Shrinking Knowledge Base Size: Dimension Reduction, Splitting & Filtering

by

Vilém Zouhar

Master Thesis

Saarland University, Faculty of Arts, Language Science and Technology
Groningen University, Faculty of Arts, Linguistics Research Master
LCT Double Degree Master

Supervision
Dietrich Klakow (UdS), Gosse Bouma (RUG),
Marius Mosbach (UdS, advisor)

April 19, 2022

UNIVERSITÄT DES SAARLANDES

university of groningen

Declaration of Authorship

Eidesstattliche Erklärung

Ich erkläre hiermit an Eides Statt, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe. Ich versichere, dass die gedruckte und die elektronische Version der Masterarbeit inhaltlich übereinstimmen.

Statement in Lieu of an Oath

I hereby confirm that I have written this thesis on my own and that I have not used any other media or materials than the ones referred to in this thesis. I assure that the electronic version is identical in content to the printed version of the Master's thesis.

Datum/Date: ______________________________

Unterschrift/Signature: ________________________

SAARLAND UNIVERSITY, GRONINGEN UNIVERSITY

Abstract

Shrinking Knowledge Base Size:
Dimension Reduction, Splitting & Filtering
by Vilém Zouhar

Recently neural network based approaches to knowledge-intensive NLP tasks, such as question answering, started to rely heavily on the combination of neural retrievers and readers. Retrieval is typically performed over a large textual knowledge base which requires significant memory and compute resources, especially when scaled up. On HotpotQA we explore various filtering & splitting criteria. Primarily, we systematically investigate reducing the size of the KB index by means of dimensionality (sparse random projections, PCA, autoencoders) and numerical precision reduction.

Our results show that PCA is an easy solution that requires very little data and is only slightly worse than autoencoders, which are less stable. All methods are sensitive to pre- and post-processing and data should always be centered and normalized both before and after dimension reduction. Finally, we show that it is possible to combine PCA with using 1bit per dimension. Overall we achieve (1) 100× compression with 75%, and (2) 24× compression with 92% original retrieval performance.

Keywords: document retrieval, knowledge base, dimension reduction, dimensionality, efficiency

Acknowledgements

During my studies I was generously supported financially and non-financially by the Erasmus Mundus European Masters Program in Language and Communication Technologies. Thank you to the administrators, Bobbye Pernice, Jürgen Trouvain and Ivana Krujiff-Korbayová.

Thanks to everyone who helped in reviewing this thesis, among others Luuk Suurmeijer (HHU), Rishu Kumar (CUNI), Mrinmaya Sachan (ETHz), Ryan Cotterell (ETHz) and Jakub Zavrel.

A part of this work was also funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) - Project-ID 232722074 - SFB 1102.

Content

Declaration of Authorship **ii**

Abstract **iii**

Acknowledgements **iv**

1 Introduction **1**

1.1 Related Work . . . 2

1.2 Retrieval pipelines . . . 4

1.3 Setup . . . 8

1.3.1 Document Retrieval . . . 8

1.3.2 Evaluation . . . 9

1.3.3 Similarity Functions . . . 10

1.3.4 Data . . . 12

1.3.5 Model Comparison . . . 12

1.3.6 Embedding Model Overview . . . 14

2 Splitting & Filtering **18**

2.1 Splitting . . . 18

2.2 Filtering . . . 20

2.2.1 Heuristics . . . 20

2.2.2 Automatic filtering . . . 20

2.2.3 Discussion . . . 25

3 Dimension Reduction **26**

3.1 Problem Statement . . . 26

3.1.1 Pre-processing Transformations . . . 27

3.2 Compression Methods . . . 28

3.2.1 Random Projection . . . 28

3.2.2 Principal Component Analysis . . . 29

3.2.3 Autoencoder 31
3.2.4 Precision Reduction 32
3.2.5 Combination of PCA and Precision Reduction 35
3.3 Analysis 37
3.3.1 Model Comparison 37
3.3.2 Speed. 37
3.3.3 Data size 37
3.3.4 Retrieval errors 38

4 Discussion **41**
4.1 Recommendations 41
4.1.1 Importance of Pre-/post-processing 41
4.1.2 Method recommendation 41
4.1.3 Splitting 42
4.2 Pitfalls of Reconstruction Loss 42
4.2.1 Distance Learning 42
4.3 Limitation 45
4.4 Summary 46

A Fusion Discussion **47**

List of Figures **50**

List of Tables **53**

Bibliography **55**

Chapter 1

Introduction

Recent approaches to knowledge-intensive NLP tasks combine neural network based models with a retrieval component that leverages dense vector representations [1–3]. The most straightforward example is question answering, where the retriever receives a question as an input and returns relevant documents to be used by the reader (both encoder and decoder), which outputs the answer [4, 5]. The same approach can also be applied for models for other tasks, such as fact-checking [6], retrieval-enhanced language modelling [7–9], or knowledgable dialogue [10, 11]. Moreover, this paradigm can also be applied to systems that utilize e.g. caching of contexts from the training corpus to provide better output, such as the k-nearest neighbours language model proposed by Khandelwal et al. [12] or the dynamic gating language model mechanism by Yogatama et al. [13]. All these pipelines are generalized as retrieving an artefact from a knowledge base [14] on which the reader is conditioned together with the query. Overall, they represent a trend of combining the advantages of non-parametric (scalability, auditability) and non-parametric models (performance, automatic optimization).

Crucially, all of the previous examples rely on the quality of the retrieval component and the knowledge base. The knowledge base is usually indexed by dense vector representations[1] and the retrieval component performs maximum similarity search, commonly using the inner product or the L^2 distance, to retrieve documents[2] from the knowledge base. Only the index alone takes up a large amount of size of the knowledge base (150GB), making deployment and experimentation very difficult.[3] The retrieval speed is also dependent on the dimensionality of the index vector. An example of a large knowledge base is the work of Borgeaud et al. [8] which performs retrieval over a database of 1.8 billion documents.

[1]Sparse representations via BM25 [15] are also commonly used but not the focus of this work.

[2]We refer to the retrieved objects as documents though they commonly range from spans of text (e.g. 100 tokens) to the full documents. This is explained and disambiguated in Chapter 2.

[3]For experimenting with this index, a memory of up to 1TB is required.

The possible impacts of reducing the knowledge base size are (1) reduced deployment and research constraints and (2) improved retrieval efficiency. The constraints are retaining as much of the original retrieval performance as possible. This thesis focuses on the issue of compressing the knowledge base size primarily through the dimensionality and precision reduction of the index and makes the following contributions:

- Analysis of negative results of various splitting and filtering methods.
- Comparison of various unsupervised index compression methods in retrieval experiments, including random projections, PCA, autoencoder, precision reduction and their combination.
- Examination of effective pre- and post-processing transformations, showing that centering and normalization are necessary for boosting the performance.
- Analysis on the impact of adding irrelevant documents and retrieval errors. Recommendations for use by practitioners.

This thesis is split into two main chapters: introduction to the problem and negative results of Splitting & Filtering (Chapter 2) and Dimension Reduction (Chapter 3, the primary content of this thesis). We provide further analysis in Section 3.3 and conclude with usage recommendations in Chapter 4.

The repository for this thesis is available open-source.[4]

1.1 Related Work

This thesis is largely based on two papers written by the same author: Zouhar et al. [14, 16] with a more detailed background and Chapter 2 added.

Knowledge-intensive NLP Overview. A comprehensive overview of earlier work on neural model-based information retrieval systems together with a general introduction has been done by Mitra and Craswell [17]. More recently models such as BERT have been utilized successfully for the task of retrieval itself [18, 19].

The aim of KILT [3] is to provide a common knowledge base for a number of different NLP tasks (ranging from question answering to fact verification) and to stimulate research in task-agnostic memory architectures. Reformulating various NLP tasks to all

[4]github.com/zouharvi/kb-shrink

use the same knowledge base format provides a stepping stone for the formalisms of artefact retrieval.

Defining a task-agnostic abstract model is closely related to multi-task learning. The goal of this approach is to improve the performance by training the model on multiple tasks rather than on individual ones [20]. The hope is that representations and generalizations learned for one task will help on another one and vice versa. A strong requirement for this is that the instantiations for different tasks (in the multi-task setup) share significant portions of the model.

An edge-case of this is using a pre-trained BERT model and then fine-tuning it for the new task and/or possibly adding extra layers to match the input and output shapes. Even for BERT, however, it was shown several times [3, 21–23] that training on multiple tasks improves the performance [24]. This would not be possible without a common model shared among the tasks. Further related work is discussed in the respective sections when presenting individual NLP models and how they fit into this schema.

Reducing index size. A thorough overview of the issue of dimensionality reduction in information retrieval in the context of dual encoders has been done by Luan et al. [25]. Though in-depth and grounded in formal arguments, their study is focused on the limits and properties of dimension reduction in general (even with sparse representations) and the effect of document length on performance. In contrast to their work, this thesis aims to compare more methods and give practical advice with experimental evidence.

A baseline for dimensionality reduction has been recently proposed by Izacard et al. [26] in which they perform the reduction while training the document (and query) encoder by adding a low dimensional linear projection layer as the final output layer. Compared to our work, their approach is supervised.

In the concurrent work of Ma et al. [27], PCA is also used to reduce the size of the document index. Compared to our work, they perform PCA using the combination of all question and document vectors. We show in Figures 3.2 and 3.6 that this is not needed and the PCA transformation matrix can be estimated much more efficiently. Moreover, we use different unsupervised compression approaches for comparison and perform additional analysis of our findings.

An orthogonal approach to the issue of memory cost has been proposed by Yamada et al. [28]. Instead of moving to another continuous vector representation, their proposed method maps original vectors to vectors of binary values which are trained using the signal from the downstream task. The pipeline, however, still relies on re-ranking using the uncompressed vectors. This method is different from ours and in Section 3.2.4 we show that they can be combined.

Finally, He et al. [9] investigate filtering and k-means pruning for the task of kNN language modelling. This work also circumvents the issue of having to always perform an expensive retrieval of a large data store by determining whether the retrieval is actually needed for a given input.

Effect of normalization. Normalization in information retrieval is historically connected with normalization with respect to the document length or term frequency [29–31]. In the context of this thesis, we refer to vector normalization and in general post-processing functions for vectors.

Timkey and van Schijndel [32] examine how dominating embedding dimensions can worsen retrieval performance. They study the contribution of individual dimensions find that normalization is key for document retrieval based on dense vector representation when BERT-based embeddings are used. Compared to our work, they study pre-trained BERT directly, while we focus on DPR.

1.2 Retrieval pipelines

Historically there has been a large focus on non-parametric models for knowledge-intensive NLP tasks. Those are not models which don't have any parameters but rather they don't have a fixed amount of parameters. This approach is particularly suited for tasks such as question answering because the knowledge is retrieved from a database and can be easily expanded or audited. Opposed to that are models which store the knowledge in parameters, such as pre-trained language models, which became largely adopted in the NLP community.

The parametric models suffer from becoming outdated very quickly [33], providing very little explainability in comparison to non-parametric models and performing poorly on unseen phenomena [7]. Recently there has been an emergence of the combination of these approaches, as shown in Table 1.1. They work by retrieving something (*artefacts*) from a knowledge base (or any persistent storage) and merging it in the computation of a parametric model.

An abstract pipeline in Figure 1.1, described in detail by Zouhar et al. [14], shows how these models operate. The key components are *encoder*, *retriever*, *aggregator* and *model* (some may be joined together in some works, commonly the retriever and the aggregator).

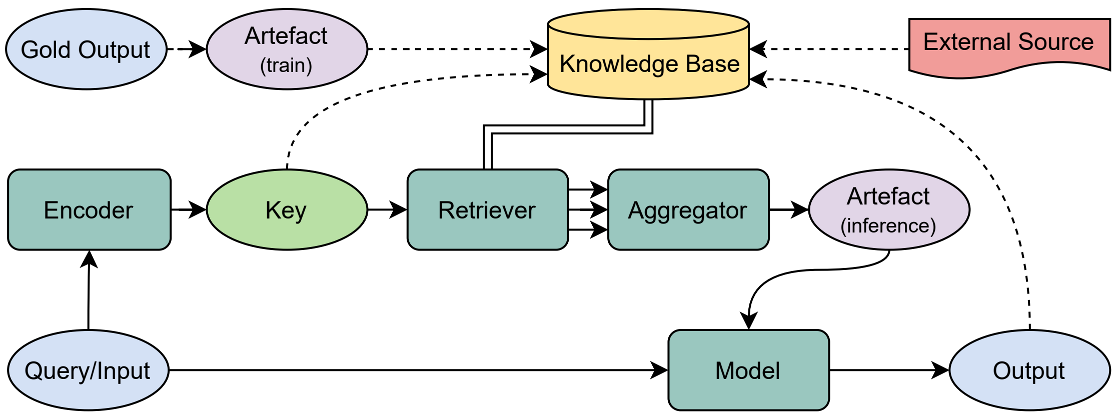

FIGURE 1.1: General scheme of NLP models utilizing artefacts by retrieving them from a knowledge base and fusing them into the model in order to produce a better output. Dashed links are utilized only in knowledge base creation and usually not all at once.

The *input* is specific to the given task. For language modelling, it is the previous context, for question answering the question, for slot-filling usually the entity and the relation, and for fact-checking the fact to be verified. In the context of this work, a *knowledge base* is a collection of items, usually (but not necessarily) with a pre-built index that maps keys to values. Prototypically, it is a collection of *documents*, though it can also be a collection of gold training data input-output pairs or a knowledge graph. *Candidates* are values retrieved from the knowledge base, which may be later post-processed (e.g. reranking or averaging) by the aggregator to form an artefact. *key* is an object through which the retriever finds suitable artefacts. Commonly the key is a dense vector representation of the input, though it is not necessarily a vector and may be dependent also on an intermediate model computation. An *artefact* is an object which is (1) dependent on elements retrieved from the knowledge base (e.g. the concatenation of k retrieved documents) and (2) can be used to improve the performance during training and inference. In the simplest example, it is the retrieved value itself, though it can also be multiple retrieved values or their combination.

In the context of question answering, we start with the query, then usually compute an embedding of it or use TF-IDF, then we give this to the retriever, which does maximum similarity search over a knowledge base, then the aggregator reranks and concatenates them and we get the artefact. This is then fused into the model, for example, in the form of priming. The model then produces the answer because it's conditioned both on the query and the artefact.

This formalism works also for other tasks that rely on retrieval, such as fact checking, knowledge base-enhanced language modelling or knowledgeable dialogue [14]. Most systems used in the literature differ in the encoder, retriever, and model design. We bring attention to four properties that characterize the differences between such systems.

- **Fusion** (early, late, other)
- **Specificity** (sample, task, class)
- **KB source** (train, external, dynamic)
- **Key & value type** (dense, sparse, other)

We consider the fusion mechanism to be of the highest interest. Priming is very common with pretrained language modelling and is an example of an early fusion. Formally the model estimates $p(y|x, \xi)$ where y is the ground-truth output, x is the query/input and ξ is a retrieved artefact. Fusion Sun et al. [34] concerns with which point is the artefact made available to the model. It can be presented to the model at the same time as the query/input, e.g. by concatenating x and ξ (early fusion), just before the output is created by the model (late fusion), or somewhere in between. We can think of any model as a composition of functions $f_1, \ldots, f_n$. In the simplest example of feed-forward networks, these correspond to single layers and activation functions and on a higher level, they correspond to whole encoder/decoder blocks. The distinction as to what counts as early and late is not clear and for presentation purposes, we consider early fusion at the level of f_1 and late fusion at the last stage, f_n. These functions themselves may, however, still be composed of multiple others. In **early fusion**, the artefact is the input together with the query to the first function f_1, while in **late fusion** the query is the single input to f_1 and artefact is considered only for f_n.

$$\begin{array}{lr}
\text{No fusion:} & f_n \circ \ldots \circ f_2 \circ f_1(q) \\
\text{Early fusion:} & f_n \circ \ldots \circ f_2 \circ f_1(q, \xi) \\
\text{Late fusion:} & f_n(f_{n-1} \circ \ldots \circ f_1(q), \xi) \\
\text{Intermediate fusion:} & f_n \circ \ldots \circ f_k(f_{k-1} \circ \ldots \circ f_1(q), \xi)
\end{array}$$

Improvements for models for one task can also transfer to others. This formalism also allows us to ask questions such as how do different fusion mechanisms affect the model computation. The key component in all of these pipelines is retrieval. This thesis focuses on a specific setup, described in Section 1.3, with the goal of reducing the knowledge base size.

See Appendix A for a discussion on . For a description of specific systems and how they fit into this typology see the appendix in Zouhar et al. [14].

Model	**Fusion**	**KB Source**	**Keys**	**Values**	**Aggregation**
k-NN LM [12]	Very late Static convex combination	Train-time	Prefix embd., L^2	Target word	Softmax
Continuous Cache LM [35]	Very late Static convex combination	Dynamic	Prefix embd., inner product	Target word	Softmax
Dynamic Gating LM [13]	Late Dyn. convex combination	Train-time	Prefix encoding, inner product	Target word	Softmax sum
Knowledge Graph LM [7]	Intermediate Constraints	External	Entity+relation Discrete struct.	Matching entity	None
Dense Passage Retrieval [36]	Early Input	External	Passage embd., inner product	Passages	None
Nearest Neighbour QA [37]	No model	Train-time	Passage embd., inner product	Answers	None
CBR-KBQA [38]	Query creation	Train-time External	Query embd., inner product	Logical forms	New query
PullNet [39]	Subgraph creation	Multiple External	Entities	Docs and Facts	Iterative join
Universal Schema QA [40]	Intermediate Retrieval	Multiple External	Query embd. Attention	Facts	Iterative projection
FAKTA [4]	Early Input	External Online	Condensed query	Docs	Re-ranking, Filtering
Wizards of Wikipedia [10]	Intermediate Addition	External	Context+topic, inverted index	Passages	Attention (topic)

TABLE 1.1: Categorization of described NLP systems in terms of the artefact retrieval typology. *Fusion* describe both where it occurs and what mechanism it employs, *Keys* describes not only the key type but also the retrieval mechanism (e.g. metric).

1.3 Setup

This section introduces concepts in document retrieval and describes the evaluation and data setup for experiments in Chapters 2 and 3. It also shows the base performance of a selection of pre-trained language models used for building the index for retrieval.

1.3.1 Document Retrieval

Conceptually, dense vector-based retrieval approaches work by first encoding all the documents and then at test-time, the query is also encoded (not necessarily using the same model). Given a query q, we retrieve top k relevant documents $Z = \{d_1, d_2, \ldots, d_k\}$ from a large collection of documents $\mathcal{D}$ so that the relevance of d with q is maximized. For this, the query and the document embedding functions $f_Q : \mathcal{Q} \to \mathbb{R}^d$ and $f_D : \mathcal{D} \to \mathbb{R}^d$ are used to map the query and *all* documents to a shared embedding space and a similarity function sim : $\mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ approximates the relevance between query and documents. Here, we consider either the inner product or the L^2 distance as sim.[5] This is conceptualized in Figure 1.2 and the following set of equations. These functions are commonly finetuned pretrained language models [23, 36, 41, 42] but can also be TF-IDF- or BM25-based [15].

$$
\begin{aligned}
Z &= \underset{d \in \mathcal{D}}{\arg \text{top-k}}\ \text{rel.}(q, d)\ \text{, with} \\
\text{rel.}(q, d) &\approx \text{sim}(f_{\text{Q}}(q), f_{\text{D}}(d))
\end{aligned}
$$

The approximation in (2) was shown to work well in practice for inner product and L^2 distance [43]. When dealing with multiple downstream tasks that share a single (large) knowledge base, typically only f_Q is fine-tuned for a specific task while f_D remains fixed [2, 3]. This assumes that the organization of the document vector space is sufficient across tasks and that only the mapping of the queries to this space needs to be trained.[6] Hence, this work is motivated primarily by finding a good r_D (because of the dominant size of the document index), though we note that r_Q is equally important and necessary because even without any vector semantics, the key and the document embeddings must have the same dimensionality.

The issue with this approach is that the pre-trained language models do not capture the meaning of the whole document in a 768-dimensional vector well [42, 44]. Although it is possible to extend the dimensionality to 2048-dimensional vectors to better capture the

[5]Cosine similarity could also be used but for computation reasons we skip it. Results are the same as for the inner product and L^2 distance when the vectors are normalized.

[6]Guu et al. [1] provide evidence that this assumption can lead to worse results in some cases.

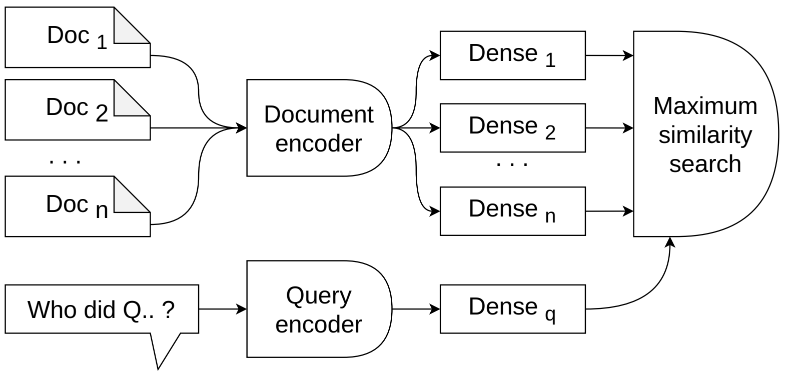

FIGURE 1.2: Conceptualized overview of retrieval on whole documents. Not actually used in practice with dense vector representations.

meaning [44, 45], this creates other issues, such as spurious matches. To alleviate this, the documents are split into *spans* over which the maximum similarity search is performed, as shown in Figure 1.3. Note that large vector approaches, i.e. TF-IDF/BM25 are an exception and can operate also on long documents, usually [46].

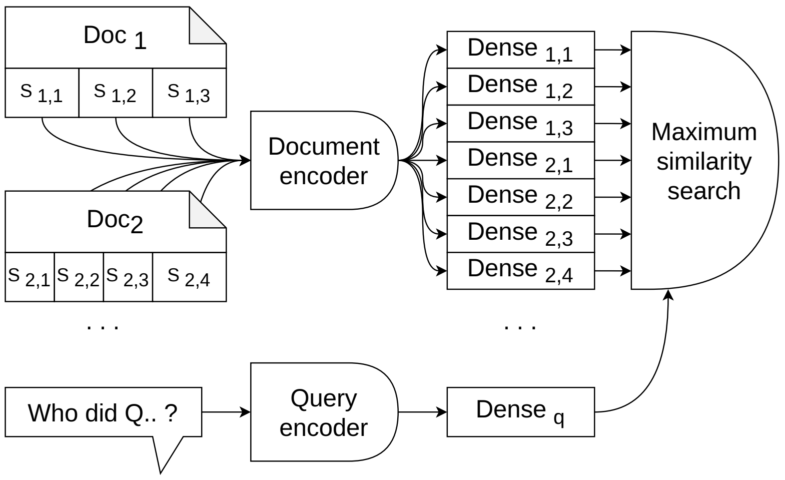

FIGURE 1.3: Conceptualized overview of retrieval on document spans.

The various splitting mechanisms are described in Chapter 2. For this chapter and most of the experiments in this thesis, we use splitting by non-overlapping spans of length 100 tokens. This is slightly suboptimal, but commonly used [36].

1.3.2 Evaluation

There are a plethora of ways of evaluating document retrieval systems, as surveyed briefly by Bama et al. [47]. To evaluate retrieval performance we use two metrics.

For dimension reduction (Chapter 3) we compute *R-Precision* averaged over queries q_i: (relevant documents among top k passages in $Z)/r$, $k =$ number of passages in relevant documents, in the same way as Petroni et al. [3]. For splitting and filtering (Chapter 2) we compute *top-10 accuracy* averaged over queries q_i: $\mathbf{1}_{r \cap (arg\,top\text{-}10_{d_j}\,\mathrm{sim}(q_i,d_j)) \neq \emptyset}$.[7] The reason for using R-Precision is that it is commonly used [47, 48]. This metric can, however, not be used for the splitting and filtering experiments because there we modify the number of relevant passages which makes the comparison even within one experiment impossible and could lead to false conclusions. Using two metrics in one thesis is further warranted by the lack of necessity to compare results between the two chapters.

1.3.3 Similarity Functions

There is no unified mathematical definition of what a similarity function is. A working definition for our context of dense document encoding retrieval is that it is a function which is high for similar vectors and low for dissimilar ones:

$$\mathrm{sim} : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$$

A common approach is to take the inverse or the negation of the L^2 (Euclidean) distance. Note that in contrast to the L^2 distance, which is non-negative, the similarity function is unbounded. Another mathematical function often used for similarity is the inner product (vector dot product).

$$\begin{aligned} L^2(a,b) &= \sqrt{(a-b)^2} = \sqrt{\Sigma_1^d (a_i - b_i)^2} \\ \mathrm{sim}_{L^2}(a,b) &= -L^2(a,b) = -\sqrt{\Sigma_1^d (a_i - b_i)^2} \\ \mathrm{sim}_{\mathrm{IP}}(a,b) &= \mathrm{IP}(a,b) = \Sigma_1^d (a_i \cdot b_i) \end{aligned}$$

A key property of these two functions is that they are recursively decomposable [49] and various improvements can be used to speed up the retrieval. There are other metrics, such as the cosine distance (equivalent to the inner product in normalized space), L^1, L^∞, Canberra [50], Bray-Curtis dissimilarity [51], Jensen-Shannon divergence [52], Mahalanobis distance [53] and many others [54–56]. We focus only on the inner product (IP) and the L^2 distance as the similarity function because either the other metrics are

[7] $\mathbf{1}$ is the indicator function. Top-k accuracy for a single query is 1 if the top k retrieved documents contain at least one relevant document, otherwise 0.

Test	**Type**	**Query Similarity**
HotpotQA: The district that the village of Asamang is located in was split on what date?	Query	
Asamang is a village in the Atwima Nwabiagya district, a district in the Ashanti Region of Ghana.	Relevant span	IP $= 0.476$ $L^2 = -1.023$
The Atwima Nwabiagya District formerly the Atwima District is one of the twenty-seven (27) districts in the Ashanti Region of Ghana. Its capital is Nkawie. In 2003, part of the district was split off by a decree of president John Agyekum Kufuor on November 12, 2003, to form the new Atwima Kwanwoma District and Atwima Mponua District.	Relevant span	IP $= 0.440$ $L^2 = -1.058$
Lowery married filmmaker Augustine Frizzell in 2010. As of 2013, they live in Dallas. Lowery identifies as an atheist, and has been a vegan since around 1996.	Irrelevant span	IP $= -0.176$ $L^2 = -1.534$
Natural Questions: minister of energy and power development in zimbabwe	Query	
The Ministry of Energy and Power Development is a government ministry, responsible for energy and electricity in Zimbabwe. The incumbent minister is Ambassador Joram Gumbo.	Relevant span	IP $= 0.758$ $L^2 = -0.696$
both in June 2017, in protest at Trump's decision to withdraw the United States from the Paris Agreement on climate change.	Irrelevant span	IP $= -0.023$ $L^2 = -1.430$

TABLE 1.2: Example of questions from HotpotQA and Natural Questions. Similarity is measured on centered and normalized embeddings from DPR-CLS. Higher always means more similar (L^2 is for this reason negated).

not widely adopted in the community or the FAISS framework does not fully support them.[8] See Table 1.2 for vector similarities between specific textual spans.

[8]For example, the cosine similarity can be used only for normalized vectors, which would prevent us from experimenting with the unnormalized vectors.

1.3.4 Data

As the knowledge base we use documents from English Wikipedia dump 2019 and follow the setup described by Petroni et al. [3]. We mark spans (original articles split into 100 token pieces, 50 million in total) as relevant for a query if they come from the same Wikipedia article as one of the provenances.[9] In order to make our experiments computationally feasible and easy to reproduce we experiment with a modified version of this knowledge base where we keep only spans of documents that are relevant to at least one query from the training or validation set of our downstream tasks. As downstream tasks, we use HotpotQA [57] for all main experiments and Natural Questions [41] to verify that the results transfer to other datasets as well. They are both widely used by the research community [58–63] and we chose them because of their popularity and availability within the KILT framework. HotpotQA has been sourced through careful crowdsourcing and is aimed at multi-hop reasoning. Natural Questions were created by aggregating and anonymizing queries issued to the Google search engine. We show examples from the two datasets in Table 1.2 together with the vector similarity between the query and selected spans. Note that the vector similarity is higher for the relevant spans than for the irrelevant ones.

This datset preprocessing to over 2 million encoded spans for HotpotQA (see Table 1.3 for dataset sizes). The 768-dimensional embeddings (32-bit floats) of this dataset (both queries and documents) add up to 7GB (146GB for the whole unpruned dataset).

Dataset	**Train queries**	**Dev queries**	**Documents**
HotpotQA	69k	6k	49.7 Mio.*
HotpotQA (pruned)	69k	6k	2.1 Mio.
Natural Questions (pruned)	78k	2k	1.6 Mio.

Table 1.3: Number of training and dev queries and documents for the different datasets used. *Whole Wikipedia dump.

1.3.5 Model Comparison

To establish baselines for uncompressed performance we use models based on BERT [64]. We consider (1) vanilla BERT, (2) SentenceBERT [42] and (3) DPR [36], which was specifically trained for document retrieval. See Section 1.3.6 for a brief overview of

[9]Spans of the original text which help in answering the query.

how these models were trained and their differences. To obtain document embeddings, we use either the last hidden state representation at `[CLS]` or the average across tokens of the last layer (both are commonly used to get the vector representation of the input).

Our first experiment compares the retrieval performance of the different models on HotpotQA. The result is shown in Figure 1.4. In alignment with previous works [42] an immediately noticeable conclusion is that vanilla BERT has a poor performance,[10] especially when taking the hidden state representation for the `[CLS]` token. Next, to make the computation of experiments tractable on available hardware, we repeat the experiment using FAISS [49],[11] which is a framework for fast approximate similarity search. We find that the performance loss across models is systematic, which warrants the use of this approximation for comparisons and all our following experiments will use FAISS on the DPR-CLS model.[12]

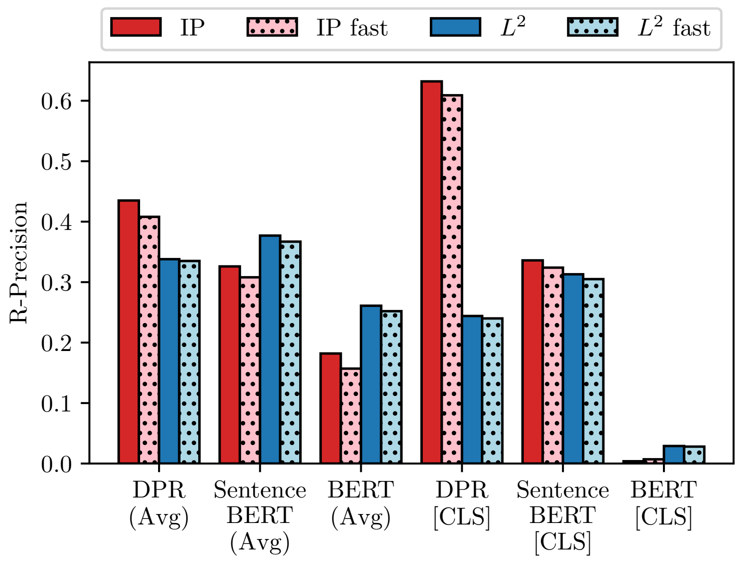

FIGURE 1.4: Comparison of different BERT-based embedding models and versions when using faster but slightly inaccurate nearest neighbour search. **[CLS]** is the specific token embedding from the last layer while **(Avg)** is all token average.

To provide further intuition to why normalization may help we show specific query and document vectors in Figure 1.6. Note that while the centering of one dimension is independent of centering of the other dimensions, the normalization of one dimension is dependen on other dimensions. For this reason, we consider only the first two dimensions of the 768-dimensional vectors. Also note that the preprocessing of queries is

[10]See Section 1.3.6 for explanation.
[11]Parameters: IndexIVFFlat, nlist=200, nprobe=100.
[12]The authors of DPR also used FAISS in their paper when presenting results.

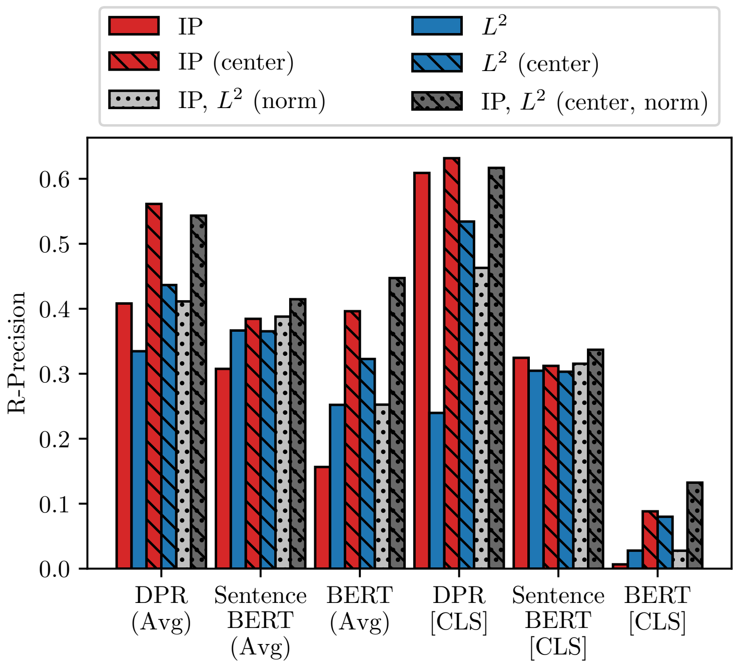

FIGURE 1.5: Effect of data centering and normalization on performance (evaluated with FAISS).

independent on the preprocessing of documents. For this reason we can see that while centering preserves the shape, it moves the queries down-left, while the documents are shifted up-left. The normalization sets the vectors along a hypersphere (a circle in 2D case). Formally, this slightly reduces the information content of one vectors by almost one dimension.[13] With this, the vectors are differentiated only by the angles and not their absolute magnitude. This further allows us to unify the ordering given by L^2, the inner product and the cosine similarity.

1.3.6 Embedding Model Overview

In this subsection, we provide a brief background of the design and training of the pre-trained models used for embedding spans: BERT, SentenceBERT and DPR. While many more models exist, we chose these three because they are commonly used and the later two are based on the first one and vastly improve on it.

[13]If we know 767 dimensions $d_1, \ldots d_{767}$, we can deduce that the last element is going to be $d_{768} = \pm\sqrt{1 - \sum_1^{767} d_i^2}$. If we used polar coordinates, we would need exactly 767 numbers because the magnitude is always 1.

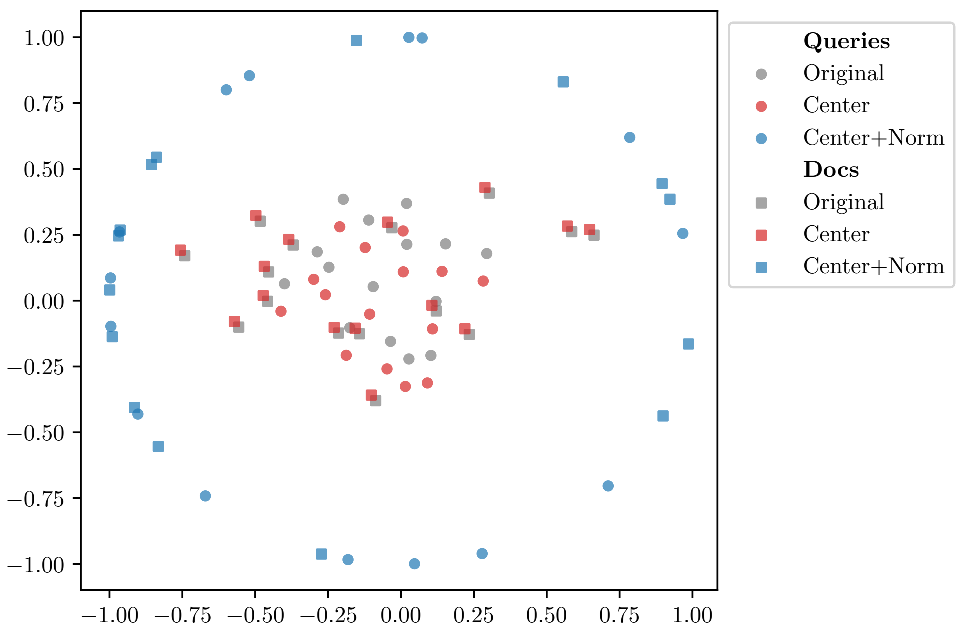

FIGURE 1.6: First two dimensions of 15 random queries and 15 random documents and their positions after preprocessing.

1.3.6.1 BERT

Bidirectional Encoder Representations from Transformers (BERT [65]) is based on the encoder part of the transformers [66] architecture (multiple blocks of embedding, multihead self-attention, feed forward and layer normalization). The goal is to obtain *some* representation of the input such that it can be finetuned and used for various other tasks either directly or with a small degree of finetuning.[14] The way this is done is by pretraining the model on large amounts of data on two tasks: language modelling and next-sentence prediction. Consider the following two inputs (each on separate line):

[CLS] [MASK] hedgehog fell [MASK] . [SEP] Gabriel García Márquez was born in 1927 . [SEP]
[CLS] He grew up with [MASK] grandparent . [SEP] His grandfather used to be a colonel . [SEP]

The goal of masked language modelling is to predict the masked tokens from context (variable number of tokens were masked, 15%). For the two examples the gold answers are *The*, *asleep* and *his*, respectively. The goal of next sentence prediction is to determine whether the second sentence follows the first. In the first example the sentences are not probable to be consecutive while in the second example they are. For the pretraining the last block in the model is followed up with classification layers. Notice the special tokens

[14]The original paper used natural language understanding but it has been applied for a plethora other NLP tasks [67–76]. Because of its popularity, it has also been adopted into other languages [77–84] and exists in a multilingual version as mBERT.

`[CLS]`, `[MASK]` and `[SEP]` which provide some structure to the input. Furthermore, BERT does not use whole tokens as the input units but subword made by WordPiece [85] and therefore the vocabulary consists of parts of words (this is not depicted in the example).

1.3.6.2 SentenceBERT

While both BERT and Roberta [86] improved the state of the art for tasks relying sentence pair similarity, the main issue is that the two sentences need to be put into the model. To find the most semantically similar sentence (to a specific one in a dataset of 50 Mio. sentences would require 50 Mio. inferences of the large model, which is highly impractical from compute resources point of view.

The authors of SentenceBERT try to alleviate this by trying to find a function f for which the following holds (sim is a vector similarity function, in their case cosine similarity):

$$\arg\max_i \ \text{sim}(f(s_i), f(s_j)) \quad \Leftrightarrow \quad s_i \text{ is semantically similar for } s_j$$

That is, they build a model which computes the embedding of the sentence such that in the new vector space, similar sentences are close together (based on cosine similarity). We can precompute $f(s_i)$ for every sentence and during inference, we only need to compute $f(s_j)$ and run a maximum similarit search over the precomputed dataset. The model is based on BERT but with a modified objective function, which (among others experimented with) is the following (given random positive and negative sentences s_p and s_n):

$$\max(||f(s_i) - f(s_p)|| - ||f(s_i) - f(s_n)|| + \epsilon, 0)$$

In practice, this minimizes $||f(s_i) - f(s_p)||$ while maximizing $||f(s_i) - f(s_n)||$. The ϵ is just a stabilization constant that ensures that the difference between the two distances is at least ϵ. The model is trained on the SNLI and Multi-Genre SNLI dataset which classifies pairs of sentences with *contradiction*, *entailnment* and *neutral*.

Dense Passage Retrieval

Because the DPR model is the one used for most experiments in this thesis, it is the most important. The goal for document retrieval is to have functions f_d and f_q for which the following holds:

$$\arg\max_i \ \ \text{sim}(f_d(d_i), f_q(q_j)) \quad \Leftrightarrow \quad d_i \text{ is relevant for } q_j$$

Note that for BERT and SentenceBert we use $f_d = f_q$ despite text spans $d_i \in \mathcal{D}$ and $q_j \in \mathcal{Q}$ coming from different distribution. One of the advantages of DPR is that it distinguishes these two functions. They start with two pretrained BERT base uncased models and optimize them in the following loss using contrastive learning (given batch i, query q_i, a relevant span p_i and set of negative samples N_i):

$$\mathcal{L}(q_i, p_i, N_i) = -\log \frac{e^{\text{sim}(q_i,p_i)}}{e^{\text{sim}(q_i,p_i)} + \sum_j e^{\text{sim}(q_i,N_{i,j})}}$$

Note that q_i, p_i and N_i are all the results of computations via f_q and f_d to which loss can be backpropagated. This essentially computes a softmax of the following vector at the first position:

$$[\text{sim}(q_i, p_i), \text{sim}(q_i, N_{i,1}), \text{sim}(q_i, N_{i,2}), \ldots, \text{sim}(q_i, N_{i,|N_i|})]$$

The whole fraction is in the interval $(0, 1)$. We want the nominator to be large (high similarity with the relevant span) while the denominator to be small (low similarity with irrelevant spans). Minimizing the negative log of this fraction pushes the fraction to be close to 1.

The positive span is easily available from the dataset while the negative spans need to be sampled carefully. The authors discuss multiple strategies for this selection, including (1) random sampling, (2) top false output of BM25, (3) positive spans for other questions in the batch, also called in-batch negatives, and (4) the combination of them, which results in their best model.

Chapter 2

Splitting & Filtering

This chapter describes the results of two methods aimed at improving retrieval performance while decreasing knowledge base size. Although the results are negative (no reasonable applicable improvement or insights have been achieved), they explore several baseline ideas and empirically answer the questions that many researchers interested in dense retrieval could ask.

In this section, we consider the pruned Wikidata with DPR-CLS encoding and centering and normalization. The effect of post-processing is detailed in Chapter 3.

2.1 Splitting

As introduced in Chapter 1, the documents are rarely used whole for retrieval. Instead, we split them into smaller chunks using a function $s(d) \in 2^d$ and create a new set of documents $\mathcal{S} = \bigcup_{d \in \mathcal{D}} s(d)$.[1]Recomputing span relevancy is not an issue because we consider any span that leads to the correct article to be a *hit* (given set of provenances for a query $\mathcal{P}_q$): $\text{hit}(q, d) \overset{def}{\Leftrightarrow} (\exists D \in \mathcal{D}, p \in \mathcal{P}_q : d \subseteq D \wedge p \subseteq D)$. It is common to use non-overlapping spans of 100 tokens [36] but this can create several issues which are demonstrated using the following two crafted examples:

0 can refer to either the most or least significant bit depending on | *the context.*

The first programmable computer, built by K. Zuse, | *used binary notation for numbers.*

In the first one, we split at the end and create a three token span that does not hold any relevant information and only takes up space. In the second example, we split in the

[1]Note that this notation allows for the spans to also be of various lengths, non-continuous and overlapping .

middle of the sentence and the compositional meaning required to answer the following question is lost:

Who built the first programmable computer that used binary notation for number?

This issue arises because the splitting is not done on syntactic nor semantic boundaries. We examine it empirically by using different splitting schemas, either splitting on tokens with different span sizes (including using overlap with previous and following tokens) or splitting at sentence boundaries. Figure 2.1 shows the results of these methods with the aforementioned setting (DPR-CLS, HotpotQA pruned). Note that we are not interested in just improving the retrieval performance but also in reducing the knowledge base size. Naturally, smaller spans lead to a higher index size because each has to be represented with a 768-dimensional vector. The reason why the difference in passage counts between e.g. Sent 1 and Sent 2 is larger than the difference between Sent 5 and Sent 6 is that not many paragraphs are longer than 5 sentences.

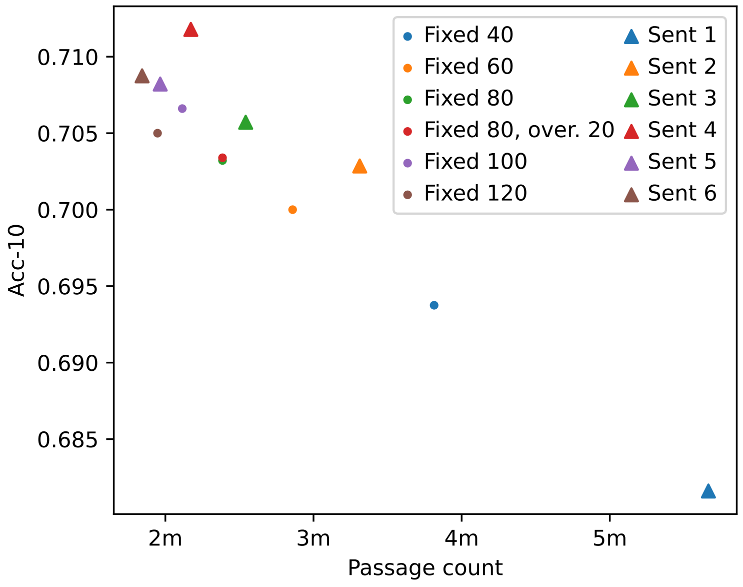

FIGURE 2.1: Retrieval performance and knowledge base size after being split using different methods. Note the cut-off y-axis.

Surprisingly, the various splitting strategies are not vastly better than the default one (Fixed 100). This can be attributed to using the DPR model, which is fine-tuned for retrieval and anything other than 100-token spans is outside of its finetuned input distribution. An exception is that splitting on sentence boundaries (specifically Sent 5 and

Sent 6) seems to be slightly systematically better than splitting at token count boundaries, even with the same number of passage counts. This can be possibly explained via the examples introduced previously in this section.

2.2 Filtering

This section is concerned with reducing the knowledge-base size by simply filtering some of the spans which are deemed to not hold any interesting information and do not help with matching to the relevant document. Either handcrafted or automatically derived, we are trying to find a function $h(d) \in \{0, 1\}$ that classifies whether a given span should be retained and we construct a new set of spans as $\mathcal{S}' = \{d | d \in \mathcal{S} \wedge h(d)\}$. For the following experiments, we use the splitting by 100 tokens so that the results are comparable across the thesis. Again, we are interested in reducing the passage count while retaining the retrieval performance.

2.2.1 Heuristics

The most straightforward option is to remove spans that are too short, as shown in the introductory example. Figure 2.2 shows the results with various filters based on either token or character count, i.e. $h_x(d) \Leftrightarrow \text{count}_{\text{word/char}}(d) \geq x$. Unfortunately, any filtering measures show a strong regression against not using any filtering and thus this trade-off is not worth it. Counterintuitively, this is true for also seemingly noninvasive heuristics, such as the number of words or characters being larger than 2 or 10, respectively. The reason for this may be that these filtered short spans are not actual outliers, as shown by the distribution in Figure 2.3.

2.2.2 Automatic filtering

The issue with filtering using handcrafted heuristics is that it requires arbitrary human decision making and can not be automated. Simple evaluation metrics, such as accuracy, consider top k most similar spans to a query (k is fixed). The idea for automatic filtering is to simply remove span s which satisfies the following two conditions given the training queries Q_T:

- *At least one negative*: There exists at least one query for which this span is not relevant but for which it has been retrieved.
- *Never positive*: No query for which this span is relevant retrieves it.

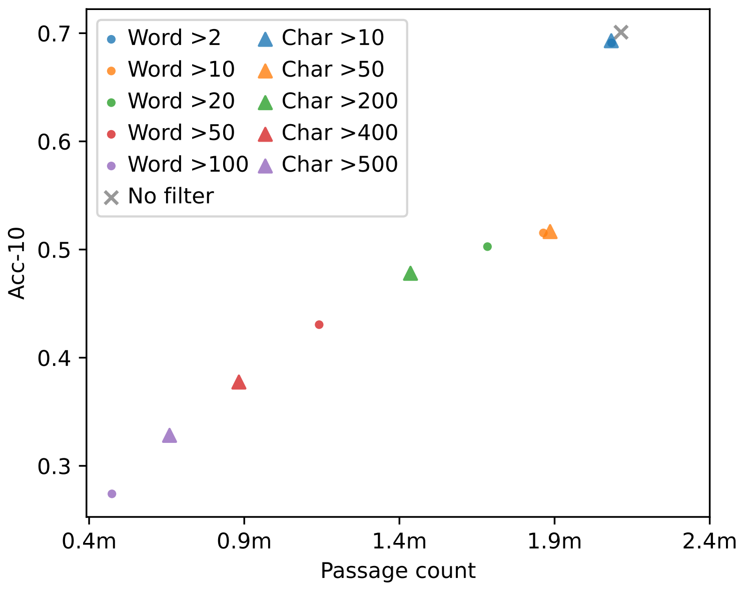

FIGURE 2.2: Retrieval performance and knowledge base size after being filtered using length-based heuristics. Note the cut-off y-axis.

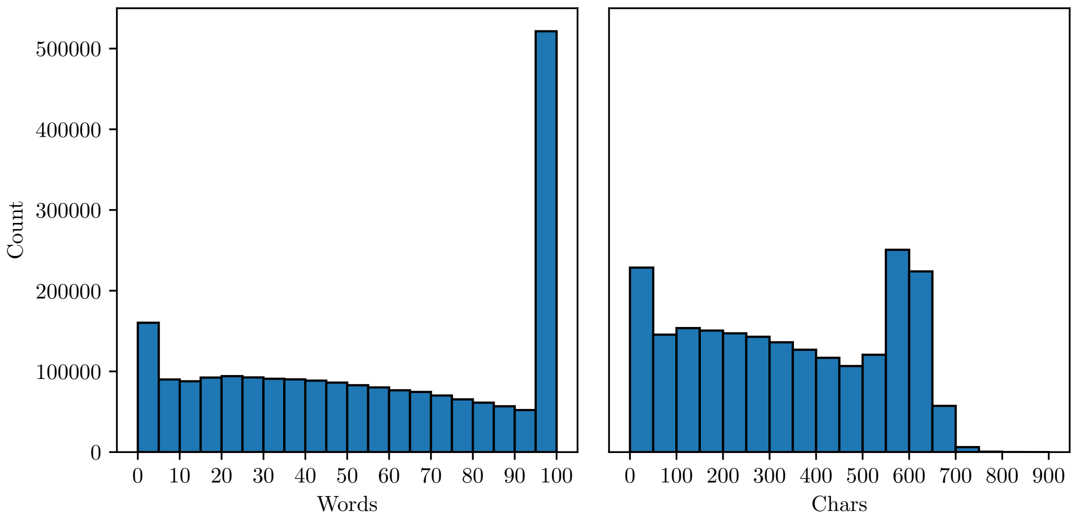

FIGURE 2.3: Distribution of span length either with word or characte count.

This filtering can be applied iteratively and two steps are visualized in Figure 2.4. This algorithm is also described in pseudocode in Listing 2.1.

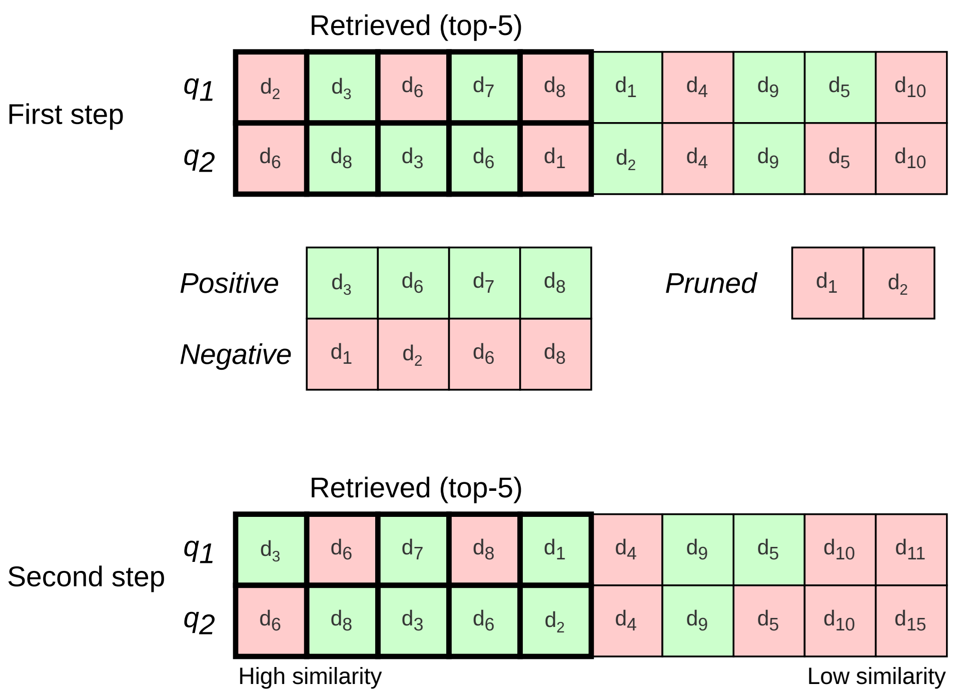

FIGURE 2.4: Example of two retrieval and one pruning steps on two queries. For both queries, the spans are sorted from left to right by decreasing similarity. Spans with bold borders are those considered as retrieved. Spans in green are relevant for the given query and spans in red are not relevant.

```
$\textsc{FilterStep}(D, Q, k)$ :                          *spans and queries*
    Negative $\leftarrow \{\}$
    Positive $\leftarrow \{\}$
    $\textsc{For}\ (q_i, r_i) \in Q$ :                    *query and relevant spans*
        $d \leftarrow \textsc{Retrieve}_k(q_i)$
        Negative $\leftarrow$ Negative $\cup (d \setminus r_i)$
        Positive $\leftarrow$ Positive $\cup (d \cap r_i)$
    Negative $\leftarrow$ Negative $\setminus$ Positive
    $D' \leftarrow D \setminus$ Negative                  *prune spans*
    $out \leftarrow (D'$, Positive, Negative$)$
```

LISTING 2.1: Definition of a single filtering step that prunes the spans and provides negative and positive samples

It is clear that by repeated application of this filtering step, the training retrieval performance is monotonous non-decreasing because we are never filtering spans that contribute positively to the metric. Training accuracy is shown in the left of Figure 2.5. The k_f by which the top-k_f spans in the filtering are considered can be the same as the k_a in the accuracy evaluation metric. If it is smaller than k_a, then the monotonicity property on the training data may not hold. If it is larger, then we may get faster convergence, as shown on the right of Figure 2.5. This is because we consider more negative and positive spans overall.

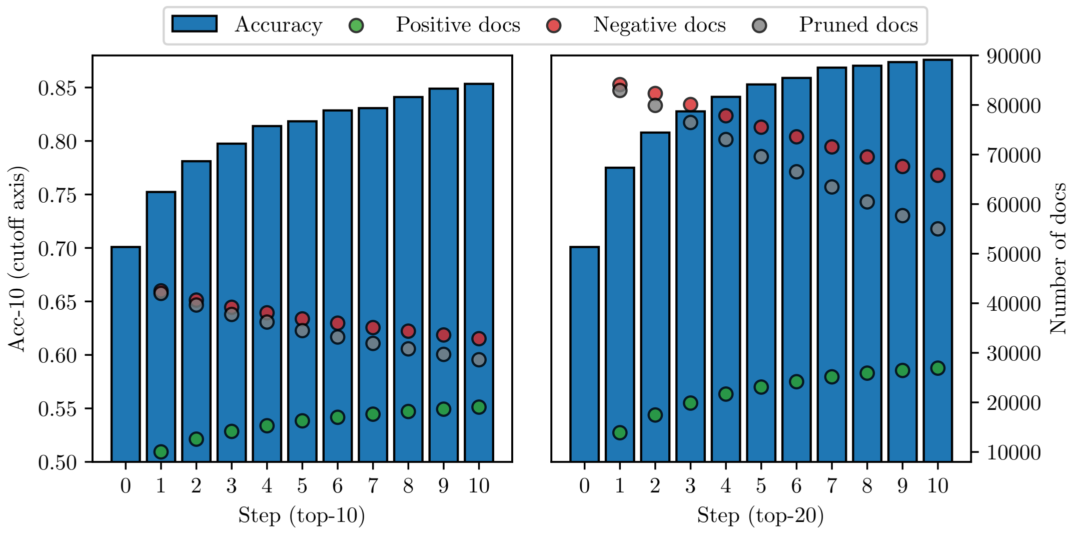

FIGURE 2.5: Autofiltering performance of dev queries on spans filtered using the same dev queries. Filtering is done using either top-10 or top-20 spans.

We are however interested in the performance on queries not seen during training. There are two approaches that we consider: (1) using pruning based on training queries and (2) training a classifier to predict whether span is positive or only negative. The results for the first one are shown in Figure 2.6 and unfortunately, the performance only deteriorates. This is because even though the training and dev queries are drawn from the same distribution, this method has no way of generalizing and filtering only spans that are never relevant to any query (i.e. spans without any factual knowledge).

The other approach, which uses logistic regression to determine whether a span should be pruned, does not work either and by removing ~5% of spans regresses to 56% retrieval accuracy in the first step. Its performance is only marginally better than the most common class classifier, as shown in Figure 2.7.

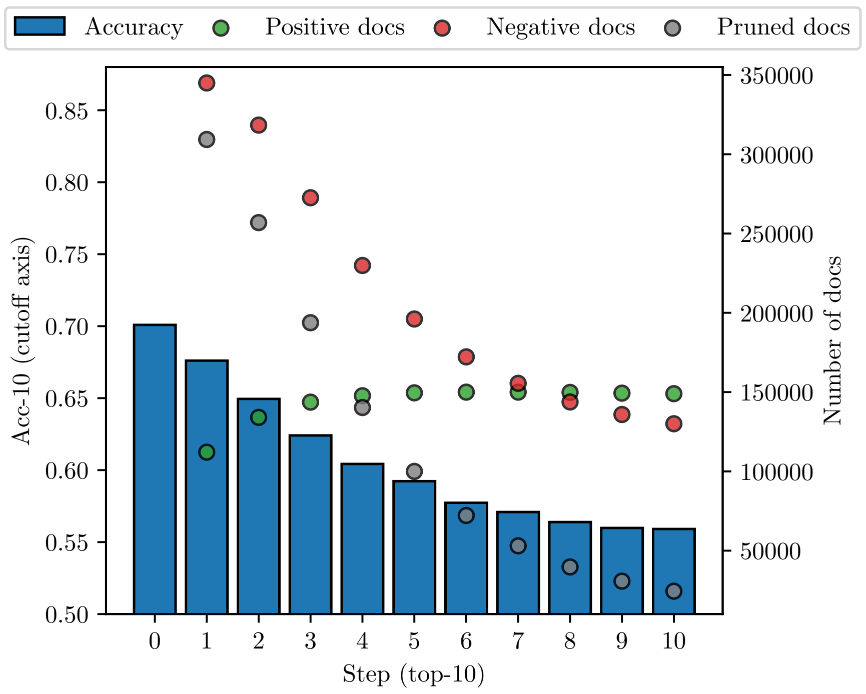

FIGURE 2.6: Autofiltering performance of dev queries on spans filtered using train queries. Filtering is done using top-10 spans.

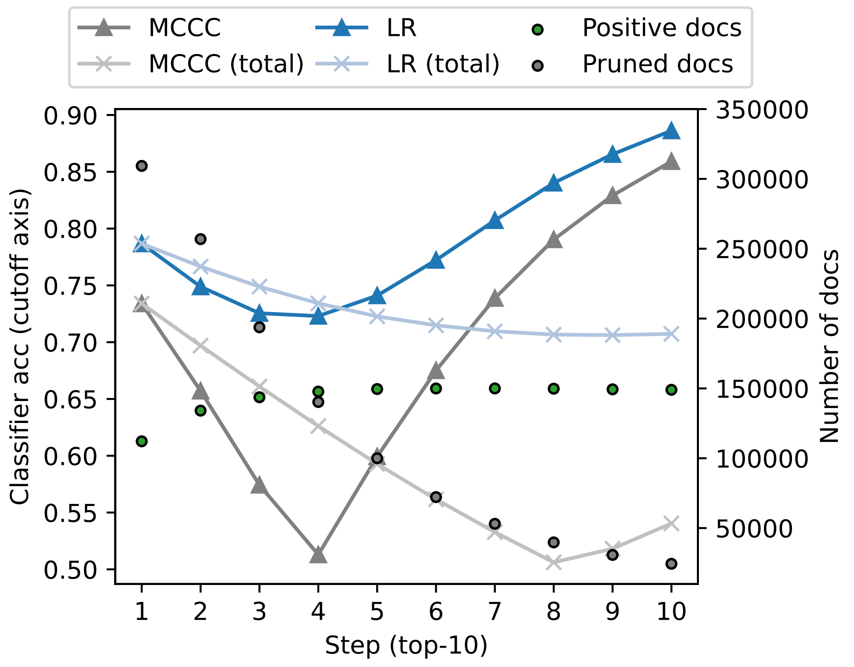

FIGURE 2.7: Training classifier accuracy for most common class classifier and logistic regression based on data from individual filtering steps or cummulative data up to a certain step (total).

2.2.3 Discussion

Neither manual based on span length nor automatic filtering showed any applicable results. Manual filtering could be further enhanced with named entity recognition and allow only spans with e.g. at least one entity.

For automatic filtering, we would like to point out that we manipulated only positive and negative sets of spans and disregarded spans that were never retrieved. Focusing on those is a venue for future work and could reduce the knowledge base size without affecting the retrieval performance neither positively nor negatively. A possible solution to filter irretrievable spans is to consider spans that are least similar to the training queries, possibly via outlier detection methods. Finally, we considered this filtering mechanism only on the level of vectors but it would be possible to also use the textual form of the span as well.

Chapter 3

Dimension Reduction

In Section 3.1, we describe the problem setup of dimensionality reduction for document retrieval. We discuss the results of different compression methods (random projections, PCA, autoencoder, precision reduction and their combination) in Section 3.2 and provide further analysis in Section 3.3.

3.1 Problem Statement

To speed up the similarity computation over a large set of documents and to decrease memory usage (f_D is usually precomputed), we apply **dimension reduction functions** $r_Q : \mathbb{R}^d \rightarrow \mathbb{R}^{d'}$ and $r_D : \mathbb{R}^d \rightarrow \mathbb{R}^{d'}$ for the query and document embeddings respectively. Formally, we are solving the following problem (extended from Section 1.3):

$$Z = \underset{d \in \mathcal{D}}{\arg \text{top-k}}\ \text{rel.}(q, d)\ , \text{with} \tag{3.1}$$

$$\text{rel.}(q, d) \approx \text{sim}(f_\text{Q}(q), f_\text{D}(d)) \tag{3.2}$$

$$\approx \text{sim}(r_\text{Q}(f_\text{Q}(q)), r_\text{D}(f_\text{D}(d))) \tag{3.3}$$

Since f_Q is commonly fine-tuned for a specific downstream task, it is desirable in (3) for the functions r_Q and r_D to be differentiable so that they can propagate the signal. These dimension-reducing functions need not be the same because even though they project to a shared vector space, the input distribution may still be different. Similarly to the query and document embedding functions, they can be fine-tuned.

3.1.1 Pre-processing Transformations

Figure 1.4 also shows that model performance, especially for DPR, depends heavily on what similarity metric is used for retrieval. This is because none of the models produces normalized vectors by default.

Figure 1.5 shows that performing only normalization $\left(\frac{\boldsymbol{x}}{||\boldsymbol{x}||}\right)$ sometimes hurts the performance but when joined with centering beforehand $\left(\frac{\boldsymbol{x}-\bar{\boldsymbol{x}}}{||\boldsymbol{x}-\bar{\boldsymbol{x}}||}\right)$, it improves the results (compared to no pre-processing) in all cases. The normalization and centering is done for queries and documents separatedly. Moreover, if the vectors are normalized, then the retrieved documents are the same for L^2 and inner product. [1]

Nevertheless, we argue it still makes sense to study the compression capabilities of L^2 and the inner product separately, since the output of the compression of normalized vectors need not be normalized.

Another common approach before any feature selection is to use z-scores ($\frac{x-\bar{x}}{\sigma}$) instead of the original values. Its boost in performance is however similar to that of centering and normalization. The effects of each pre-processing step are in Table 3.1. The significant differences in performance show the importance of data pre-processing (agnostic to model selection).

	IP	**L^2**
DPR-CLS	0.609	0.240
Center	0.630	0.353
Z-Score	0.632	0.525
Norm.	0.463	
Center + norm.	0.618	
Z-Score + norm.	0.621	

Table 3.1: Effect of pre-processing transformations on embeddings produced by DPR-CLS. Means and standard deviations are computed separately for documents and queries. Transformation into z-scores includes centering.

[1] $\arg\max_k -||\boldsymbol{a}-\boldsymbol{b}||^2 = \arg\max_k -\langle \boldsymbol{a}, \boldsymbol{a}\rangle^2 - \langle \boldsymbol{b}, \boldsymbol{b}\rangle^2 + 2\cdot\langle \boldsymbol{a}, \boldsymbol{b}\rangle = \arg\max_k\ 2\cdot\langle \boldsymbol{a}, \boldsymbol{b}\rangle - 2 = \arg\max_k\ \langle \boldsymbol{a}, \boldsymbol{b}\rangle$

3.2 Compression Methods

Having established the retrieval performance of the uncompressed baseline, we now turn to methods for compressing the dense document index and the queries.

3.2.1 Random Projection

The simplest way to perform dimension reduction for a given index $\boldsymbol{x} \in \mathbb{R}^d$ is to randomly preserve only certain d' dimensions and drop all other dimensions:

$$f_{\text{drop.}}(\boldsymbol{x}) = (x_{m_1}, x_{m_2}, \ldots, x_{m_{d'}})$$

Another approach is to greedily search which dimensions to drop (those that, when omitted, either improve the performance or lessen it the least):

$$\begin{aligned}
&p_i(\boldsymbol{x}) = (x_0, x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_{768}) \\
&\mathcal{L}_i = \text{R-Prec}(p_i(Q), p_i(D)) \\
&m = \text{sort}^{\text{desc.}}_{\mathcal{L}}([1 \ldots 768]) \\
&f_{\text{greedy drop.}}(\boldsymbol{x}) = (x_{m_1}, x_{m_2}, \ldots, x_{m_{d'}})
\end{aligned}$$

The advantage of these two approaches is that they can be represented easily by a single $\mathbb{R}^{768\times d}$ matrix. We consider two other standard random projection methods: Gaussian random projection and Sparse random projection [87]. Such random projections are suitable mostly for inner product [88] though the differences are removed by normalizing the vectors (which also improves the performance).

Results. The results of all random projection methods are shown in Figure 3.1. Gaussian random projection seems to perform equally to sparse random projection. The performance is not fully recovered for the two methods. Interestingly, simply dropping random dimensions led to better performance than that of sparse or Gaussian random projection. The greedy dimension dropping even improves the performance slightly over random dimension dropping in some cases before saturating and is deterministic. As shown in Table 3.4, the greedy dimension dropping with post-processing achieves the best performance among all random projection methods. Without post-processing, L_2 distance works better compared to the inner product.

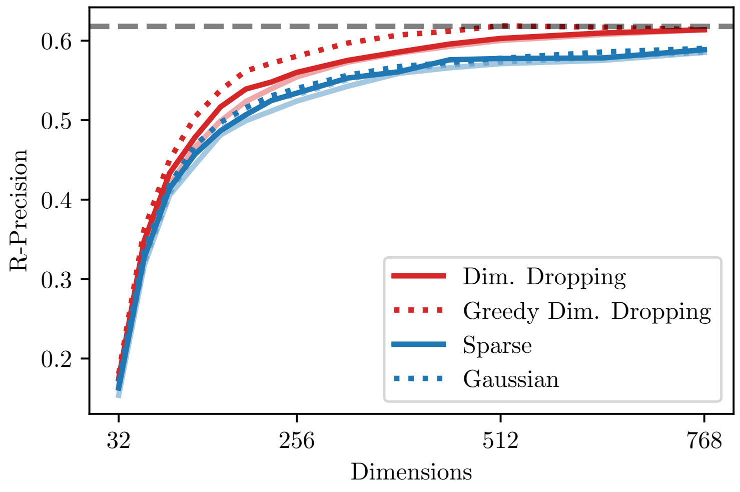

FIGURE 3.1: Dimension reduction using different random projections methods. Presented values are the max of 3 runs (except for greedy dimension dropping, which is deterministic), semi-transparent lines correspond to the minimum. Embeddings are provided by centered and normalized DPR-CLS. Final vectors are also post-processed by centering and normalization.

3.2.2 Principal Component Analysis

Another natural candidate for dimensionality reduction is principal component analysis (PCA) [89]. PCA considers the dimensions with the highest variance and omits the rest. This leads to a projection matrix that projects the original data onto the principal components using an orthonormal basis T. The following loss is minimized $\mathcal{L} = \text{MSE}(\mathbf{T}'\mathbf{T}\boldsymbol{x}, \boldsymbol{x})$. Note that we fit PCA on the covariance matrix of either the document index, query embeddings or both and the trained dimension-reducing projection is then applied to both the document and query embeddings.

Results. The results of performing PCA are shown in Figure 3.2. First, we find that the uncompressed performance, as well as the effect of compression, is highly dependent on the data pre-processing. This should not be surprising as the PCA algorithm assumes centered and pre-processed data. Nevertheless, we stress and demonstrate the importance of this step. This is given by the normalization of the input vectors and also that the column vectors of PCA are orthonormal.

Second, when the data is not centered, the PCA is sensitive to what it is trained on. Figure 3.2 show systematically that training on the set of available queries provides better performance than training on the documents or a combination of both. Subsequently, after centering the data, it does not matter anymore what is used for fitting: **both the queries and the documents provide good estimates of the data variance** and

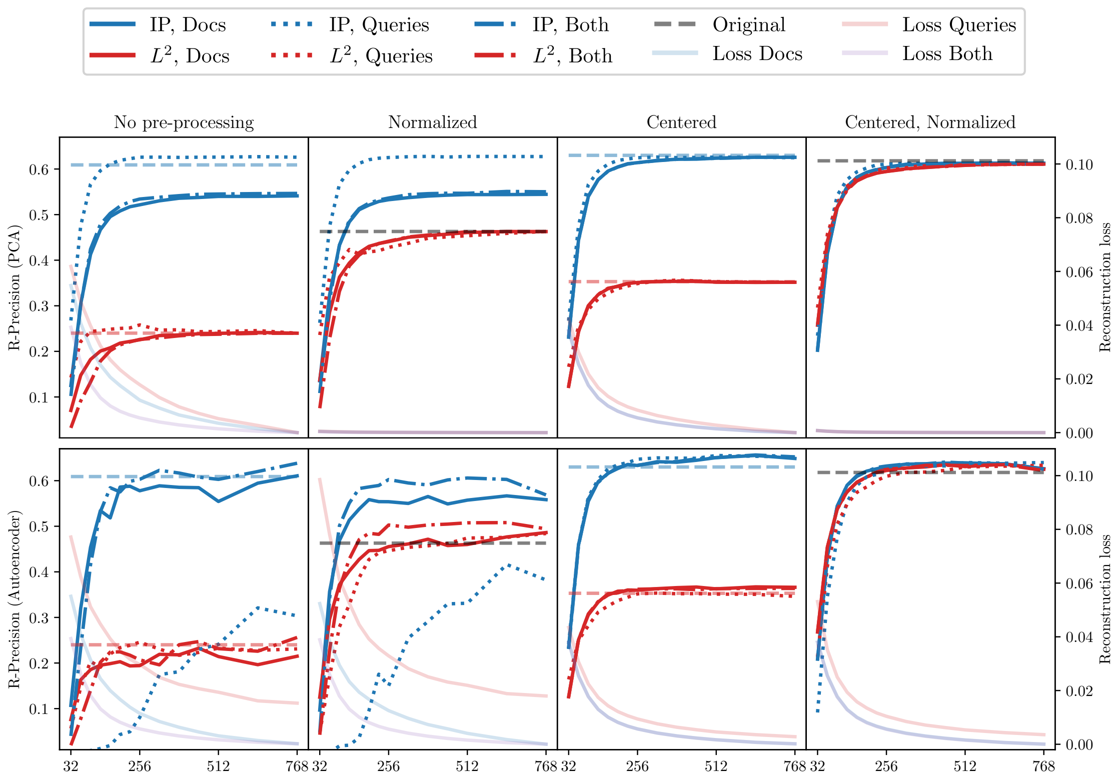

FIGURE 3.2: Dimension reduction using PCA (top) and Autoencoder (bottom) trained either on document index, query embeddings or both. Each figure corresponds to one of the four possible combinations of centering and normalizing the input data. The output vectors are not post-processed. Reconstruction loss (MSE, average for both documents and queries) is shown in transparent colour and computed in original data space. Horizontal lines show uncompressed performance. Embeddings are provided by DPR-CLS.

the dependency on training data size for PCA is explored explicitly in Section 3.3.1. The reason why queries provide better results without centering is that they are more centered in the first place, as shown in Table 3.2.

	Avg. L^1 (std)	**Avg. L^2 (std)**
Documents	243.0 (20.1)	12.3 (0.6)
Queries	137.0 (7.5)	9.3 (0.2)

TABLE 3.2: Average L^1 and L^2 norms of document and query embeddings from DPR-CLS without pre-processing.

In all cases, the PCA performance starts to plateau around 128 dimensions and is within 95% of the uncompressed performance. Finally, we note that while PCA is concerned with minimizing reconstruction loss, Figure 3.2 shows that even after vastly decreasing the reconstruction loss, no significant improvements in retrieval performance are achieved. We further discuss this finding in Section 4.2.

Component Scaling. One potential issue of PCA is that there may be dimensions that dominate the vector space. Mu et al. [90] suggest to simply remove the dimension corresponding to the highest eigenvalue though we find that simply scaling down the top k eigenvectors systematically outperforms standard PCA. For simplicity, we focused on the top 5 eigenvectors and performed a small-scale grid-search of the scaling factors. The best performing one was $(0.5, 0.8, 0.8, 0.9, 0.8)$ and Table 3.4 shows that it provides a small additional boost in retrieval performance.

3.2.3 Autoencoder

A straightforward extension of PCA for dimensionality reducing is to use autoencoders, which has been widely explored [91, 92]. Usually, the model is described by an encoder $e : \mathbb{R}^d \to \mathbb{R}^b$, a function from a higher dimension to the target (bottleneck) dimension and a decoder $r : \mathbb{R}^b \to \mathbb{R}^d$, which maps back from the target dimension to the original vector space. The final (reconstruction) loss is then commonly computed as $\mathcal{L} = \text{MSE}((r \circ e)(\boldsymbol{x}), \boldsymbol{x})$. To reduce the dimensionality of a dataset, only the function e is applied to both the query and the document embedding. We consider three models with the bottleneck: [2]

1. A linear projection similar to PCA but without the restriction of orthonormal columns:

$$e_1(\boldsymbol{x}) = L_{128}^{768}$$
$$r_1(\boldsymbol{x}) = L_{768}^{128}$$

2. A multi-layer feed forward neural network with tanh activation:

$$e_2(\boldsymbol{x}) = L_{512}^{768} \circ \tanh \circ L_{256}^{512} \circ \tanh \circ L_{128}^{256}$$
$$r_2(\boldsymbol{x}) = L_{256}^{128} \circ \tanh \circ L_{512}^{256} \circ \tanh \circ L_{768}^{512}$$

[2] L_b^a symbolizes a fully connected linear layer from a dimensions to b dimensions. Layer composition is done with $\circ$.

3. The same encoder as in the previous model but with a shallow decoder:

$$
\begin{aligned}
e_3(\boldsymbol{x}) &= L^{768}_{512} \circ \tanh \circ L^{512}_{256} \circ \tanh \circ L^{256}_{128} \\
r_3(\boldsymbol{x}) &= L^{128}_{768}
\end{aligned}
$$

Compared to PCA, it is able to model non-pairwise interaction between dimensions (in the case of models 2 and 3 also non-linear interaction). Hyperparameters are listed in Table 3.3.

Hyperparameters	
Batch size	128
Optimizer	Adam
Learning rate	10^{-3}
L_1 regularization	$10^{-5.9}$

TABLE 3.3: Hyperparameters of autoencoder architectures. L_1 regularization is used only when explicitly mentioned.

Results. We explore the effects of training data and pre-processing with results for the first model shown in Figure 3.2. Surprisingly, the Autoencoder is even more sensitive to proper pre-processing than PCA, most importantly centering which makes the results much more stable.

The rationale for the third model is that we would like the hidden representation to require as little post-processing as possible to become the original vector again. The higher performance of the model with shallow decoder, shown in Table 3.4 supports this reasoning. An alternative way to reduce the computation (modelling dimension relationships) in the decoder is to regularize the weights in the decoder. We make use of L_1 regularization explicitly because L_2 regularization is conceptually already present in Adam's weight decay. This improves each of the three models.

Similarly to the other reconstruction loss-based method (PCA), without post-processing, the inner product works yields better results.

3.2.4 Precision Reduction

Lastly, we also experiment with reducing index size by lowering the float precision from 32 bits to 16 and 8 bits. Note that despite their quite high retrieval performance, they

Method	**Compression**	**Original**		**Center + Norm.**
		IP	L^2	{IP, L^2} (% original)
Original	1×	0.609	0.240	0.618 (100%)
Gaussian Projection (128)	6×	0.413	0.453	0.468 (76%)
Sparse Projection (128)	6×	0.398	0.448	0.457 (74%)
Dimension Dropping (128)	6×	0.426	0.466	0.478 (77%)
Greedy Dimension Dropping (128)	6×	0.447	0.478	0.504 (82%)
PCA (128)	6×	0.577	0.562	0.579 (94%)
PCA (128, scaled top 5)	6×	0.586	0.572	0.592 (96%)
Autoencoder (128, single layer)	6×	0.585	0.569	0.588 (95%)
Autoencoder (128, full)	6×	0.564	0.560	0.588 (95%)
Autoencoder (128, shallow decoder)	6×	0.599	0.582	0.599 (97%)
Autoencoder (128, single layer) + L_1	6×	0.600	0.587	0.601 (97%)
Autoencoder (128, full) + L_1	6×	0.573	0.569	0.589 (95%)
Autoencoder (128, shallow decoder) + L_1	6×	0.601	0.591	0.601 (97%)
Precision 16-bit	2×	0.612	0.610	0.615 (100%)
Precision 8-bit	4×	0.613	0.610	0.614 (99%)
Precision 1-bit (offset 0.5)	32×	0.559	0.556	0.561 (91%)
Precision 1-bit (offset 0)	32×	0.530	0.556	0.561 (91%)
PCA (245) + Precision 1-bit (offset 0.5)	100×	0.459	0.458	0.461 (75%)
PCA (128) + Precision 8-bit	24×	0.558	0.553	0.567 (92%)

TABLE 3.4: Overview of compression method performance (from 768) using either L^2 or inner product for retrieval. Inputs are based on centered and normalized output of DPR-CLS and the outputs optionally post-processed again. Performance is measured by R-Precision on the pruned HotpotQA dataset.

only reduce the size by 2 and 4 respectively (as opposed to 6 by dimension reduction via PCA to 128 dimensions). Another drawback is that retrieval time is not affected because the dimensionality remains the same. For more intuition, Figure 3.3 and table 3.5 illustrate the effect of floating point precision reduction.

Using only one bit per dimension is a special case of precision reduction suggested by Yamada et al. [28]. Because we use centered data, we can define the element-wise transformation function as:

$$f_\alpha(x_i) = \begin{cases} 1-\alpha & x_i \geq 0 \\ 0-\alpha & x_i < 0 \end{cases}$$

Bit `1` would then correspond to $1-\alpha$ and `0` to $0-\alpha$. While Yamada et al. [28] use values 1 and 0, we work with 0.5 and -0.5 in order to be able to distinguish between

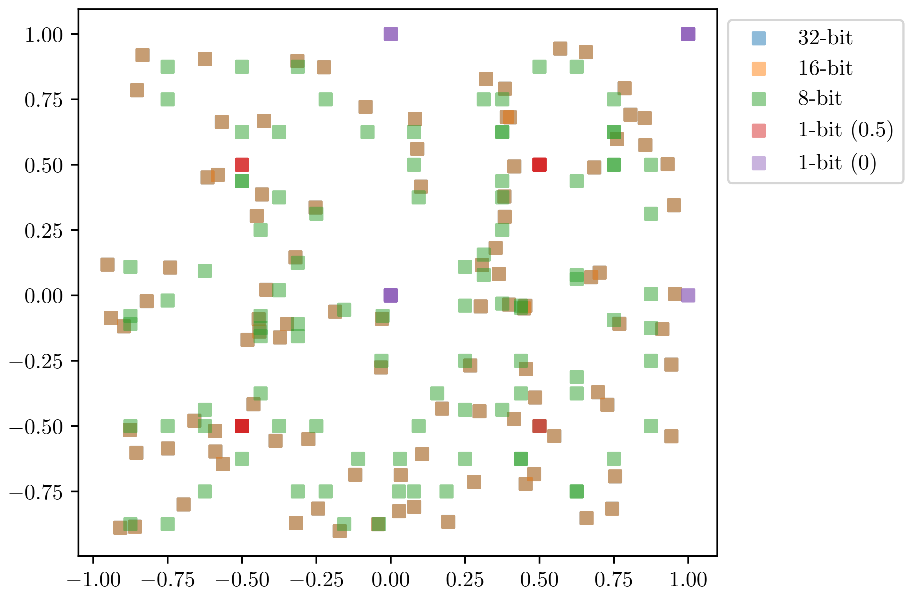

FIGURE 3.3: 100 randomly generated points with 5 precision reductions methods. 32-bit and 16-bit overlap, 1-bit has only 4 options.

32-bit	**16-bit**	**8-bit**	**1-bit (0.5)**	**1-bit (0)**
0.10159580514915101	0.10162353515625	0.09375	0.5	1.0
0.41629564523620965	0.416259765625	0.375	0.5	1.0
-0.41819052217411135	-0.418212890625	-0.375	-0.5	0.0
0.02165521039532603	0.0216522216796875	0.01953125	0.5	1.0
0.7858939086953094	0.7861328125	0.75	0.5	1.0
0.7925861778668761	0.79248046875	0.75	0.5	1.0
-0.7488293790723275	-0.7490234375	-0.625	-0.5	0.0
-0.5855142437236265	-0.58544921875	-0.5	-0.5	0.0

TABLE 3.5: Decimal representation of randomly generated numbers with 5 precision reduction methods.

certain cases when using IP-based similarity.[3] As shown in Table 3.4, this indeed yields a slight improvement. When applying post-processing, however, the two approaches are equivalent. While this method achieves extreme 32x compression on the disk and retains most of the retrieval performance. The downside of 1-bit and 8-bit is that if one wishes to use standard retrieval pipelines, these variables would have to be converted to a supported, larger, data type.

[3]When using 0 and 1, the IP similarity of 0 and 1 is the same as 0 and 0 while for -0.5 and 0.5 they are -0.25 and 0.25 respectively.

Finally, reducing precision can be readily combined with dimension reduction methods (see Section 3.2.5), such as PCA (prior to changing the data type). As shown in the last row of Table 3.4, this can lead to the compressed size be 100x smaller while retaining 75% retrieval performance on HotpotQA and 89% for NaturalQuestions (see Table 3.6).

3.2.5 Combination of PCA and Precision Reduction

It is possible to combine methods for dimension reduction with methods for reducing data type precision. The results in Figure 3.4 show that PCA can be combined with e.g. 8-bit precision reduction with negligible loss in performance.

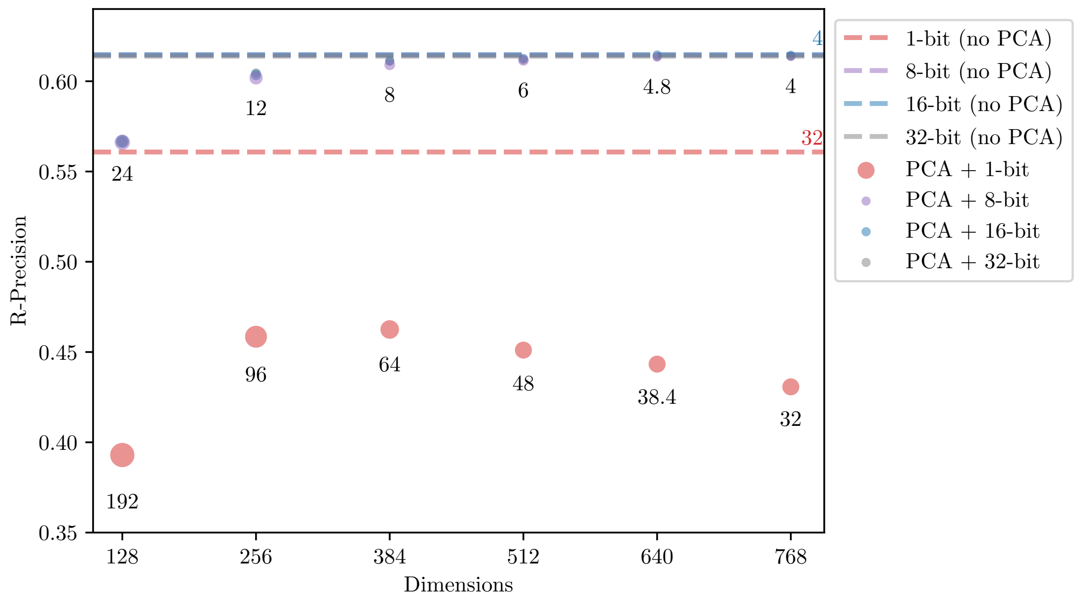

FIGURE 3.4: Combination of PCA and precision reduction. Compression ratio is shown in text. 16-bit and 32-bit values overlap with 8-bit and their compression ratios are not shown. Measured on HotpotQA with DPR-CLS.

Method	**Compression**	**Original**		**Center + Norm.**
		IP	L^2	{IP, L^2} (% original)
Original	1×	0.934	0.758	0.920 (100%)
Gaussian Projection	6×	0.825	0.848	0.855 (93%)
Sparse Projection	6×	0.826	0.848	0.856 (93%)
Dimension Dropping	6×	0.840	0.863	0.867 (94%)
Greedy Dimension Dropping	6×	0.845	0.873	0.873 (95%)
PCA	6×	0.908	0.907	0.910 (99%)
PCA (scaled top 5)	6×	0.916	0.910	0.920 (100%)
Autoencoder (single layer)	6×	0.915	0.910	0.914 (99%)
Autoencoder (full)	6×	0.903	0.907	0.910 (99%)
Autoencoder (shallow decoder)	6×	0.916	0.918	0.919 (100%)
Autoencoder + L_1 (single layer)	6×	0.918	0.918	0.921 (100%)
Autoencoder + L_1 (full)	6×	0.909	0.910	0.913 (99%)
Autoencoder + L_1 (shallow decoder)	6×	0.918	0.917	0.919 (100%)
Precision 16-bit	2×	0.921	0.917	0.920 (100%)
Precision 8-bit	4×	0.920	0.921	0.922 (100%)
Precision 1-bit (offset 0.5)	32×	0.902	0.902	0.904 (98%)
Precision 1-bit (offset 0)	32×	0.892	0.902	0.904 (98%)
PCA (245) + Precision 1-bit (offset 0.5)	100×	0.854	0.862	0.858 (93%)
PCA (128) + Precision 8-bit	24×	0.906	0.904	0.909 (99%)

Table 3.6: Overview of compression method performance (from 768) using either L^2 or inner product for retrieval. Inputs are based on (1) original and (2) centered and normalized output of DPR-CLS. Performance is measured by R-Precision on NaturalQuestions.

3.3 Analysis

3.3.1 Model Comparison

The comparison of all discussed dimension reduction methods is shown in Table 3.4. It also shows the role of centering and normalization post-encoding which systematically improves the performance. The best performing model for dimension reduction is the autoencoder with L_1 regularization and either just a single projection layer for the encoder and decoder or with the shallow decoder (6x compression with 97% retrieval performance).

We also show the major experiments in Table 3.6 (table structure equivalent to that for the pruned dataset in Table 3.4) on Natural Question [41] with identical dataset pre-processing. The performance is overall larger because the task is different and the set of documents is lower (1.5 million spans) but comparatively the trends are in line with the previous conclusions of the thesis.

3.3.2 Speed.

Despite the autoencoder providing slightly better retrieval performance and PCA being generally easier to use (due to the lack of hyperparameters), there are several tradeoffs in model selection. Once the models are trained, the runtime performance (encoding) is comparable though for PCA it is a single matrix projection while for the autoencoder it may be several layers and activation functions.

Depending on the specific library used for implementation, however, the results differ. Figure 3.5 shows that the autoencoder (implemented in PyTorch) is much slower than any other model when run on a CPU but the fastest when run on a GPU. Similarly, PCA works best if used from the PyTorch library (whether on CPU or GPU) and from the standard Scikit package. Except for Scikit, there seems to be little relation between the target dimensionality and computation time.

3.3.3 Data size

A crucial aspect of the PCA and autoencoder methods is how much data they need for training. In the following, we experimented with limiting the number of training samples for PCA and the linear autoencoder. Results are shown in Figure 3.6.

[4]PyTorch 1.9.1, scikit-learn 0.23.2, RTX 2080 Ti (CUDA 11.4), 64×2.1GHz Intel Xeon E5-2683 v4, 1TB RAM.

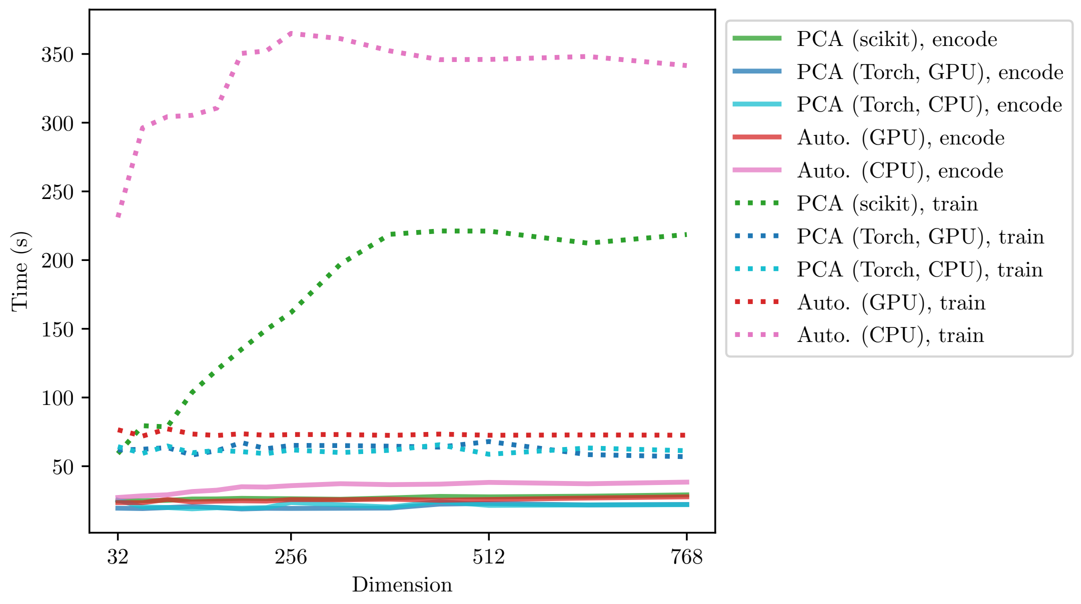

FIGURE 3.5: Speed comparison of PCA and autoencoder (model 3) implemented in PyTorch and Scikit[4]split into training and encoding parts. Models were trained on documents and queries jointly (normalized).

While Ma et al. [27] used a much larger training set to fit PCA, we find that PCA requires very few samples (lower-bounded by 128 which is also the number of dimensions used for this experiment). This is because in the case of PCA training data is used to estimate the data covariance matrix which has been shown to work well when using a few samples [93]. Additionally, we find that overall the autoencoder needs more data to outperform PCA.

Next, we experimented with adding more (potentially irrelevant) documents to the knowledge base. For this, we kept the training data for the autoencoder and PCA to the original size. The results are shown as dashed lines in Figure 3.6. Retrieval performance quickly deteriorates for both models (faster than for the uncompressed case), highlighting the importance of filtering irrelevant documents from the knowledge base.

3.3.4 Retrieval errors

So far, our evaluation focused on quantitative comparisons. In the following, we compare the distribution of documents retrieved before and after compression to investigate if there are systematic differences. We carry out this analysis using HotpotQA which, by design, requires two documents in order to answer a given query. We compare retrieval with the original document embeddings to retrieval with PCA and 1-bit compression.

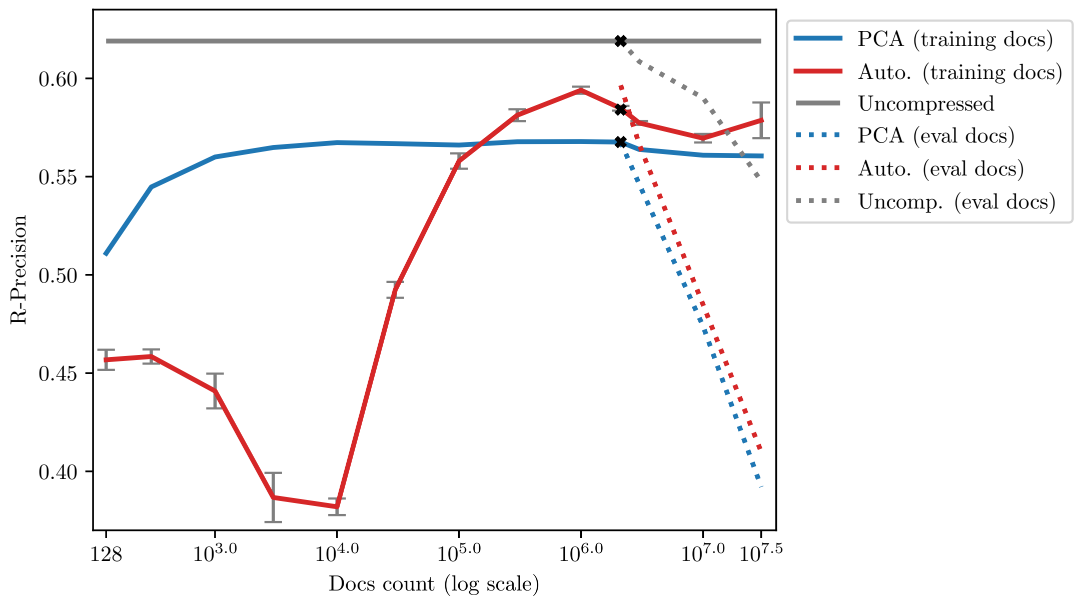

FIGURE 3.6: Dependency of PCA and autoencoder performance (evaluated on HotpotQA dev data, trained on document encodings) by modifying training data (solid lines) and by adding irrelevant documents to the retrieval pool (dashed lines). Black crosses indicate the original training size. Note the log scale on the x-axis and the truncation of the y-axis.

	Uncompressed	PCA	1bit
Uncompressed	1.00		
PCA	0.87	1.00	
1bit	0.81	0.80	1.00

TABLE 3.7: Correlation of the number of retrieved documents for HotpotQA queries in different retrieval modes: uncompressed, PCA (128) and 1-bit precision with R-Precisions (centered & normalized) of 0.618, 0.579 and 0.561, respectively.

We find that there are no systematic differences compared to the uncompressed retrieval. This is demonstrated by the small off-diagonal values in Figure 3.7. This result shows that if the retriever working with uncompressed embeddings returns two relevant documents in the top-k for a given query, also the retriever working with the compressed index is very likely to include the same two documents in the top-k. This is further shown by the Pearson correlation in Table 3.7.

Overall this suggests that the compressed index can be used on downstream tasks with predictable performance loss based on the slightly worsened retrieval performance. Furthermore, there do not seem to be any systematic differences even between the two vastly different compression methods used for this experiment (PCA and 1-bit precision). This indicates that, despite their methodological differences, the two compression approaches seem to remove the same redundances in the uncompressed data. We leave a more detailed exploration of these findings for future work.

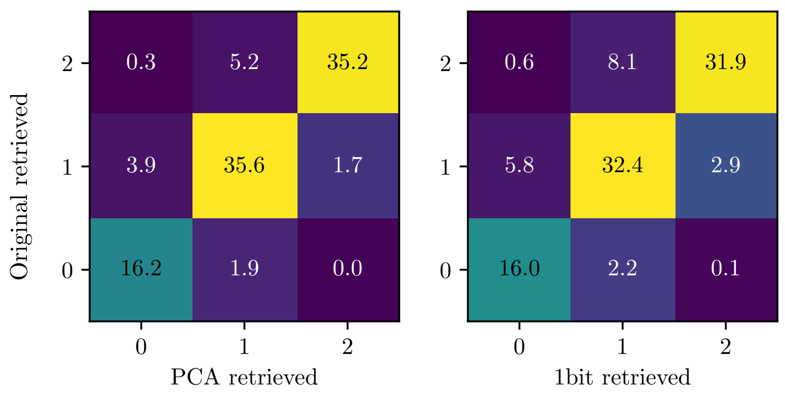

FIGURE 3.7: Distribution of the number of retrieved documents for HotpotQA queries before and after compression: PCA (128) and 1-bit precision with R-Precisions (centered & normalized).

Chapter 4

Discussion

In this section, we briefly discuss the main conclusions from our experiments and analysis in the form of recommendations for NLP practitioners. We also discuss the general approach to dimensionality reduction from the point of view of distance learning and the limitations. We summarize together with ideas for future work.

4.1 Recommendations

4.1.1 Importance of Pre-/post-processing

As our results show, for all methods (and models), centering and normalization should be done before and after dimension reduction, as it boosts the performance of every model.

4.1.2 Method recommendation

While most compression methods achieve similar retrieval performance and compression ratios (cf. Table 3.4 and Table 3.6), PCA stands out in the following regards:

- It requires only minimal implementation effort and no tuning of hyper-parameters beyond selecting how many principal components to keep.
- As our analysis shows, the PCA matrix can be estimated well with only 1000 document or query embeddings. It is not necessary to learn a transformation matrix on the full knowledge base.
- PCA can easily be combined with precision reduction based approaches.

4.1.3 Splitting

Small gains (both performance and smaller knowledge base size) can be achieved by splitting on sentence boundaries (e.g. 6 sentences instead of 100 tokens). Further improvements can be possibly made by using the same splitting scheme in model pretraining.

4.2 Pitfalls of Reconstruction Loss

Despite PCA and autoencoder being the most successful methods, low reconstruction loss provides no theoretical guarantee to the retrieval performance. Consider a simple linear projection that can be represented as a diagonal matrix that projects to a space of the same dimensionality. This function has a trivial inverse and therefore no information is lost when it is applied. The retrieval is however disrupted, as it will mostly depend on the first dimension and nothing else. This is a major flaw of approaches that minimize the vector reconstruction loss because the optimized quantity is different to the actual goal.

$$R = \begin{pmatrix} 10^{99} & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 \end{pmatrix} \qquad R^{-1} = \begin{pmatrix} 10^{-99} & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 \end{pmatrix}$$

$$\forall v \in \mathbf{R}^d : L^2((R^{-1} \times R)v, v) = L^2(\mathrm{I}v, v) = 0 \quad \Rightarrow \quad \mathcal{L} = 0$$

4.2.1 Distance Learning

In this subsection, we discuss a more general solution to dimension reduction for document retrieval: distance/manifold learning. This task is also concerned with creating a new vector space that has some key properties transferred from the original, higher-dimensional, space (distances, ordering etc.). Figure 4.1 illustrates dimension reduction on random vectors from 3D to 2D.[1] The various methods are discussed later. Note the distances between the three selected points in the 2D space.

[1]The 3D visualization is in itself a kind of dimension reduction but we add additional features (leading verticals and grid) to make use of the human visual processing.

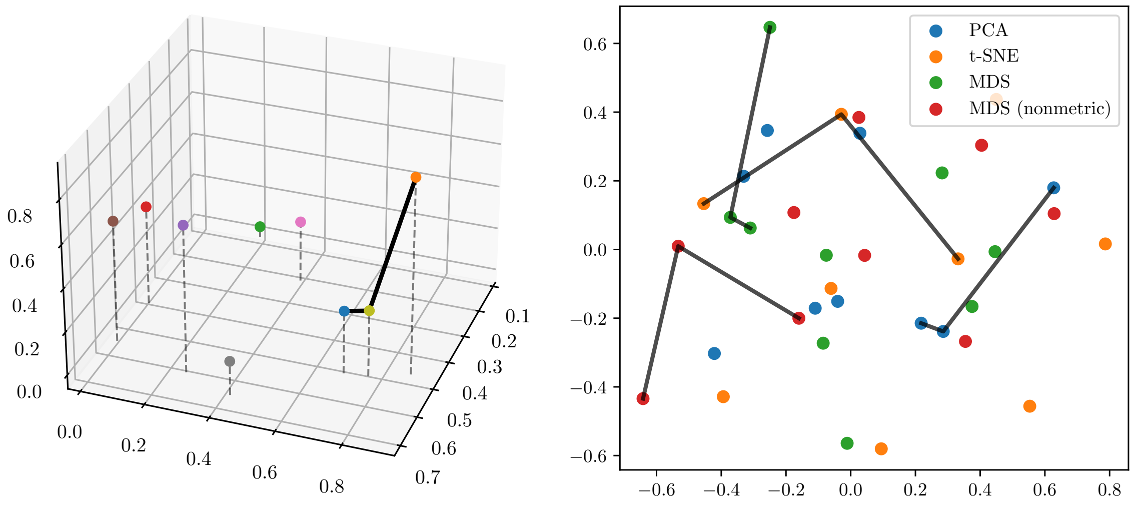

FIGURE 4.1: Example of dimension reduction using PCA, t-SNE and parametric and non-parametric MDS from 3 to 2 dimensions.

4.2.1.1 Existing Approaches

The task of dimensionality reduction has been explored by standard statistical methods by the name manifold learning.

The most used method is t-distributed stochastic neighbor (t-SNE) embedding built on the work of Hinton and Roweis [94] or multidimensional scaling [95, 96]. They organize a new vector space (of lower dimensionality) so that the L^2 distances follow those of the original space (extensions to other metrics also exist). Although the optimization goal is more in line with our task of vector space compression with the preservation of nearest neighbours, methods of manifold learning are limited by the large computation costs[2] and the fact that they do not construct a function but rather move the discrete points in the new space to lower the optimization loss. This makes it not applicable for online purposes (i.e. adding new samples that need to be compressed as well).

Multidimensional Scaling. A widely used technique for dimensionality reduction is MDS. An extension to this is using an arbitrary metric for the original space, though the new organization is still based on the L^2 distance. An important variant of this method is nonmetric MDS which preserves the ordering of neighbours instead of trying to model all distances accurately. This optimization goal is important for the task at hand, dimensionality reduction for vectors used for document retrieval. A downside of

[2]The common fast implementation for t-SNE, Barnes-Hut [97, 98] is based on either quadtrees or octrees and is limited to 3 dimensions.

MDS is that the support is limited to metrics only in the formal mathematical sense, which the inner product does not satisfy and has very large computation costs.

Uniform Manifold Approximation and Projection for Dimension Reduction. While the recent technique UMAP [99] solves some of the issues of t-SNE (target dimension limits), it is still too slow for practical purposes when applied to the whole knowledge base.

All of the manifold learning methods are also very dependent on the hyperparameters, which makes working with the techniques (in comparison to PCA) more difficult, also yielding suboptimal results.

4.2.1.2 Gradient-based optimization

The main disadvantage of the approaches based on reconstruction loss is that their optimization goal strays from what we are interested in, namely preserving distances between vectors. We tried to reformulate the problem in terms of deep learning and gradient-based optimization to alleviate the issue of speed and extensibility of standard manifold learning approaches. We try to learn a function that maps the original vector space to a lower-dimensional one while preserving similarities. That can be either a simple linear projection A or generally a more complex differentiable function f:

$$\mathcal{L} = \mathrm{MSE}(\mathrm{sim}(f(t_i), f(t_j)), \mathrm{sim}(t_i, t_j))$$

After the function f is fitted, both the training and new data can be compressed by its application. As opposed to manifold learning which usually leverages specific properties of the metrics, here they can be any differentiable functions. The optimization was, however, too slow, underperforming (between sparse projection and PCA) and did not currently provide any benefits (results not shown in this thesis).

Similar to manifold learning, we may create completely new vectors instead of relying on differentiable transformations. Given an already existing lower-dimensional representation of the data, we may consider the following loss, computed the gradient and updated the values. Assuming t_i and t_j are randomly sampled vectors and the new vectors n_i and n_j are trainable parameters. The elements in the pairs of vectors need not be from the same distribution. In this case, t_i and n_i may be queries and t_j and n_j documents. The optimization goal is to have same similarities in the new vector space as in the original one.

$$\mathcal{L} = \text{MSE}(\text{sim}(n_i, n_j), \text{sim}(t_i, t_j))$$

Naturally, this is very susceptible to the initial organization of the vector space. We considered two solutions for initialization of every dimension: uniform sampling from $(-1, 1)$ and sampling from a normal distribution with $\mu = 1$ and $\sigma^2 = 1$. We also experimented with the normalization of the whole vector to the norm of 1. This method does not, however, solve the issue that the result of the computation is not applicable to new data and only reorganizes data at train-time.

We also tried to use unsupervised contrastive learning by considering close neighbours in the original space as positive samples and distant neighbours as negative samples but reached similar results:

$$\mathcal{L} = \frac{\sum_{t_i, t_j \text{ close}} e^{\text{sim}(f(t_i), f(t_j))}}{\sum_{t_i, t_j \text{ distant}} e^{\text{sim}(f(t_i), f(t_j))}}$$

4.3 Limitation

We are fairly confident, based on prior existing research on this topic, that the explored methods will not yield unexpected undesirable results. We aimed at making deployment and experiments with large knowledge bases easier from compute and memory resrouces point of view. A persisting bottleneck is the embedding model inference, which is required to generate the index for the whole knowledge base. Additionally, for extremely resource limited scenarios, the application of these methods may not be straightforward.[3]

The key thing that could prevent from the described methods to be used by document retrieval practicioners is the lack of quantification of how the loss in retrieval performance affects the downstream task performance. While we shown that there are no probably no systematic differences in the kinds of documents that are retrieved, it is still unknown whether 90% of original retrieval performance means that a specific question answering system will have 90% exact match. Because this depends heavily on the specific task, we leave this question open for future survey and would like to see curves downstream task performance - retrieval retrieval performance for multiple systems and tasks.

[3]For example, to avoid having the whole 150GB index in memory, one must interweave embedding model inference with dimension reduction steps.

4.4 Summary

In this work, we examined several simple unsupervised methods for dimensionality reduction for retrieval-based NLP tasks: random projections, PCA, autoencoder and precision reduction and their combination. We also documented the data requirements of each method and their reliance on pre- and post-processing. We explored several options for splitting and filtering and reported that the commonly used practices are adequate. We stress the importance of pre- and post-processing (centering + normalization).

Future research

As shown in prior works, dimension reduction can take place also during training where the loss is more in-line with the retrieval goal. Methods for dimension reduction after training, however, rely mostly on reconstruction loss, which is suboptimal. Therefore more research for dimension reduction methods is needed, such as fast manifold or distance-based learning.

Despite negative results with gradient-based metric learning, we believe that with enough research, it may show to be useful. It also leads to many new questions, for example, if the loss uses the inner product as the similarity in the new space and L^2 in the original, it is unclear to which extent the trained function would be able to capture the original L^2 structure with the inner product as the similarity metric.

Appendix A

Fusion Discussion

Intuitively it makes more sense to prefer early fusion, to maximize the model's access to extra information [36, 100]. However, this can also be a disadvantage, as the signal from the artefact can get lost during the long computation. In the case of an artefact which is the gold output of a similar query from the training data, later fusion makes more sense. This also allows for a degree of explainability. By examining the forward pass of the last function we could determine what the contribution of the artefact was to the produced output.

The decision of how late the fusion should be depends heavily on the artefact type. The application of every function in the chain of computation projects the input to some latent space. The final function f_n is special because it projects the output of previous functions to the space of possible outputs for the whole model. In this space, there is the prediction $\hat{y}$ and also the true output y. The task performance metric is defined in this space. During inference, adding an artefact should ideally move the prediction in the output space closer to the correct output. Assuming c is the intermediate computation and there are two (overloaded) functions that produce a prediction: $\hat{y}_x = f_n(c)$ and $\hat{y}_\xi = f_n(c, \xi)$. In circumstances in which adding the artefact helps, $L(\hat{y}_\xi, y) < L(\hat{y}_x, y)$, where L is a loss function such as cross-entropy. This is illustrated in the first row of Figure A.1 for $n = 2$.

Assume that we can create an inverse of the last projection and see where the correct output lies in the intermediate representation. There may be multiple such $c_t : f_n(c_t) = y$ or none, if too much information was lost by the first projection f_1. Further, assume that there is always at least one such c_t. We may then define an intermediate loss L_i for each model computation by measuring the distance of the partial computations to the back-projection. Similarly to late fusion, we consider two overloaded functions that produce the intermediate representation $c_x = f_1(q)$ and $c_\xi = f_1(q, \xi)$. Adding the artefact then ideally moves the intermediate representation closer to the back-projection and reduces the intermediate loss: $L^i(c_\xi, c_t) < L^i(c_x, c_t)$.[1]

This is illustrated in the second row of Figure A.1, which depicts a model with only two computational steps: $f_2 \circ f_1$. Early fusion (second row) adds the artefact to f_1, while late fusion adds it in the next step. For simplicity in the figure, we consider the standard L^2 distance loss between the points. In both cases, adding the artefact reduced the target loss. For early function, the intermediate loss was also reduced and the target loss was lower. This does not always happen and complex computations may still at some point project the intermediate computation to the same point regardless of whether an artefact was added earlier or not. It may also be the case that training with artefacts takes a longer time and the intermediate loss is higher but that the presence of the artefacts will make the model converge to a better optimum (lower generalization error).

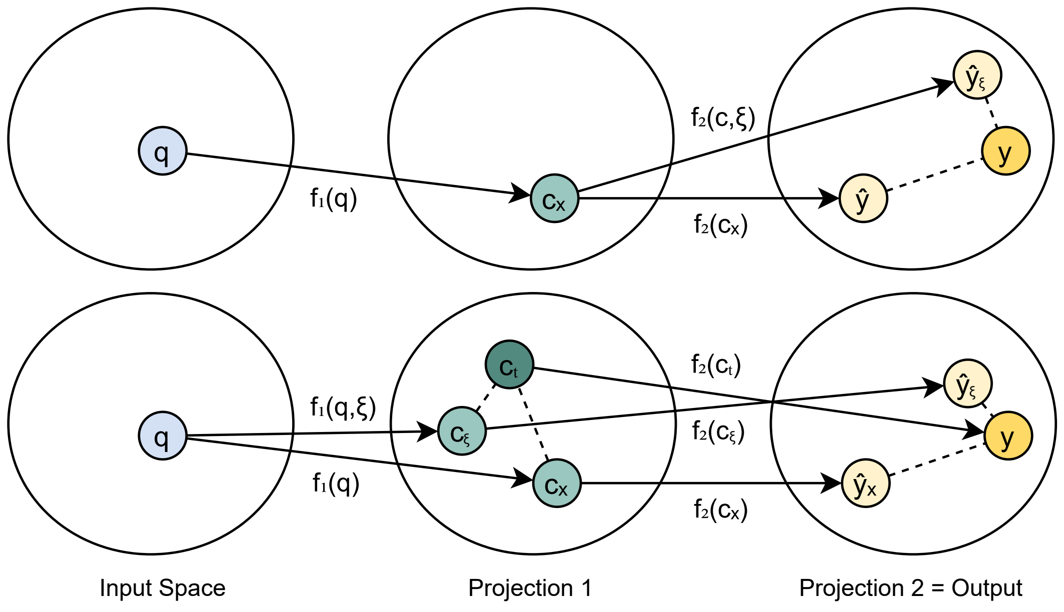

FIGURE A.1: Example of how late (top) and early (bottom) fusions affect the projections into intermediate spaces. The lengths of dashed lines correspond to the loss (longer is greater loss). Adding an artefact ξ to the computation decreases the loss in both early and late fusions.

Even though the invertibility of projections in this paper is only used as an illustration for the artefact fusion and an intermediate loss does not have to be defined in practice, invertible neural networks are an ongoing topic of research [101, 102]. The combination

[1]In case of multiple elements that map to y, we can define a loss that considers the minimum distance to any of them: $L^{i'} = \min L^i(c, c_t)$. If there is no such element in the projection space, then we may consider the elements that project close to the target $C_t = \arg\min L(f_2(c), y)$.

of artefact retrieval and invertible neural networks has not yet been explored to our knowledge.

List of Figures

Figure 1.1 General scheme of NLP models utilizing artefacts by retrieving them from a knowledge base and fusing them into the model in order to produce a better output. Dashed links are utilized only in knowledge base creation and usually not all at once. 5

Figure 1.2 Conceptualized overview of retrieval on whole documents. Not actually used in practice with dense vector representations. 9

Figure 1.3 Conceptualized overview of retrieval on document spans. 9

Figure 1.4 Comparison of different BERT-based embedding models and versions when using faster but slightly inaccurate nearest neighbour search. [**CLS**] is the specific token embedding from the last layer while **(Avg)** is all token average. 13

Figure 1.5 Effect of data centering and normalization on performance (evaluated with FAISS). 14

Figure 1.6 First two dimensions of 15 random queries and 15 random documents and their positions after preprocessing. 15

Figure 2.1 Retrieval performance and knowledge base size after being split using different methods. Note the cut-off y-axis. 19

Figure 2.2 Retrieval performance and knowledge base size after being filtered using length-based heuristics. Note the cut-off y-axis. 21

Figure 2.3 Distribution of span length either with word or characte count. . . 21

Figure 2.4 Example of two retrieval and one pruning steps on two queries. For both queries, the spans are sorted from left to right by decreasing similarity. Spans with bold borders are those considered as retrieved. Spans in green are relevant for the given query and spans in red are not relevant. 22

Figure 2.5 Autofiltering performance of dev queries on spans filtered using the same dev queries. Filtering is done using either top-10 or top-20 spans. 23

Figure 2.6 Autofiltering performance of dev queries on spans filtered using train queries. Filtering is done using top-10 spans. 24

Figure 2.7 Training classifier accuracy for most common class classifier and logistic regression based on data from individual filtering steps or cumulative data up to a certain step (total). 24

Figure 3.1 Dimension reduction using different random projections methods. Presented values are the max of 3 runs (except for greedy dimension dropping, which is deterministic), semi-transparent lines correspond to the minimum. Embeddings are provided by centered and normalized DPR-CLS. Final vectors are also post-processed by centering and normalization. 29

Figure 3.2 Dimension reduction using PCA (top) and Autoencoder (bottom) trained either on document index, query embeddings or both. Each figure corresponds to one of the four possible combinations of centering and normalizing the input data. The output vectors are not post-processed. Reconstruction loss (MSE, average for both documents and queries) is shown in transparent colour and computed in original data space. Horizontal lines show uncompressed performance. Embeddings are provided by DPR-CLS. 30

Figure 3.3 100 randomly generated points with 5 precision reductions methods. 32-bit and 16-bit overlap, 1-bit has only 4 options. 34

Figure 3.4 Combination of PCA and precision reduction. Compression ratio is shown in text. 16-bit and 32-bit values overlap with 8-bit and their compression ratios are not shown. Measured on HotpotQA with DPR-CLS. 35

Figure 3.5 Speed comparison of PCA and autoencoder (model 3) implemented in PyTorch and Scikit[2]split into training and encoding parts. Models were trained on documents and queries jointly (normalized). 38

Figure 3.6 Dependency of PCA and autoencoder performance (evaluated on HotpotQA dev data, trained on document encodings) by modifying training data (solid lines) and by adding irrelevant documents to the retrieval pool (dashed lines). Black crosses indicate the original training size. Note the log scale on the x-axis and the truncation of the y-axis. 39

Figure 3.7 Distribution of the number of retrieved documents for HotpotQA queries before and after compression: PCA (128) and 1-bit precision with R-Precisions (centered & normalized). 40

Figure 4.1 Example of dimension reduction using PCA, t-SNE and parametric and non-parametric MDS from 3 to 2 dimensions. 43

Figure A.1 Example of how late (top) and early (bottom) fusions affect the projections into intermediate spaces. The lengths of dashed lines correspond to the loss (longer is greater loss). Adding an artefact ξ to the computation decreases the loss in both early and late fusions. 48

List of Tables

Table 1.1 Categorization of described NLP systems in terms of the artefact retrieval typology. *Fusion* describe both where it occurs and what mechanism it employs, *Keys* describes not only the key type but also the retrieval mechanism (e.g. metric). 7

Table 1.2 Example of questions from HotpotQA and Natural Questions. Similarity is measured on centered and normalized embeddings from DPR-CLS. Higher always means more similar (L^2 is for this reason negated). . 11

Table 1.3 Number of training and dev queries and documents for the different datasets used. *Whole Wikipedia dump. 12

Table 3.1 Effect of pre-processing transformations on embeddings produced by DPR-CLS. Means and standard deviations are computed separately for documents and queries. Transformation into z-scores includes centering. 27

Table 3.2 Average L^1 and L^2 norms of document and query embeddings from DPR-CLS without pre-processing. 30

Table 3.3 Hyperparameters of autoencoder architectures. L_1 regularization is used only when explicitly mentioned. 32

Table 3.4 Overview of compression method performance (from 768) using either L^2 or inner product for retrieval. Inputs are based on centered and normalized output of DPR-CLS and the outputs optionally post-processed again. Performance is measured by R-Precision on the pruned HotpotQA dataset. 33

Table 3.5 Decimal representation of randomly generated numbers with 5 precision reduction methods. 34

Table 3.6 Overview of compression method performance (from 768) using either L^2 or inner product for retrieval. Inputs are based on (1) original and (2) centered and normalized output of DPR-CLS. Performance is measured by R-Precision on NaturalQuestions. 36

Table 3.7 Correlation of the number of retrieved documents for HotpotQA queries in different retrieval modes: uncompressed, PCA (128) and 1-bit precision with R-Precisions (centered & normalized) of 0.618, 0.579 and 0.561, respectively. 39

Bibliography

[1] Kelvin Guu, Kenton Lee, Zora Tung, Panupong Pasupat, and Ming-Wei Chang. Realm: Retrieval-augmented language model pre-training. *arXiv preprint arXiv:2002.08909*, 2020.

[2] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, et al. Retrieval-augmented generation for knowledge-intensive nlp tasks. *arXiv preprint arXiv:2005.11401*, 2020.

[3] Fabio Petroni, Aleksandra Piktus, Angela Fan, Patrick Lewis, Majid Yazdani, Nicola De Cao, James Thorne, Yacine Jernite, Vladimir Karpukhin, Jean Maillard, et al. Kilt: a benchmark for knowledge intensive language tasks. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 2523–2544, 2021.

[4] Moin Nadeem, Wei Fang, Brian Xu, Mitra Mohtarami, and James Glass. Fakta: An automatic end-to-end fact checking system. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics (Demonstrations)*, pages 78–83, 2019.

[5] Charles L Chen. *Neural Network Models for Tasks in Open-Domain and Closed-Domain Question Answering.* Ohio University, 2020.

[6] Andon Tchechmedjiev, Pavlos Fafalios, Katarina Boland, Malo Gasquet, Matthäus Zloch, Benjamin Zapilko, Stefan Dietze, and Konstantin Todorov. Claimskg: A knowledge graph of fact-checked claims. In *International Semantic Web Conference*, pages 309–324. Springer, 2019.

[7] Robert Logan, Nelson F Liu, Matthew E Peters, Matt Gardner, and Sameer Singh. Barack's wife hillary: Using knowledge graphs for fact-aware language modeling. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 5962–5971, 2019.

[8] Sebastian Borgeaud, Arthur Mensch, Jordan Hoffmann, Trevor Cai, Eliza Rutherford, Katie Millican, George van den Driessche, Jean-Baptiste Lespiau, Bogdan Damoc, Aidan Clark, Diego de Las Casas, Aurelia Guy, Jacob Menick, Roman

Ring, Tom Hennigan, Saffron Huang, Loren Maggiore, Chris Jones, Albin Cassirer, Andy Brock, Michela Paganini, Geoffrey Irving, Oriol Vinyals, Simon Osindero, Karen Simonyan, Jack W. Rae, Erich Elsen, and Laurent Sifre. Improving language models by retrieving from trillions of tokens, 2021.

[9] Junxian He, Graham Neubig, and Taylor Berg-Kirkpatrick. Efficient nearest neighbor language models. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 5703–5714, 2021.

[10] Emily Dinan, Stephen Roller, Kurt Shuster, Angela Fan, Michael Auli, and Jason Weston. Wizard of wikipedia: Knowledge-powered conversational agents. *arXiv preprint arXiv:1811.01241*, 2018.

[11] Sixing Wu, Ying Li, Dawei Zhang, Yang Zhou, and Zhonghai Wu. Topicka: Generating commonsense knowledge-aware dialogue responses towards the recommended topic fact. In *IJCAI*, volume 2020, pages 3766–3772, 2020.

[12] Urvashi Khandelwal, Omer Levy, Dan Jurafsky, Luke Zettlemoyer, and Mike Lewis. Generalization through memorization: Nearest neighbor language models. *arXiv preprint arXiv:1911.00172*, 2019.

[13] Dani Yogatama, Cyprien de Masson d'Autume, and Lingpeng Kong. Adaptive semiparametric language models. *Transactions of the Association for Computational Linguistics*, 9:362–373, 2021.

[14] Vilém Zouhar, Marius Mosbach, Debanjali Biswas, and Dietrich Klakow. Artefact retrieval: Overview of NLP models with knowledge base access. In *Workshop on Commonsense Reasoning and Knowledge Bases*, 2021. URL `https://openreview.net/forum?id=9_oCNR6R9l2`.

[15] Stephen E Robertson, Steve Walker, Susan Jones, Micheline M Hancock-Beaulieu, Mike Gatford, et al. Okapi at trec-3. *Nist Special Publication Sp*, 109:109, 1995.

[16] Vilém Zouhar, Marius Mosbach, Miaoran Zhang, and Dietrich Klakow. Knowledge base index compression via dimensionality and precision reduction. *arXiv preprint arXiv:2204.02906*, 2022.

[17] Bhaskar Mitra and Nick Craswell. Neural models for information retrieval. *arXiv preprint arXiv:1705.01509*, 2017.

[18] Rodrigo Nogueira, Wei Yang, Kyunghyun Cho, and Jimmy Lin. Multi-stage document ranking with bert. *arXiv preprint arXiv:1910.14424*, 2019.

[19] Amir Soleimani, Christof Monz, and Marcel Worring. Bert for evidence retrieval and claim verification. *Advances in Information Retrieval*, 12036:359, 2020.

[20] Jean Maillard, Vladimir Karpukhin, Fabio Petroni, Wen-tau Yih, Barlas Oguz, Veselin Stoyanov, and Gargi Ghosh. Multi-task retrieval for knowledge-intensive tasks. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 1098–1111, Online, August 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.acl-long.89. URL `https://aclanthology.org/2021.acl-long.89`.

[21] Xiaodong Liu, Pengcheng He, Weizhu Chen, and Jianfeng Gao. Multi-task deep neural networks for natural language understanding. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4487–4496, 2019.

[22] Chi Sun, Xipeng Qiu, Yige Xu, and Xuanjing Huang. How to fine-tune bert for text classification? In *China National Conference on Chinese Computational Linguistics*, pages 194–206. Springer, 2019.

[23] Hyun Kim, Joon-Ho Lim, Hyun-Ki Kim, and Seung-Hoon Na. Qe bert: bilingual bert using multi-task learning for neural quality estimation. In *Proceedings of the Fourth Conference on Machine Translation (Volume 3: Shared Task Papers, Day 2)*, pages 85–89, 2019.

[24] Armen Aghajanyan, Anchit Gupta, Akshat Shrivastava, Xilun Chen, Luke Zettlemoyer, and Sonal Gupta. Muppet: Massive multi-task representations with prefinetuning. *arXiv preprint arXiv:2101.11038*, 2021.

[25] Yi Luan, Jacob Eisenstein, Kristina Toutanova, and Michael Collins. Sparse, dense, and attentional representations for text retrieval. *Transactions of the Association for Computational Linguistics*, 9:329–345, 2021.

[26] Gautier Izacard, Fabio Petroni, Lucas Hosseini, Nicola De Cao, Sebastian Riedel, and Edouard Grave. A memory efficient baseline for open domain question answering. *arXiv preprint arXiv:2012.15156*, 2020.

[27] Xueguang Ma, Minghan Li, Kai Sun, Ji Xin, and Jimmy Lin. Simple and effective unsupervised redundancy elimination to compress dense vectors for passage retrieval. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 2854–2859, 2021.

[28] Ikuya Yamada, Akari Asai, and Hannaneh Hajishirzi. Efficient passage retrieval with hashing for open-domain question answering. *arXiv preprint arXiv:2106.00882*, 2021.

[29] Mahboob Alam Khalid, Valentin Jijkoun, and Maarten de Rijke. The impact of named entity normalization on information retrieval for question answering. In *European Conference on Information Retrieval*, pages 705–710. Springer, 2008.

[30] Seung-Hoon Na. Two-stage document length normalization for information retrieval. *ACM Transactions on Information Systems (TOIS)*, 33(2):1–40, 2015.

[31] Dwaipayan Roy, Debasis Ganguly, Sumit Bhatia, Srikanta Bedathur, and Mandar Mitra. Using word embeddings for information retrieval: How collection and term normalization choices affect performance. In *Proceedings of the 27th ACM international conference on information and knowledge management*, pages 1835–1838, 2018.

[32] William Timkey and Marten van Schijndel. All bark and no bite: Rogue dimensions in transformer language models obscure representational quality. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 4527–4546, 2021.

[33] Angeliki Lazaridou, Adhiguna Kuncoro, Elena Gribovskaya, Devang Agrawal, Adam Liska, Tayfun Terzi, Mai Gimenez, Cyprien de Masson d'Autume, Sebastian Ruder, Dani Yogatama, et al. Pitfalls of static language modelling. *arXiv preprint arXiv:2102.01951*, 2021.

[34] Haitian Sun, Bhuwan Dhingra, Manzil Zaheer, Kathryn Mazaitis, Ruslan Salakhutdinov, and William W Cohen. Open domain question answering using early fusion of knowledge bases and text. 2018.

[35] Edouard Grave, Armand Joulin, and Nicolas Usunier. Improving neural language models with a continuous cache. *arXiv preprint arXiv:1612.04426*, 2016.

[36] Vladimir Karpukhin, Barlas Oguz, Sewon Min, Patrick Lewis, Ledell Wu, Sergey Edunov, Danqi Chen, and Wen-tau Yih. Dense passage retrieval for open-domain question answering. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 6769–6781, 2020.

[37] Patrick Lewis, Pontus Stenetorp, and Sebastian Riedel. Question and answer test-train overlap in open-domain question answering datasets. In *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume*, pages 1000–1008, 2021.

[38] Rajarshi Das, Manzil Zaheer, Dung Thai, Ameya Godbole, Ethan Perez, Jay-Yoon Lee, Lizhen Tan, Lazaros Polymenakos, and Andrew McCallum. Case-based reasoning for natural language queries over knowledge bases. *arXiv preprint arXiv:2104.08762*, 2021.

[39] Haitian Sun, Tania Bedrax-Weiss, and William Cohen. Pullnet: Open domain question answering with iterative retrieval on knowledge bases and text. 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 2380–2390, 2019.

[40] Rajarshi Das, Manzil Zaheer, Siva Reddy, and Andrew McCallum. Question answering on knowledge bases and text using universal schema and memory networks. *arXiv preprint arXiv:1704.08384*, 2017.

[41] Tom Kwiatkowski, Jennimaria Palomaki, Olivia Redfield, Michael Collins, Ankur Parikh, Chris Alberti, Danielle Epstein, Illia Polosukhin, Jacob Devlin, Kenton Lee, et al. Natural questions: a benchmark for question answering research. *Transactions of the Association for Computational Linguistics*, 7:453–466, 2019.

[42] Nils Reimers and Iryna Gurevych. Sentence-bert: Sentence embeddings using siamese bert-networks. 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 3973–3983, 2019.

[43] Jimmy Lin. A proposed conceptual framework for a representational approach to information retrieval. *arXiv preprint arXiv:2110.01529*, 2021.

[44] Iz Beltagy, Matthew E Peters, and Arman Cohan. Longformer: The long-document transformer. *arXiv preprint arXiv:2004.05150*, 2020.

[45] Srikar Appalaraju, Bhavan Jasani, Bhargava Urala Kota, Yusheng Xie, and R Manmatha. Docformer: End-to-end transformer for document understanding. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 993–1003, 2021.

[46] Yuanhua Lv and ChengXiang Zhai. When documents are very long, bm25 fails! In *Proceedings of the 34th international ACM SIGIR conference on Research and development in Information Retrieval*, pages 1103–1104, 2011.

[47] S Sathya Bama, MI Ahmed, and A Saravanan. A survey on performance evaluation measures for information retrieval system. *International Research Journal of Engineering and Technology*, 2(2):1015–1020, 2015.

[48] Tetsuya Sakai. On the reliability of information retrieval metrics based on graded relevance. *Information processing & management*, 43(2):531–548, 2007.

[49] Jeff Johnson, Matthijs Douze, and Hervé Jégou. Billion-scale similarity search with gpus. *IEEE Transactions on Big Data*, 2019.

[50] Godfrey N Lance and William T Williams. Computer programs for hierarchical polythetic classification ("similarity analyses"). *The Computer Journal*, 9(1):60–64, 1966.

[51] Narina Thakur, Deepti Mehrotra, Abhay Bansal, and Manju Bala. Analysis and implementation of the bray–curtis distance-based similarity measure for retrieving information from the medical repository. In *International Conference on Innovative Computing and Communications*, pages 117–125. Springer, 2019.

[52] Jianhua Lin. Divergence measures based on the shannon entropy. *IEEE Transactions on Information theory*, 37(1):145–151, 1991.

[53] Prasanta Chandra Mahalanobis. On the generalized distance in statistics. National Institute of Science of India, 1936.

[54] MK Vijaymeena and K Kavitha. A survey on similarity measures in text mining. *Machine Learning and Applications: An International Journal*, 3(2):19–28, 2016.

[55] Rajesh Joshi and Satish Kumar. A dissimilarity measure based on jensen shannon divergence measure. *International Journal of General Systems*, 48(3):280–301, 2019.

[56] Pinky Sitikhu, Kritish Pahi, Pujan Thapa, and Subarna Shakya. A comparison of semantic similarity methods for maximum human interpretability. In *2019 artificial intelligence for transforming business and society (AITB)*, volume 1, pages 1–4. IEEE, 2019.

[57] Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William Cohen, Ruslan Salakhutdinov, and Christopher D Manning. Hotpotqa: A dataset for diverse, explainable multi-hop question answering. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 2369–2380, 2018.

[58] Rishav Chakravarti, Anthony Ferritto, Bhavani Iyer, Lin Pan, Radu Florian, Salim Roukos, and Avirup Sil. Towards building a robust industry-scale question answering system. In *Proceedings of the 28th International Conference on Computational Linguistics: Industry Track*, pages 90–101, 2020.

[59] Man Luo, Shuguang Chen, and Chitta Baral. A simple approach to jointly rank passages and select relevant sentences in the obqa context. *arXiv preprint arXiv:2109.10497*, 2021.

[60] Ronghan Li, Lifang Wang, Shengli Wang, and Zejun Jiang. Asynchronous multi-grained graph network for interpretable multi-hop reading comprehension. 2021.

[61] Jiayu Ding, Siyuan Wang, Qin Chen, and Zhongyu Wei. Reasoning chain based adversarial attack for multi-hop question answering. *arXiv preprint arXiv:2112.09658*, 2021.

[62] Nicolas Gontier, Siva Reddy, and Christopher Pal. Does entity abstraction help generative transformers reason? *arXiv preprint arXiv:2201.01787*, 2022.

[63] Luiz Bonifacio, Hugo Abonizio, Marzieh Fadaee, and Rodrigo Nogueira. Inpars: Data augmentation for information retrieval using large language models. *arXiv preprint arXiv:2202.05144*, 2022.

[64] 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, 2019.

[65] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding, 2019.

[66] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017.

[67] Chen Qu, Liu Yang, Minghui Qiu, W Bruce Croft, Yongfeng Zhang, and Mohit Iyyer. Bert with history answer embedding for conversational question answering. In *Proceedings of the 42nd international ACM SIGIR conference on research and development in information retrieval*, pages 1133–1136, 2019.

[68] Zekun Yang, Noa Garcia, Chenhui Chu, Mayu Otani, Yuta Nakashima, and Haruo Takemura. Bert representations for video question answering. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pages 1556–1565, 2020.

[69] Rohit Kumar Kaliyar, Anurag Goswami, and Pratik Narang. Fakebert: Fake news detection in social media with a bert-based deep learning approach. *Multimedia tools and applications*, 80(8):11765–11788, 2021.

[70] Yang Liu. Fine-tune bert for extractive summarization. *arXiv preprint arXiv:1903.10318*, 2019.

[71] Yang Liu and Mirella Lapata. Text summarization with pretrained encoders. *arXiv preprint arXiv:1908.08345*, 2019.

[72] Kai Hakala and Sampo Pyysalo. Biomedical named entity recognition with multilingual bert. In *Proceedings of The 5th Workshop on BioNLP Open Shared Tasks*, pages 56–61, 2019.

[73] Subendhu Rongali, Luca Soldaini, Emilio Monti, and Wael Hamza. Don't parse, generate! a sequence to sequence architecture for task-oriented semantic parsing. In *Proceedings of The Web Conference 2020*, pages 2962–2968, 2020.

[74] Han He and Jinho Choi. Establishing strong baselines for the new decade: Sequence tagging, syntactic and semantic parsing with bert. In *The Thirty-Third International Flairs Conference*, 2020.

[75] Zhengjie Gao, Ao Feng, Xinyu Song, and Xi Wu. Target-dependent sentiment classification with bert. *Ieee Access*, 7:154290–154299, 2019.

[76] Manish Munikar, Sushil Shakya, and Aakash Shrestha. Fine-grained sentiment classification using bert. In *2019 Artificial Intelligence for Transforming Business and Society (AITB)*, volume 1, pages 1–5. IEEE, 2019.

[77] Pieter Delobelle, Thomas Winters, and Bettina Berendt. Robbert: a dutch roberta-based language model. *arXiv preprint arXiv:2001.06286*, 2020.

[78] Branden Chan, Stefan Schweter, and Timo Möller. German's next language model. *arXiv preprint arXiv:2010.10906*, 2020.

[79] Jakub Sido, Ondřej Pražák, Pavel Přibáň, Jan Pašek, Michal Seják, and Miloslav Konopík. Czert–czech bert-like model for language representation. *arXiv preprint arXiv:2103.13031*, 2021.

[80] Yiming Cui, Wanxiang Che, Ting Liu, Bing Qin, and Ziqing Yang. Pre-training with whole word masking for chinese bert. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 29:3504–3514, 2021.

[81] Antti Virtanen, Jenna Kanerva, Rami Ilo, Jouni Luoma, Juhani Luotolahti, Tapio Salakoski, Filip Ginter, and Sampo Pyysalo. Multilingual is not enough: Bert for finnish. *arXiv preprint arXiv:1912.07076*, 2019.

[82] Zihan Wang, Stephen Mayhew, Dan Roth, et al. Extending multilingual bert to low-resource languages. *arXiv preprint arXiv:2004.13640*, 2020.

[83] Stefan Daniel Dumitrescu, Andrei-Marius Avram, and Sampo Pyysalo. The birth of romanian bert. *arXiv preprint arXiv:2009.08712*, 2020.

[84] Wietse de Vries, Andreas van Cranenburgh, Arianna Bisazza, Tommaso Caselli, Gertjan van Noord, and Malvina Nissim. Bertje: A dutch bert model. *arXiv preprint arXiv:1912.09582*, 2019.

[85] Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, et al. Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144*, 2016.

[86] 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. *arXiv preprint arXiv:1907.11692*, 2019.

[87] Imola K Fodor. A survey of dimension reduction techniques. Technical report, Citeseer, 2002.

[88] Samuel Kaski. Dimensionality reduction by random mapping: Fast similarity computation for clustering. In *1998 IEEE International Joint Conference on Neural Networks Proceedings. IEEE World Congress on Computational Intelligence (Cat. No. 98CH36227)*, volume 1, pages 413–418. IEEE, 1998.

[89] Karl Pearson F.R.S. Liii. on lines and planes of closest fit to systems of points in space. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 2(11):559–572, 1901. doi: 10.1080/14786440109462720.

[90] Jiaqi Mu, Suma Bhat, and Pramod Viswanath. All-but-the-top: Simple and effective postprocessing for word representations. *arXiv preprint arXiv:1702.01417*, 2017.

[91] Changjie Hu, Xiaoli Hou, and Yonggang Lu. Improving the architecture of an autoencoder for dimension reduction. In *2014 IEEE 11th Intl Conf on Ubiquitous Intelligence and Computing and 2014 IEEE 11th Intl Conf on Autonomic and Trusted Computing and 2014 IEEE 14th Intl Conf on Scalable Computing and Communications and Its Associated Workshops*, pages 855–858. IEEE, 2014.

[92] Yasi Wang, Hongxun Yao, and Sicheng Zhao. Auto-encoder based dimensionality reduction. *Neurocomputing*, 184:232–242, 2016.

[93] Saldju Tadjudin and David A Landgrebe. Covariance estimation with limited training samples. *IEEE Transactions on Geoscience and Remote Sensing*, 37(4): 2113–2118, 1999.

[94] Geoffrey Hinton and Sam T Roweis. Stochastic neighbor embedding. In *NIPS*, volume 15, pages 833–840. Citeseer, 2002.

[95] Joseph B Kruskal. Nonmetric multidimensional scaling: a numerical method. *Psychometrika*, 29(2):115–129, 1964.

[96] Ingwer Borg and Patrick JF Groenen. *Modern multidimensional scaling: Theory and applications*. Springer Science & Business Media, 2005.

[97] Josh Barnes and Piet Hut. A hierarchical o (n log n) force-calculation algorithm. *nature*, 324(6096):446–449, 1986.

[98] Laurens Van Der Maaten. Barnes-hut-sne. *arXiv preprint arXiv:1301.3342*, 2013.

[99] Leland McInnes, John Healy, and James Melville. Umap: Uniform manifold approximation and projection for dimension reduction. *arXiv preprint arXiv:1802.03426*, 2018.

[100] Gautier Izacard and Edouard Grave. Leveraging passage retrieval with generative models for open domain question answering. *arXiv preprint arXiv:2007.01282*, 2020.

[101] Lynton Ardizzone, Jakob Kruse, Sebastian Wirkert, Daniel Rahner, Eric W Pellegrini, Ralf S Klessen, Lena Maier-Hein, Carsten Rother, and Ullrich Köthe. Analyzing inverse problems with invertible neural networks. *arXiv preprint arXiv:1808.04730*, 2018.

[102] Jens Behrmann, Paul Vicol, Kuan-Chieh Wang, Roger Grosse, and Jörn-Henrik Jacobsen. Understanding and mitigating exploding inverses in invertible neural networks. In *International Conference on Artificial Intelligence and Statistics*, pages 1792–1800. PMLR, 2021.