Reproducing "Dense Passage Retrieval" (Karpukhin et al., EMNLP 2020)

Harrison Bay, Arnav Das, Pemi Nguyen
University of Washington
185 E Stevens Way NE, Seattle, WA 98195-2350
`{hcybay, arnavmd2, peming}@uw.edu`

Reproducibility Summary

Scope of Reproducibility

The main claim of the original paper [**1**] is that it is possible to create a state-of-the-art passage retrieval system for question answering applications through fine-tuning a pair of BERTs–one encoding the question and one encoding the passage. The compute used to train the passage retrieval system is prohibitively expensive; the authors trained on a machine with 8 32GB GPUs. As such, we do not attempt to reproduce *all* of the results in the paper, just a *few* selected ones. Additionally, our final experiments are performed on a subset of Wikipedia rather than the entirety of Wikipedia, decreasing our computational burden while still maintaining the relevance of the reproduction (see [**4**]).

Methodology

We used the authors' BM25 results for the NQ dataset to train our model. We re-implement the vast majority of retriever code, reproducing selected results from Tables 2 and 3 of the original paper. We also perform two additional experiments; one, we train the model with DistilBERT [**2**] instead of BERT-Base, and two, we explore a larger range of top-k accuracies than the paper for both models across both the development and test sets.

Since we did not have access to the same high-powered compute as the original paper, we downscaled the experiments to decrease our computational burden as mentioned above. We utilized a machine with with 4 NVIDIA GTX 1080 12GB GPUs, and our final experiments, including ablations, took roughly 3 days to produce.

Results

Overall, our results support the main claim of the original paper. We achieve roughly 60% top-20 and 70% top-100 recall in our experiments. Note that these numbers are still substantially less than what is described in the paper since our experimental setup attempts to reduce computational costs; see section 3.5 of this report for more information.

What was easy

It was easy to get the preprocessed data from the original authors' GitHub repository. This is quite important; if this was not the case, we would have had to re-implement BM25 before attempting to reproduce the main claim of the paper. Additionally, the architecture of the model translates to succinct code. It is important to note that we did *not* reproduce the end-to-end QA system, i.e., we did not implement the neural reader, so we did not verify if it is straightforward to implement one.

What was difficult

Implementing a `DistributedDataParallel` version of our model was much more time consuming than we expected. While this is necessary to get good performance when utilizing multiple GPUs, we found the API's documentation to be suboptimal–we spent a significant amount of time debugging subtle issues and re-training models.

Communication with original authors

We had regular (every week or so) contact with Sewon Min, one of the original authors of the paper.

1 Introduction

The goal of this paper is to demonstrate that a BERT-based encoder architecture can create dense representations of questions and passages that can be used for query based document retrieval. Traditional retrieval systems typically use sparse TF-IDF/BM25 based vectors lack the semantic richness of the representations produced by neural networks. Thus, the main finding of this paper is that BERT-based embeddings have the ability to yield representations that are more conducive to the task of passage retrieval when compared to more traditional approaches. The performance improvements are empirically demonstrated on several commonly used QA benchmarks.

2 Scope of Reproducibility

Previous state-of-the-art passage retrieval systems use TF-IDF/BM25 to rank and retrieve passages that are relevant to a search query, and the main claim of the paper is that we can do significantly better with neural encoders. We attempt to reproduce that main claim. In particular, we verify:

- "It is possible to create a state-of-the-art passage retrieval system by training a pair of encoders using contrastive loss."
- The single-training, DPR top-20 and top-100 accuracies of NQ's test set (in the middle block of Table 2)
- The G.+BM25 negatives with 31+32 in-batch examples' top-5, top-20, and top-100 accuracies of the NQ dataset's development set (in the bottom block of Table 3)

There is an **important caveat** to our results: our top-k retrieval accuracies are based on a *subset* of Wikipedia (roughly 1.6 million passages) while the paper's top-k retrieval accuracies in Tables 2 and 3 are based on the *entirety* of Wikipedia (roughly 21 million passages). Therefore, our numbers are *not* identically comparable, though they should be pretty similar. Furthermore, if we produce higher accuracies with a DPR model and a subset of Wikipedia than with a BM25 model on the entirety of Wikipedia, we will have successfully verified the main claim above since it has been observed that evaluating on a subset of Wikipedia leads to a decrease in recall by a few percentage points (see the bottom of the `README` of [**4**]). The subset selection was done using a script from repository for a project related to the original paper [**4**].

Additionally, the results in Table 2 in the original paper correspond to DPR models trained with batch size 128; therefore, our reproductions for these observations will again be slightly lower because we use batch size 32. See section 3.3 for more details.

3 Methodology

3.1 Model descriptions

The model used in the original paper consists of two separate BERT encoders, one for embedding the question and one for embedding the passage. They are optimized simultaneously with a contrastive loss function (described below).

The learning objective introduces a "similarity" score for the embedded question and passage vectors:

$$\mathrm{sim}(q,p) = E_Q(q)^T E_P(p)$$

q, p are tokenized question and passage vectors and $E_Q(\cdot), E_P(\cdot)$ are the question and passage encoders. In English, to compute the similarity score, we take the inner product of the embedded question and passage vectors. The overall contrastive loss can be expressed as

$$L(q_i, p_i^+, p_{i,1}^-, \ldots, p_{i,n}^-) = -\log \frac{e^{\mathrm{sim}(q_i, p_i^+)}}{e^{\mathrm{sim}(q_i, p_i^+)} + \sum_{j=1}^{n} e^{\mathrm{sim}(q_i, p_{i,j}^-)}}$$

The i index represents the ith question in our dataset and p_i^+ and p_i^- correspond to the embeddings of the positive and negative passages, respectively.

Every question is paired with a single positive sample and n negative samples (in the case of in-batch negative sampling, if k is our batch size, then we have $n = 2k - 1$ negative samples, one of which is a BM25 hard-negative). Thus, our objective function aims to maximize $\text{sim}(q_i, p_i^+)$ while minimizing $\text{sim}(q_i, p_{i,j}^-)$. As we can see, this is very familiar: our objective is simply the negative log softmax of all of our similarity scores within the batch.

Since our model is made up of two pre-trained BERT encoders that are fine-tuned, it is quite large–each BERT has 110 million parameters, so the DPR model has a total of 220 million parameters. DistilBERT, which we utilize for one of our additional experiments, has a total of 66 million parameters–that "distilled" model has a total of 132 million parameters.

3.2 Datasets

The models we train for this reproducibility report are all trained and evaluated on the Natural Questions (NQ) training set. NQ consists of questions from Google search queries and passages extracted from Wikipedia articles [**6**]. We obtain the data from the download script available in the original DPR repository [**3**]. In particular, the data is available in JSON format pre-labeled with BM25 scores. For each question, there are three different lists of passages: `positive_ctxs`, `negative_ctxs`, and `hard_negative_ctxs`. Two are relevant: positive contexts (passages) contain an answer string (also available in the JSON object; it is possible for a question to have more than one answer), and *hard* negative contexts are passages that are very lexically similar to the question, i.e. high BM25, but do not contain the answer.

In the case that there are no hard negative passages associated with a question, we choose a negative example from `negative_ctxs`. This happens only once or twice out of the tens of thousands training examples so its impact on our results is trivial.

Additionally, the datasets are pre-split into training, development, and test sets, each with the below cardinalities:

Table 1: Dataset Size

	Train	**Dev.**	**Test**
NQ	58,880	8,757	3,610

A few clarifications about our experimental setup and the structure of the datasets are necessary:

- JSON-formatted datasets are only available for the training and development sets, `nq-train.json` and `nq-dev.json`, respectively. This is because the BM25 metric for choosing negative examples does not matter for the official evaluation metric. Additionally, `nq-dev.json` is used for calculating the *pseudo-evaluation* metric during training; it is *not* used for performing the official evaluation (`nq-dev.csv` is used for that). See section 3.4 for more details.
- Due to the way PyTorch's `DistributedSampler` is utilized in our code, it is necessary for us to drop off the tail end of our development and test sets when performing the official evaluation; i.e., if our world size is 4 and we are performing the official evaluation on the test set, we must drop off the last two examples to make the dataset size divisible by 4 ($3610 \mod 4 = 2$). This is done via the `drop_last` flag in `DistributedSampler()`; see section 4.2 for more details.
- The original NQ training set has 79,168 examples, but after filtering, we are left with 58,880 examples. See the original DPR paper for more information.

Finally, for the official evaluation (see section 3.4 for more details), we utilized a subset of Wikipedia containing roughly 1.6 million passages along with the titles of the articles from which they came from [**4**]. This is substantially pared down compared to the full Wikipedia database the original DPR paper uses (roughly 21 million passages). This was necessary for the success of our reproduction since we have to perform a forward pass for each passage in the Wikipedia database. Therefore, our results are not directly comparable to the results shown in the paper.

3.3 Hyperparameters

We obtain the hyperparameters from the paper and the code release, and we did not conduct any additional hyperparameter searches. For quick reference, the important hyperparameters are:

- Adam optimizer with adam epsilon 1e-8
- Learning rate: 1e-5
- Dropout: 0.1

- Epochs: 40. In our communications, Sewon Min noted that while the pseudo-evaluation (see section 3.4) may stay stagnant at around 90% or so from epoch to epoch, later checkpoints generalize better for the official evaluation. Therefore, we perform the official evaluation on only the final model checkpoint. This is again an effort to decrease computational burden because we would otherwise have to create embeddings for the entire (subset of) Wikipedia database for each checkpoint.
- Batch size: 32. Batch size is very important for our experimental setup since we use in-batch negative sampling as described in the paper. Essentially, we use the positive and hard negative passages for all other questions in the batch as additional negative samples for a particular question. For example, if our batch size is 32, each question in the batch will have 1 positive passage and 63 negative passages–we then perform the log softmax operation described in section 3.1. For batch size 128, each question insteadhas 1 positive passage and 255 negative passages. Observe that as batch size increases, the task becomes "harder" to learn; however, it also becomes more similar to the open-domain task we attempt to solve.

  The paper experiments with both batch sizes 32 and 128. Unfortunately, we did not have access to GPUs with sufficient memory, so we were only able to perform experiments with batch size 32.

3.4 Implementation

We re-implemented the vast majority of the DPR model, available in our repository [**5**]; however, we did utilize the original DPR code release to assist in our implementation of cross-GPU in-batch negative sampling in `DistributedDataParallel` mode. In particular, we utilize slightly-modified versions of `dist_utils.py` and the `_calc_loss` function in `train_dense_encoder.py`. Additionally, we utilize `utils.py` for checkpointing which we obtained from a previous course (CSE 490G1/CSE 599 au20 taught by Joseph Redmon).

We use python. The `environment.yml` file lists nearly all dependencies and should work out-of-the-box with `conda`, but in the case that you do not want to use the environment file, the major dependencies are `Huggingface Transformers`, `PyTorch`, `pandas`, `h5py`, `FAISS` (CPU version is fine), `seaborn`, `matplotlib`, and optionally `wandb` (Weights and Biases).

We also provide ablation options: one for DistilBERT vs BERT, and one for varying the hard-negative sampling process (we currently just choose the "hardest" BM25 negative as our negative example). Note that the sampling process ablation was not tested due to compute constraints, though we are fairly confident it should work smoothly.

Pseudo-evaluation vs official evaluation Above, we mention two different types of evaluations: the pseudo-evaluation and the official evaluation. The *pseudo-evaluation* is the top-1 accuracy of the model when choosing the positive passage out of the pool of passages inside of the batch. So for example, in our experimental setup with batch size 32, the model is choosing the positive passage out of a pool of 64. In contrast, the *official evaluation* is the top-k recall of the model when choosing a positive passage out of the *entirety* of the Wikipedia subset. A success is counted i, after normalization, any of the top-k passages contain any of the answer strings associated with a question.

The reason for the pseudo-evaluation is that it is computationally infeasible to perform the official evaluation after every epoch since you have to embed the entirety of the Wikipedia database and all of the questions, per evaluation trial. Therefore, we use the pseudo-evaluation as a heuristic to observe while training.

3.5 Computational requirements

Since our experiments are not directly comparable to the experiments performed in the paper, we do not have directly-observed estimates for the amount of time the original setup takes per experiment. Similarly, we do not have estimates for the average runtime per epoch, the total number of trials, and total number of GPU hours. However, Sewon Min stated that that a single experiment took roughly 18 hours to perform on a machine with 8 32GB GPUs (this is over the entire Wikipedia database, not the subset). Additionally, the paper states that they spent around 40 epochs training the DPR model on large datasets such as NQ.

All experiments presented in this report were performed utilizing 4 NVIDIA GTX 1080 12GB GPUs for both training and evaluation. Again, note that this is not enough compute for the experiments presented in the original paper that use a batch size of 128 and embed the full Wikipedia database. The average runtime per epoch was roughly 45 minutes for the BERT version and 30 minutes for the DistilBERT version. Our final results ran a single trial for each of BERT and DistilBERT, and we utilized roughly 5-7 days of GPU time per experiment. Similar to the original paper, each experiment trained for 40 epochs. Overall, our actual compute used roughly matched our expectations from before we ran the experiments.

As we can see, the paper is very computationally expensive, and there are less-visible factors such as hyperparameter search that further increase the total computational cost of the original paper past the above estimations. Additionally, we only reproduce the results reported for the retrieval portion of a question answering system. The authors of the paper also implemented an end-to-end question answering system which includes implementing and training a neural reader to extract answer spans.

Since the neural passage retrieval system is the main contribution of the paper, we are able to significantly reduce our computational burden while still retaining the value of our reproduction. Specifically, we:

- Utilize BM25-preprocessed data;
- Do not perform hyperparameter search;
- Reproduce models based on batch size 32 instead of 128;
- Perform an additional experiment with a lighter-weight BERT model, DistilBERT;
- Omit implementation of the end-to-end QA system;
- Train and evaluate only on NQ instead of a variety of datasets;
- Generate dense embeddings for a subset of Wikipedia rather than its entirety.

Table 2: Experimental Results

	$k = 1$	$k = 5$	$k = 20$	$k = 100$	$k = 500$	$k = 1000$
BERT-Dev.	24.3%	44.45%	58.86%	69.67%	76.40%	78.73%
BERT-Test	24.7%	45.59%	60.48%	71.09%	78.16%	80.59%
DistilBERT-Dev.	26.50%	49.09%	62.44%	72.11%	78.27%	80.43%
DistilBERT-Test	27.63%	50.01%	63.88%	73.50%	79.99%	82.31%

3.6 Results

Overall, our results support the main claim and statistics we sought out to reproduce, stated in section 2.1. We reiterate that our results are not directly comparable to the original paper's due to differences in Wikipedia databases and batch sizes, though as noted, they should be considered underestimates of true performance.

First, since our top-k accuracy seems to at least match the top-k accuracy for BM25 in the original paper (in Figure 1 for [**1**]), we have successfully verified that it is possible to train a neural passage retrieval system that matches traditional retrieval systems. In fact, in the case of DistilBERT, we show that such retrieval systems can outperform BM25–see section 3.8.2 for more details. Furthermore, the BM25 recall metric in the aforementioned figure is on the *full* Wikipedia database whereas our metric is on a *subset*; we hypothesize that our reproductions would actually outperform BM25 if we evaluated on the full Wikipedia database, as previous experiments have demonstrated [**4**].

Second, rows 2 and 4 of Table 2 correspond to the respective metrics in the middle block of Table 2 in the original paper. We hypothesize the ~15 point performance decrease is mainly due to a smaller batch size and different Wikipedia database.

Finally, rows 1 and 3 correspond to the respective metrics in the bottom block of Table 3 in the original paper. We hypothesize the ~15 point performance decrease is mainly due to a different Wikipedia database.

3.7 Additional results not present in the original paper

3.7.1 Wide-range Top-k Recall

The original paper only includes results for $k = 20, 100$ in top-k accuracy. We augment these results by testing additional values for k as shown in Table 2 above. Interestingly, as we increase k, we see that performance saturates. This could indicate that when DPR fails to retrieve relevant passages, it fails quite badly.

3.7.2 DistilBERT Backbone Architecture

DistilBERT is a compressed version of BERT that has ~40% less parameters and is ~60% more efficient than the original model while only losing a few percentage points of accuracy [**2**]. Thus, we test if it is possible cut down on the computational requirements at both training and inference time while maintaining the original performance by replacing the BERT encoders with DistilBERTs. The results are shown in Table 2. Surprisingly, we see that the DistilBERT architecture consistently outperforms the BERT architecture. We hypothesize two potential explanations:

- Our BERT DPR reproduction is overfitting due to overparameterization–we see in both the DistilBERT and BERT training loops that losses approach 0 after 15 or so epochs.
- Our BERT DPR reproduction is undertrained. Since BERT has been shown to perform better than DistilBERT on a variety of benchmarks, it may have higher capacity and thus take longer to optimize. We train both DistilBERT and BERT versions of DPR for 40 epochs.

Nevertheless, we have demonstrated that it is indeed possible to achieve nearly identical (or perhaps even better) passage recall performance with an appropriate substitution of “distilled” encoder.

4 Discussion

We believe the evidence produced in our reproduction fully supports the claims made in the original paper. Our reproduction is valuable because it produces great retriever results at a significantly-reduced computational cost. Additionally, our success with DistilBERT further demonstrates that the dot-product similarity metric is indeed a viable way to retrieve passages in the open-domain QA setting. However, our focus on reducing computational cost could also be seen as a weakness: we perform substantially fewer experiments than the original paper, so our results can reasonably be interpreted as less robust.

4.1 What was easy

It was easy to get the preprocessed data from the original authors’ GitHub repository. This is quite important; if this was not the case, we would have had to re-implement BM25 before attempting to reproduce the main claim of the paper. Additionally, the architecture of the model and its evaluation both translate to succinct code, minus the `DistributedDataParallel` portion. FAISS deserves a special shoutout–it works incredibly fast and the CPU version is simple to set up and use. Finally, the model is intuitive. At a high level, it seems reasonable to use an encoder to encode questions, another encoder to encode passages, and then optimize to find the most relevant question-passage pairs. The objective and training scenario for the retriever is “friendly” as well: the dot product is a very familiar operation, and the loss function is similar to the softmax function which is also quite familiar.

Ultimately, if you have a need to retrieve relevant passages to questions in the open-domain setting and are familiar with `DistributedDataParallel`, then re-implementation of the paper should be doable.

4.2 What was difficult

Utilizing PyTorch’s `DistributedDataParallel` was by far the most difficult part of implementing the paper. This module is necessary to ensure that we can train the model on multiple GPUs in the most efficient way possible. When implementing, we encountered four significant obstacles:

1. As newcomers to larger-scale distributed training, we found the documentation of `DistributedDataParallel` and related classes in PyTorch to be suboptimal and confusing. Our unfamiliarity with distributed behavior led to a few significant setbacks, described below.
2. When we first wrapped our model with DDP, we were quite lost and spent several hours looking for a good tutorial. While we found a great tutorial that guided us through most of the process [**7**], we still had obstacles

to smooth out. For example, due to memory constraints, each GPU could only fit 8 questions (and thus 16 passages). However, since our desired batch size was 32, each question would need to be evaluated versus 63 negative examples–this meant mini-batches of passages from each GPU needed to be pooled globally. This cross-GPU communication was difficult to figure out and we ultimately solved the problem by slightly repurposing methods from the original DPR code release (see section 3.4).

3. Our first run of the model produced terrible results, getting us 24% top-20 and 33% top-100 recall. Questioning our implementation, we found out that `DistributedSamplers` do not shuffle data for you, from epoch to epoch, unless you call the `set_epoch()` method. In other tasks such as classification tasks, this may not be a big deal; however, since our task is contrastive, it was severely hurting our ability to generalize–in short, for a particular epoch, each question in the training data would be paired with the exact same in-batch negatives it was paired with for all other epochs.
4. Once we fixed this bug and re-trained, our model ended up performing even worse than before, getting us 6% top-20 and 12% top-100 recall. It was completely counterintuitive that we get worse performance when we fix our training. Again questioning our implementation, we found another bug–if you do not set the flag `drop_last` as false, a `DistributedSampler` will sample data as if the data points are looped; that is, some data points at the beginning of a dataset end up being sampled twice (see section 3.2). To solve some of the issues we encounter in distributed training, we save an embedding file per GPU when generating embeddings and load them in and re-sort them by index when we perform the official evaluation. This process in combination with the twice-sampled data points ended up causing off-by-one/two/three errors in both our question and passage similarity searches. Once we fixed this and re-embedded our questions and passages, we ended up with our final results.

Note that simply looking at the original DPR code release was not a possible solution. The original authors do not use PyTorch's built-in `DistributedSampler` and instead implement their own `DataIterator`.

4.3 Recommendations for reproducibility

The main barrier to reproducibility in this area is computational cost, and authors in this field should prioritize making these models more computationally accessible. In our reproduction, we attempted to cut down on computational cost as much as possible, and we *still* had to commit a large amount of compute to reproduce a fraction of the results (and they aren't even directly comparable). This is true for other areas of NLP as well; state-of-the-art improvements in natural language processing seem to heavily depend on the amount of computational resources available for training. As this trend continues, reproducibility becomes less and less feasible.

If the reader plans on reproducing this paper as well, instructions for reproducing our experimental results discussed in this report are extensively documented in our repository. If you plan on re-implementing yourself without the help of our code release or the original DPR code release, make sure to carefully study documentation of your dependencies, especially PyTorch–there are many subtleties that you should understand before committing large amounts of compute to a training or evaluation loop. [**5**]

Communication with original authors

We had regular (roughly weekly) contact with Sewon Min, one of the original authors of the paper.

5 References

[1] Vladimir Karpukhin Barlas Oğuz Sewon Min Patrick Lewis Ledell Wu Sergey Edunov Danqi Chen Wen-tau Yih. (2020) Dense Passage Retrieval for Open-Domain Question Answering. *The 2020 Conference on Empirical Methods in Natural Language Processing*,

[2] Victor Sanh Lysandre Debut Julien Chaumond Thomas Wolf. (2019) DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter. *Advances in Neural Information Processing Systems 33*

[3] Dense Passage Retrieval, `https://github.com/facebookresearch/DPR`

[4] Efficient Open-Domain Question Answering, `https://github.com/efficientqa/retrieval-based-baselines`

[5] CSE 517 Final Project, `https://github.com/peminguyen/CSE517-final-project`

[6] Google's Natural Question Dataset, `https://ai.google.com/research/NaturalQuestions`

[7] Distributed data parallel training in Pytorch by Kevin Yang