can be seen that the most significant feature is the score retrieved from the ChatNoir, then there is a Divergence from Independence based on Chi-square [Kocabas et al., 2013] and the existence of comparison objects in the document.

Document ranking using neural information retrieval based on BERT Contextualized language models such as BERT can be much more efficient for ranking tasks because they contain vast relationships between language units. In the proposed work we use a reranking model from OpenNIR [MacAvaney, 2020][8] based on "Vanilla" Transformer architecture [Vaswani et al., 2017b].

BERT receives a query and document and processes it jointly. A distinctive feature of the BERT reranker is injection token similarity matrices on each layer, which considerably improves performance [MacAvaney et al., 2019].

First, we pretrain this reranker on the Antique dataset. We clean this dataset from incorrect symbols and makeups. We also left from the dataset documents of length more than 300 characters, since the length of the ChatNoir retrieves usually does not exceed 300. The training process lasted for 500 epochs with 0.001 learning rate and 56 objects in every batch. Finally, our model gives NDCG@5 0.3362 on a validation set. We fine-tune the model on 40 train topics from the previous year for 50 epochs with the same configuration. Fine-tuning increased the score on validation up to 0.412.

Table 6.9: Results on validation set

Method	NDCG@5	Time, ms
Random Forest	0.408	127.168
XGBoost	0.547	128.848
LightGBM	**0.572**	131.244
Bert Ranker	0.412	1560.947
Baseline'20	0.534	-

Results on validation set The result for every proposed approach obtained on the validation part of data from the previous year competition is in Table 6.9. We also evaluate the previous year's baseline on the validation set. The best scores come from the LightGBM model, which also outperforms the baseline. XGBoost has fewer scores, Random Forest as a simple algorithm has the smallest score. Bert overtakes Random Forest a little.

[8] https://github.com/Georgetown-IR-Lab/OpenNIR

In the right column, we also added the time required to train each model. It can be seen that the ensemble-based models have approximately the same time complexity, while the Bert requires much more time to train.

Results on the test set For final testing, the retrieved documents were manually labelled with a score from 0 to 3. Judgment was made on two independent criteria: the relevance of the document to the given topic and the quality of the text. The quality criterion includes good language styling, easy reading, and proper sentence structure, and the absence of typos.

For each criterion, a separate file with the assessor's scores is available. The results of two evaluations are presented in Table 6.10 and Table 6.11. The runs of our team Katana have the best result between all teams in terms of relevance and the second result in terms of text quality.

As in the validation set, XGBoost and LightGBM give the best performance. It is well explained, since the loss of these models is based on the ranking quality functions, NDCG in the XGBoost case, and LambdaMART in the LightGBM case. The first model describes the relevance a bit better (0.489) and has the first place among all participants. For quality, in contrast, LightGBM is better. It archives 0.684 and takes second place in a quality table, slightly surrendering to Top 1. The random forest method has scores just below the baseline in both cases. It can be explained by a more elementary algorithm to build an ensemble. Bert gives quite a good result for quality and a weak one for relevance. Perhaps the data from the adjacent task (factoid QA) used for the training is the reason for not a very accurate solution.

Table 6.10: NDCG@5 Relevance scores

Method	NDCG@5
Random Forest	0.393
XGBoost (Top 1)	**0.489**
LightGBM	0.460
Bert Ranker	0.091
ChatNoir baseline	0.422
Thor team (Top 2)	0.478

Table 6.11: NDCG@5 Quality scores

Method	NDCG@5
Random Forest	0.630
XGBoost	0.675
LightGBM (Top 2)	**0.684**
Bert Ranker	0.466
ChatNoir baseline	0.636
Rayla team (Top 1)	0.688

We present our solution to the Argument retrieval shared task. We pay attention to ensembles methods and use statistic approaches, language modelling, and comparative structure extraction to retrieve features for it. We also use a neural reranker based on the Bert technique to use information from a contextualised model in our task.

The best results were obtained by gradient boosting methods, training on ranking cost function: XGBoost and LightGBM. The proposed approaches outperform baseline and take first and second places in relevance and quality ranking, respectively. Bert contextualized model shows the need for large learning data.

Table 6.12: Example of documents with the different relevance to query "Is admission rate in Stanford higher than that of MIT?"

Is admission rate in Stanford higher than that of MIT?	
LightGBM Top-3	Baseline Top-3
1. Stanford and Harvard have a similar admissions rate of about 7%. MIT comes with a somewhat greater rate of success admitting just under 10% or 1742 for the class of 2015. Harvard, Stanford and MIT are global leaders in culture, commerce and governmental policies.	1. Stanford and Harvard have a similar admissions rate of about 7%. MIT comes with a somewhat greater rate of success admitting just under 10% or 1742 for the class of 2015. Harvard, Stanford and MIT are global leaders in culture, commerce and governmental policies
2. For more than a decade, I have served as an admissions officer for MIT. In that time, i've read more than 10,000 applications and have watched thousands of new students enter MIT. It is a privilege to work at the most dynamic and exciting university in the world.	2. For more than a decade, I have served as an admissions officer for MIT. In that time, i've read more than 10,000 applications and have watched thousands of new students enter MIT. It is a privilege to work at the most dynamic and exciting university in the world.
3. Our primary enhancement was targeted at families earning less than $75,000 — making mit tuition free and eliminating	3. All of this factual information, plus a lot of other detail, can be found in the mit admissions literature. In fact, this year, mit will award $74 million in undergraduate aid.

Table 6.13: Example of documents with the different relevance to query "Which smartphone has a better battery life: Xperia or iPhone?"

Which smartphone has a better battery life: Xperia or iPhone?	
LightGBM Top-3	Baseline Top-3
1. 1. The power saver app that will turn down settings when battery life is low to get as much juice out of the battery as possible. Sony has set the benchmark with its 12 megapixel camera inside the Xperia S.	1. The iPhone 4 is apple's thinnest smartphone yet, but offers a much better screen, faster processor, video calling, and many other enhancements.
2. How to increase the battery life of apple's iPhone 4s many of those with an iphone 4s have complaints about the battery life. Apple has acknowledged these problems, and is working to fix them.	2. Sony Xperia's review: an above average smartphone gizmotraker', as far as battery life is concerned, it last about 7 hr 30 min in talktime, 450 hrs in standby.
3. Sony Ericsson includes an 8gb card in the sales package the Sony Ericsson Xperia arc s has below average battery life. Most users will get around 24 hours of life out of the Xperia. X27's 1600mah battery before it needs a recharge, but heavy users may need an injection of power before then.	3. How to increase the battery life of Apple's Iphone 4s many of those with an iphone 4s have complaints about the battery life. Apple has acknowledged these problems, and is working to fix them.

6.3.2 Retrieving Comparative Arguments using Deep Language Models

In this part, we describe a submission to the Touché lab's Task 2 on Argument Retrieval for Comparative Questions. We continue attemting to solve the passage-retireval task in the aim to answering a question in the Comparative case and employ approaches based on pre-trained deep language model architecture ColBERT [Khattab and Zaharia, 2020]. This BERT-based architecture is adapted to the text ranking task by learning to represent both queries and documents as vectors and measuring the similarity between them. We use a model trained on a question-answering dataset MSMARCO, with the proposed weights and weights pre-trained by us. We also customize ColBERT for the comparative retrieval domain by fine-tuning the model on the data from the previous years' Touché competitions. The proposed experiments verify the usefulness of the transfer learning from a large pre-trained

ranking language models to the problem of arguments extraction for comparative topics. Our solutions rank third in both relevance, quality, and stance prediction evaluations.

The Touché lab's Task 2 on Argument Retrieval in 2022 [Bondarenko et al., 2022] proposes to select passages from a corpus of 1 million texts that are most relevant to the user's comparative queries, as well as to determine their position - which object in the text is proposed as the most suitable. We employ neural-network based approach with a simplified scheme for comparing query and document embeddings. In addition to using the pre-trained large language model, we further trained the model on documents ranked for comparative queries.

On the validation dataset, the approach shows competitive performance, but less than the ensemble-based method from the previous section 6.3.1. This work shows the possibility and efficiency of the neural network technique based on the matching of the query and document representations relatively to a specific comparative case of informational retrieval.

The main difficulty in finding relevant documents on the web is the large size of the text corpus. Traditionally, search engine systems depict documents using statistic-based features, the computation of which is not complex.

A large volume of texts imposes a limitation on the use of neural networks for ranking documents in a corpus. There are two ways of neural approaches to information retrieval tasks: representation-based models [Huang et al., 2013] and interaction-based models [Mitra et al., 2017]. The first one computes the representation of the topic and passage separately and only counts the score of interaction for the pair. Interaction-based methods match the query and document in a token or phrase-level. This set of methods is more expensive but most effective. In the proposed paper we deploy architecture, which combines the advantages of both these methods.

Data Provided for The Task

The organizers offer the participants 50 comparative questions (topics), for which it was necessary to extract and rank passages from the text corpus. Topics for the

competition are available online[9]. The organizers also provide a corpus of about 0.9 million texts for passage extraction. For stance detection, every topic comprises objects that are compared in it. For stance detection support, a dataset created from comparative questions of the MSMARCO dataset[10] is proposed. The dataset includes relevant answers with highlighted objects of comparison in it and their position in the documents. Every text in a dataset has a detected stance.

For model validation purposes, the task presents 100 topics and corresponding relevance annotations of the previous year's competition [Bondarenko et al., 2020c, 2021]. These documents were also retrieved from ChatNoir and ranked manually to 0 (not relevant), 1 (relevant), or 2 (highly relevant) scores. The 2020 year assessment contains a common ranking, last year's competition has a separate judgment for relevance and quality. We use this data to fine-tune the model to comparative sub-task in document retrieval. Besides, last year's team submissions are available too.

The standard learning object for argument ranking consists of a triple: query, positive passage (relevant text), negative passage (irrelevant text). Reading comprehension dataset MSMARCO (Microsoft Machine Reading Comprehension) [Nguyen et al., 2016] includes 1,010,916 anonymized questions from Bing's query and 8 million passages extracted from the search system Bing. For the training BERT-based model we use MSMARCO-Passage-Ranking, which comprises triplets from the mentioned questions and passages.

We use data from the previous years' Touche tasks to generate a validation dataset and dataset for fine-tuning the ColBERT model. For every topic, we retrieve up to 100 texts from the ClueWeb12 [11] corpus using the ChatNoir [Potthast et al., 2012] system, according to Tocuhe'20-21 task rules. The validation dataset was created on 10 topics from 2021 with corresponding quality and relevance qrels. The rest 40 topics and 50 topics from 2020 produce data for adapting the pre-trained model for text ranking in terms of argumentative objects comparison.

The 2020 year task topics have only one assessment dimension in qrels. If the

[9] https://webis.de/events/touche-22/shared-task-2.html
[10] https://microsoft.github.io/msmarco
[11] http://lemurproject.org/clueweb12

score in this is 1 or 2, we treat this text as relevant. Irrelevant pairs were selected from documents with ratings less than 1 or from the search results for different topics, provided that they were not presented in the search results for the current query. In the case of an assessment of 21 years, there are separate judgments among two axes: quality and relevance. We calculate a sum of a quality and relevance score and consider relevant documents having a score equal to or more than 3, otherwise - irrelevant. The statistic of mentioned datasets is in Table 6.14.

Table 6.14: Statistics of datasets used in training from scratch and fine-tuning.

Dataset	Task	Number of triples
MSMARCO-Passage-Ranking	train	39 780 810
Dataset based on Touché 2021	fine-tune	46 450

For document ranking, we use the ColBERT [Khattab and Zaharia, 2020] model, pre-trained in several ways. Using the model, in the test stage, we created an index of all documents in the provided collection of text passages. Using this index, we select the top-k most relevant texts to each of the topics. We use auxiliary information about objects under comparison to find them in every ranked document and define document stance using Comparative argumentation machine CAM [Schildwächter et al., 2019] functionality. We execute produced solutions on the web evaluation platform Tira [Potthast et al., 2019]. The retrieved documents will be manually assessed for both metrics: general relevance and comparison quality. Relevance represents proximity to the topic and the presence of sufficient argumentative support. Quality refers to good structure, understandable news, and text styling.

In the validation phase, we use topics of the previous year's competition as queries. The corpus on which the model builds the index consists of documents from the Chat Noir issue that are relevant to topics. We retrieve documents for every question and compare them to official Qrels judgments.

Document Ranking

The main architecture we used in the document retrieval task is Contextualized Late Interaction over BERT (ColBERT). ColBERT provides a trade-off between

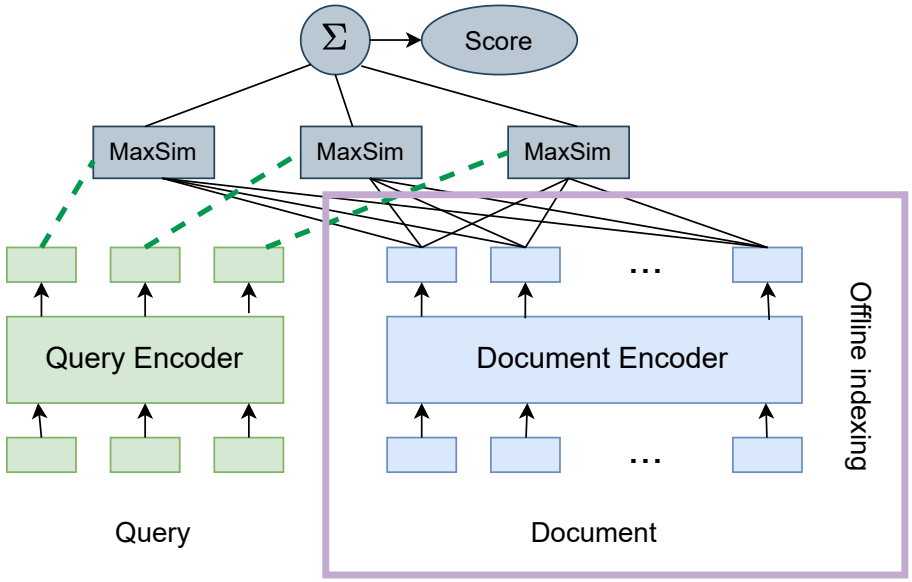


Figure 6-4: The late interaction matching scheme, which is used in the ColBERT model. The similarity of the query and document is the sum of the scores between every query token and the most similar document token. Source of the image: [Khattab and Zaharia, 2020].

representation-based models with low computational cost and well-performed token interaction-based models. Actually, for approaches with a full interaction matrix between query and document tokens, ColBERT reduces complexity by providing a convolution over the document tokens.

The query and document processing in the ColBERT architecture contains 2 steps:

- To encode query, we add $[Q]$ after $[CLS]$ token, process padded query by BERT, apply convolution and normalization
- To encode document, we add $[D]$ after $[CLS]$ token, process padded passage by BERT, apply convolution and normalization, also filter out punctuation symbols.
- The conception of Late Interaction (Figure 6-4) from the entire document considers only the token that has the highest similarity to the given query token. The relevance of the document is estimated as the sum of maximum similarities across all query tokens.

- To retrieve in a large-scale set of passages, we use the Faiss library [Johnson et al., 2019]. Faiss library implements the k-nearest neighbour approximation of text indexing, which balances the computational complexity and approximation quality.

Thus, the ColBERT approach fine-tunes BERT main encoder and learns from the scratch linear layers, filter and embeddings for $[q]$ and $[d]$ symbols. Leveraging on triplets of query, document with high relevance and document with low relevance $< q, d^+, d^- >$, the model optimizes the pairwise softmax cross-entropy loss.

ColBERT Models

For passage retrieval in the Touche task, we use three different types of pre-trained ColBERT architecture.

ColBERT original The first is a checkpoint, generated at the University of Glasgow [12] on MSMARCO triples using instruction from the official ColBERT repository [13].

ColBERT from scratch We also pre-trained ColBERT architecture, provided in repository, from scratch by ourselves. We use L2 distance between a query and document instead of cosine similarity, since the original paper noted that the faiss index works faster on a square distance. The training process was carried out in a 3 epochs with the learning rate 3e−6, batch size 64, passage length no more than 180, query length 32, similarity $l2$, and took about two weeks on a single GPU card.

ColBERT fine-tune We also tried to fine-tune the resulting model on data for a comparative question-answer system obtained from information from past competitions and described in section 6.14. The pre-training procedure was carried out with the following parameters: learning rate 1e−7, batch size 64, passage length no more than 180, query length 32, similarity $L2$. The weights are updated using the AdamW optimizer during 10 epochs.

[12] http://www.dcs.gla.ac.uk/~craigm/colbert.dnn.zip
[13] https://github.com/stanford-futuredata/ColBERT

Stance Detection

An additional challenge within the task was to determine the stances of retrieved documents. Stance defines the document's attitude towards the compared objects: pro first object, pro-second object, neutral, or the absence of attitude. To detect the stance of a given document, we note objects from topic auxiliary data, found them in the document, and consider text between objects' locations. Comparative Argumentative Machine (CAM) offers the possibility of classification those pieces of text. It decodes them into feature vectors using Infersent [Conneau et al., 2017] and applies a pre-trained XGBoost classifier to features [Schildwächter et al., 2019]. The output of CAM is considered to be a document stance class.

Results

We run the proposed approaches in two stages: in the validation stage, the model retrieves and ranks documents for the previous year's topic over the ChatNoir output, and in the test stage the model ranks passages for a given topics over proposed corpus, at the same time designating their stance.

Table 6.15: NDCG@5 results for quality and relevance of retrieved document on validation set.

Method	Quality	Relevance
Baseline'21	0.427	0.649
Best Answer'21	0.421	0.591
ColBERT original	0.413	0.474
ColBERT from scratch	0.342	0.314
ColBERT fine-tune	0.322	0.365

The result for every proposed approach obtained on the validation part of data from the previous year's competition is in Table 6.15. We compare ColBERT-based approaches to the previous year's baseline and LGBM Ranker, considered the best answer. The best scores come from the frequency-based feature baseline approach, the second place belongs to the ensembles over statistic and comparative features set. Pre-trained ColBERT provides slightly worse results in terms of quality. In terms of accuracy, the decrease is more significant, but the same in order as the difference between the first and second places scores. ColBERT, trained by our

team from scratch, provides a worse result than pre-trained ColBERT. This can be explained by the insufficient training time of the model relative to the original work. Fine-tuning this version on the dataset from the previous year's task gives a noticeable increase in relevance but makes the model perform slightly worse on quality. This may be due to the properties of the Touche-based dataset used for model fine-tuning. It contains passages that are less complete and grammatically correct than MSMARCO objects, but at the same time they are more suitable specifically for the comparative subset of questions.

The documents retrieved were manually evaluated for two dimensions. The first criterion is relevance describing how supportive answer is contained in passage, the second is rhetorical quality - good styling and well-understoodness of the text. The results also contain the macro-clssification scores F1 for the stance detection. The results for three criteria for our tean Katana and Top-1 approach for each metric are in Tables 6.16.

For the ranking document task, ColBERT, trained on the MSMARCO dataset has the best performance according to fine-tuning the model. The difference between the model with downloaded weights and the model trained by us from scratch is not significant. Pre-trained model achive 3rd place in terms of relevance, while model trained from scratch has 3rd place in the quality table. Fine-tuning comparative data impairs the results. It may be due to the quality difference between texts from the main and fine-tuning data - in the MSMARCO case, well-formed natural language passages were composed by humans on the basis of the search system outputs [Nguyen et al., 2016]. The quality of the stance detection towards the objects expectedly depends on the ranking performance - the ColBERT with pre-trained weights also takes third place.

Table 6.16: Final evaluation scores on the test set for Katana team as compared to the Top-1 approaches.

Method	NDCG@5 relevance	NDCG@5 quality	F1 stance detection
ColBERT original	0.618 (Top-3)	0.643	0.229 (Top-3)
ColBERT from scratch	0.601	0.644 (Top-3)	0.221
ColBERT fine-tune	0.574	0.637	0.212
Top-1 approach	0.758	0.774	0.313

6.4 FC Layers Compression in Comparative QA Neural Models

In the preceding section, we discussed the utilisation of the BERT-based ColBERT model for information retrieval tasks. In this section, we delve into compressing deep language models for information retrieval and question answering, employing SVD and TTM decomposition.

The architecture of ColBERT consists of the BERT pre-trained model and an additional output linear layer. We focus on the compression of the Transformer-based part and apply compression techniques to FC layers in all 16 Transformer blocks. We employ SVD with a rank of 40, as well as TTM decomposition with a rank determined through an analysis of the singular values in the internal SVD step of the TTM decomposition Algorithm 2.

Hyperparameters selection To determine the rank for the SVD, we refer to the desired compression rate of the model outlined in Section 5. For the decomposition of TTM, we select the core dimensions so that the dimensions products within each core were approximately equivalent across all $\mathcal{G}_k$. In Section 5 we provide a detailed explanation of this choice. As a result, we represent the BERT FC layers matrices, initially sized as [3072, 768], as a series of cores $[1, 4, 8, R]$, $[R, 6, 8, R2]$, $[R, 8, 6, R3]$, $[R, 4, 8, 1]$. The following paragraph describes the methodology for selecting the layers which is best for TTM decomposition, as well as its desired ranks.

Selection of layers to compress and corresponding TTM ranks The rank parameter is crucial in the TTM decomposition. To determine the rank for each core in the sequence, we employ the truncated SVD method for the corresponding unfolding matrix of the original tensor (like in the Algorithm 2). We extract and visualise the spectrum of each SVD for all modules in the BERT architecture. As we have 4 cores in decomposition, every module has 3 SVD spectra, one for each inner rank of the TTM decomposition. Our observations led to the following conclusions:

- Linear layers in the transformer architecture in common do not have exact

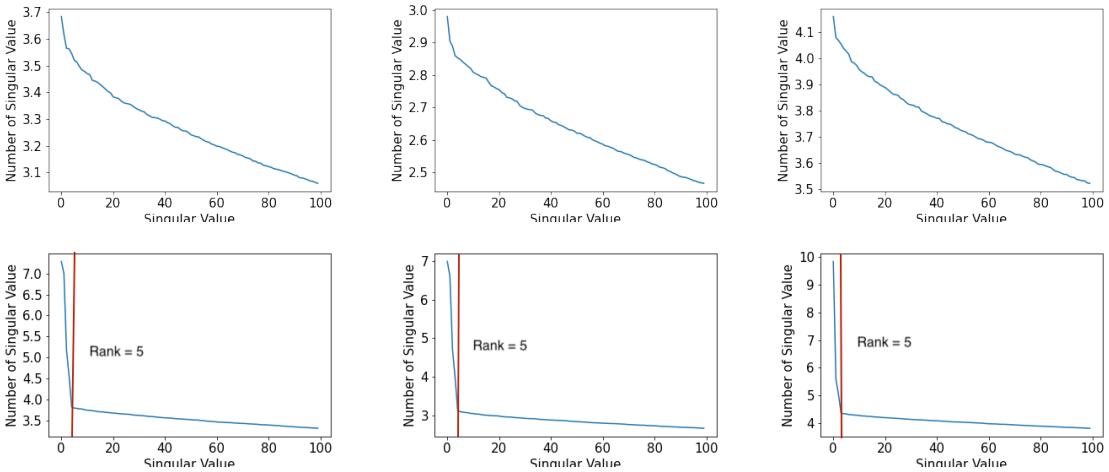


Figure 6-5: First row: singular values for the 1-st module layer, **"uncompressible"**, in ColBERT. Gradual slope of spectrum is less sharp. Second row: singular values for the 4-th module layer,**"compressible"**, in ColBERT. Gradual slope of spectrum is more sharp.

low-ranking structures (with singular values close to 0).

- Some layers have values with a more gradual slope without inflection points. Some layers have inflection points in the plot and values decreasing by approximately 2 times in the first 5-10 steps before reaching a plateau. We label the first type of layers as **"incompressible"** and the latter as **"compressible"** with a rank equal to the inflexion point (see Figure 6-5).

- For a particular weight matrix, all three sets of singular values within one layer are usually correlated in terms of compressibility or incompressibility.

- The compressibility (or incompressibility) property is typically maintained within the module. The module inside BERT consists of intermediate and output linear layers. For instance, if one layer in the module appears to be compressible, the other layer likely will be also compressible.

To compress the ColBERT, we chose modules 3, 4, and 5 from its architecture. In these modules, all linear layers show singular values decrease by approximately half among the all unfoldings. We identified that the number of singular values at which they obtain a half decrease is approximately 5 for dense layers and 20 for output layers inside the modules. We utilized these numbers to determine the desired ranks

for the TTM decomposition. It results in a model that retained 80% of the original parameters. We kept this hyperparameter — model compression rate — constant throughout the experiments detailed in this section. Based on it, we chose the rank value for SVD - 40.

6.5 Experimental Setup

ColBERT original In this setup we employ the ColBERT model, loaded from the checkpoint without any compression.

ColBERT TTM In this configuration, we use the TTM structure proposed in Section 4 to replace the modules that are most appropriate based on the rank selection procedure. We use the compression algorithm 2 to compress the intermediate FC part in the module with rank 5 and output with rank 20.

ColBERT TTM Round (TNTorch) In this section, we provide the compression method using the TNTorch framework [Usvyatsov et al., 2022]. First, we create full-rank cores by reshaping the proper unfolding without any truncation. Then, we reduce the core ranks to the desired level by applying the truncation SVD method to the corresponding unfoldings.

ColBERT SVD To achieve the desired model compression, we replace each FC layer in the modules by its SVD-compressed version with the desired rank.

ColBERT SVD Selection In these experiments, we apply SVD compression to modules, which are selected based on the above heuristics for TTM decomposition. The results (Table 6.17) show that, despite the use of TTM algorithm artefacts, this choice can also work for SVD compression.

6.5.1 Results

We select the three "compressible" modules (under numbers 3, 4 ,5) using the proposed heuristic. Table 6.17 shows that the downstream problem's results under this

Table 6.17: NDCG@5 results for quality and relevance of retrieved document on MSMARCO validation set. The column "Layers" provides number of modules, which have been selected for compression. Column "Rank" describes the corresponding compression ranks. In the case of SVD, the intermediate and output layers were compressed with the same rank, in the case of TTM decomposition, with different.

Method	Parameters	Layers	Rank	MRR@10	Success@5	Recall@50
ColBERT original	109 580 544	-	-	0.40	0.50	0.74
ColBERT SVD	94 051 584	2-4-6-8-10	40	0.20	0.30	0.59
ColBERT SVD Select	95 447 808	3-4-5	1	0.29	0.43	0.70
ColBERT TTM	81 952 992	2-4-6-8-10	50	0.09	0.14	0.37
ColBERT TTM (Custom)	95 485 518	3-4-5	20-5	0.27	0.40	0.68
ColBERT TTM (Tntorch)	95 485 518	3-4-5	20-5	0.26	0.41	0.68

Table 6.18: NDCG@5 results for quality and relevance of retrieved document on validation set. The column "Layers" provides number of modules, which have been selected for compression. Column "Rank" describes the corresponding compression ranks. In the case of SVD, the intermediate and output layers were compressed with the same rank, in the case of TTM decomposition, with different.

Method	Number of Parameter	Layers	Rank	Quality	Relevance
ColBERT original	109 580 544	-	-	0.49	0.42
ColBERT SVD	95 447 808	3-4-5	40	0.31	0.17
ColBERT TTM	95 485 518	3-4-5	20-5	0.34	0.17

selection are improved significantly compared to the absence of selection. Selecting for TTM decomposition-only modules, which are better compressible increases quality. However, if SVD compresses the same set of selected layers with an extremely low rank 1, the scores will also not drop but increase. At the same time, adding to the set of compressible modules at least one non-compressible (under number 6) drops quality for SVD. This can be interpreted as the model based on the Transformer architecture consists of modules with different significance. The best common strategy would be to compress the FC from the less significant ones. According to the results, FC in the ColBERT pre-trained model are better compressed by low-rank methods like SVD than TTM decomposition.

As can be seen from the Table 6.18, on the small comparative dataset TTM with modules selection gives the same relevance and a slight increase in quality according

to the SVD method.

Table 6.19 provides qualitative examples of the responses recovered for the full and compressed models. The quality characteristics of the found texts for TTM and SVD are approximately the same: it addresses the question a little worse than the answer found by the full model. However, in both versions of the texts retrieved by the compressed models, key concepts such as the product name or using a cooking appliance are presented.

Table 6.19: Example of documents with the different relevance to query "What is better at reducing fever in children, Ibuprofen or Aspirin ?"

What are the best rice cookers? ?		
ColBERT Pre-trained	ColBERT Compressed SVD	ColBERT Compressed TTM
1.This thing is the best. Finally I found a rice cooker of good quality, easy to clean, and doesn't leave a mess all over my counter!	1. If you're new to brown rice, try mixing half white rice and half brown rice together.	1. What they do: rice cookers, as their name indicates, are an electric appliance that can make perfectly cooked rice of all varieties.
2. To find the best, read rice cooker reviews: to help others make good rice cooker buying decisions, take a minute and review your current rice cooker.	2. The main benefit of a rice cooker is being able to cook rice better and faster than by traditional stovetop cooking.	2. It cooks up to 14 cups of moist, fluffy rice and can simultaneously steam vegetables and meats for easy, all-in-one meals.
3. Also, my other rice cookers would always make a mess by spewing out water everywhere. no mess with this cooker.	3. Take the guesswork out of making rice with this rice cooker and food steamer from aroma.	3. I grew up on rice cooker rice, so I always think the quality is the best.

6.6 Conclusion

In this Chapter, we compare different methods to answer questions within the comparative case. To do this, we propose our system demonstration pipeline, which

utilizes a Transformer-based CTRL model for generation, document and sentence retrieval methods, and combining answers using templates. We evaluate each approach using computational metrics and human opinions and find that the Transformer CTRL model approach performs the best.

We also present our solution for Argument Retrieval for Comparative Questions – a ranking task over a corpus of textual passages. We solved retrieval question-answering assignments by seeking the proper answer for data in the custom comparative dataset and dataset MSMARCO. In this competition, we pay attention to ensemble methods. To retrieve features for it, we use statistic approaches, language modelling and comparative structure extraction. We also use neural rankers based on the Transformer-based BERT architecture and ColBERT — model, which based on computationally effective late interaction architecture. Inside this competition, the best results are obtained by gradient boosting methods, training on ranking cost functions: XGBoost and LightGBM. Among the Transformer-based models, the best result is related to the ColBERT.

We took a ColBERT and applied TTM compression and SVD to the linear layers inside the model. As experiments show, both compression methods give approximately the same result on the dataset from the comparatively question-answer case and the MSMARCO dataset. We selected the layers most suitable for compression based on singular value analysis. The correct choice of layers seems critical - for example, compressing "proper" modules with a lower rank performs better than compressing all layers with a higher rank. Simultaneously, adding an "improper" layer to the compression set impairs the results.

The source code for information retrieval competitions is available online [14], as well as the System Demonstration code [15].

[14] https://github.com/sayankotor/touche
[15] https://s-nlp.github.io/coqas

Chapter 7

Transformer-based Encoders Compression using TTM Decomposition

This Chapter based on the paper "A Computational Study of Matrix Decomposition Methods for Compression of Pre-trained Transformers" (cf. Section "Publications" at page 4).

7.1 Introduction

Since the performance of large language models has advanced jointly with its size, there has been growing interest in developing techniques to compress these models while maintaining their effectiveness.

Normally, information is lost when compressing models and the quality drops noticeably. In this case, the models are fine-tuned until a certain quality is achieved on the task. However, fine-tuning is also resource intensive, even for a compressed model. To make it more efficient, we use the alignment of the low-rank compression objective and the task objective [Hsu et al., 2022]. This makes the compressed model more consistent with further fine-tuning.

One approach to model compression is to apply matrix factorization techniques to the heaviest part of the Transformer – FC layers (see Table 7.1). The most

popular and simplest choice is to use the SVD to reduce the number of parameters while retaining the expressive power of the model.

Applying SVD to a matrix can decrease its expressibility [Yang et al., 2018a]. However, additional techniques are employed to ensure a satisfactory quality of the resultant model. Hsu et al. [2022] introduces the Fisher Weighted SVD (FWSVD) approach, which considers the significance of each parameter for the model's performance during the compression process based on gradient values.

Another method for compressing large language models is Tensor-train matrix decomposition, or simply TTM decomposition [Oseledets, 2011b]. TTM decomposition transforms a weight matrix into a high-order tensor, which is then expressed as a product of lower-dimensional objects. In this study, we expand the application of the Fisher Weighted SVD (FWSVD) approach to TTM decomposition, creating a novel approach called FWTTM.

Our contributions can be summarized as follows:

- We extend the previous work by Hsu et al. [2022] and incorporate weighting based on Fisher information inside the TTM decomposition (we denote this approach as FWTTM).
- We provide a comprehensive analysis of the performance of the BERT model compressed with SVD, TTM, FWSVD, and FWTTM on various ranks on the GLUE benchmark tasks and the BART model in sequence-to-sequence tasks of text summarization and text detoxification.
- We provide an implementation of the studied methods widely applicable to pre-trained Transformer models, such as those at the Huggingface repository.

Table 7.1: Number of parameters for different modules in various Transformer architectures.

Module/Model	BERT		BART	
Full model	109 M	100%	140 M	100%
Fully connected layers	57 M	52%	84 M	60%
Embeddings	24 M	22%	38 M	27%
Attention	28 M	26%	23 M	16%

7.2 Related Work

This section reviews methods related to model size reduction. It contains knowledge distillation, quantisation, pruning, and low-rank approximation techniques.

The first approach, Knowledge distillation (KD), learns a student model with a smaller parameter budget guided by a larger trainer model. These methods can also be applied to transfer knowledge from a large teacher model to a smaller student model [Hinton et al., 2015b]. KD can improve the generalisation performance of the student model and reduce its size and computational cost [Jiao et al., 2020]. We use DistilBERT [Sanh et al., 2019a] – a distilled version of the BERT model – as one of the strong baselines in our work.

Pruning is another powerful technique to reduce the number of parameters in the deep neural network. The goal of neural network pruning is to identify and remove unimportant connections to reduce model size without affecting network accuracy. Movement pruning [Sanh et al., 2020] is a very efficient method for pruning unstructured networks. This method gives high sparsity in the model, while preserving the original quality score. On the other hand, such models will show effectiveness only with specialised hardware and may not give any benefits to standardised devices such as GPUs.

Block pruning is another effective method for reducing the number of parameters in the deep neural network. This approach involves removing entire blocks of unimportant connections rather than individual connections. This can result in a more structured and efficient network architecture. An example of block pruning is filter pruning, where all filters in a convolutional neural network are removed [Li et al., 2017].

For attention-based Transformer architectures, head pruning Michel et al. [2019] in which attention and self-attention heads are cut is also relevant.

The quantisation approach enables the reduction of the model size without compromising the parameter count, achieved by reducing the number of bits allocated to each parameter. The concept of quantisation-aware training, which involves training the model with the reduced weights, originated in the realm of general deep

learning models [Hawks et al., 2021] and has been extended to transformer-based encoders [Wang et al., 2022].

Low-rank approximation techniques provide an alternative way to achieve model compression. One such technique is SVD, which has been successfully applied to compress various components of neural networks, such as word embeddings [Lan et al., 2020], attention matrices [Michel et al., 2019], and transformer layers [Hu et al., 2021]. Another approximation technique is TTM, which decomposes high-order tensors into a sequence of low-order tensors Oseledets [2011b]. TTM has been employed for compressing word embeddings [Hrinchuk et al., 2020], CNNs [Garipov et al., 2016], and even visual transformers [Pham Minh et al., 2022].

7.3 Low-rank Compression Methods

In this section, we describe four low-rank approximation methods used in our computational study to compress feedforward layers of Transformers: SVD, TTM, FWSVD, and FWTTM, with the last one being a novel approach.

7.3.1 Layer Structures

We perform two types of replacement layers and, respectively, weight compression -SVD and TTM decomposition. To implement these techniques, we employ the corresponding PyTorch-compatible layer classes as described in Chapter 4.

Singular Value Decomposition (SVD) We compress the initial model by replacing Fully-Connected layers with their SVD analogues. To implement it, we employ the corresponding PyTorch compatible layer classes as described in Chapter 4. To obtain the SVD, we use the built-in Python function to decompose pretrained weights.

Tensor Train Matrix (TTM) decomposition We obtain a TTM-based layer, we also use the layer class which is described in Chapter 4. For decomposition, we harness an implementation based on the TNTorch [Usvyatsov et al., 2022] li-

brary with additions to achieve more memory and time-aware calculation set out in Chapter 4 and custom implementation of the algorithm from [Oseledets, 2010].

Fisher Weighted SVD (FWSVD) We inject Fisher's information into the decomposition algorithms to minimise the gap between the decomposition and task-oriented objectives. Fisher information determines the importance of parameters in relation to a given task [Bishop, 2007]. We find an approximation of the Fisher matrix $\mathbf{I_W}$ using the dataset $D = \{d_1, \ldots, d_{|D|}\}$ and loss $L(d_i)$ on dataset element as described by Hsu et al. [2022], for each weight matrix $\mathbf{W} \in \mathbb{R}^{I \times J}$:

$$\begin{aligned} \mathbf{I_W} &= \mathbb{E}\left[\left(\frac{\partial}{\partial \mathbf{W}} \log p(D|\mathbf{W})\right)^2\right], \\ \mathbf{I_W} &\approx \frac{1}{|D|}\sum_{i=1}^{|D|} \frac{\partial}{\partial \mathbf{W}} L(d_i; \mathbf{W}). \end{aligned} \tag{7.1}$$

Having this, ideally, we would want to solve weighted low-rank approximation:

$$\|\sqrt{\mathbf{I_W}} * (\mathbf{W} - \hat{\mathbf{W}})\|^2 \rightarrow \min_{\text{rank}\,\hat{W}=r}. \tag{7.2}$$

Unfortunately, this problem does not have a closed-form solution. Therefore, Hsu et al. [2022] proposes to sum Fisher matrix by rows and solve low-rank approximation with row-wise weighting, which can be done using SVD:

$$\begin{aligned} \tilde{\mathbf{I}}_\mathbf{W} &= \text{diag}\left(\mathbf{I_W} \cdot \mathbf{1}\right), \\ \hat{\mathbf{W}} &= \tilde{\mathbf{I}}_\mathbf{W}\mathbf{W} = \mathbf{U}\mathbf{S}\mathbf{V}^\mathbf{T}, \end{aligned} \tag{7.3}$$

where $\mathbf{1} = (1, \ldots, 1) \in \mathbb{R}^{J \times 1}$, *diag* - diagonal matrix with size $I \times I$.

The resulted weighted factors for initial matrix $\mathbf{W} \approx \hat{\mathbf{U}}\hat{\mathbf{S}}\hat{\mathbf{V}}^\mathbf{T}$ are computed as follows:

$$\hat{\mathbf{U}} = \tilde{\mathbf{I}}_\mathbf{W}^{-\mathbf{1}}\mathbf{U}, \hat{\mathbf{S}} = \mathbf{S}, \hat{\mathbf{V}} = \mathbf{V}. \tag{7.4}$$

As a result, we get low-rank approximations, which account for parameter importances for the target task.

Fisher Weighted TTM (FWTTM) Our contribution is the Algorithm 12 incorporating Fisher information into the TTM decomposition. The algorithm consists of the following steps:

1. Compute the Fisher matrix $\mathbf{I_W}$ for the original layer matrix $\mathbf{W}$.

2. Apply the same transformations to $\mathbf{I_W}$ as to $\mathbf{W}$ to obtain a "Fisher tensor" $\hat{\mathbf{I}}_\mathbf{W}$.

3. For each SVD step of the TTM algorithm, we use the Fisher matrix exactly as in the FWSVD setup. The first low-rank term is reshaped to obtain another core in TTM, the second term goes to the next iteration, and the Fisher matrix unfolds using $\mathbf{U}$ to keep the shape with the second term.

Algorithm 12 Fisher-Weighted TTM decomposition.

Input: Matrix of layer weights $\mathbf{W}$, matrix of Fisher weights $\mathbf{I_W}$, shapes $I_1, J_1, \ldots, I_d, J_d$, ranks $r_0, \ldots, r_d$

Output: Cores $\mathcal{G}^k$, $k = 1 \ldots d$ of the TTM decomposition

$\mathcal{B} = \mathbf{W}.\text{reshape}\,(I_1, J_1, \ldots, I_d, J_d)$
$\mathcal{B}_\mathcal{I} = \mathbf{I_W}.\text{reshape}\,(I_1, J_1, \ldots, I_d, J_d)$
$\mathcal{C} = \mathcal{B}.\text{permute}\,(1, d+1, 2, d+2, \ldots, d, 2d)$
$\mathcal{C}_\mathcal{I} = \mathcal{B}_\mathcal{I}.\text{permute}\,(1, d+1, 2, d+2, \ldots, d, 2d)$ $N_r = I_1 J_1 \ldots I_d J_d$
for k in $\{1, \ldots, d-1\}$ **do**
    $N_k = I_k J_k$
    $N_r = \frac{N_r}{N_k}$
    $r = r_k$
    Unfolding $\mathbf{M} = \mathcal{C}.\text{reshape}\,(N_k, rN_r)$,
    Unfolding $\mathbf{M_I} = \mathcal{C}_\mathcal{I}.\text{reshape}\,(N_k, rN_r)$
    $\tilde{\mathbf{M}}_\mathbf{I} = \text{diag}\,(M_I)$
    $\tilde{\mathbf{M}}_\mathbf{I}\mathbf{M} = \mathbf{U}\mathbf{S}\mathbf{V}^\mathbf{T}$ truncated to r_k
    $\tilde{\mathbf{U}} = \tilde{\mathbf{M}}_\mathbf{I}^{-1}\mathbf{U}$
    $\mathbf{M} = \mathbf{S}\mathbf{V}^\mathbf{T}$
    $\mathbf{M_I} = \mathbf{U}^\mathbf{T}\mathbf{M_I}$
    $\mathcal{G}^k = \tilde{\mathbf{U}}.\text{reshape}\,(r_k, n_k, r_{k+1})$
    $\mathcal{G}^k = \mathcal{G}^k.\text{permute}\,(2, 1, 3)$
end for

7.4 Transformer Compression Setup

This section describes our setup for compressing Transformer models using low-rank approximation approaches. We focus on two methods: TTM and SVD, *with* and *without* using Fisher information. We aim to reduce the number of parameters in the model while maintaining its performance. Furthermore, we assume that we can access the task-orientated model-tuning process. We use the information obtained within this process to improve the quality of the compression, and thus speed up the tuning by the desired values.

We run two setups for compressing and evaluating models on the GLUE [Wang et al., 2019], ParaDetox [Logacheva et al., 2022], and XSUM [Narayan et al., 2018] datasets:

- **Single-train**. We fine-tune a model for each task, compress it, and measure performance.
- **Double-train**. We follow the same steps as for the Single-train and fine-tune the compressed model again on the same task.

All datasets in this work are based on English.

7.4.1 Baselines

We compare our compressed model to the model obtained by distillation [Jiao et al., 2020], Block Pruning [Sanh et al., 2020] and inference of the original model with floating-point precision equal to 16. Note that Distillation and Block Pruning are train-aware methods. It means they require fine-tuning for the desired task, so we can use it only in the Double-train pipeline.

For mixed precision training and evaluation, we use the FP16 library, which is built-in in PyTorch [Paszke et al., 2019b]. We set the optimisation level to 01 and patched all torch functions and tensor methods, except those that benefit from FP32 precision (softmax, etc.) For the two models analysed, we obtained a compression up to 52% for the BERT model and 54% for the BART model. However, since the resulting Tables 7.5, 7.8, 7.7, 7.6 show compression of the models relative to

the number of their parameters, but the quantisation of FP16 keeps the number of parameters the same, we indicate the actual number of parameters with a dagger (†).

7.4.2 Experimental Setup

We compare the performance of four proposed methods based on the BERT and BART models. We use three compression ratios for the selected models and report the corresponding ranks in Table 7.2. Finally, we tested compressed BERT on nine NLU tasks, including language acceptability, sentiment analysis, paraphrasing, and natural language inference, compressed BART – on text summarization, and detoxification tasks.

Table 7.2: Ranks for different compression approaches.

BERT			BART		
C. Rate	SVD	TTM	C. Rate	SVD	TTM
48% (53 M)	6	10	60% (83 M)	10	10
63% (69 M)	183	60	74% (102 M)	210	64
95% (102 M)	534	110	90% (125 M)	460	96

7.4.3 Selection of Hyperparameters

The proposed layer structure assumes two sets of hyperparameters: TTM *core shapes* for TTM decomposition and *ranks* for SVD and TTM decomposition.

We state that for the maximum compression rate in TTM, cores' non-rank shapes should be as close to each other as possible. We choose $I_k \cdot J_k$ so that they are equal to each other and approximately equal to $(I \cdot J)^{1/D}$. The selection of shapes is implemented with a custom algorithm, which will be presented source code. As cores, we take objects with sizes $[1 \times 32 \times 12 \times R]$, $[10 \times 3 \times 2 \times R]$, $[R \times 2 \times 2 \times R]$, $[R \times 16 \times 16 \times 1]$.

The rank r for the truncation in SVD and the set of $R_1 \dots R_{M-1}$ is selected according to the desired compression level.

7.4.4 Selection the Layers for Compression

As mentioned in section 6, certain modules within the transformer architecture lend themselves better to compression than others. This chapter proposes two methods to distinguish between favourable and unfavourable modules.

The first approach, similar to the one described in the previous part, involves analysing the singular values of the SVD expansions for the SVD and TTM layers. Those layers which have "sharp" graphics of the singular values will be marked as compressible with ranks depending on the inflection point of the graphics. We also involve the second approach, a method based on analysing Fisher information inside the layer.

As an additional method of selecting layers for compression, we use a metric called Fisher information variance $\varphi(\mathbf{W})$ [Hua et al., 2022], calculated as the variance of its corresponding Fisher information $\mathbf{I}_\mathbf{w}$. We rank the calculated $\varphi(\mathbf{W})$ for each layer in the ascending order. As candidates, we take the layers with the lowest variance. In the following experiments, we use the same number of layers for compression and the same ranks as in the singular values method to get a similar model size after compression.

As indicated in Tables 7.3,7.4, the quality of compression for both SVD and TTM decomposition is highly dependent on the selection of "good" layers. However, the singular value-based method outperforms the Fisher matrix statistics in terms of overall improvement.

Table 7.3: The results of different types of selection of BERT modules for compression in the Single-train pipeline. All compressed models has approximately 91 mln parameters.

Compression	Selection	AVG	STSB	CoLA	MNLI	MRCP	QNLI	QQP	RTE	SST2	WNLI
SVD	No select	0.64	0.79	0.1	0.76	0.6	0.79	0.84	0.52	0.89	0.51
SVD	FisherS	0.58	0.82	0.26	0.6	0.32	0.63	0.82	0.5	0.8	0.49
SVD	SingularS	0.71	0.86	0.43	0.72	0.78	0.83	0.84	0.56	0.9	0.52
FWSVD	No select	0.77	0.87	0.54	0.83	0.83	0.88	0.87	0.63	0.91	0.52
FWSVD	FisherS	0.64	0.85	0.35	0.68	0.48	0.78	0.81	0.52	0.84	0.48
FWSVD	SingularS	0.73	0.87	0.42	0.75	0.8	0.82	0.79	0.62	0.9	0.56
TTM	No select	0.54	0.81	0.07	0.46	0.4	0.58	0.68	0.53	0.78	0.52
TTM	FisherS	0.4	0.53	0.01	0.35	0.16	0.49	0.46	0.48	0.58	0.5
TTM	SingularS	0.53	0.82	0	0.51	0.16	0.74	0.69	0.50	0.82	0.55

Table 7.4: The results of different types of selection of BERT modules for compression in the Double-train pipeline. All compressed models has approximately 91 mln parameters.

Compression	Selection	AVG	STSB	CoLA	MNLI	MRCP	QNLI	QQP	RTE	SST2	WNLI
SVD	No select	0.76	0.88	0.52	0.83	0.85	0.9	0.74	0.66	0.9	0.51
SVD	FisherS	0.77	0.88	0.46	0.83	0.87	0.9	0.89	0.66	0.9	0.52
SVD	SingularS	0.78	0.88	0.53	0.83	0.87	0.9	0.88	0.67	0.92	0.51
FWSVD	No select	0.67	0.88	0.51	0.83	0.86	0.89	0.32	0.62	0.62	0.53
FWSVD	FisherS	0.77	0.88	0.46	0.83	0.86	0.9	0.89	0.65	0.9	0.52
FWSVD	SingularS	0.78	0.89	0.53	0.83	0.87	0.9	0.89	0.67	0.91	0.52
TTM	No select	0.67	0.87	0.1	0.83	0.84	0.87	0.46	0.62	0.9	0.56
TTM	FisherS	0.76	0.88	0.43	0.83	0.87	0.9	0.89	0.63	0.9	0.48
TTM	SingularS	0.76	0.88	0.42	0.83	0.87	0.89	0.88	0.65	0.9	0.52

Table 7.5: The results of different types of compression of BERT for experiment with task-oriented fine-tuning and further compression (Single-train). The best results at each model size are in **bold**, best overall results are underlined.

Method	C.Rate	AVG	STSB	CoLA	MNLI	MRCP	QNLI	QQP	RTE	SST2	WNLI
Full	100 %	0.79	0.88	0.57	0.84	0.9	0.91	0.87	0.67	0.92	0.54
DistilBERT	61 %	0.76	0.87	0.51	0.82	0.87	0.89	**0.88**	0.59	0.91	0.48
FP16 eval.	100%†	0.78	0.88	0.55	0.83	0.88	0.90	0.88	0.67	0.91	0.48
Block Pruning (75%)	61%	0.72	0.85	0.24	0.83	0.83	0.86	0.87	0.52	0.88	0.56
SVD		0.37	0.24	0.00	0.36	0.20	0.50	**0.47**	0.48	0.52	0.51
FWSVD		0.38	0.25	0.00	0.33	**0.39**	0.50	0.40	0.49	0.51	**0.56**
TTM	49 %	**0.44**	**0.58**	0.02	**0.37**	0.25	**0.56**	0.43	**0.50**	**0.72**	0.51
FWTTM		**0.44**	**0.59**	0.02	**0.37**	0.27	0.54	0.42	**0.50**	**0.71**	0.51
SVD		0.45	0.63	0.01	0.36	0.22	0.51	0.54	0.54	0.78	0.48
FWSVD		**0.55**	0.54	**0.07**	**0.52**	**0.55**	0.62	**0.70**	**0.58**	**0.79**	0.55
TTM	63 %	0.44	0.65	0.01	0.40	0.16	0.54	0.52	0.48	0.74	0.48
FWTTM		0.47	**0.71**	0.01	0.44	0.17	**0.64**	0.53	0.48	0.72	**0.56**
SVD		0.70	0.81	0.26	0.82	0.69	0.88	0.87	0.53	**0.90**	0.53
FWSVD		**0.78**	**0.88**	**0.55**	**0.84**	**0.87**	**0.90**	**0.88**	0.64	**0.92**	**0.55**
TTM	95 %	0.76	0.87	0.52	0.79	0.86	0.87	0.86	**0.65**	0.91	0.48
FWTTM		0.77	0.87	0.53	0.81	0.86	0.88	0.87	**0.65**	0.91	0.54

7.5 Experiments with NLU Tasks

In this section, we perform an evaluation of encoder-based Transformers using the BERT model [Devlin et al., 2019] as the base model. We use `bert-base-uncased` checkpoint from the HuggingFace [Wolf et al., 2019] model hub.

7.5.1 Experimental Settings

We perform experiments on the General Language Understanding Evaluation (GLUE) benchmark [Wang et al., 2019] using the evaluation script and metrics provided by

Table 7.6: The results of different types of compression of BERT for experiments with task-oriented fine-tuning, compression, and further fine-tuning (Double-train).

Method	C.Rate	AVG	STSB	CoLA	MNLI	MRCP	QNLI	QQP	RTE	SST2	WNLI
Full (109 mln.)	100 %	0.79	0.88	0.57	0.84	0.90	0.91	0.87	0.67	0.92	0.54
DistilBERT	61%	0.76	0.87	0.51	0.82	0.87	0.89	0.88	0.59	0.91	0.48
FP16 eval.	100%†	0.78	0.88	0.55	0.83	0.88	0.90	0.88	0.67	0.91	0.48
Block Pruning (75%)	61%	0.72	0.85	0.24	0.83	0.83	0.86	0.87	0.52	0.88	0.56
SVD		0.68	**0.83**	0.00	**0.79**	0.79	**0.85**	**0.87**	0.59	**0.87**	0.49
FWSVD		0.68	0.82	0.04	**0.79**	0.79	**0.85**	**0.87**	0.56	0.86	**0.54**
TTM	49%	**0.69**	**0.83**	**0.15**	0.78	**0.81**	0.84	**0.87**	**0.60**	0.86	0.43
FWTTM		**0.69**	**0.83**	**0.15**	0.78	**0.81**	0.84	**0.87**	**0.60**	0.86	0.49
SVD		0.75	0.86	0.43	**0.83**	0.84	**0.89**	**0.88**	0.64	**0.90**	0.50
FWSVD		**0.77**	**0.87**	**0.47**	**0.83**	**0.85**	**0.89**	**0.88**	**0.65**	**0.90**	**0.56**
TTM	63%	0.70	0.85	0.10	0.81	0.81	0.86	**0.88**	0.61	0.88	0.49
FWTTM		0.70	0.85	0.08	0.81	0.82	0.86	0.87	0.61	0.88	0.53
SVD		0.78	**0.89**	**0.56**	**0.84**	**0.88**	**0.91**	**0.89**	0.68	**0.91**	0.44
FWSVD		**0.79**	**0.89**	**0.56**	**0.84**	**0.88**	0.90	**0.89**	**0.69**	**0.91**	0.51
TTM	95%	0.77	0.88	0.52	0.83	0.83	0.89	0.88	0.68	0.90	0.51
FWTTM		0.78	0.88	0.52	0.83	0.87	0.90	0.88	0.68	0.90	**0.54**

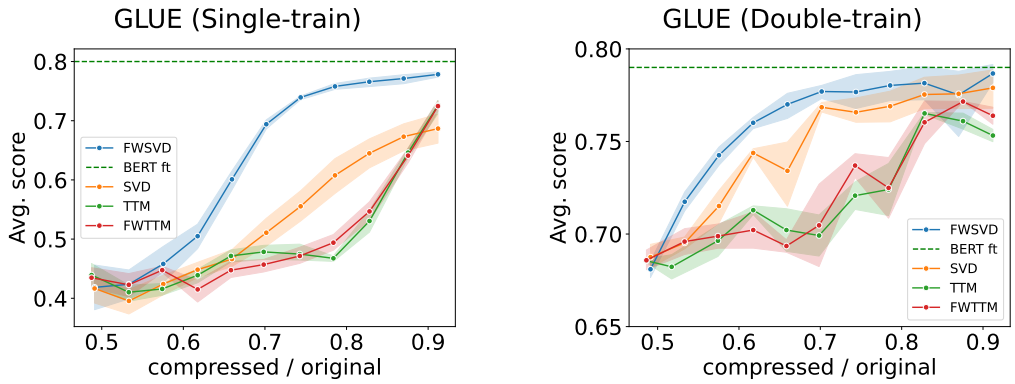


Figure 7-1: Trade-off between accuracy and compression rate for GLUE tasks: Single-train and Double-train average over all tasks.

HuggingFace library [Wolf et al., 2019].[1] Additionally, we run our experiments with five different random seeds and report the average performance across runs to ensure the robustness of our results.

Results The evaluation results of the BERT model on the GLUE benchmark using different compression methods are reported in Tables 7.5 and 7.6, respectively, for single and double train setups. Pre-trained models are available at Hugginface [2].

The results with standard deviations are given in the Appendix, Chapter A. Fore-

[1] https://github.com/huggingface/transformers/tree/main/examples/pytorch/text-classification

[2] https://huggingface.co/s-nlp/

most, one may observe that the overall absolute scores for the Single-train setup are much lower than for the Double-train which is also accounted in the prior research. Certainly, modifying the low-rank-based compression structure of the layers necessitates additional fine-tuning to restore the model's performance to a reasonably acceptable level. Please note that in these Thesis, we consistently present both setups, since Single-train can be utilized for numerous large models that exceed the memory capacity of the available GPU. Typically, for fine-tuning, it is necessary to double the size of the model to store gradients.

Secondly, we observe that the TTM decomposition outperforms SVD in low ranks, that is, in the area of high compression (49% of the original model parameters), while SVD performs better at higher ranks. This difference is large for a Single-train and for a Double-train it is not so pronounced.

Thirdly, we observe that incorporating Fisher information improves SVD consistently and slightly improves TTM at high ranks while not degrading its performance at other ranks (compression levels). TTM performs poorly in some tasks, such as CoLA, and better in other tasks, such as STSB (see Figure 7-1). Low-rank compression methods, especially FWSVD, outperform fine-tuned baseline models of approximately the same size at medium compression rates.

Finally, one can observe that for Double-train FWSVD (63%) compares comparably or better to Distillation and Pruning baselines while it could be combined with F16 quantization in principle.

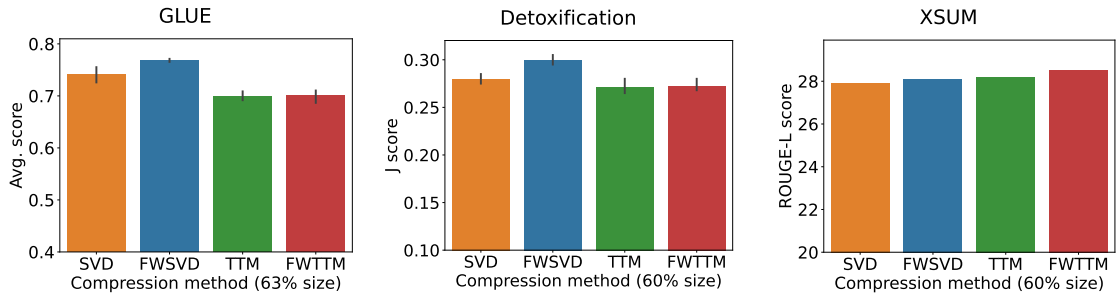


Figure 7-2: Comparison of compression methods for GLUE, Detox, XSUM with Double-train setup.

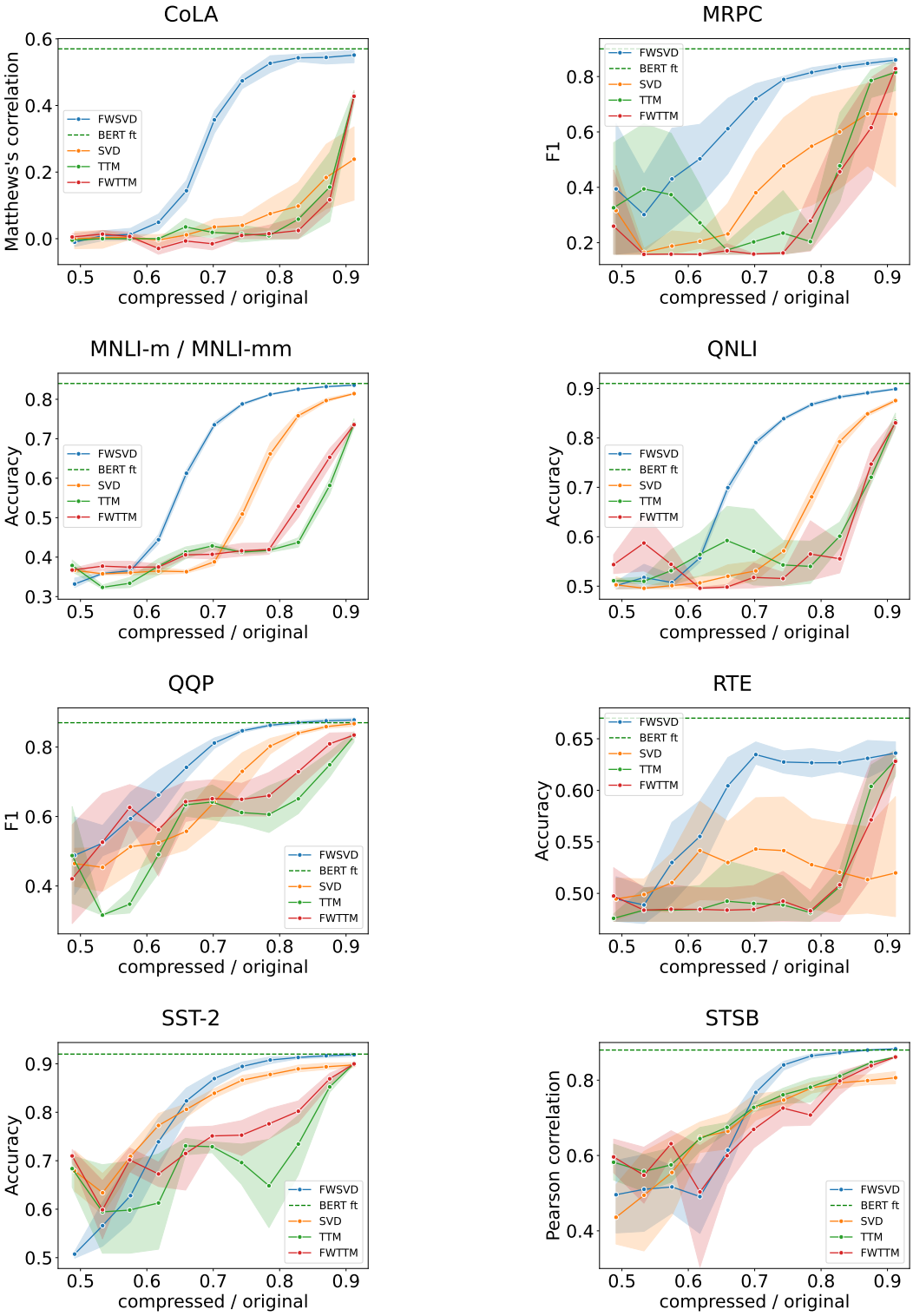


Figure 7-3: Results for GLUE benchmark for *bert-base-uncased* model, with task-oriented fine-tuning and further compression (Single-train).

Table 7.7: The results of different types of compression of BART for experiments on XSUM dataset with task-oriented fine-tuning and further compression (Single-train and Double-train). The best results at each model size are in **bold**, best overall results are underlined.

Pipeline		Single-train			Double-train		
Metric		ROUGE			ROUGE		
Method	C. Rate	1	2	L	1	2	L
`bart-base`	100%	42.4	19.6	34.5	42.4	19.6	34.5
FP16 eval.	100%	32.8	11.0	25.5	32.8	11.0	25.5
Block Pruning (95%)	63%	-	-	-	23.4	5.7	18.8
Block Pruning (65%)	74%	-	-	-	34.6	12.2	27.9
SVD	60%	6.2	**0.5**	5.4	35.2	13.1	27.9
FWSVD		**11.1**	0.3	**8.5**	35.5	13.3	28.1
TTM		3.6	0.1	3.1	35.8	13.3	28.2
FWTTM		7.4	**0.4**	6.3	**36.0**	**13.8**	**28.5**
SVD	74%	6.4	0.4	5.4	39.8	17.1	32.2
FWSVD		**18.5**	**3.2**	**14.5**	**40.6**	**17.8**	**32.9**
TTM		5.6	0.2	4.7	38.3	15.9	33.6
FWTTM		5.2	0.3	5.5	39.0	16.2	32.1
SVD	90%	32.8	11.0	25.5	41.8	18.9	33.9
FWSVD		**38.9**	**19.1**	**31.2**	**41.9**	**19.1**	**34.1**
TTM		26.4	6.8	19.8	41.2	18.6	33.6
FWTTM		23.2	5.1	16.3	38.1	19.5	33.2

7.6 Experiments With Sequence-to-sequence Models

We test different layer compression methods on the encoder-decoder model BART [Lewis et al., 2020] on two sequence-to-sequence tasks. Namely, we test different compression methods on text summarization tasks and a subtask of textual style transfer - text detoxification. In our experiments, we use `bart-base` checkpoint from the HuggingFace [Wolf et al., 2019] model hub.

7.6.1 Text Summarization

In our text summarization experiments, we use the XSUM English dataset [Narayan et al., 2018], which contains news articles and their corresponding single-sentence summaries. We aim to train BART to generate accurate and concise summaries of

Table 7.8: The results of different types of BART compression for detoxification experiments in Single-train and Double-train pipelines. The best results at each model size are in **bold**, best overall results are underlined. *Italic* results represent senseless model outputs.

Pipeline		Single-train				Double-train			
Method	C. Rate	STA	SIM	FL	J	STA	SIM	FL	J
`bart-base`	100 %	-	-	-	-	0.89	0.60	0.82	0.44
FP16 eval.	100%†	-	-	-	-	0.89	0.60	0.82	0.44
Block Pruning (95%)	63%	-	-	-	-	*0.92*	*0.34*	*0.30*	*0.12*
Block Pruning (65%)	74%	-	-	-	-	0.82	0.60	0.73	0.36
SVD	60%	*0.97*	*0.18*	*0.10*	*0.01*	0.75	**0.59**	0.65	0.28
FWSVD		*0.32*	*0.46*	*0.58*	*0.07*	**0.78**	**0.59**	**0.68**	**0.30**
TTM		*0.97*	*0.19*	*0.16*	*0.03*	0.74	0.58	0.64	0.27
FWTTM		*0.99*	*0.18*	*0.19*	*0.03*	0.74	0.58	0.65	0.27
SVD	74%	*0.85*	*0.21*	*0.14*	*0.03*	0.82	0.60	0.77	0.38
FWSVD		*0.32*	*0.46*	*0.58*	*0.07*	**0.87**	**0.61**	**0.80**	**0.42**
TTM		*0.99*	*0.17*	*0.06*	*0.01*	0.82	**0.61**	0.75	0.37
FWTTM		*0.99*	*0.18*	*0.16*	*0.03*	0.84	0.60	0.75	0.38
SVD	90%	**0.85**	0.42	0.72	0.25	0.86	**0.61**	**0.81**	**0.43**
FWSVD		0.70	**0.64**	**0.82**	**0.35**	**0.87**	**0.61**	**0.81**	**0.43**
TTM		0.49	0.60	0.71	0.18	0.86	**0.61**	0.80	0.41
FWTTM		0.82	0.46	0.59	0.22	0.86	**0.61**	0.80	0.41

the input articles.

We evaluate the performance of models using the **ROUGE** metrics [Lin, 2004]: we use **ROUGE-1** and **ROUGE-2** to measure the overlap between the generated and reference summaries at the unigram and bigram levels. We also use **ROUGE-L** to measure the longest common subsequence between the generated and reference summaries.

Results The results for summarization are presented in Table 7.7. In the Single-train setup, FWSVD outperforms other methods across different compression levels by all metrics. However, in the Double-train pipeline, FWTTM emerges as the top performer at low ranks, while TTM excels only in the Rouge-2 metric at high ranks. In terms of the remaining metrics, SVD demonstrates superior performance. Notably, the tensor and matrix compression techniques employed in the Double-train setup exhibit improved results as compared to the baselines.

7.6.2 Text Detoxification

Text detoxification aims to rewrite a sentence in a rude form into a neutrally formulated sentence while preserving its meaning.

We use the English parallel dataset ParaDetox [Logacheva et al., 2022] in our experiments. This dataset contains pairs of sentences in rude and neutral forms, which allows us to train text detoxification models in a way similar to neural machine translation. The scale of the dataset also makes training faster and more convenient. We follow the evaluation pipeline presented by Logacheva et al. [2022] and measure the performance of our models using three metrics: **STA** (style transfer accuracy), **SIM** (similarity), and **FL** (fluency of generated text). **STA** measures how well the model transfers the style of the input sentence from rude to neutral. **SIM** measures how similar the meaning of the generated sentence is to the input sentence. **FL** measures how fluent and natural the generated sentence is.

Results We show the results of our compression experiments in Table 7.8 and provide the extended version of the table with variations included in Table A.3. Text generation also preserves the trend shown in language comprehension. However, unlike GLUE, in the task of detoxification, all the compressed models in the Single-train pipeline are hallucinating and generating senseless tokens at low and medium ranks. We represent these results with *italic*. Pre-trained models are available at Hugginface [3].

In general, the TTM and SVD approaches show the best results in low ranks on all tasks. At medium and high ranks, FWSVD breaks ahead. FWSVD demonstrates the most significant benefits at medium ranks, while the impact becomes nearly imperceptible at higher compression levels. Therefore, it is essential to note that the Fisher information acquired for the language modelling task also contributes to improvements in metrics that are not directly related to LM (such as **STA** and **FL**); moreover, they are obtained by the auxiliary neural network model.

[3] https://huggingface.co/s-nlp/

7.7 Conclusion

In the proposed work, we explore the Transformer compression techniques that involve low-rank and tensor decomposition of its most heavy part – fully-connected layers (featuring 50-60% of parameters depending on the model). We test the performance of compressed BERT and BART models on natural language understanding and generation tasks and compare the suggested compression with other popular approaches, such as pruning, distillation, and quantisation. Furthermore, we for the first time implement and adapt the method proposed by Hsu et al. [2022] to the decomposition of TTMs incorporating Fisher information, which measures the importance of individual layer parameters with respect to training objectives.

Our experiments, summarised in concise form in Figure 7-2, show that incorporating Fisher information (FW* models) consistently improves the quality of the compression method, in the case of SVD and depending on the setup, improves or does not degrade the quality of TTM. At a medium compression level, baseline approaches based on training-aware distillation, pruning, and quantisation demonstrate inferror or performance comparable to the respective methods in each group decomposition-based methods.

In summary, we can say that the representation of pretrained layers in the TTM form is a working method, which, however, does not give an unambiguous result for any given compression settings. In the future, it would be interesting to test this method not only for compressing the entire model, but also for compressing weight gradients, as is done, for example, in the LoRA [Hu et al., 2022] approach.

Chapter 8

Conclusion

In this dissertation, we used compressed representation of tensor structures to address limitation of contemporary Natural Language Processing caused by large data sizes and large sizes of linguistic models. We met three statements that appeared in the Thesis Objectives. First, we showed the efficacy of Canonical Polyadic Decomposition in compressing the embedding layer of a Knowledge Base model while accounting for data characteristics. This approach yielded favourable results in the explicit embedding evaluation and downstream tasks, simplifying the training process, and reducing memory consumption.

Then, we reduce the size of the Transformer models by replacing the linear layer with a Tensor Train Matrix structure. To achieve this, we adapted the TTM object for the neural network to optimise signal propagation through the TTM core. The resulting TTM container required fewer parameters and memory while maintaining the same efficiency level during training and fine-tuning. Consequently, this approach reduces time and memory requirements, making it more energy efficient and suitable for low-power hardware.

With these structural components in place, we address two key questions, which finally aimed at two types of experiment. First, we investigate the expressive power of TTM compression compared to low-rank methods. To do so, we conducted experiments on the GPT-2 decoder transformer, training it from scratch and evaluating its performance on Language Modelling, Natural Language Understanding, and Text summarization tasks. Finally, our findings revealed that 1) the training environment

heavily influences the performance of the final model, and 2) under identical training conditions, a model with TTM layers consistently outperforms other methods, such as SVD.

Secondly, we explored the impact of compressing pre-trained models on their quality for specific tasks. We conducted experiments on a question-answering task within a particular comparative case. We research the different approaches to a Question-Answering task and focus on retrieval-based techniques. Next, we evaluate several retrieval-based groups of methods, statistics, tree structures, and transformer-based on a comparative dataset. As a transformer method requires large transformer models to employ, we provide compression of FC layers in them. We obtain the result that TTM decomposition and SVD give similar compression results inside comparative QA task and are highly dependent on the choice of the "proper" layer for compression. We also provide the algorithm for finding such types of layers.

We also conducted experiments with compression and further fine-tuning on the BERT architecture for the tasks of the Natural Language Understanding benchmark GLUE. We met the third point from the Objectives by injecting information about the Language Modelling loss in the compression algorithm - SVD or TTM decomposition - and show that it induces performance.

Under described experiment conditions, TTM decomposition outperforms SVD at lower ranks and loses at higher ranks. According to the third point in Objectives, we define layers inside the model that are more and less compressible. This property is not constant and varies from one Transformer architecture to another, it can even change during fine-tuning of a specific model. We can identify these layers by analysing the SVD spectrum inside the TTM-SVD algorithm.

The results of two types of experiments — training from scratch in the Chapter 5 and compression in Chapters 7, 6 — support the ideas from the Chapter 4: TTM representation of a FC layer in Transformers is effective only if we train it from scratch. In this case, we get a compressed representation of a layer matrix that is structured for TTM, and TTM approaches outperform SVD on downstream tasks. In the case of compression of yet pre-trained FC layer weights, which do not have a Kronecker structure, TTM decomposition isn't engaged to outperform low-rank

methods.

In terms of potential practical application of the proposed research, we make the following observations.

The first part of the work focusses on the practical implementation of diverse concepts, including developing an algorithm for knowledge base embeddings. These concepts hold significant potential in tackling the language grounding problem, wherein a model acquires a complete representation of a specific fact or phenomenon by considering inputs such as images, textual descriptions, and relevant information from knowledge graphs.

The TTM-based layers are invariant across different transformer models. This characteristic opens up the possibility, if desired (and with access to weights), to optimize even massive models like ChatGPT [1] and GigaChat [2]. As the size of large language models is expected to continue growing in thein the nearest future, driven by an increase in the number of transformer modules and the size of the processed text, there arises a need for a more condensed parameterized layer representation. Such a representation would offer valuable benefits in these evolving scenarios.

[1] https://openai.com/blog/chatgpt
[2] https://developers.sber.ru/portal/products/gigachat

Bibliography

Tinsaye Abye, Tilmann Sager, and Anna Juliane Triebel. An open-domain web search engine for answering comparative questions. In Linda Cappellato, Carsten Eickhoff, Nicola Ferro, and Aurélie Névéol, editors, *Working Notes of CLEF 2020 - Conference and Labs of the Evaluation Forum, Thessaloniki, Greece, September 22-25, 2020*, volume 2696 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2020. URL http://ceur-ws.org/Vol-2696/paper_130.pdf.

Martin Andrews and Sam Witteveen. Unsupervised natural question answering with a small model. In *Proceedings of the Second Workshop on Fact Extraction and VERification (FEVER)*, pages 34–38, Hong Kong, China, November 2019. doi:10.18653/v1/D19-6606. URL https://www.aclweb.org/anthology/D19-6606.

Jatin Arora, Sumit Agrawal, Pawan Goyal, and Sayan Pathak. Extracting entities of interest from comparative product reviews. In *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*, CIKM '17, pages 1975–1978, New York, NY, USA, 2017. ACM. ISBN 978-1-4503-4918-5. doi:10.1145/3132847.3133141. URL http://doi.acm.org/10.1145/3132847.3133141.

Sören Auer, Christian Bizer, Georgi Kobilarov, Jens Lehmann, Richard Cyganiak, and Zachary G. Ives. Dbpedia: A nucleus for a web of open data. In Karl Aberer, Key-Sun Choi, Natasha Fridman Noy, Dean Allemang, Kyung-Il Lee, Lyndon J. B. Nixon, Jennifer Golbeck, Peter Mika, Diana Maynard, Riichiro Mizoguchi, Guus Schreiber, and Philippe Cudré-Mauroux, editors, *The Semantic Web, 6th International Semantic Web Conference, 2nd Asian Semantic Web Conference, ISWC 2007 + ASWC 2007, Busan, Korea, November 11-15, 2007*, volume 4825 of *Lecture Notes in Computer Science*, pages 722–735. Springer, 2007. doi:10.1007/978-3-540-76298-0_52. URL https://doi.org/10.1007/978-3-540-76298-0_52.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015. URL http://arxiv.org/abs/1409.0473.

Ivana Balazevic, Carl Allen, and Timothy M. Hospedales. Hypernetwork knowledge graph embeddings. *CoRR*, abs/1808.07018, 2018. URL http://arxiv.org/abs/1808.07018.

Ivana Balazevic, Carl Allen, and Timothy Hospedales. TuckER: Tensor factorization for knowledge graph completion. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 5185–5194, Hong Kong, China, November 2019. Association for Computational Linguistics. doi:10.18653/v1/D19-1522. URL https://aclanthology.org/D19-1522.

Casey Battaglino, Grey Ballard, and Tamara G. Kolda. A practical randomized CP tensor decomposition. *SIAM J. Matrix Anal. Appl.*, 39(2):876–901, 2018. doi:10.1137/17M1112303. URL https://doi.org/10.1137/17M1112303.

Jacob Biamonte and Ville Bergholm. Tensor networks in a nutshell, 2017.

Chris Biemann. Chinese whispers - an efficient graph clustering algorithm and its application to natural language processing problems. In *Proceedings of TextGraphs: the First Workshop on Graph Based Methods for Natural Language Processing*, pages 73–80, New York City, June 2006. Association for Computational Linguistics. URL https://aclanthology.org/W06-3812.

Lukas Biewald. Experiment tracking with weights and biases, 2020. URL https://www.wandb.com/. Software available from wandb.com.

Christopher M. Bishop. *Pattern recognition and machine learning, 5th Edition*. Information science and statistics. Springer, 2007. ISBN 9780387310732. URL https://www.worldcat.org/oclc/71008143.

Kurt Bollacker, Colin Evans, Praveen Paritosh, Tim Sturge, and Jamie Taylor. Freebase: A collaboratively created graph database for structuring human knowledge. pages 1247–1250, 01 2008a. doi:10.1145/1376616.1376746.

Kurt D. Bollacker, Colin Evans, Praveen K. Paritosh, Tim Sturge, and Jamie Taylor. Freebase: a collaboratively created graph database for structuring human knowledge. In Jason Tsong-Li Wang, editor, *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2008, Vancouver, BC, Canada, June 10-12, 2008*, pages 1247–1250. ACM, 2008b. doi:10.1145/1376616.1376746. URL https://doi.org/10.1145/1376616.1376746.

Alexander Bondarenko, Pavel Braslavski, Michael Völske, Rami Aly, Maik Fröbe, Alexander Panchenko, Chris Biemann, Benno Stein, and Matthias Hagen. Comparative web search questions. pages 52–60, 01 2020a. doi:10.1145/3336191.3371848.

Alexander Bondarenko, Maik Fröbe, Meriem Beloucif, Lukas Gienapp, Yamen Ajjour, Alexander Panchenko, Chris Biemann, Benno Stein, Henning Wachsmuth, Martin Potthast, and Matthias Hagen. Overview of touché 2020: Argument retrieval: Extended abstract. In *Experimental IR Meets Multilinguality, Multimodality, and Interaction: 11th International Conference of the CLEF Association, CLEF 2020, Thessaloniki, Greece, September 22–25, 2020, Proceedings*, page 384–395, Berlin, Heidelberg, 2020b. Springer-Verlag. ISBN 978-3-

030-58218-0. doi:10.1007/978-3-030-58219-7_26. URL https://doi.org/10.1007/978-3-030-58219-7_26.

Alexander Bondarenko, Maik Fröbe, Meriem Beloucif, Lukas Gienapp, Yamen Ajjour, Alexander Panchenko, Chris Biemann, Benno Stein, Henning Wachsmuth, Martin Potthast, and Matthias Hagen. Overview of Touché 2020: Argument Retrieval. In Linda Cappellato, Carsten Eickhoff, Nicola Ferro, and Aurélie Névéol, editors, *Working Notes Papers of the CLEF 2020 Evaluation Labs*, volume 2696 of *CEUR Workshop Proceedings*, September 2020c. URL http://ceur-ws.org/Vol-2696/.

Alexander Bondarenko, Lukas Gienapp, Maik Fröbe, Meriem Beloucif, Yamen Ajjour, Alexander Panchenko, Chris Biemann, Benno Stein, Henning Wachsmuth, Martin Potthast, and Matthias Hagen. Overview of Touché 2021: Argument Retrieval. In Djoerd Hiemstra, Maria-Francine Moens, Josiane Mothe, Raffaele Perego, Martin Potthast, and Fabrizio Sebastiani, editors, *Advances in Information Retrieval. 43rd European Conference on IR Research (ECIR 2021)*, volume 12036 of *Lecture Notes in Computer Science*, pages 574–582, Berlin Heidelberg New York, March 2021. Springer. doi:10.1007/978-3-030-72240-1_67. URL https://link.springer.com/chapter/10.1007/978-3-030-72240-1_67.

Alexander Bondarenko, Maik Fröbe, Johannes Kiesel, Shahbaz Syed, Timon Gurcke, Meriem Beloucif, Alexander Panchenko, Chris Biemann, Benno Stein, Henning Wachsmuth, Martin Potthast, and Matthias Hagen. Overview of touché 2022: Argument retrieval. In *Experimental IR Meets Multilinguality, Multimodality, and Interaction: 13th International Conference of the CLEF Association, CLEF 2022, Bologna, Italy, September 5–8, 2022, Proceedings*, page 311–336, Berlin, Heidelberg, 2022. Springer-Verlag. ISBN 978-3-031-13642-9. doi:10.1007/978-3-031-13643-6_21. URL https://doi.org/10.1007/978-3-031-13643-6_21.

Antoine Bordes, Nicolas Usunier, Alberto García-Durán, Jason Weston, and Oksana Yakhnenko. Translating embeddings for modeling multi-relational data. In Christopher J. C. Burges, Léon Bottou, Zoubin Ghahramani, and Kilian Q. Weinberger, editors, *Advances in Neural Information Processing Systems 26: 27th Annual Conference on Neural Information Processing Systems 2013. Proceedings of a meeting held December 5-8, 2013, Lake Tahoe, Nevada, United States*, pages 2787–2795, 2013. URL https://proceedings.neurips.cc/paper/2013/hash/1cecc7a77928ca8133fa24680a88d2f9-Abstract.html.

Antoine Bordes, Nicolas Usunier, Sumit Chopra, and Jason Weston. Large-scale simple question answering with memory networks. *CoRR*, abs/1506.02075, 2015. URL http://dblp.uni-trier.de/db/journals/corr/corr1506.html#BordesUCW15.

Liora Braunstain, Oren Kurland, David Carmel, Idan Szpektor, and Anna Shtok. Supporting human answers for advice-seeking questions in CQA sites. In Nicola Ferro, Fabio Crestani, Marie-Francine Moens, Josiane Mothe, Fabrizio Silvestri, Giorgio Maria Di Nunzio, Claudia Hauff, and Gianmaria Silvello, editors, *Advances in Information Retrieval - 38th European Conference on IR Research,*

ECIR 2016, Padua, Italy, March 20-23, 2016. Proceedings, volume 9626 of *Lecture Notes in Computer Science*, pages 129–141. Springer, 2016. doi:10.1007/978-3-319-30671-1_10. URL https://doi.org/10.1007/978-3-319-30671-1_10.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. *CoRR*, abs/2005.14165, 2020. URL https://arxiv.org/abs/2005.14165.

Christopher Burges, Robert Ragno, and Quoc Le. Learning to Rank with Nonsmooth Cost Functions. In B. Schölkopf, J. Platt, and T. Hoffman, editors, *Advances in Neural Information Processing Systems*, volume 19. MIT Press, 2007. URL https://proceedings.neurips.cc/paper/2006/file/af44c4c56f385c43f2529f9b1b018f6a-Paper.pdf.

Mikhail Burtsev, Alexander Seliverstov, Rafael Airapetyan, Mikhail Arkhipov, Dilyara Baymurzina, Nickolay Bushkov, Olga Gureenkova, Taras Khakhulin, Yuri Kuratov, Denis Kuznetsov, Alexey Litinsky, Varvara Logacheva, Alexey Lymar, Valentin Malykh, Maxim Petrov, Vadim Polulyakh, Leonid Pugachev, Alexey Sorokin, Maria Vikhreva, and Marat Zaynutdinov. DeepPavlov: Open-source library for dialogue systems. In *Proceedings of ACL 2018, System Demonstrations*, pages 122–127, Melbourne, Australia, July 2018. Association for Computational Linguistics. doi:10.18653/v1/P18-4021. URL https://aclanthology.org/P18-4021.

Viktoria Chekalina and Alexander Panchenko. Retrieving comparative arguments using deep pre-trained language models and NLU. In Linda Cappellato, Carsten Eickhoff, Nicola Ferro, and Aurélie Névéol, editors, *Working Notes of CLEF 2020 - Conference and Labs of the Evaluation Forum, Thessaloniki, Greece, September 22-25, 2020*, volume 2696 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2020. URL http://ceur-ws.org/Vol-2696/paper_210.pdf.

Viktoriia Chekalina, Alexander Bondarenko, Chris Biemann, Meriem Beloucif, Varvara Logacheva, and Alexander Panchenko. Which is better for deep learning: Python or MATLAB? Answering Comparative Questions in Natural Language. In *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: System Demonstrations*, pages 302–311, Online, April 2021. Association for Computational Linguistics. URL https://www.aclweb.org/anthology/2021.eacl-demos.36.

Ting Chen, Ji Lin, Tian Lin, Song Han, Chong Wang, and Denny Zhou. Adaptive mixture of low-rank factorizations for compact neural modeling, 2019. URL https://openreview.net/forum?id=r1xFE3Rqt7.

Hao Cheng, Yaoliang Yu, Xinhua Zhang, Eric Xing, and Dale Schuurmans. Scalable and sound low-rank tensor learning. In Arthur Gretton and Christian C. Robert, editors, *Proceedings of the 19th International Conference on Artificial Intelligence and Statistics*, volume 51 of *Proceedings of Machine Learning Research*, pages 1114–1123, Cadiz, Spain, 09–11 May 2016. PMLR. URL https://proceedings.mlr.press/v51/cheng16.html.

Andrzej Cichocki, Namgil Lee, Ivan V. Oseledets, Anh Huy Phan, Qibin Zhao, and Danilo P. Mandic. Low-rank tensor networks for dimensionality reduction and large-scale optimization problems: Perspectives and challenges PART 1. *CoRR*, abs/1609.00893, 2016. URL http://arxiv.org/abs/1609.00893.

Maximin Coavoux, Hady Elsahar, and Matthias Gallé. Unsupervised aspect-based multi-document abstractive summarization. In *Proceedings of the 2nd Workshop on New Frontiers in Summarization*, pages 42–47, Hong Kong, China, November 2019. Association for Computational Linguistics. doi:10.18653/v1/D19-5405. URL https://aclanthology.org/D19-5405.

Alexis Conneau, Douwe Kiela, Holger Schwenk, Loïc Barrault, and Antoine Bordes. Supervised learning of universal sentence representations from natural language inference data. *CoRR*, abs/1705.02364, 2017. URL http://arxiv.org/abs/1705.02364.

Misha Denil, Babak Shakibi, Laurent Dinh, Marc'Aurelio Ranzato, and Nando de Freitas. Predicting parameters in deep learning. *CoRR*, abs/1306.0543, 2013. URL http://arxiv.org/abs/1306.0543.

Tim Dettmers, Pasquale Minervini, Pontus Stenetorp, and Sebastian Riedel. Convolutional 2d knowledge graph embeddings. *CoRR*, abs/1707.01476, 2017. URL http://arxiv.org/abs/1707.01476.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics. doi:10.18653/v1/N19-1423. URL https://aclanthology.org/N19-1423.

Daniel M. Dunlavy, Tamara G. Kolda, and Evrim Acar. Temporal link prediction using matrix and tensor factorizations. *ACM Trans. Knowl. Discov. Data*, 5(2): 10:1–10:27, 2011. doi:10.1145/1921632.1921636. URL https://doi.org/10.1145/1921632.1921636.

Ondřej Dušek and Filip Jurčíček. A context-aware natural language generator for dialogue systems. In *Proceedings of the 17th Annual Meeting of the Special Interest Group on Discourse and Dialogue*, pages 185–190, Los Angeles, September 2016. Association for Computational Linguistics. doi:10.18653/v1/W16-3622. URL https://www.aclweb.org/anthology/W16-3622.

Ali Edalati, Marzieh S. Tahaei, Ahmad Rashid, Vahid Partovi Nia, James J. Clark, and Mehdi Rezagholizadeh. Kronecker decomposition for GPT compression. *CoRR*, abs/2110.08152, 2021. URL https://arxiv.org/abs/2110.08152.

Qiaoqing Fan and Yu Fang. An answer summarization method based on keyword extraction. In *BIO Web of Conferences*, volume 8, page 03015. EDP Sciences, 2017.

Markus Freitag and Scott Roy. Unsupervised natural language generation with denoising autoencoders. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 3922–3929, Brussels, Belgium, October-November 2018. Association for Computational Linguistics. doi:10.18653/v1/D18-1426. URL https://www.aclweb.org/anthology/D18-1426.

Jerome Friedman. Stochastic Gradient Boosting. *Computational Statistics & Data Analysis*, 38:367–378, 02 2002. doi:10.1016/S0167-9473(01)00065-2.

Michael Fromm, Evgeniy Faerman, and Thomas Seidl. TACAM: topic and context aware argument mining. pages 99–106, 2019. doi:10.1145/3350546.3352506. URL https://doi.org/10.1145/3350546.3352506.

Murthy Ganapathibhotla and Bing Liu. Mining opinions in comparative sentences. In Donia Scott and Hans Uszkoreit, editors, *COLING 2008, 22nd International Conference on Computational Linguistics, Proceedings of the Conference, 18-22 August 2008, Manchester, UK*, pages 241–248, 2008. URL https://aclanthology.org/C08-1031/.

Leo Gao, Stella Biderman, Sid Black, Laurence Golding, Travis Hoppe, Charles Foster, Jason Phang, Horace He, Anish Thite, Noa Nabeshima, Shawn Presser, and Connor Leahy. The pile: An 800gb dataset of diverse text for language modeling. *CoRR*, abs/2101.00027, 2021. URL https://arxiv.org/abs/2101.00027.

Timur Garipov, Dmitry Podoprikhin, Alexander Novikov, and Dmitry P. Vetrov. Ultimate tensorization: compressing convolutional and FC layers alike. *CoRR*, abs/1611.03214, 2016. URL http://arxiv.org/abs/1611.03214.

Timur Garipov, Pavel Izmailov, Dmitrii Podoprikhin, Dmitry P. Vetrov, and Andrew Gordon Wilson. Loss Surfaces, Mode Connectivity, and Fast Ensembling of DNNs. In *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018*, pages 8803–8812, 2018. URL https://proceedings.neurips.cc/paper/2018/hash/be3087e74e9100d4bc4c6268cdbe8456-Abstract.html.

Aaron Gokaslan and Vanya Cohen. Openwebtext corpus, 2019. http://Skylion007.github.io/OpenWebTextCorpus.

Ian J. Goodfellow and Oriol Vinyals. Qualitatively characterizing neural network optimization problems, 2015. URL http://arxiv.org/abs/1412.6544.

Priya Goyal, Piotr Dollár, Ross B. Girshick, Pieter Noordhuis, Lukasz Wesolowski, Aapo Kyrola, Andrew Tulloch, Yangqing Jia, and Kaiming He. Accurate, large minibatch SGD: training imagenet in 1 hour. *CoRR*, abs/1706.02677, 2017. URL http://arxiv.org/abs/1706.02677.

Richard A. Harshman, Paul E. Green, Yoram Wind, and Margaret E. Lundy. A model for the analysis of asymmetric data in marketing research. *Marketing Science*, 1(2):205–242, 1982. ISSN 07322399, 1526548X. URL http://www.jstor.org/stable/183857.

Benjamin Hawks, Javier M. Duarte, Nicholas J. Fraser, Alessandro Pappalardo, Nhan Tran, and Yaman Umuroglu. Ps and qs: Quantization-aware pruning for efficient low latency neural network inference. *Frontiers Artif. Intell.*, 4:676564, 2021. doi:10.3389/frai.2021.676564. URL https://doi.org/10.3389/frai.2021.676564.

Karl Moritz Hermann, Tomas Kocisky, Edward Grefenstette, Lasse Espeholt, Will Kay, Mustafa Suleyman, and Phil Blunsom. Teaching machines to read and comprehend. In C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 28. Curran Associates, Inc., 2015. URL https://proceedings.neurips.cc/paper/2015/file/afdec7005cc9f14302cd0474fd0f3c96-Paper.pdf.

Geoffrey Hinton, Oriol Vinyals, and Jeffrey Dean. Distilling the knowledge in a neural network. In *NIPS Deep Learning and Representation Learning Workshop*, 2015a. URL http://arxiv.org/abs/1503.02531.

Geoffrey E. Hinton, Oriol Vinyals, and Jeffrey Dean. Distilling the knowledge in a neural network. *CoRR*, abs/1503.02531, 2015b. URL http://arxiv.org/abs/1503.02531.

F. L. Hitchcock. The expression of a tensor or a polyadic as a sum of products. *J. Math. Phys*, 6(1):164–189, 1927.

David Hong, Tamara G. Kolda, and Jed A. Duersch. Generalized canonical polyadic tensor decomposition. *SIAM Review*, 62(1):133–163, Jan 2020. ISSN 1095-7200. doi:10.1137/18m1203626. URL http://dx.doi.org/10.1137/18M1203626.

Oleksii Hrinchuk, Valentin Khrulkov, Leyla Mirvakhabova, Elena D. Orlova, and Ivan V. Oseledets. Tensorized embedding layers. In Trevor Cohn, Yulan He, and Yang Liu, editors, *Findings of the Association for Computational Linguistics: EMNLP 2020, Online Event, 16-20 November 2020*, volume EMNLP 2020 of *Findings of ACL*, pages 4847–4860. Association for Computational Linguistics, 2020. doi:10.18653/v1/2020.findings-emnlp.436. URL https://doi.org/10.18653/v1/2020.findings-emnlp.436.

Wan-Ting Hsu, Chieh-Kai Lin, Ming-Ying Lee, Kerui Min, Jing Tang, and Min Sun. A unified model for extractive and abstractive summarization using inconsistency loss. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 132–141, Melbourne, Australia,

July 2018. Association for Computational Linguistics. doi:10.18653/v1/P18-1013. URL https://www.aclweb.org/anthology/P18-1013.

Yen-Chang Hsu, Ting Hua, Sungen Chang, Qian Lou, Yilin Shen, and Hongxia Jin. Language model compression with weighted low-rank factorization, 2022. URL https://arxiv.org/abs/2207.00112.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models. In *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*. OpenReview.net, 2022. URL https://openreview.net/forum?id=nZeVKeeFYf9.

Peng Hu, Xi Peng, Hongyuan Zhu, Mohamed M. Sabry Aly, and Jie Lin. OPQ: compressing deep neural networks with one-shot pruning-quantization. In *Thirty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2021, Thirty-Third Conference on Innovative Applications of Artificial Intelligence, IAAI 2021, The Eleventh Symposium on Educational Advances in Artificial Intelligence, EAAI 2021, Virtual Event, February 2-9, 2021*, pages 7780–7788. AAAI Press, 2021. URL https://ojs.aaai.org/index.php/AAAI/article/view/16950.

Ting Hua, Yen-Chang Hsu, Felicity Wang, Qian Lou, Yilin Shen, and Hongxia Jin. Numerical optimizations for weighted low-rank estimation on language model. *arXiv preprint arXiv:2211.09718*, 2022.

Po-Sen Huang, Xiaodong He, Jianfeng Gao, Li Deng, Alex Acero, and Larry Heck. Learning deep structured semantic models for web search using clickthrough data. In *Proceedings of the 22nd ACM International Conference on Information and Knowledge Management*, CIKM '13, page 2333–2338, New York, NY, USA, 2013. Association for Computing Machinery. ISBN 9781450322638. doi:10.1145/2505515.2505665. URL https://doi.org/10.1145/2505515.2505665.

Xiao Huang, Jingyuan Zhang, Dingcheng Li, and Ping Li. Knowledge graph embedding based question answering. In *Proceedings of the Twelfth ACM International Conference on Web Search and Data Mining*, WSDM '19, page 105–113, New York, NY, USA, 2019. Association for Computing Machinery. ISBN 9781450359405. doi:10.1145/3289600.3290956. URL https://doi.org/10.1145/3289600.3290956.

Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Quantized neural networks: Training neural networks with low precision weights and activations. *CoRR*, abs/1609.07061, 2016. URL http://arxiv.org/abs/1609.07061.

Masaru Isonuma, Junichiro Mori, and Ichiro Sakata. Unsupervised neural single-document summarization of reviews via learning latent discourse structure and its ranking. pages 2142–2152, 2019. doi:10.18653/v1/p19-1206. URL https://doi.org/10.18653/v1/p19-1206.

Shaoxiong Ji, Shirui Pan, Erik Cambria, Pekka Marttinen, and Philip S. Yu. A survey on knowledge graphs: Representation, acquisition and applications. *CoRR*, abs/2002.00388, 2020. URL https://arxiv.org/abs/2002.00388.

Yuwang Ji, Qiang Wang, Xuan Li, and Jie Liu. A survey on tensor techniques and applications in machine learning. *IEEE Access*, 7:162950–162990, 2019. doi:10.1109/ACCESS.2019.2949814.

Xiaoqi Jiao, Yichun Yin, Lifeng Shang, Xin Jiang, Xiao Chen, Linlin Li, Fang Wang, and Qun Liu. Tinybert: Distilling BERT for natural language understanding. In Trevor Cohn, Yulan He, and Yang Liu, editors, *Findings of the Association for Computational Linguistics: EMNLP 2020, Online Event, 16-20 November 2020*, volume EMNLP 2020 of *Findings of ACL*, pages 4163–4174. Association for Computational Linguistics, 2020. doi:10.18653/v1/2020.findings-emnlp.372. URL https://doi.org/10.18653/v1/2020.findings-emnlp.372.

Nitin Jindal and Bing Liu. Mining comparative sentences and relations. In *Proceedings, The Twenty-First National Conference on Artificial Intelligence and the Eighteenth Innovative Applications of Artificial Intelligence Conference, July 16-20, 2006, Boston, Massachusetts, USA*, pages 1331–1336. AAAI Press, 2006. URL http://www.aaai.org/Library/AAAI/2006/aaai06-209.php.

Jeff Johnson, Matthijs Douze, and Hervé Jégou. Billion-scale similarity search with GPUs. *IEEE Transactions on Big Data*, 7(3):535–547, 2019.

Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B. Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models. *CoRR*, abs/2001.08361, 2020. URL https://arxiv.org/abs/2001.08361.

Seyed Mehran Kazemi and David Poole. Simple embedding for link prediction in knowledge graphs. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, NIPS'18, page 4289–4300, Red Hook, NY, USA, 2018. Curran Associates Inc.

Nitish Shirish Keskar, Bryan McCann, Lav R Varshney, Caiming Xiong, and Richard Socher. Ctrl: A conditional transformer language model for controllable generation. *arXiv preprint arXiv:1909.05858*, 2019.

Wiltrud Kessler and Jonas Kuhn. A gold standard corpus of comparisons in product reviews. In *LREC-2014*, Reykjavik, Iceland, May 2014.

Urvashi Khandelwal, Kevin Clark, Dan Jurafsky, and Lukasz Kaiser. Sample efficient text summarization using a single pre-trained transformer. *CoRR*, abs/1905.08836, 2019. URL http://arxiv.org/abs/1905.08836.

Omar Khattab and Matei Zaharia. Colbert: Efficient and effective passage search via contextualized late interaction over BERT. volume abs/2004.12832, 2020. URL https://arxiv.org/abs/2004.12832.

Valentin Khrulkov, Oleksii Hrinchuk, Leyla Mirvakhabova, and Ivan V. Oseledets. Tensorized embedding layers for efficient model compression. volume abs/1901.10787, 2019. URL http://arxiv.org/abs/1901.10787.

Ilker Kocabas, Bekir Dincer, and Bahar Karaoğlan. A Nonparametric Term Weighting Method for Information Retrieval Based on Measuring the Divergence from Independence. *Information Retrieval*, pages 1–24, 05 2013. doi:10.1007/s10791-013-9225-4.

Tamara G. Kolda and Brett W. Bader. Tensor decompositions and applications. *SIAM Rev.*, 51(3):455–500, August 2009a. ISSN 0036-1445. doi:10.1137/07070111X. URL https://doi.org/10.1137/07070111X.

Tamara G. Kolda and Brett W. Bader. Tensor decompositions and applications. *SIAM Review*, 51(3):455–500, 2009b. doi:10.1137/07070111X. URL https://doi.org/10.1137/07070111X.

Timothee Lacroix, Nicolas Usunier, and Guillaume Obozinski. Canonical tensor decomposition for knowledge base completion. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 2863–2872. PMLR, 10–15 Jul 2018. URL https://proceedings.mlr.press/v80/lacroix18a.html.

Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. ALBERT: A lite BERT for self-supervised learning of language representations. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020. URL https://openreview.net/forum?id=H1eA7AEtvS.

Adam Lerer, Ledell Wu, Jiajun Shen, Timothée Lacroix, Luca Wehrstedt, Abhijit Bose, and Alex Peysakhovich. Pytorch-biggraph: A large scale graph embedding system. In Ameet Talwalkar, Virginia Smith, and Matei Zaharia, editors, *Proceedings of Machine Learning and Systems 2019, MLSys 2019, Stanford, CA, USA, March 31 - April 2, 2019*. mlsys.org, 2019. URL https://proceedings.mlsys.org/book/282.pdf.

Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Veselin Stoyanov, and Luke Zettlemoyer. BART: denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. In Dan Jurafsky, Joyce Chai, Natalie Schluter, and Joel R. Tetreault, editors, *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, ACL 2020, Online, July 5-10, 2020*, pages 7871–7880. Association for Computational Linguistics, 2020. doi:10.18653/v1/2020.acl-main.703. URL https://doi.org/10.18653/v1/2020.acl-main.703.

Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. Pruning filters for efficient convnets. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017. URL https://openreview.net/forum?id=rJqFGTslg.

Xin Li and Dan Roth. Learning question classifiers. 2002. URL https://aclanthology.org/C02-1150.

Chin-Yew Lin. ROUGE: A package for automatic evaluation of summaries. In *Text Summarization Branches Out*, pages 74–81, Barcelona, Spain, July 2004. Association for Computational Linguistics. URL https://aclanthology.org/W04-1013.

Marina Litvak and Mark Last. Graph-based keyword extraction for single-document summarization. In *Coling 2008: Proceedings of the workshop Multi-source Multilingual Information Extraction and Summarization*, pages 17–24, Manchester, UK, August 2008. Coling 2008 Organizing Committee. URL https://aclanthology.org/W08-1404.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized BERT pretraining approach. *CoRR*, abs/1907.11692, 2019. URL http://arxiv.org/abs/1907.11692.

Charles Van Loan and Nikos Pitsianis. Approximation with kronecker products. 1992.

Varvara Logacheva, Daryna Dementieva, Sergey Ustyantsev, Daniil Moskovskiy, David Dale, Irina Krotova, Nikita Semenov, and Alexander Panchenko. Paradetox: Detoxification with parallel data. In Smaranda Muresan, Preslav Nakov, and Aline Villavicencio, editors, *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2022, Dublin, Ireland, May 22-27, 2022*, pages 6804–6818. Association for Computational Linguistics, 2022. doi:10.18653/v1/2022.acl-long.469. URL https://doi.org/10.18653/v1/2022.acl-long.469.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019. URL https://openreview.net/forum?id=Bkg6RiCqY7.

Sean MacAvaney. OpenNIR: A Complete Neural Ad-Hoc Ranking Pipeline. In James Caverlee, Xia (Ben) Hu, Mounia Lalmas, and Wei Wang, editors, *WSDM '20: The Thirteenth ACM International Conference on Web Search and Data Mining, Houston, TX, USA, February 3-7, 2020*, pages 845–848. ACM, 2020. doi:10.1145/3336191.3371864. URL https://doi.org/10.1145/3336191.3371864.

Sean MacAvaney, Andrew Yates, Arman Cohan, and Nazli Goharian. CEDR: Contextualized Embeddings for Document Ranking. pages 1101–1104, 2019. doi:10.1145/3331184.3331317. URL https://doi.org/10.1145/3331184.3331317.

Craig Macdonald and Nicola Tonellotto. Declarative Experimentation in Information Retrieval using PyTerrier. In Krisztian Balog, Vinay Setty, Christina Lioma, Yiqun Liu, Min Zhang, and Klaus Berberich, editors, *ICTIR '20: The 2020 ACM*

SIGIR International Conference on the Theory of Information Retrieval, Virtual Event, Norway, September 14-17, 2020, pages 161–168. ACM, 2020. URL https://dl.acm.org/doi/10.1145/3409256.3409829.

Craig Macdonald, Nicola Tonellotto, and Sean MacAvaney. IR from bag-of-words to BERT and beyond through practical experiments. In Gianluca Demartini, Guido Zuccon, J. Shane Culpepper, Zi Huang, and Hanghang Tong, editors, *CIKM '21: The 30th ACM International Conference on Information and Knowledge Management, Virtual Event, Queensland, Australia, November 1 - 5, 2021*, page 4861. ACM, 2021. doi:10.1145/3459637.3482028. URL https://doi.org/10.1145/3459637.3482028.

Chirantana Mallick, Ajit Kumar Das, Madhurima Dutta, Asit Kumar Das, and Apurba Sarkar. Graph-based text summarization using modified textrank. In *Soft Computing in Data Analytics*, pages 137–146. Springer, 2019.

Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. Pointer sentinel mixture models. *CoRR*, abs/1609.07843, 2016. URL http://arxiv.org/abs/1609.07843.

Paul Michel, Omer Levy, and Graham Neubig. Are sixteen heads really better than one? pages 14014–14024, 2019. URL https://proceedings.neurips.cc/paper/2019/hash/2c601ad9d2ff9bc8b282670cdd54f69f-Abstract.html.

Rada Mihalcea. Graph-based ranking algorithms for sentence extraction, applied to text summarization. In *Proceedings of the ACL Interactive Poster and Demonstration Sessions*, pages 170–173, 2004.

George A. Miller. WordNet: A lexical database for English. In *Speech and Natural Language: Proceedings of a Workshop Held at Harriman, New York, February 23-26, 1992*, 1992. URL https://aclanthology.org/H92-1116.

Bhaskar Mitra, Fernando Diaz, and Nick Craswell. Learning to match using local and distributed representations of text for web search. In Rick Barrett, Rick Cummings, Eugene Agichtein, and Evgeniy Gabrilovich, editors, *Proceedings of the 26th International Conference on World Wide Web, WWW 2017, Perth, Australia, April 3-7, 2017*, pages 1291–1299. ACM, 2017. doi:10.1145/3038912.3052579. URL https://doi.org/10.1145/3038912.3052579.

Ramesh Nallapati, Bowen Zhou, Cicero dos Santos, Caglar Gulcehre, and Bing Xiang. Abstractive text summarization using sequence-to-sequence RNNs and beyond. In *Proceedings of the 20th SIGNLL Conference on Computational Natural Language Learning*, pages 280–290, Berlin, Germany, August 2016. Association for Computational Linguistics. doi:10.18653/v1/K16-1028. URL https://aclanthology.org/K16-1028.

Shashi Narayan, Shay B. Cohen, and Mirella Lapata. Don't give me the details, just the summary! topic-aware convolutional neural networks for extreme summarization. In Ellen Riloff, David Chiang, Julia Hockenmaier, and Jun'ichi Tsujii,

editors, *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, Brussels, Belgium, October 31 - November 4, 2018*, pages 1797–1807. Association for Computational Linguistics, 2018. doi:10.18653/v1/d18-1206. URL https://doi.org/10.18653/v1/d18-1206.

Deepak Nathani, Jatin Chauhan, Charu Sharma, and Manohar Kaul. Learning attention-based embeddings for relation prediction in knowledge graphs. *CoRR*, abs/1906.01195, 2019. URL http://arxiv.org/abs/1906.01195.

Dai Quoc Nguyen, Tu Dinh Nguyen, Dat Quoc Nguyen, and Dinh Phung. A novel embedding model for knowledge base completion based on convolutional neural network. In *Proceedings of the 16th Annual Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT)*, pages 327–333, 2018.

Tri Nguyen, Mir Rosenberg, Xia Song, Jianfeng Gao, Saurabh Tiwary, Rangan Majumder, and Li Deng. MS MARCO: A human generated machine reading comprehension dataset. *CoRR*, abs/1611.09268, 2016. URL http://arxiv.org/abs/1611.09268.

Maximilian Nickel, Volker Tresp, and Hans-Peter Kriegel. A three-way model for collective learning on multi-relational data. In *Proceedings of the 28th International Conference on International Conference on Machine Learning*, ICML'11, page 809–816, Madison, WI, USA, 2011. Omnipress. ISBN 9781450306195.

Kyosuke Nishida, Itsumi Saito, Kosuke Nishida, Kazutoshi Shinoda, Atsushi Otsuka, Hisako Asano, and Junji Tomita. Multi-style generative reading comprehension. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 2273–2284, Florence, Italy, July 2019. Association for Computational Linguistics. doi:10.18653/v1/P19-1220. URL https://www.aclweb.org/anthology/P19-1220.

Alexander Novikov, Dmitrii Podoprikhin, Anton Osokin, and Dmitry P Vetrov. Tensorizing neural networks. In C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 28. Curran Associates, Inc., 2015. URL https://proceedings.neurips.cc/paper/2015/file/6855456e2fe46a9d49d3d3af4f57443d-Paper.pdf.

I. V. Oseledets. Approximation of $2^d \times 2^d$ matrices using tensor decomposition. *SIAM Journal on Matrix Analysis and Applications*, 31(4):2130–2145, 2010. doi:10.1137/090757861. URL https://doi.org/10.1137/090757861.

I. V. Oseledets. Tensor-train decomposition. *SIAM Journal on Scientific Computing*, 33(5):2295–2317, 2011a. doi:10.1137/090752286. URL https://doi.org/10.1137/090752286.

Ivan V. Oseledets. Tensor-train decomposition. *SIAM J. Sci. Comput.*, 33(5):2295–2317, 2011b. doi:10.1137/090752286. URL https://doi.org/10.1137/090752286.

Alexander Panchenko, Eugen Ruppert, Stefano Faralli, Simone P. Ponzetto, and Chris Biemann. Building a web-scale dependency-parsed corpus from CommonCrawl. In *LREC-2018*, Miyazaki, Japan, May 2018. URL https://www.aclweb.org/anthology/L18-1286.

Alexander Panchenko, Alexander Bondarenko, Mirco Franzek, Matthias Hagen, and Chris Biemann. Categorizing comparative sentences. In *Proceedings of the 6th Workshop on Argument Mining*, pages 136–145, Florence, Italy, August 2019. doi:10.18653/v1/W19-4516. URL https://www.aclweb.org/anthology/W19-4516.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019a. URL http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Z. Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In Hanna M. Wallach, Hugo Larochelle, Alina Beygelzimer, Florence AlcheBuc, Emily B. Fox, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pages 8024–8035, 2019b. URL https://proceedings.neurips.cc/paper/2019/hash/bdbca288fee7f92f2bfa9f7012727740-Abstract.html.

David A. Patterson, Joseph Gonzalez, Quoc V. Le, Chen Liang, Lluis-Miquel Munguia, Daniel Rothchild, David R. So, Maud Texier, and Jeff Dean. Carbon emissions and large neural network training. *CoRR*, abs/2104.10350, 2021. URL https://arxiv.org/abs/2104.10350.

Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. Deep Contextualized Word Representations. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2018*, pages 2227–2237. Association for Computational Linguistics, 2018. doi:10.18653/v1/n18-1202. URL https://doi.org/10.18653/v1/n18-1202.

Hoang Pham Minh, Nguyen Nguyen Xuan, and Son Tran Thai. Tt-vit: Vision transformer compression using tensor-train decomposition. In Ngoc Thanh Nguyen,

Yannis Manolopoulos, Richard Chbeir, Adrianna Kozierkiewicz, and Bogdan Trawiński, editors, *Computational Collective Intelligence*, pages 755–767, Cham, 2022. Springer International Publishing. ISBN 978-3-031-16014-1.

Martin Potthast, Matthias Hagen, Benno Stein, Jan Graßegger, Maximilian Michel, Martin Tippmann, and Clement Welsch. ChatNoir: A Search Engine for the ClueWeb09 Corpus. In Bill Hersh, Jamie Callan, Yoelle Maarek, and Mark Sanderson, editors, *35th International ACM Conference on Research and Development in Information Retrieval (SIGIR 2012)*, page 1004. ACM, August 2012. ISBN 978-1-4503-1472-5. doi:10.1145/2348283.2348429.

Martin Potthast, Tim Gollub, Matti Wiegmann, and Benno Stein. TIRA Integrated Research Architecture. In Nicola Ferro and Carol Peters, editors, *Information Retrieval Evaluation in a Changing World*, The Information Retrieval Series. Springer, Berlin Heidelberg New York, September 2019. ISBN 978-3-030-22948-1. doi:10.1007/978-3-030-22948-1_5.

Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. 2019a.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. *OpenAI Blog*, 1(8), 2019b.

Jack W. Rae, Sebastian Borgeaud, Trevor Cai, Katie Millican, Jordan Hoffmann, H. Francis Song, John Aslanides, Sarah Henderson, Roman Ring, Susannah Young, Eliza Rutherford, Tom Hennigan, Jacob Menick, Albin Cassirer, Richard Powell, George van den Driessche, Lisa Anne Hendricks, Maribeth Rauh, Po-Sen Huang, Amelia Glaese, Johannes Welbl, Sumanth Dathathri, Saffron Huang, Jonathan Uesato, John Mellor, Irina Higgins, Antonia Creswell, Nat McAleese, Amy Wu, Erich Elsen, Siddhant M. Jayakumar, Elena Buchatskaya, David Budden, Esme Sutherland, Karen Simonyan, Michela Paganini, Laurent Sifre, Lena Martens, Xiang Lorraine Li, Adhiguna Kuncoro, Aida Nematzadeh, Elena Gribovskaya, Domenic Donato, Angeliki Lazaridou, Arthur Mensch, Jean-Baptiste Lespiau, Maria Tsimpoukelli, Nikolai Grigorev, Doug Fritz, Thibault Sottiaux, Mantas Pajarskas, Toby Pohlen, Zhitao Gong, Daniel Toyama, Cyprien de Masson d'Autume, Yujia Li, Tayfun Terzi, Vladimir Mikulik, Igor Babuschkin, Aidan Clark, Diego de Las Casas, Aurelia Guy, Chris Jones, James Bradbury, Matthew Johnson, Blake A. Hechtman, Laura Weidinger, Iason Gabriel, William S. Isaac, Edward Lockhart, Simon Osindero, Laura Rimell, Chris Dyer, Oriol Vinyals, Kareem Ayoub, Jeff Stanway, Lorrayne Bennett, Demis Hassabis, Koray Kavukcuoglu, and Geoffrey Irving. Scaling language models: Methods, analysis & insights from training gopher. *CoRR*, abs/2112.11446, 2021. URL https://arxiv.org/abs/2112.11446.

Thomas Rebele, Fabian M. Suchanek, Johannes Hoffart, Joanna Biega, Erdal Kuzey, and Gerhard Weikum. YAGO: A multilingual knowledge base from wikipedia, wordnet, and geonames. In *The Semantic Web - ISWC 2016 - 15th International*

Semantic Web Conference, Kobe, Japan, October 17-21, 2016, Proceedings, Part II, pages 177–185, 2016. doi:10.1007/978-3-319-46547-0_19. URL https://doi.org/10.1007/978-3-319-46547-0_19.

Stephen E. Robertson, Hugo Zaragoza, and Michael J. Taylor. Simple BM25 extension to multiple weighted fields. In David A. Grossman, Luis Gravano, ChengXiang Zhai, Otthein Herzog, and David A. Evans, editors, *Proceedings of the 2004 ACM CIKM International Conference on Information and Knowledge Management, Washington, DC, USA, November 8-13, 2004*, pages 42–49. ACM, 2004. doi:10.1145/1031171.1031181. URL https://doi.org/10.1145/1031171.1031181.

Daniel Ruffinelli, Samuel Broscheit, and Rainer Gemulla. You CAN teach an old dog new tricks! on training knowledge graph embeddings. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=BkxSmlBFvr.

Ivan Rybin, Vladislav Korablinov, Pavel Efimov, and Pavel Braslavski. Rubq 2.0: An innovated russian question answering dataset. In Ruben Verborgh, Katja Hose, Heiko Paulheim, Pierre-Antoine Champin, Maria Maleshkova, Óscar Corcho, Petar Ristoski, and Mehwish Alam, editors, *The Semantic Web - 18th International Conference, ESWC 2021, Virtual Event, June 6-10, 2021, Proceedings*, volume 12731 of *Lecture Notes in Computer Science*, pages 532–547. Springer, 2021. doi:10.1007/978-3-030-77385-4_32. URL https://doi.org/10.1007/978-3-030-77385-4_32.

Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. Distilbert, a distilled version of BERT: smaller, faster, cheaper and lighter. *CoRR*, abs/1910.01108, 2019a. URL http://arxiv.org/abs/1910.01108.

Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. Distilbert, a distilled version of BERT: smaller, faster, cheaper and lighter. *CoRR*, abs/1910.01108, 2019b. URL http://arxiv.org/abs/1910.01108.

Victor Sanh, Thomas Wolf, and Alexander M. Rush. Movement pruning: Adaptive sparsity by fine-tuning, 2020. URL https://proceedings.neurips.cc/paper/2020/hash/eae15aabaa768ae4a5993a8a4f4fa6e4-Abstract.html.

Teven Le Scao, Angela Fan, Christopher Akiki, Ellie Pavlick, Suzana Ilic, Daniel Hesslow, Roman Castagné, Alexandra Sasha Luccioni, François Yvon, Matthias Gallé, Jonathan Tow, Alexander M. Rush, Stella Biderman, Albert Webson, Pawan Sasanka Ammanamanchi, Thomas Wang, Benoît Sagot, Niklas Muennighoff, Albert Villanova del Moral, Olatunji Ruwase, Rachel Bawden, Stas Bekman, Angelina McMillan-Major, Iz Beltagy, Huu Nguyen, Lucile Saulnier, Samson Tan, Pedro Ortiz Suarez, Victor Sanh, Hugo Laurençon, Yacine Jernite, Julien Launay, Margaret Mitchell, Colin Raffel, Aaron Gokaslan, Adi Simhi, Aitor Soroa, Alham Fikri Aji, Amit Alfassy, Anna Rogers, Ariel Kreisberg Nitzav, Canwen Xu, Chenghao Mou, Chris Emezue, Christopher Klamm, Colin Leong, Daniel van Strien, David Ifeoluwa Adelani, and et al. BLOOM: A 176b-parameter open-access multilingual language model. *CoRR*, abs/2211.05100,

2022. doi:10.48550/arXiv.2211.05100. URL https://doi.org/10.48550/arXiv.2211.05100.

Matthias Schildwächter, Alexander Bondarenko, Julian Zenker, Matthias Hagen, Chris Biemann, and Alexander Panchenko. Answering comparative questions: Better than ten-blue-links? *arXiv preprint arXiv:1901.05041*, 2019.

Matthias Schildwächter, Alexander Bondarenko, Julian Zenker, Matthias Hagen, Chris Biemann, and Alexander Panchenko. Answering comparative questions: Better than ten-blue-links? *CoRR*, abs/1901.05041, 2019. URL http://arxiv.org/abs/1901.05041.

Abigail See, Peter J. Liu, and Christopher D. Manning. Get to the point: Summarization with pointer-generator networks. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1073–1083, Vancouver, Canada, July 2017. Association for Computational Linguistics. doi:10.18653/v1/P17-1099. URL https://www.aclweb.org/anthology/P17-1099.

Mohammad Shoeybi, Mostofa Patwary, Raul Puri, Patrick LeGresley, Jared Casper, and Bryan Catanzaro. Megatron-lm: Training multi-billion parameter language models using model parallelism. *CoRR*, abs/1909.08053, 2019. URL http://arxiv.org/abs/1909.08053.

Daniel G. A. Smith and Johnnie Gray. Opt_einsum - A python package for optimizing contraction order for einsum-like expressions. *J. Open Source Softw.*, 3(26): 753, 2018. doi:10.21105/joss.00753. URL https://doi.org/10.21105/joss.00753.

Fabian M. Suchanek, Gjergji Kasneci, and Gerhard Weikum. Yago: a core of semantic knowledge. In *The Web Conference*, 2007.

Jian-Tao Sun, Xuanhui Wang, Dou Shen, Hua-Jun Zeng, and Zheng Chen. CWS: a comparative web search system. In *Proceedings of the 15th international conference on World Wide Web*, pages 467–476, Edinburgh, Scotland, UK, 2006.

Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. Sequence to sequence learning with neural networks. In Zoubin Ghahramani, Max Welling, Corinna Cortes, Neil D. Lawrence, and Kilian Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems 2014, December 8-13 2014, Montreal, Quebec, Canada*, pages 3104–3112, 2014. URL https://proceedings.neurips.cc/paper/2014/hash/a14ac55a4f27472c5d894ec1c3c743d2-Abstract.html.

Charles Sutton and Andrew McCallum. An Introduction to Conditional Random Fields. *Found. Trends Mach. Learn.*, 4(4):267–373, 2012. URL https://doi.org/10.1561/2200000013.

Urmish Thakker, Jesse G. Beu, Dibakar Gope, Ganesh Dasika, and Matthew Mattina. Rank and run-time aware compression of NLP applications. *CoRR*, abs/2010.03193, 2020. URL https://arxiv.org/abs/2010.03193.

Kristina Toutanova and Danqi Chen. Observed versus latent features for knowledge base and text inference. In *Proceedings of the 3rd Workshop on Continuous Vector Space Models and their Compositionality*, pages 57–66, Beijing, China, July 2015. Association for Computational Linguistics. doi:10.18653/v1/W15-4007. URL https://aclanthology.org/W15-4007.

Théo Trouillon, Christopher R. Dance, Éric Gaussier, Johannes Welbl, Sebastian Riedel, and Guillaume Bouchard. Knowledge graph completion via complex tensor factorization. *J. Mach. Learn. Res.*, 18:130:1–130:38, 2017. URL http://jmlr.org/papers/v18/16-563.html.

Théo Trouillon, Johannes Welbl, Sebastian Riedel, Eric Gaussier, and Guillaume Bouchard. Complex embeddings for simple link prediction. In Maria Florina Balcan and Kilian Q. Weinberger, editors, *Proceedings of The 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pages 2071–2080, New York, New York, USA, 20–22 Jun 2016. PMLR. URL http://proceedings.mlr.press/v48/trouillon16.html.

L. R. Tucker. Some mathematical notes on three-mode factor analysis. *Psychometrika*, 31:279–311, 1966c.

Mikhail Usvyatsov, Rafael Ballester-Ripoll, and Konrad Schindler. tntorch: Tensor network learning with pytorch. *J. Mach. Learn. Res.*, 23:208:1–208:6, 2022. URL http://jmlr.org/papers/v23/21-1197.html.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017a. URL https://proceedings.neurips.cc/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017b. URL https://proceedings.neurips.cc/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *CoRR*, abs/1706.03762, 2017c. URL http://arxiv.org/abs/1706.03762.

Henning Wachsmuth, Martin Potthast, Khalid Al-Khatib, Yamen Ajjour, Jana Puschmann, Jiani Qu, Jonas Dorsch, Viorel Morari, Janek Bevendorff, and Benno Stein. Building an Argument Search Engine for the Web. In Kevin Ashley, Claire Cardie, Nancy Green, Iryna Gurevych, Ivan Habernal, Diane Litman, Georgios Petasis, Chris Reed, Noam Slonim, and Vern Walker, editors,

4th Workshop on Argument Mining (ArgMining 2017) at EMNLP, pages 49–59. Association for Computational Linguistics, September 2017. URL https://www.aclweb.org/anthology/W17-5106.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R. Bowman. GLUE: A multi-task benchmark and analysis platform for natural language understanding. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019. URL https://openreview.net/forum?id=rJ4km2R5t7.

Rui Wang, Bicheng Li, Shengwei Hu, Wenqian Du, and Min Zhang. Knowledge graph embedding via graph attenuated attention networks. *IEEE Access*, 8:5212–5224, 2020. doi:10.1109/ACCESS.2019.2963367. URL https://doi.org/10.1109/ACCESS.2019.2963367.

Xiaozhi Wang, Tianyu Gao, Zhaocheng Zhu, Zhengyan Zhang, Zhiyuan Liu, Juanzi Li, and Jian Tang. Kepler: A unified model for knowledge embedding and pre-trained language representation. *Transactions of the Association for Computational Linguistics*, 9:176–194, 2021. doi:10.1162/tacl_a_00360.

Yining Wang, Liwei Wang, Yuanzhi Li, Di He, and Tie-Yan Liu. A theoretical analysis of ndcg type ranking measures. In Shai Shalev-Shwartz and Ingo Steinwart, editors, *Proceedings of the 26th Annual Conference on Learning Theory*, volume 30 of *Proceedings of Machine Learning Research*, pages 25–54, Princeton, NJ, USA, 12–14 Jun 2013. PMLR. URL https://proceedings.mlr.press/v30/Wang13.html.

Zhen Wang, Jianwen Zhang, Jianlin Feng, and Zheng Chen. Knowledge graph embedding by translating on hyperplanes. In Carla E. Brodley and Peter Stone, editors, *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence, July 27 -31, 2014, Québec City, Québec, Canada*, pages 1112–1119. AAAI Press, 2014. URL http://www.aaai.org/ocs/index.php/AAAI/AAAI14/paper/view/8531.

Zheng Wang, Juncheng B. Li, Shuhui Qu, Florian Metze, and Emma Strubell. Squat: Sharpness- and quantization-aware training for BERT, 2022. URL https://doi.org/10.48550/arXiv.2210.07171.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, and Jamie Brew. Huggingface's transformers: State-of-the-art natural language processing. *CoRR*, abs/1910.03771, 2019. URL http://arxiv.org/abs/1910.03771.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander M. Rush. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages

38–45, Online, October 2020. Association for Computational Linguistics. URL https://www.aclweb.org/anthology/2020.emnlp-demos.6.

Qiang Wu, Chris J.C. Burges, Krysta M. Svore, and Jianfeng Gao. Adapting bboosting for information retrieval measures. *Information Retrieval*, 13:254–270, June 2010. URL https://www.microsoft.com/en-us/research/publication/adapting-boosting-for-information-retrieval-measures/.

Le Xu, Lei Cheng, Ngai Wong, and Yik-Chung Wu. Tensor train factorization and completion under noisy data with prior analysis and rank estimation, 2022.

Bishan Yang, Wen-tau Yih, Xiaodong He, Jianfeng Gao, and Li Deng. Embedding entities and relations for learning and inference in knowledge bases. 2015. URL http://arxiv.org/abs/1412.6575.

Linyao Yang, Hongyang Chen, Zhao Li, Xiao Ding, and Xindong Wu. Chatgpt is not enough: Enhancing large language models with knowledge graphs for fact-aware language modeling, 2023.

Zhilin Yang, Zihang Dai, Ruslan Salakhutdinov, and William W. Cohen. Breaking the softmax bottleneck: A high-rank RNN language model. In *International Conference on Learning Representations*, 2018a. URL https://openreview.net/forum?id=HkwZSG-CZ.

Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William W. Cohen, Ruslan Salakhutdinov, and Christopher D. Manning. Hotpotqa: A dataset for diverse, explainable multi-hop question answering. In Ellen Riloff, David Chiang, Julia Hockenmaier, and Jun'ichi Tsujii, editors, *EMNLP-2018*, pages 2369–2380, Brussels, Belguim, November 2018b. doi:10.18653/v1/d18-1259. URL https://doi.org/10.18653/v1/d18-1259.

Seid Muhie Yimam, Iryna Gurevych, Richard Eckart de Castilho, and Chris Biemann. WebAnno: A flexible, web-based and visually supported system for distributed annotations. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, pages 1–6, Sofia, Bulgaria, August 2013. Association for Computational Linguistics. URL https://www.aclweb.org/anthology/P13-4001.

Chunxing Yin, Bilge Acun, Xing Liu, and Carole-Jean Wu. Tt-rec: Tensor train compression for deep learning recommendation models. *CoRR*, abs/2101.11714, 2021. URL https://arxiv.org/abs/2101.11714.

Shuai Zhang, Yi Tay, Lina Yao, and Qi Liu. Quaternion knowledge graph embeddings. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. URL https://proceedings.neurips.cc/paper/2019/file/d961e9f236177d65d21100592edb0769-Paper.pdf.

Susan Zhang, Stephen Roller, Naman Goyal, Mikel Artetxe, Moya Chen, Shuohui Chen, Christopher Dewan, Mona Diab, Xian Li, Xi Victoria Lin, Todor Mihaylov, Myle Ott, Sam Shleifer, Kurt Shuster, Daniel Simig, Punit Singh Koura, Anjali Sridhar, Tianlu Wang, and Luke Zettlemoyer. Opt: Open pre-trained transformer language models. arXiv, 2022. doi:10.48550/ARXIV.2205.01068. URL https://arxiv.org/abs/2205.01068.

Qibin Zhao, Guoxu Zhou, Shengli Xie, Liqing Zhang, and Andrzej Cichocki. Tensor ring decomposition, 2016. URL http://arxiv.org/abs/1606.05535.

Appendix A

Additional results for Transformer-based Encoders compression using TTM decomposition

Table A.1: The results of different types of compression of BERT for experiment with task-oriented fine-tuning and further compression (Single-train). The best results at each model size are in **bold**, best overall results are underlined.

Method	Size	AVG	STSB	CoLA	MNLI	MRCP	QNLI	QQP	RTE	SST2	WNLI
`bert-base`	100%	0.79	0.88	0.57	0.84	0.9	0.91	0.87	0.67	0.92	0.54
DistilBERT	61%	0.76	0.87	0.51	0.82	0.87	0.89	**0.88**	0.59	0.91	0.48
`bert-base` FP-16	100%†	0.78	0.88	0.55	0.83	0.88	0.90	0.88	0.67	0.91	0.48
SVD	49%	0.37 ± 0.01	0.24 ± 0.12	0.00 ± 0.00	0.36 ± 0.01	0.20 ± 0.09	0.50 ± 0.02	0.47 ± 0.10	0.48 ± 0.01	0.52 ± 0.02	0.51 ± 0.07
FWSVD		0.38 ± 0.03	0.25 ± 0.15	**0.10 ± 0.01**	0.33 ± 0.01	0.39 ± 0.33	0.50 ± 0.01	0.40 ± 0.11	0.49 ± 0.03	0.51 ± 0.01	**0.56 ± 0.00**
TTM		**0.44 ± 0.02**	**0.58 ± 0.06**	0.02 ± 0.03	**0.37 ± 0.01**	0.25 ± 0.22	**0.56 ± 0.01**	0.43 ± 0.17	**0.50 ± 0.03**	**0.72 ± 0.02**	0.51 ± 0.07
FWTTM		**0.44 ± 0.02**	**0.59 ± 0.06**	0.02 ± 0.03	**0.37 ± 0.01**	0.27 ± 0.24	0.54 ± 0.01	0.42 ± 0.18	**0.50 ± 0.03**	**0.71 ± 0.02**	0.51 ± 0.07
SVD	63%	0.45 ± 0.02	0.63 ± 0.07	0.01 ± 0.02	0.36 ± 0.01	0.22 ± 0.11	0.51 ± 0.03	0.54 ± 0.06	0.54 ± 0.06	0.78 ± 0.03	0.48 ± 0.07
FWSVD		**0.55 ± 0.03**	0.54 ± 0.10	**0.07 ± 0.03**	**0.52 ± 0.02**	**0.55 ± 0.20**	0.62 ± 0.02	**0.70 ± 0.07**	**0.58 ± 0.05**	**0.79 ± 0.05**	0.55 ± 0.02
TTM		0.44 ± 0.02	0.65 ± 0.03	0.01 ± 0.02	0.40 ± 0.02	0.16 ± 0.00	0.54 ± 0.06	0.52 ± 0.14	0.48 ± 0.00	0.74 ± 0.03	0.48 ± 0.08
FWTTM		0.47 ± 0.02	**0.71 ± 0.02**	0.01 ± 0.04	0.44 ± 0.01	0.17 ± 0.01	**0.64 ± 0.06**	0.53 ± 0.11	0.48 ± 0.03	0.72 ± 0.04	**0.56 ± 0.00**
SVD	95%	0.70 ± 0.02	0.81 ± 0.02	0.26 ± 0.14	0.82 ± 0.00	0.69 ± 0.22	0.88 ± 0.00	0.87 ± 0.01	0.53 ± 0.06	0.90 ± 0.01	0.53 ± 0.08
FWSVD		**0.78 ± 0.01**	**0.88 ± 0.00**	**0.55 ± 0.02**	**0.84 ± 0.00**	**0.87 ± 0.01**	**0.90 ± 0.00**	**0.88 ± 0.01**	0.64 ± 0.01	**0.92 ± 0.01**	**0.55 ± 0.04**
TTM		0.76 ± 0.01	0.87 ± 0.00	0.52 ± 0.02	0.79 ± 0.00	0.86 ± 0.01	0.87 ± 0.01	0.86 ± 0.00	**0.65 ± 0.01**	0.91 ± 0.01	0.48 ± 0.01
FWTTM		0.77 ± 0.01	0.87 ± 0.00	0.53 ± 0.02	0.81 ± 0.00	0.86 ± 0.01	0.88 ± 0.00	0.87 ± 0.00	**0.65 ± 0.01**	0.91 ± 0.01	0.54 ± 0.01

Method	Size	AVG	STSB	CoLA	MNLI	MRCP	QNLI	QQP	RTE	SST2	WNLI
`bert-base`	100%	**0.79**	0.88	**0.57**	**0.84**	**0.90**	**0.91**	0.87	0.67	**0.92**	0.54
DistilBERT	61%	0.76	0.87	0.51	0.82	0.87	0.89	0.88	0.59	0.91	0.48
`bert-base` FP-16	100%†	0.78	0.88	0.55	0.83	0.88	0.90	0.88	0.67	0.91	0.48
B.P. (75%)	61%	0.72	0.85	0.24	0.83	0.83	0.86	0.87	0.52	0.88	0.56
SVD	49%	0.68 ± 0.01	**0.83 ± 0.00**	0.00 ± 0.01	**0.79 ± 0.01**	0.79 ± 0.01	**0.85 ± 0.00**	**0.87 ± 0.00**	0.59 ± 0.01	**0.87 ± 0.00**	0.49 ± 0.08
FWSVD		0.68 ± 0.01	0.82 ± 0.01	0.04 ± 0.05	**0.79 ± 0.00**	0.79 ± 0.00	**0.85 ± 0.01**	**0.87 ± 0.00**	0.56 ± 0.03	0.86 ± 0.00	**0.54 ± 0.06**
TTM		**0.69 ± 0.01**	**0.83 ± 0.00**	**0.15 ± 0.01**	0.78 ± 0.00	**0.81 ± 0.01**	0.84 ± 0.00	**0.87 ± 0.00**	**0.60 ± 0.01**	0.86 ± 0.01	0.43 ± 0.08
FWTTM		**0.69 ± 0.01**	**0.83 ± 0.00**	**0.15 ± 0.04**	0.78 ± 0.00	**0.81 ± 0.01**	0.84 ± 0.00	**0.87 ± 0.00**	**0.60 ± 0.01**	0.86 ± 0.01	0.49 ± 0.00
SVD	63%	0.75 ± 0.02	0.86 ± 0.00	0.43 ± 0.02	**0.83 ± 0.00**	0.84 ± 0.01	**0.89 ± 0.00**	**0.88 ± 0.01**	0.64 ± 0.02	**0.90 ± 0.01**	0.50 ± 0.10
FWSVD		**0.77 ± 0.00**	**0.87 ± 0.00**	**0.47 ± 0.02**	**0.83 ± 0.00**	**0.85 ± 0.01**	**0.89 ± 0.01**	**0.88 ± 0.01**	**0.65 ± 0.01**	**0.90 ± 0.01**	**0.56 ± 0.01**
TTM		0.70 ± 0.01	0.85 ± 0.00	0.10 ± 0.10	0.81 ± 0.00	0.81 ± 0.01	0.86 ± 0.00	**0.88 ± 0.01**	0.61 ± 0.01	0.88 ± 0.00	0.49 ± 0.09
FWTTM		0.70 ± 0.02	0.85 ± 0.00	0.08 ± 0.08	0.81 ± 0.00	0.82 ± 0.00	0.86 ± 0.00	0.87 ± 0.01	0.61 ± 0.01	0.88 ± 0.00	0.53 ± 0.07
SVD	95%	0.78 ± 0.01	**0.89 ± 0.00**	**0.56 ± 0.02**	**0.84 ± 0.00**	**0.88 ± 0.02**	**0.91 ± 0.00**	**0.89 ± 0.01**	0.68 ± 0.01	**0.91 ± 0.01**	0.44 ± 0.08
FWSVD		**0.79 ± 0.04**	**0.89 ± 0.00**	**0.56 ± 0.03**	**0.84 ± 0.00**	**0.88 ± 0.01**	0.90 ± 0.00	**0.89 ± 0.01**	**0.69 ± 0.01**	**0.91 ± 0.01**	0.51 ± 0.08
TTM		0.77 ± 0.05	0.88 ± 0.00	0.52 ± 0.03	0.83 ± 0.00	0.83 ± 0.06	0.89 ± 0.00	0.88 ± 0.00	0.68 ± 0.02	0.90 ± 0.00	0.51 ± 0.07
FWTTM		0.78 ± 0.03	0.88 ± 0.00	0.52 ± 0.02	0.83 ± 0.00	0.87 ± 0.01	0.90 ± 0.00	0.88 ± 0.00	0.68 ± 0.02	0.90 ± 0.00	**0.54 ± 0.06**

Table A.2: The results of different types of compression of BERT for experiments with task-oriented fine-tuning, compression, and further fine-tuning (Double-train). The best results at each model size are in **bold**, best overall results are underlined.

Table A.3: The results of different types of compression for the `bart-base` model for experiments with detoxification with task-oriented fine-tuning, compression, and further fine-tuning (Single-train and Double-train). The best results at each model size are in **bold**, best overall results are underlined. *Italic* results represent senseless model outputs.

Pipeline		Single-train				Double-train			
Method	C. Rate	STA	SIM	FL	J	STA	SIM	FL	J
`bart-base`	100 %	-	-	-	-	0.89	0.60	0.82	0.44
FP16 *eval.*†	100 %	-	-	-	-	0.89	0.60	0.82	0.44
B.P. (95%)	63%	-	-	-	-	0.92	0.34	0.30	0.12
B.P. (65%)	74%	-	-	-	-	0.82	0.60	0.73	0.36
SVD	60%	*0.97 ± 0.04*	*0.18 ± 0.01*	*0.10 ± 0.05*	*0.01 ± 0.02*	0.75 ± 0.01	**0.59 ± 0.01**	0.65 ± 0.01	0.28 ± 0.01
FWSVD		*0.32 ± 0.01*	*0.46 ± 0.01*	*0.58 ± 0.01*	*0.07 ± 0.01*	**0.78 ± 0.02**	**0.59 ± 0.01**	**0.68 ± 0.00**	**0.30 ± 0.01**
TTM		*0.97 ± 0.04*	*0.19 ± 0.02*	*0.16 ± 0.04*	*0.03 ± 0.01*	0.74 ± 0.02	0.58 ± 0.01	0.64 ± 0.02	0.27 ± 0.01
FWTTM		*0.99 ± 0.15*	*0.18 ± 0.01*	*0.19 ± 0.05*	*0.03 ± 0.02*	0.74 ± 0.02	0.58 ± 0.00	0.65 ± 0.00	0.27 ± 0.01
SVD	74%	*0.85 ± 0.06*	*0.21 ± 0.01*	*0.14 ± 0.05*	*0.03 ± 0.01*	0.82 ± 0.01	0.60 ± 0.01	0.77 ± 0.01	0.38 ± 0.01
FWSVD		*0.32 ± 0.01*	*0.46 ± 0.01*	*0.58 ± 0.01*	*0.07 ± 0.01*	**0.87 ± 0.00**	**0.61 ± 0.01**	**0.80 ± 0.01**	**0.42 ± 0.01**
TTM		*0.99 ± 0.02*	*0.17 ± 0.01*	*0.06 ± 0.07*	*0.01 ± 0.01*	0.82 ± 0.01	**0.61 ± 0.01**	0.75 ± 0.01	0.37 ± 0.01
FWTTM		*0.99 ± 0.03*	*0.18 ± 0.01*	*0.16 ± 0.05*	*0.03 ± 0.01*	0.84 ± 0.01	0.60 ± 0.01	0.75 ± 0.01	0.38 ± 0.01
SVD	90%	**0.85 ± 0.01**	0.42 ± 0.01	0.72 ± 0.01	0.25 ± 0.01	0.86 ± 0.00	**0.61 ± 0.00**	**0.81 ± 0.00**	**0.43 ± 0.00**
FWSVD		0.70 ± 0.10	**0.64 ± 0.01**	**0.82 ± 0.00**	**0.35 ± 0.01**	**0.87 ± 0.01**	**0.61 ± 0.00**	**0.81 ± 0.00**	**0.43 ± 0.00**
TTM		0.49 ± 0.17	0.60 ± 0.01	0.71 ± 0.00	0.18 ± 0.01	0.86 ± 0.01	**0.61 ± 0.00**	0.80 ± 0.01	0.41 ± 0.01
FWTTM		0.82 ± 0.06	0.46 ± 0.01	0.59 ± 0.01	0.22 ± 0.01	0.86 ± 0.01	**0.61 ± 0.00**	0.80 ± 0.01	0.41 ± 0.00