The authors look to improve dense passage retrieval's space requirement. Methods like HNSW require a lot of space, but this can be improved by learning a binary encoding. The authors jointly train a retriever and ranker over binary codes, which shows promising performance at a fraction of the space requirement. It's unclear if hamming distances can be applied in more complex data spaces where candidate generation plays a larger role in performance.

Efficient Passage Retrieval with Hashing for Open-domain Question Answering

Ikuya Yamada[†,‡] **Akari Asai**[*] **Hannaneh Hajishirzi**[*,§]

[†]Studio Ousia [‡]RIKEN [*]University of Washington

[§]Allen Institute for AI

ikuya@ousia.jp {akari,hannaneh}@cs.washington.edu

Abstract

Most state-of-the-art open-domain question answering systems use a neural retrieval model to encode passages into continuous vectors and extract them from a knowledge source. However, such retrieval models often require large memory to run because of the massive size of their passage index. In this paper, we introduce *Binary Passage Retriever* (*BPR*), a memory-efficient neural retrieval model that integrates a *learning-to-hash* technique into the state-of-the-art Dense Passage Retriever (DPR) (Karpukhin et al., 2020) to represent the passage index using compact binary codes rather than continuous vectors. BPR is trained with a multi-task objective over two tasks: efficient candidate generation based on binary codes and accurate reranking based on continuous vectors. Compared with DPR, BPR substantially reduces the memory cost from 65GB to 2GB without a loss of accuracy on two standard open-domain question answering benchmarks: Natural Questions and TriviaQA. Our code and trained models are available at `https://github.com/studio-ousia/bpr`.

1 Introduction

Open-domain question answering (QA) is the task of answering arbitrary factoid questions based on a knowledge source (e.g., Wikipedia). Recent state-of-the-art QA models are typically based on a two-stage *retriever–reader* approach (Chen et al., 2017) using a *retriever* that obtains a small number of relevant passages from a large knowledge source and a *reader* that processes these passages to produce an answer. Most recent successful retrievers encode questions and passages into a common continuous embedding space using two independent encoders (Lee et al., 2019; Karpukhin et al., 2020; Guu et al., 2020). Relevant passages are retrieved using a nearest neighbor search on the index containing the passage embeddings with a question embedding as a query.

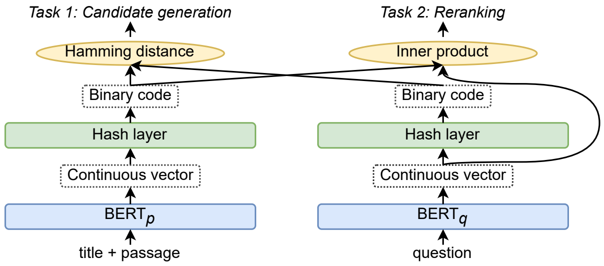

Figure 1: Architecture of BPR, a BERT-based model generating compact binary codes for questions and passages. The passages are retrieved in two stages: (1) efficient candidate generation based on the Hamming distance using the binary code of the question and (2) accurate reranking based on the inner product using the continuous embedding of the question.

These retrievers often outperform classical methods (e.g., BM25), but they incur a large memory cost due to the massive size of their passage index, which must be stored entirely in memory at runtime. For example, the index of a common knowledge source (e.g., Wikipedia) requires dozens of gigabytes.[1] A reduction in the index size is critical for real-world QA that requires large knowledge sources such as scientific databases (e.g., PubMed) and web-scale corpora (e.g., Common Crawl).

In this paper, we introduce *Binary Passage Retriever* (*BPR*), which learns to hash continuous vectors into compact binary codes using a multi-task objective that simultaneously trains the encoders and hash functions in an end-to-end manner (see Figure 1). In particular, BPR integrates our *learning-to-hash* technique into the state-of-the-art Dense Passage Retriever (DPR) (Karpukhin et al., 2020) to drastically reduce the size of the

[1] The passage index of the off-the-shelf DPR model (Karpukhin et al., 2020) requires 65GB for indexing the 21M English Wikipedia passages, which have 13GB storage size.

passage index by storing it as binary codes. BPR computes binary codes by applying the sign function to continuous vectors. As the sign function is not compatible with back-propagation, we approximate it using the scaled tanh function during training. To improve search-time efficiency while maintaining accuracy, BPR is trained to obtain both binary codes and continuous embeddings for questions with multi-task learning over two tasks: (1) candidate generation based on the Hamming distance using the binary code of the question and (2) reranking based on the inner product using the continuous embedding of the question. The former task aims to detect a small number of candidate passages efficiently from the entire passages and the latter aims to rerank the candidate passages accurately.

We conduct experiments using the Natural Questions (NQ) (Kwiatkowski et al., 2019) and TriviaQA (TQA) (Joshi et al., 2017) datasets. Compared with DPR, our BPR achieves similar QA accuracy and competitive retrieval performance with a substantially reduced memory cost from 65GB to 2GB. Furthermore, using an improved reader, we achieve results that are competitive with those of the current state of the art in open-domain QA. Our code and trained models are available at `https://github.com/studio-ousia/bpr`.

2 Related Work

Retrieval for Open-domain QA Many recent open-domain QA models depend on the retriever to select relevant passages from a knowledge source. Early works involved the adoption of sparse representations (Chen et al., 2017) for the retriever, whereas recent works (Lee et al., 2019; Guu et al., 2020; Karpukhin et al., 2020) have often adopted dense representations based on neural networks. Our work is an extension of DPR (Karpukhin et al., 2020), which has been used in recent state-of-the-art QA models (Lewis et al., 2020; Izacard and Grave, 2020). Concurrent with our work, Izacard et al. (2020) attempted to reduce the memory cost of DPR using post-hoc product quantization with dimension reduction and filtering of passages. However, they observed a significant degradation in the QA accuracy compared with their full model. We adopt the learning-to-hash method with our multi-task objective and substantially compress the index without losing accuracy.

Learning to Hash The objective of hashing is to reduce the memory and search-time cost of the nearest neighbor search by representing data points using compact binary codes. Learning to hash (Wang et al., 2016, 2018) is a method for learning hash functions in a data-dependent manner. Recently, many *deep-learning-to-hash* methods have been proposed (Lai et al., 2015; Zhu et al., 2016; Li et al., 2016; Cao et al., 2017, 2018) to jointly learn feature representations and hash functions in an end-to-end manner. We follow Cao et al. (2017) to implement our hash functions. Similar to our work, Xu and Li (2020) used the learning-to-hash method to reduce the computational cost of the answer sentence selection task, the objective of which is to select an answer sentence from a limited number of candidates (up to 500 in their experiments). Our work is different from the aforementioned work because we focus on efficient and scalable passage retrieval from a large knowledge source (21M Wikipedia passages in our experiments) using an effective multi-task approach. In addition to hashing-based methods, improving approximate neighbor search has been actively studied (Jégou et al., 2011; Malkov and Yashunin, 2020; Guo et al., 2020). We use Jégou et al. (2011) and Malkov and Yashunin (2020) as baselines in our experiments.

3 Model

Given a question and large-scale passage collection such as Wikipedia, a retriever finds relevant passages that are subsequently processed by a reader. Our retriever is built on DPR (Karpukhin et al., 2020), which is a retriever based on BERT (Devlin et al., 2019). In this section, we first introduce DPR and then explain our model.

3.1 Dense Passage Retriever (DPR)

DPR uses two independent BERT encoders to encode question q and passage p into d-dimensional continuous embeddings:

$$\mathbf{e}_q = \mathrm{BERT}_q(q), \quad \mathbf{e}_p = \mathrm{BERT}_p(p), \qquad (1)$$

where $\mathbf{e}_q \in \mathbb{R}^d$ and $\mathbf{e}_p \in \mathbb{R}^d$. We use the uncased version of BERT-base; therefore, $d = 768$. The output representation of the `[CLS]` token is obtained from the encoder. To create passage p, the passage title and text are concatenated (`[CLS]` *title* `[SEP]` *passage* `[SEP]`). The relevance score of passage p, given question q, is computed using the inner product of the corresponding vectors, $\langle \mathbf{e}_q, \mathbf{e}_p \rangle$.

BPR achieves retrieval based on Hamming distance of their binary code and rerank using the full embedding. Training is done jointly on both.

Training Let $\mathcal{D} = \{\langle q_i, p_i^+, p_{i,1}^-, \cdots, p_{i,n}^- \rangle\}_{i=1}^m$ be m training instances consisting of a question q_i, a passage that answers the question (positive passage), p_i^+, and n passages that are irrelevant for the question (negative passages), $p_{i,j}^-$. The model is trained by minimizing the negative log-likelihood of the positive passage:

$$\mathcal{L}_{\text{dpr}} = -\log \frac{\exp(\langle \mathbf{e}_{q_i}, \mathbf{e}_{p_i^+} \rangle)}{\exp(\langle \mathbf{e}_{q_i}, \mathbf{e}_{p_i^+} \rangle) + \sum_{j=1}^{n} \exp(\langle \mathbf{e}_{q_i}, \mathbf{e}_{p_{i,j}^-} \rangle)}. \quad (2)$$

Inference DPR creates a passage index by applying the passage encoder to each passage in the knowledge source. At runtime, it retrieves the top-k passages employing maximum inner product search with the question embedding as a query.

3.2 Model Architecture

Figure 1 shows the architecture of BPR. BPR builds a passage index by computing a binary code for each passage in the knowledge source. To compute the binary codes for questions and passages, we add a *hash layer* on top of the question and passage encoders of DPR. Given embedding $\mathbf{e} \in \mathbb{R}^d$ computed by an encoder, the hash layer computes its binary code, $\mathbf{h} \in \{-1, 1\}^d$, as

$$\mathbf{h} = \text{sign}(\mathbf{e}), \quad (3)$$

where $\text{sign}(\cdot)$ is the sign function such that for $i = 1, ..., d$, $\text{sign}(h_i) = 1$ if $h_i > 0$; otherwise, $\text{sign}(h_i) = -1$. However, the sign function is incompatible with back-propagation because its gradient is zero for all non-zero inputs and is ill-defined at zero. Inspired by Cao et al. (2017), we address this by approximating the sign function using the scaled tanh function during the training:

$$\tilde{\mathbf{h}} = \tanh(\beta \mathbf{e}), \quad (4)$$

where β is a scaling parameter. When β increases, the function gradually becomes non-smooth, and as $\beta \rightarrow \infty$, it converges to the sign function. At each training step, we increase β by setting $\beta = \sqrt{\gamma \cdot step + 1}$, where *step* is the number of finished training steps. We set $\gamma = 0.1$ and explain the effects of changing it in Appendix B.

3.3 Two-stage Approach

To reduce the computational cost without losing accuracy, BPR splits the task into candidate generation and reranking stages. At the candidate generation stage, we efficiently obtain the top-l candidate passages using the Hamming distance between the binary code of question $\mathbf{h}_q$ and that of each passage, $\mathbf{h}_p$. We then rerank the l candidate passages using the inner product between the continuous embedding of question $\mathbf{e}_q$ and $\mathbf{h}_p$ and select the top-k passages from the reranked candidates. We perform candidate generation using binary code $\mathbf{h}_q$ for search-time efficiency, and reranking using expressive continuous embedding $\mathbf{e}_q$ for accuracy. We set $l = 1000$ and describe the effects of using different l values in Appendix C.

3.4 Training

To compute effective representations for both the candidate generation and reranking stages, we combine the loss functions of the two tasks:

$$\mathcal{L} = \mathcal{L}_{\text{cand}} + \mathcal{L}_{\text{rerank}}. \quad (5)$$

Task #1 for Candidate Generation The objective of this task is to improve candidate generation using the ranking loss with the approximated hash code of question $\tilde{\mathbf{h}}_q$ and that of passage $\tilde{\mathbf{h}}_p$:

$$\mathcal{L}_{\text{cand}} = \textstyle\sum_{j=1}^{n} \max(0, -(\langle \tilde{\mathbf{h}}_{q_i}, \tilde{\mathbf{h}}_{p_i^+} \rangle + \langle \tilde{\mathbf{h}}_{q_i}, \tilde{\mathbf{h}}_{p_{i,j}^-} \rangle) + \alpha). \quad (6)$$

We set $\alpha = 2$ and investigate the effects of selecting different α values and using the cross-entropy loss instead of the ranking loss in Appendix D. Note that the retrieval performance based on the Hamming distance can be optimized using this loss function because the Hamming distance and inner product can be used interchangeably for binary codes.[2]

Task #2 for Reranking We improve the reranking stage using the following loss function:

$$\mathcal{L}_{\text{rerank}} = -\log \frac{\exp(\langle \mathbf{e}_{q_i}, \tilde{\mathbf{h}}_{p_i^+} \rangle)}{\exp(\langle \mathbf{e}_{q_i}, \tilde{\mathbf{h}}_{p_i^+} \rangle) + \sum_{j=1}^{n} \exp(\langle \mathbf{e}_{q_i}, \tilde{\mathbf{h}}_{p_{i,j}^-} \rangle)}. \quad (7)$$

This function is equivalent to $\mathcal{L}_{\text{dpr}}$, with the exception that $\tilde{\mathbf{h}}_p$ is used instead of $\mathbf{e}_p$.

3.5 Algorithms for Candidate Generation

To perform candidate generation, we test two standard algorithms: (1) linear scan based on efficient Hamming distance computation,[3] and (2) hash table lookup implemented by building a hash table that maps each binary code to the corresponding passages and querying it multiple times by increasing the Hamming radius until we obtain l passages.

[2] Given two binary codes, $\mathbf{h}_i$ and $\mathbf{h}_j$, there exists a relationship between their Hamming distance, $\text{dist}_H(\cdot, \cdot)$, and inner product, $\langle \cdot, \cdot \rangle$: $\text{dist}_H(\mathbf{h}_i, \mathbf{h}_j) = \frac{1}{2}(const - \langle \mathbf{h}_i, \mathbf{h}_j \rangle)$.

[3] The Hamming distance can be computed more efficiently than the inner product using the POPCNT CPU instruction.

top 1k seems to be enough to capture 100% recall.

Is hamming distance flexible when presented with noisy data domains?

Model	Top 1		Top 20		Top 100		QA Acc. (EM)		Index size	Query time
	NQ	TQA	NQ	TQA	NQ	TQA	NQ	TQA		
DPR	**46.0**	**53.5**	78.4	**79.4**	85.4	**85.0**	41.5	**56.8**	64.6GB	456.9ms
DPR + HNSW	45.7	53.2	**78.8**	78.8	85.2	84.2	41.2	56.6	151.0GB	1.8ms
DPR + Simple LSH	21.5	28.4	63.9	65.2	77.2	76.9	35.8	48.1	2.0GB	28.8ms
DPR + PQ	32.5	42.8	72.2	73.2	81.2	80.4	38.4	52.0	2.0GB	44.0ms
BPR (linear scan; $l = 1000$)	41.1	49.7	77.9	77.9	**85.7**	84.5	**41.6**	**56.8**	2.0GB	85.3ms
BPR (hash table lookup; $l = 1000$)	"	"	"	"	"	"	"	"	2.2GB	38.1ms

Table 1: Top k recall and exact match (EM) QA accuracy on test sets with the index size and query time of BPR and baselines. All models use the same reader based on BERT-base to evaluate the QA accuracy.

Model	Top 1		Top 20		Top 100		Query time
	NQ	TQA	NQ	TQA	NQ	TQA	
BPR ($l = 1000$)	41.1	49.7	77.9	77.9	85.7	84.5	38.1ms
BPR w/o reranking	38.0	46.1	76.5	75.9	84.9	83.4	37.9ms
BPR w/o candidate generation	41.1	49.7	77.9	77.9	85.7	84.5	457.8ms

Table 2: Results of our ablation study. Hash table lookup is used to implement candidate generation.

4 Experiments

Datasets We conduct experiments using the NQ and TQA datasets and English Wikipedia as the knowledge source. We use the following pre-processed data available on the DPR website:[4] Wikipedia corpus containing 21M passages and the training/validation datasets for the retriever containing multiple positive, *random* negative, and *hard* negative passages for each question.

Baselines We compare our BPR with DPR with linear scan and DPR with Hierarchical Navigable Small World (HSNW) graphs (Malkov and Yashunin, 2020) – which builds a multi-layer structure consisting of a hierarchical set of proximity graphs, following Karpukhin et al. (2020) – for our primary baselines. We also apply two popular post-hoc quantization algorithms to the DPR passage index: simple locality sensitive hashing (LSH) (Neyshabur and Srebro, 2015) and product quantization (PQ) (Jégou et al., 2011). We configure these algorithms such that their passage representations have the same size as that of BPR: the number of bits per passage of the LSH is set as 768, and the number of centroids and the code size of the PQ are configured as 96 and 8 bits, respectively.

Experimental settings Our experimental setup follows Karpukhin et al. (2020). We evaluate our model based on its top-k recall (the percentage of positive passages in the top-k passages), retrieval efficiency (the index size and query time), and exact match (EM) QA accuracy measured by combining our model with a reader. We use the same BERT-based reader as that used by DPR. Our model is trained using the same method as DPR. We conduct experiments on servers with two Intel Xeon E5-2698 v4 CPUs and eight Nvidia V100 GPUs. The passage index are built using Faiss (Johnson et al., 2019). Further details are provided in Appendix A.

4.1 Results

Main results Table 1 presents the top-k recall (for $k \in \{1, 20, 100\}$), EM QA accuracy, index size, and query time achieved by BPR and baselines on the NQ and TQA test sets. BPR achieves similar or even better performance than DPR in both retrieval with $k \geq 20$ and EM accuracy with a substantially reduced index size from 65GB to 2GB. We observe that BPR performs worse than DPR for $k = 1$, but usually the recall in small k is less important because the reader usually produces an answer based on $k \geq 20$ passages. BPR significantly outperforms all quantization baselines. The query time of BPR is substantially shorter than that of DPR. Hash table lookup is faster than linear scan but requires slightly more storage. DPR+HNSW is faster than BPR; however, it requires 151GB of storage.

Ablations Table 2 shows the results of our ablation study. Disabling the reranking clearly degrades performance, demonstrating the effectiveness of our two-stage approach. Disabling the can-

[4] https://github.com/facebookresearch/DPR

this doesn't look very apples-to-apples to me, but the perf per parameters is impressive.

Model	Pretrained model	#params	NQ	TQA
RAG (Lewis et al., 2020)	BART-large	406M	44.5	56.1
FiD (base) (Izacard and Grave, 2020)	T5-base	220M	48.2	65.0
FiD (large) (Izacard and Grave, 2020)	T5-large	770M	**51.4**	**67.6**
BPR ($l = 1000$)	BERT-base	110M	41.6	56.8
BPR ($l = 1000$)	ELECTRA-large	335M	49.0	65.6

Table 3: Exact match QA accuracy of BPR and state of the art models. BPR achieves performance close to FiD (large) with almost half of the parameters.

didate generation (treating all passages as candidates) results in the same performance as using only top-1000 candidates, but significantly increases the query time due to the expensive inner product computation over all passage embeddings.

Comparison with State of the Art Table 3 presents the EM QA accuracy of BPR combined with state-of-the-art reader models. Here, we also report the results of our model using an improved reader based on ELECTRA-large (Clark et al., 2020) instead of BERT-base. Our improved model outperforms all models except the large model of Fusion-in-Decoder (FiD), which contains more than twice as many parameters as our model.

5 Conclusion

We introduce BPR, which is an extension of DPR, based on a learning-to-hash technique and a novel two-stage approach. It reduces the computational cost of open-domain QA without a loss in accuracy.

Acknowledgement

We are grateful for the feedback and suggestions from the anonymous reviewers and the members of the UW NLP group. This research was supported by Allen Distinguished investigator award, a gift from Facebook, and the Nakajima Foundation Fellowship.

References

Yue Cao, Bin Liu, Mingsheng Long, and Jianmin Wang. 2018. HashGAN: Deep Learning to Hash With Pair Conditional Wasserstein GAN. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1287–1296.

Z Cao, M Long, J Wang, and P S Yu. 2017. HashNet: Deep Learning to Hash by Continuation. In *2017 IEEE International Conference on Computer Vision*, pages 5609–5618.

Danqi Chen, Adam Fisch, Jason Weston, and Antoine Bordes. 2017. Reading Wikipedia to Answer Open-Domain Questions. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1870–1879.

Kevin Clark, Minh-Thang Luong, Quoc V Le, and Christopher D Manning. 2020. ELECTRA: Pre-training Text Encoders as Discriminators Rather Than Generators. In *International Conference on Learning Representations*.

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.

Ruiqi Guo, Philip Sun, Erik Lindgren, Quan Geng, David Simcha, Felix Chern, and Sanjiv Kumar. 2020. Accelerating Large-Scale Inference with Anisotropic Vector Quantization. In *International Conference on Machine Learning*.

Kelvin Guu, Kenton Lee, Zora Tung, Panupong Pasupat, and Mingwei Chang. 2020. Retrieval Augmented Language Model Pre-Training. In *Proceedings of the 37th International Conference on Machine Learning*, volume 119, pages 3929–3938.

Gautier Izacard and Edouard Grave. 2020. Leveraging Passage Retrieval with Generative Models for Open Domain Question Answering. *arXiv preprint arXiv:2007.01282*.

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*.

H Jégou, M Douze, and C Schmid. 2011. Product Quantization for Nearest Neighbor Search. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 33(1):117–128.

J Johnson, M Douze, and H Jégou. 2019. Billion-Scale Similarity Search with GPUs. *IEEE Transactions on Big Data*.

Mandar Joshi, Eunsol Choi, Daniel Weld, and Luke Zettlemoyer. 2017. TriviaQA: A Large Scale Distantly Supervised Challenge Dataset for Reading Comprehension. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1601–1611.

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*, pages 6769–6781.

Tom Kwiatkowski, Jennimaria Palomaki, Olivia Redfield, Michael Collins, Ankur Parikh, Chris Alberti, Danielle Epstein, Illia Polosukhin, Jacob Devlin, Kenton Lee, Kristina Toutanova, Llion Jones, Matthew Kelcey, Ming-Wei Chang, Andrew M Dai, Jakob Uszkoreit, Quoc Le, and Slav Petrov. 2019. Natural Questions: A Benchmark for Question Answering Research. *Transactions of the Association for Computational Linguistics*, 7:453–466.

Hanjiang Lai, Yan Pan, Ye Liu, and Shuicheng Yan. 2015. Simultaneous Feature Learning and Hash Coding With Deep Neural Networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*.

Kenton Lee, Ming-Wei Chang, and Kristina Toutanova. 2019. Latent Retrieval for Weakly Supervised Open Domain Question Answering. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 6086–6096.

Patrick Lewis, Ethan Perez, Aleksandara Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, and others. 2020. Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks. In *Advances in Neural Information Processing Systems 33*.

Wu-Jun Li, Sheng Wang, and Wang-Cheng Kang. 2016. Feature Learning Based Deep Supervised Hashing with Pairwise Labels. In *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence*, pages 1711–1717.

Y A Malkov and D A Yashunin. 2020. Efficient and Robust Approximate Nearest Neighbor Search Using Hierarchical Navigable Small World Graphs. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 42(4):824–836.

Behnam Neyshabur and Nathan Srebro. 2015. On Symmetric and Asymmetric LSHs for Inner Product Search. In *Proceedings of the 32nd International Conference on Machine Learning*, volume 37, pages 1926–1934.

J Wang, W Liu, S Kumar, and S Chang. 2016. Learning to Hash for Indexing Big Data—A Survey. *Proceedings of the IEEE*, 104(1):34–57.

J Wang, T Zhang, J Song, N Sebe, and H T Shen. 2018. A Survey on Learning to Hash. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 40(4):769–790.

Dong Xu and Wu-Jun Li. 2020. Hashing Based Answer Selection. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34(05):9330–9337.

Han Zhu, Mingsheng Long, Jianmin Wang, and Yue Cao. 2016. Deep Hashing Network for Efficient Similarity Retrieval. *Proceedings of the AAAI Conference on Artificial Intelligence*, 30(1):2415–2421.

Appendix for "Efficient Passage Retrieval with Hashing for Open-domain Question Answering"

A Details of Experimental Setup

A.1 Knowledge Source

As the knowledge source, we use the preprocessed Wikipedia corpus consisting of 21,015,324 Wikipedia passages available on the website of Karpukhin et al. (2020). The corpus is based on the December 20, 2018 version of the English Wikipedia and created by filtering out semi-structured data (i.e., tables, infoboxes, lists, and disambiguation pages) and splitting the remaining Wikipedia articles into multiple, disjointed text blocks of 100 words each.

A.2 Question Answering Datasets

We conduct experiments using the NQ and TQA datasets with the training, development, and test sets as in Lee et al. (2019); Karpukhin et al. (2020). A brief description of these datasets is provided as follows:

- **NQ** is a QA dataset for which questions are obtained from Google queries and answers comprise the spans of English Wikipedia articles.
- **TQA** consists of trivia questions and their answers retrieved from the Web.

We use the preprocessed datasets available on the website of Karpukhin et al. (2020).[5] The numbers of questions contained in these datasets are listed in Table 4. For each question, the dataset contains three types of passages: (1) *positive passages* selected based on gold-standard human annotations or distant supervision, (2) *random negative passages* selected randomly from all the passages, and (3) *hard negative passages* selected based on the BM25 scores between the question and all the passages.

A.3 Details of BPR

Our training configuration follows that of Karpukhin et al. (2020). In particular, for each question, we use one positive and one hard negative passage and create a mini-batch comprising 128 questions. We use the method of *inbatch-negatives*, wherein each positive passage in a mini-batch is treated as the negative passage of each question in the mini-batch if it does not correspond to the question. Our model contains 220 million parameters, and is trained for up to 40 epochs using Adam. Regarding the hyperparameter search, we select the learning rate from the search range {1e-5, 2e-5, 3e-5, 5e-5} based on the top-100 recall on the validation set of the NQ dataset. Therefore, the number of hyperparameter search trials is 4. The detailed hyperparameters are listed in Table 5.

Dataset	Train	Validation	Test
NQ	58,880	8,757	3,610
TQA	60,413	8,837	11,313

Table 4: Number of questions in the preprocessed dataset used in our experiments.

Name	Value
Batch size	128
Maximum question length	256
Maximum passage length	256
Maximum training epochs	40
Peak learning rate	2e-5
Learning rate decay	linear
Warmup ratio	0.06
Dropout	0.1
Weight decay	0.0
Adam β_1	0.9
Adam β_2	0.999
Adam ϵ	1e-6

Table 5: Hyperparameters used to train BPR.

A.4 Details of Reader

For each passage in the top-k passages retrieved by the retriever, the reader assigns a relevance score to the passage and selects the best answer span in the passage. The final answer is the selected span from the passage with the highest relevance score.

Let $\mathbf{P}_i \in \mathbb{R}^{q \times d}$ $(1 \leq i \leq k)$ be a BERT output representation for the i-th passage, where q is the maximum token length of the passage, and d is the dimension size of the output representation. The probabilities of a passage being selected and a token being the start or end positions of an answer is computed as

$$P_{score}(i) = \text{softmax}\left(\hat{\mathbf{P}}^\top \mathbf{w}_{score}\right)_i, \quad (8)$$

$$P_{start,i}(s) = \text{softmax}\left(\mathbf{P}_i \mathbf{w}_{start}\right)_s, \quad (9)$$

$$P_{end,i}(t) = \text{softmax}\left(\mathbf{P}_i \mathbf{w}_{end}\right)_t, \quad (10)$$

where $\hat{\mathbf{P}} = [\mathbf{P}_1^{[\text{CLS}]}, \ldots, \mathbf{P}_k^{[\text{CLS}]}] \in \mathbb{R}^{d \times k}$, $\mathbf{w}_{score} \in \mathbb{R}^d$, $\mathbf{w}_{start} \in \mathbb{R}^d$, and $\mathbf{w}_{end} \in \mathbb{R}^d$.

[5] https://github.com/facebookresearch/DPR

Name	BERT-base	ELECTRA-large
Batch size	32	32
Maximum token length	350	350
Maximum training epochs	20	20
Negative passage size	23	17
Peak learning rate	2e-5	1e-5
Learning rate decay	linear	linear
Warmup ratio	0.06	0.06
Dropout	0.1	0.1
Weight decay	0.0	0.0
Adam β_1	0.9	0.9
Adam β_2	0.999	0.999
Adam ϵ	1e-6	1e-6

Table 6: Hyperparameters used to train the reader based on BERT-base and that based on ELECTRA-large.

Configuration	Top 1	Top 20	Top 100
$\gamma = 0.025$	39.4	**76.7**	83.8
$\gamma = 0.05$	39.5	76.5	84.0
$\gamma = 0.1$	**39.8**	**76.7**	**84.1**
$\gamma = 0.2$	39.6	76.3	83.9

Table 7: Top-1, top-20, and top-100 recall of our model with $\gamma \in \{0.025, 0.05, 0.1, 0.2\}$ on the validation set of the NQ dataset.

The passage selection score of the i-th passage is given as $P_{score}(i)$, and the score of the s-th to t-th tokens from the i-th passage is given as $P_{start,i}(s) \times P_{end,i}(t)$.

During the training, we sample one positive and multiple negative passages from the passages returned by the retriever. The model is trained to maximize the log-likelihood of the correct answer span in the positive passage, combined with the log-likelihood of the positive passage being selected. We use the BERT-base or ELECTRA-large as our pretrained model. Regarding the hyperparameter search, we select the learning rate from {1e-5, 2e-5, 3e-5, 5e-5} based on its EM accuracy on the validation set of the NQ dataset. Therefore, the number of hyperparameter search trials is 4. Detailed hyperparameters are listed in Table 6.

B Effects of Scaling Parameter

To investigate how the scaling parameter, γ, affects the performance, we test the performance of our model using various γ values, where $\gamma \in \{0.025, 0.05, 0.1, 0.2\}$. The retrieval performance on the validation set of the NQ dataset is shown in Table 7. Overall, the scaling parameter has a minor impact on the performance. We select $\gamma = 0.1$ because of its enhanced performance.

#candidates	Top 1		Top 20		Top 100	
	NQ	TQA	NQ	TQA	NQ	TQA
$l = 200$	41.1	49.7	77.9	77.9	85.4	84.0
$l = 500$	41.1	49.7	77.9	77.9	85.6	84.4
$l = 1000$	41.1	49.7	77.9	77.9	**85.7**	**84.5**
$l = 2000$	41.1	49.7	77.9	77.9	**85.7**	**84.5**

Table 8: Top-1, top-20, and top-100 recall of our model with $l \in \{200, 500, 1000\}$ on test sets.

Configuration	Top 1	Top 20	Top 100
Cross entropy loss	28.6	67.8	79.8
Ranking loss $\alpha = 0.0$	39.8	76.4	84.0
Ranking loss $\alpha = 1.0$	40.0	76.5	84.0
Ranking loss $\alpha = 2.0$	39.8	**76.7**	**84.1**
Ranking loss $\alpha = 4.0$	**40.3**	**76.7**	84.0

Table 9: Top-1, top-20, and top-100 recall of our model with the various settings of the loss function $\mathcal{L}_{\text{cand}}$ evaluated on the validation set of the NQ dataset.

C Effects of Number of Candidate Passages

We report the performance of our model with the varied number of candidate passages l in Table 8. Overall, BPR achieves similar performance in all settings. Increasing the number of candidate passages slightly improves the top-100 performance until it reaches $l = 1000$.

D Effects of Loss of Task #1 with Various Settings

We investigate the effects of using various settings of the loss function $\mathcal{L}_{\text{cand}}$ in Eq.(6). Instead of using the ranking loss, we test the performance with the cross-entropy loss, similar to Eq.(2), and $\tilde{\mathbf{h}}_q$ and $\tilde{\mathbf{h}}_p$ are used instead of $\mathbf{e}_q$ and $\mathbf{e}_p$, respectively. Furthermore, we also test how the parameter α affects the performance. As shown in Table 9, the cross-entropy loss clearly performs worse than the ranking loss. Furthermore, a change in the parameter α has a minor impact on the performance. Here, we select the ranking loss with $\alpha = 2.0$ because of its enhanced performance on the top-20 and top-100 performance.