Knowledge Base Index Compression via Dimensionality and Precision Reduction

Anonymous ACL submission

Abstract

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 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.

1 Introduction

Recent approaches to knowledge-intensive NLP tasks combine neural network based models with a retrieval component that leverages dense vector representations (Guu et al., 2020; Lewis et al., 2020; Petroni et al., 2021). The most straightforward example is question answering, where the retriever receives as input a question and returns relevant documents to be used by the reader (both encoder and decoder), which outputs the answer (Chen, 2020). The same approach can also be applied in other contexts, such as fact-checking (Tchechmedjiev et al., 2019) or knowledgable dialogue (Dinan et al., 2018). 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. (2019) or the dynamic gating language model mechanism by Yogatama et al. (2021). All these pipelines are generalized as retrieving an artefact from a knowledge base (Zouhar et al., 2021) on which the reader is conditioned together with the query.

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, making deployment and experimentation very difficult. 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. (2021) which performs retrieval over a database of 1.8 billion documents.

This paper focuses on the issue of compressing the index through dimensionality and precision reduction and makes the following contributions:

- 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 practicioners.

[1] Sparse representations via BM25 (Robertson et al., 1995) 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.

In Section 3, we describe the problem scenario and the experimental setup. We discuss the results of different compression methods in Section 4. We provide further analysis in Section 5 and conclude with usage recommendations in Section 6. The repository for this project is available open-source.[3]

2 Related Work

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. (2021). 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 paper 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. (2020) 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. (2021), 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 4 and 5 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. (2021). 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 4.4 we show that they can be combined.

Finally, He et al. (2021) 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. Timkey and van Schijndel (2021) 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.

3 Setup

3.1 Problem Statement and Evaluation

Given a query q, the following set of equations summarizes the conceptual progression from retrieving 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 $f_Q : \mathcal{Q} \rightarrow \mathbb{R}^d$ and $f_D : \mathcal{D} \rightarrow \mathbb{R}^d$ are used to map the query and *all* documents to a shared embedding space and a similarity function $\text{sim} : \mathbb{R}^d \times \mathbb{R}^d \rightarrow \mathbb{R}$ approximates the relevance between query and documents. Here, we consider either the inner product or the L^2 distance as sim.[4] Finally, 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:

$$Z = \underset{d \in \mathcal{D}}{\arg \text{top-k}} \ \text{rel.}(q, d) \text{ , with} \tag{1}$$

$$\text{rel.}(q, d) \approx \text{sim}(f_\text{Q}(q), f_\text{D}(d)) \tag{2}$$

$$\approx \text{sim}(r_\text{Q}(f_\text{Q}(q)), r_\text{D}(f_\text{D}(d))) \tag{3}$$

The approximation in (2) was shown to work well in practice for inner product and L^2 distance (Lin, 2021). In this case, f_Q is commonly fine-tuned for a specific downstream task. For this reason, 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

[3] Link will be available in the camera-ready version.

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

and document embedding functions, they can be fine-tuned.

Task Agnostic Representation. 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 (Lewis et al., 2020; Petroni et al., 2021). 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.[5] 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.

R-Precision. To evaluate retrieval performance we compute *R-Precision* averaged over queries: (relevant documents among top k passages in Z)/r, k = number of passages in relevant documents, in the same way as Petroni et al. (2021). Following previous work, we consider the inner product (IP) and the L^2 distance as the similarity function.

3.2 Data

As knowledge base we use documents from English Wikipedia and follow the setup described by Petroni et al. (2021). 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.[6] 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 (Yang et al., 2018) for all main experiments and Natural Questions (Kwiatkowski et al., 2019) to verify that the results transfer to other datasets as well. This leads to over 2 million encoded spans for HotpotQA (see Table 5 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).

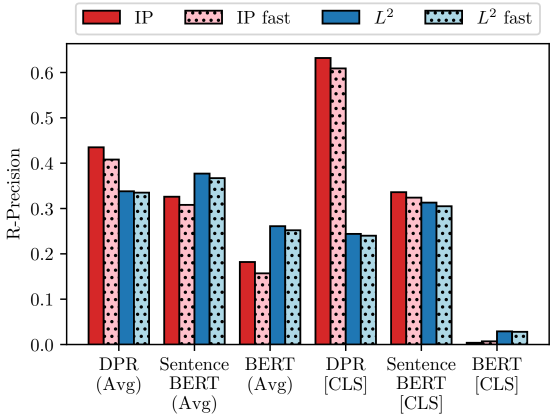

Figure 1: Comparison of different BERT-based embedding models and versions when using faster but slightly inaccurate nearest neighbour search. **[CLS]** is the specific token embedding from the last layer while **(Avg)** is all token average.

3.3 Uncompressed Retrieval Peformance

To establish baselines for uncompressed performance we use models based on BERT (Devlin et al., 2019). We consider (1) vanilla BERT, (2) Sentence-BERT (Reimers and Gurevych, 2019) and (3) DPR (Karpukhin et al., 2020), which was specifically trained for document retrieval. To obtain document embeddings, we use either the last hidden state representation at `[CLS]` or the average across tokens of the last layer.

Our first experiment compares the retrieval performance of the different models on HotpotQA. The result is shown in Figure 1. In alignment with previous works (Reimers and Gurevych, 2019) an immediately noticeable conclusion is that vanilla BERT has a poor performance, especially when taking the hidden state representation for the `[CLS]` token. Next, to make computation tractable, we repeat the experiment using FAISS (Johnson et al., 2019).[7] We find that the performance loss across models is systematic, which warrants the use of this approximate nearest neighbour search for comparisons and all our following experiments will use FAISS on the DPR-CLS model.

Pre-processing Transformations. Figure 1 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 2 shows that performing only normaliza-

[5] Guu et al. (2020) provide evidence that this assumption can lead to worse results in some cases.

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

[7] IndexIVFFlat, nlist=200, nprobe=100.

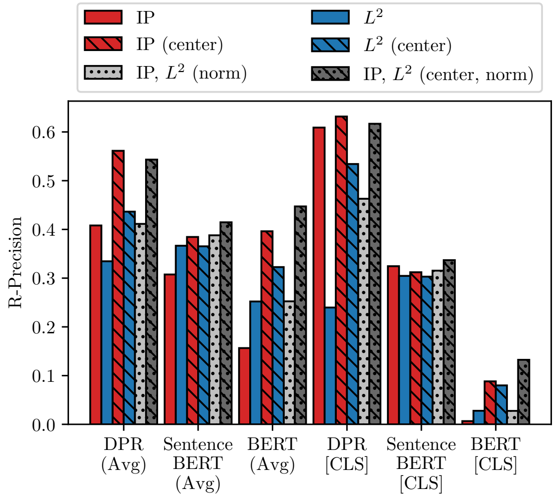

Figure 2: Effect of data centering and normalization on performance (evaluated with FAISS).

tion ($\frac{\boldsymbol{x}}{||\boldsymbol{x}||}$) sometimes hurts the performance but when joined with centering beforehand ($\frac{\boldsymbol{x}-\bar{\boldsymbol{x}}}{||\boldsymbol{x}-\bar{\boldsymbol{x}}||}$), it improves the results (compared to no preprocessing) 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. [8]

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.

4 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.

4.1 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):

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

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 (Fodor, 2002). Such random projections are suitable mostly for inner product (Kaski, 1998) though the differences are removed by normalizing the vectors (which also improves the performance).

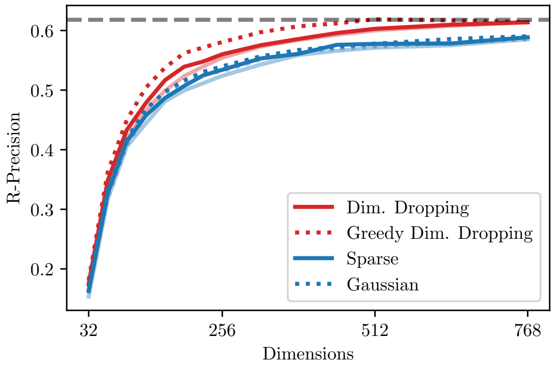

Figure 3: Dimension reduction using different random projections methods. Presented values are the max of 3 runs (except for greedy dimension dropping, which is deterministic), semi-transparent lines correspond to the minimum. Embeddings are provided by centered and normalized DPR-CLS. Final vectors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

Results. The results of all random projection methods are shown in Figure 3. 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 2, 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 inner product.

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

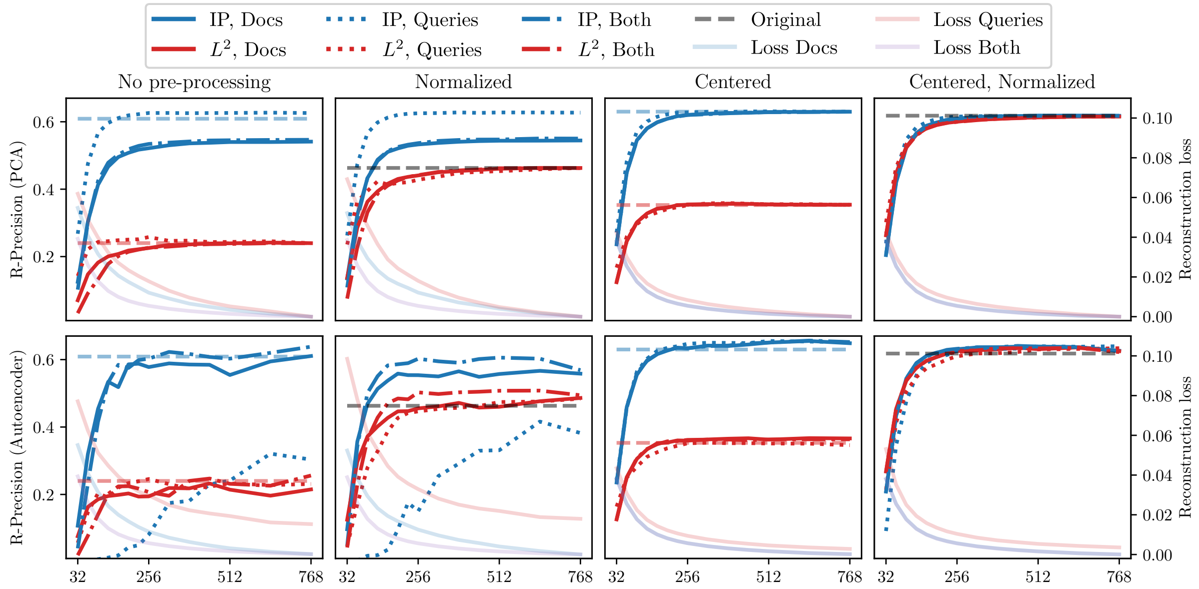

Figure 4: Dimension reduction using PCA (top) and Autoencoder (bottom) trained either on document index, query embeddings or both. Each figure corresponds to one of the four possible combinations of centering and normalizing the input data. The output vectors are not post-processed. Reconstruction loss (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.

4.2 Principal Component Analysis

Another natural candidate for dimensionality reduction is principal component analysis (PCA) (F.R.S., 1901). 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 4. 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 4 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 the dependency on training data size for PCA is explored explicitly in Section 5.1. The reason why queries provide better results without centering is that they are more centered in the first place, as shown in Table 1.

	Avg. $\mathbf{L^1}$ (std)	Avg. $\mathbf{L^2}$ (std)
Documents	243.0 (20.1)	12.3 (0.6)
Queries	137.0 (7.5)	9.3 (0.2)

Table 1: 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 4 shows that even after vastly decreasing the reconstruction loss, no significant improvements in retrieval performance are achieved. We further discuss this finding in Appendix C.

Component Scaling. One potential issue of PCA is that there may be dimensions that dominate the vector space. Mu et al. (2017) 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 2 shows that it provides a small additional boost in retrieval performance.

4.3 Autoencoder

A straightforward extension of PCA for dimensionality reducing is to use autoencoders, which has been widely explored (Hu et al., 2014; Wang et al., 2016). Usually, the model is described by an encoder $e : \mathbb{R}^d \rightarrow \mathbb{R}^b$, a function from a higher dimension to the target (bottleneck) dimension and a decoder $r : \mathbb{R}^b \rightarrow \mathbb{R}^d$, 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:

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:

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

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

$$e_3(\boldsymbol{x}) = L_{512}^{768} \circ \tanh \circ L_{256}^{512} \circ \tanh \circ L_{128}^{256}$$
$$r_3(\boldsymbol{x}) = L_{768}^{128}$$

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

Results. We explore the effects of training data and pre-processing with results for the first model shown in Figure 4. 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 2 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, inner product works yields better results.

4.4 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 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.

Using only one bit per dimension is a special case of precision reduction suggested by Yamada et al. (2021). 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 $\texttt{1}$ would then correspond to $1 - \alpha$ and $\texttt{0}$ to $0 - \alpha$. While Yamada et al. (2021) use values 1 and 0, we work with 0.5 and -0.5 in order to be able to distinguish between certain cases when using IP-based similarity.[9] As shown in Table 2, 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 is that if one wishes to use standard retrieval pipelines, these variables would have to be converted to a supported, larger, data type.

[9] 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.

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 2: 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 HotpotQA.

Finally, reducing precision can be readily combined with dimension reduction methods (see Appendix D), such as PCA (prior to changing the data type). As shown in the last row of Table 2, this can lead to the compressed size be 100x smaller while retaining 75% retrieval performance on HotpotQA and 89% for NaturalQuestions (see Table 6).

5 Analysis

5.1 Model Comparison

The comparison of all discussed dimension reduction methods is shown in Table 2. 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). Additionally, Appendix B compares training and evaluation speeds of common implementations.

5.2 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 5.

While Ma et al. (2021) 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 (Tadjudin and Landgrebe, 1999). 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 5. Retrieval performance quickly deteriorates for both models (faster than for the uncompressed case), highlighting the importance of filtering irrelevant documents from the knowledge base.

5.3 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 re-

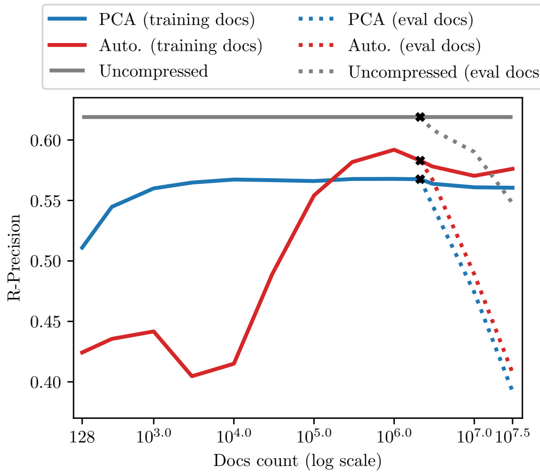

Figure 5: Dependency of PCA and autoencoder performance (evaluated on HotpotQA dev data, trained on document encodings) by modifying training data (solid lines) and by adding irrelevant documents to the retrieval pool (dashed lines). Black crosses indicate the original training size. Note the log scale on the x-axis and the truncation of the y-axis.

trieval with the original document embeddings to retrieval with PCA and 1-bit compression.

We find that there are no systematic differences compared to the uncompressed retrieval. This is demonstrated by the small off-diagonal values in Figure 6. 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 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.

6 Discussion

In this section we briefly discuss the main conclusions from our experiments and analysis in the form of recommendations for NLP practicioners.

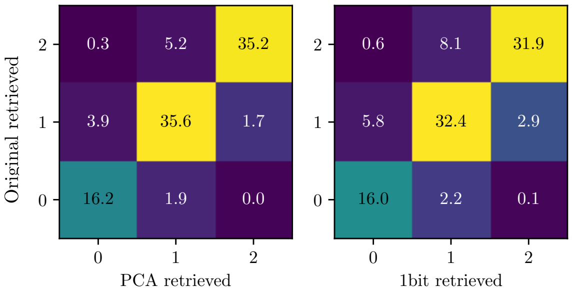

Figure 6: Distribution of the number of retrieved documents for HotpotQA queries before and after compression: PCA (128) and 1-bit precision with R-Precisions (centered & normalized) of 0.579 and 0.561, respectively.

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.

Method recommendation. While most compression methods achieve similar retrieval performance and compression ratios (cf. Table 2 and Table 6), PCA stands out in the following regards: (1) It requires only minimal implementation effort and no tuning of hyper-parameters beyond selecting which principal components to keep; (2) 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; (3) PCA can easily be combined with precision reduction based approaches.

7 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.

Future work. 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.

References

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

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

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. 2021. Improving language models by retrieving from trillions of tokens.

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

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. 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.

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

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

Karl Pearson F.R.S. 1901. 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.

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

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

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

Changjie Hu, Xiaoli Hou, and Yonggang Lu. 2014. 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.

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

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

Vladimir Karpukhin, Barlas Oguz, Sewon Min, Patrick Lewis, Ledell Wu, Sergey Edunov, Danqi Chen, and Wen-tau Yih. 2020. 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.

Samuel Kaski. 1998. 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.

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

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

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

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

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

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

Xueguang Ma, Minghan Li, Kai Sun, Ji Xin, and Jimmy Lin. 2021. 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.

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

Fabio Petroni, Aleksandra Piktus, Angela Fan, Patrick Lewis, Majid Yazdani, Nicola De Cao, James Thorne, Yacine Jernite, Vladimir Karpukhin, Jean Maillard, et al. 2021. 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.

Nils Reimers and Iryna Gurevych. 2019. 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.

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

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

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

William Timkey and Marten van Schijndel. 2021. 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.

Laurens Van Der Maaten. 2013. Barnes-hut-sne. *arXiv preprint arXiv:1301.3342*.

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

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

Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William Cohen, Ruslan Salakhutdinov, and Christopher D Manning. 2018. 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.

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

Vilém Zouhar, Marius Mosbach, Debanjali Biswas, and Dietrich Klakow. 2021. Artefact retrieval: Overview of NLP models with knowledge base access. In *Workshop on Commonsense Reasoning and Knowledge Bases*.

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

Table 3: Hyperparameters of autoencoder architectures described in Section 4.3. L_1 regularization is used only when explicitly mentioned.

A Pre-processing

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 4. 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 4: 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.

B 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 7 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.

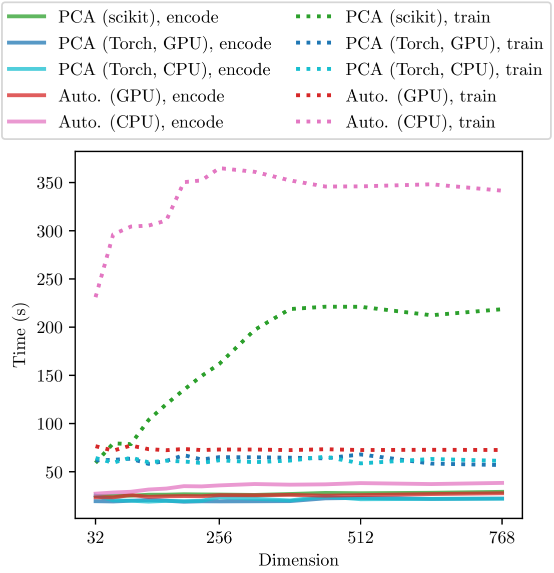

Figure 7: Speed comparison of PCA and autoencoder (model 3) implemented in PyTorch and Scikit[10]split into training and encoding parts. Models were trained on documents and queries jointly (normalized).

C 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.

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

Distance Learning. 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

[10] 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.

(t-SNE) embedding built on the work of Hinton and Roweis (2002) or multidimensional scaling (Kruskal, 1964; Borg and Groenen, 2005). 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[11] 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).

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:

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

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.

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.

D 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 8 show that PCA can be combined with e.g. 8-bit precision reduction with negligible loss in performance.

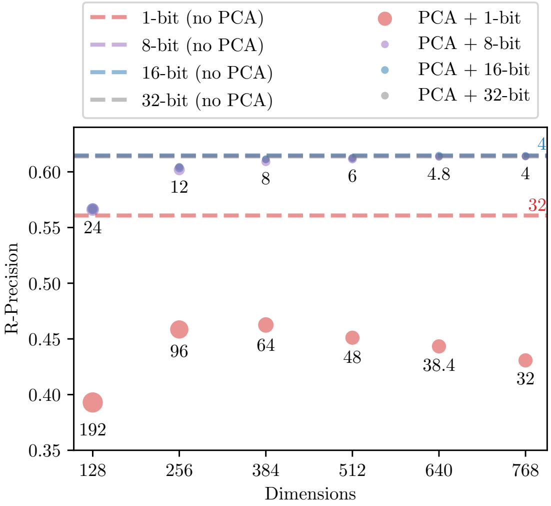

Figure 8: Combination of PCA and precision reduction. Compression ratio is shown in text. 16-bit and 32-bit values overlap with 8-bit and their compression ratios are not shown. Measured on HotpotQA with DPR-CLS.

E Comparison on Natural Questions

We also show the major experiments in Table 6 (table structure equivalent to that for the pruned dataset in Table 2) on Natural Question (Kwiatkowski et al., 2019) 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 paper.

[11] The common fast implementation for t-SNE, Barnes-Hut (Barnes and Hut, 1986; Van Der Maaten, 2013) is based on either quadtrees or octrees and is limited to 3 dimensions.

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 5: Number of training and dev queries and documents for the different datasets used.

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 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.