Unsupervised Cross-Task Generalization via Retrieval Augmentation

Anonymous ACL submission

Abstract

Prior work shows that using natural-language instructions to convert data of diverse NLP tasks can benefit massive multi-task models, which show great performance in unsupervised cross-task generalization. In this paper, we present a retrieval augmentation method for further improving such models by a dense retriever with reranking. The proposed method achieves significant improvement without using any training data of unseen tasks but only a few unlabeled examples. We will make our code and models publicly available.

1 Introduction

Recent advances in natural language processing (NLP) have shown that it is promising to train a massive multi-task NLP model by formatting data of different tasks with a shared text-to-text format, especially when the formatting templates include natural-language instructions (Ye et al., 2021; Wei et al., 2021; Sanh et al., 2021). Such massive multi-task models have shown surprisingly great performance on unseen tasks with limited or no training data. These prior works make it easier to develop techniques for efficient cross-task generalization of massive pre-trained NLP models.

For example, Ye et al. (2021) proposed CrossFit, which is a general evaluation protocol for building and evaluating such massive multi-task models in NLP. This training recipe has two stages as follows. In the upstream learning stage, we use a mixture of data from many different NLP tasks together to train a text-to-text Transformer language model (LM), e.g., BART (Lewis et al., 2020) and T5 (Raffel et al., 2020), thus creating an upstream model. In the generalization stage, we are given an unseen task and use a few labeled data points to fine-tune the upstream model so that it can quickly generalize to this unseen task. The models trained with the CrossFit pipeline show that this kind of multi-task upstream learning can indeed generalize to unseen NLP tasks (i.e., cross-task generalization).

However, it requires training instances for the target task during cross-task generalization because the templates introduced in CrossFit have task names as the hard prefixes. For example, an instance of a sentiment analysis task can be "`amazon-review:` *best cast iron skillet you will every buy.*" → "positive." Thus, it is necessary to fine-tune upstream models with at least a few examples where the prefixes are the target task names, so that the models can generalize to this unseen task.

In order to eliminate the need of training instances of unseen tasks for cross-task generalization, FLAN (Wei et al., 2021) and T0 (Sanh et al., 2021) add natural language (NL) instructions to the formatting templates used in the upstream learning stage. For example, the T0 models (Sanh et al., 2021) that are trained with data converted by the PromptSource (Bach et al., 2022) templates. These natural-language instructions make unsupervised cross-task generalization more promising because such instructions are more likely to be generalized to unseen yet similar tasks.

We study how to further improve such NL-instructed multi-task LMs for unsupervised cross-task generalization via retrieval augmentation. Experiments show that our proposed method achieves significant improvement over the base models without using any training data of the unseen tasks but only a few unlabeled examples.

2 Problem Formulation

We here formulate the problem of *unsupervised* cross-task generalization of a massive multi-task language model and point out its challenges.

Massive Multi-Task Language Models. We assume there are N different upstream tasks, $\{\mathcal{T}_1, \ldots, \mathcal{T}_N\}$. We use D to denote the collection of all labeled data (i.e., the **upstream data**) for these upstream tasks, which are used for training a massive multi-task model $\mathcal{M}$. The datasets of these upstream tasks are all converted to a shared *text-to-text* format by using natural-language instruction

templates such as PromptSource (Bach et al., 2022) for reformatting data of different tasks.

Unsupervised Cross-Task Generalization. Let us consider a collection of M unseen tasks that are not included in the above upstream tasks, and we denote them with $\{\mathcal{U}_1, \ldots, \mathcal{U}_M\}$. For each unseen task $\mathcal{U}_i$, we have only a small number of ***unlabeled*** examples, and our objective is to use these unlabeled data to improve the general performance of upstream model $\mathcal{M}$ for the task $\mathcal{U}_i$.

Specifically, we consider two sets of examples for the task $\mathcal{U}_i$: 1) a small[1] unlabeled *query set* Q_i and 2) a large labeled *test set* E_i, where $|Q_i| \ll |E_i|$. An unsupervised cross-task generalization method f thus needs to enhance $\mathcal{M}$ (by only using the inputs from Q_i) so that the updated model $\mathcal{M}_i$ can perform better on E_i. To get a comprehensive assessment for a method, we look at the overall performance of all selected unseen tasks.

Challenges. Unlike the formulation of few-shot learning for cross-task generalization presented in the CrossFit framework (Ye et al., 2021), ours is more challenging due to the lack of human-annotated supervision on the query set, leading to a *zero-shot* learning paradigm. To enhance upstream multi-task LMs for a new task U_i, one may want to use the query examples Q_i to re-weight the upstream data $\mathcal{D}$ and then re-train a new model dedicated for the target task U_i. However, such re-training methods are hardly feasible in practice because they are overly time-consuming, especially when we have a large number of target tasks (i.e., M is a large number) to generalize.

3 Retrieval Augmentation

We introduce a simple and general retrieval-augmentation framework for unsupervised cross-task generalization, which consists of two key components: a dense retriever and a re-ranking module. We first discuss the motivation as well as the overview of the proposed framework, then show the details of each component, and finally discuss the potential extensions.

3.1 Overview

Recall that our goal is to improve the upstream multi-task model $\mathcal{M}$ on an unseen task $\mathcal{U}$ by using its unlabelled query examples Q. Our key motivation is to retrieve a small subset of upstream data $R \subset D$ and then fine-tune $\mathcal{M}$ with them to create a dedicated model $\mathcal{M}'$ for future inference on the task $\mathcal{U}$. The retrieved examples R should be highly related to the query examples Q that reveal the skills required by the task $\mathcal{U}$. We believe re-learning these examples R can help us better generalize $\mathcal{M}$ to the unseen task $\mathcal{U}$, although these retrieved data has been already mixed with other data and then used for training $\mathcal{M}$ before.

To efficiently retrieve such useful R for re-learning given the query examples Q, we present a two-stage retrieval-augmentation framework. In the first stage, we build a dense index of all upstream examples, where each upstream example in D is encoded by a dense vector. Then, we can build query vectors by encoding query examples Q for quickly retrieving the top-K most relevant examples via *maximum inner product search* (MIPS). In the second stage, we want to re-rank the initial retrieved examples with a more computationally expensive module that can further improve the quality of the final retrieval results. In the following subsections, we show the details of these two stages and present several design choices which we evaluate extensively in our experiments (Sec. 4).

3.2 Dense Retriever

Efficient indexing. Considering the fact that the size of the upstream data D can be extremely large, we must ensure that the retrieval step is efficient. Therefore, we pre-compute a dense index by encoding each example $(x, y) \in D$ with a d-dimensional vector $\mathbf{x} \in \mathbb{R}^d$, so that we can get the matrix $\mathbf{D} \in \mathbb{R}^{|D| \times d}$ as the index of all upstream examples. Given a query set Q, we encode every example inside it to get a set of query vectors for retrieval augmentation on the fly. Then, we use these vectors to search the closest upstream examples over the index $\mathbf{D}$ via MIPS, producing a set of retrieved data that can be used as either the final R or as a candidate set which can be further pruned by re-ranking. We use the FAISS library (Johnson et al., 2019) to build and query these dense indices.

Example encoding. The example encoder is the key of the above dense-retrieval pipeline where we use it to encode both upstream examples in D and the query examples in Q. As we want to use MIPS to enable efficient retrieval, a simple and effective strategy is to use the same encoder function for both D and Q such that a simple L2 distance metric can work effectively. We propose to use the upstream model $\mathcal{M}$ for encoding the examples, and we also show other reasonable designs in our evaluation section. Without loss of generality, let us assume $\mathcal{M}$ to be a text-to-text Transformer that

[1] We use $|Q_i|$=16 query examples in our main experiments.

has multiple layers for its encoder and decoder, such as BART (Lewis et al., 2020) and T5 (Raffel et al., 2020). We encode an example by first collecting the hidden representation of each token in the input at the last encoder layer, and applying mean pooling to get a single vector to represent this example.

Retrieval aggregation. Note that our target size of retrieved data is $|R|$ and we have $|Q|$ query examples. To retrieve $|R|$ examples, we search for the top-K examples for each query example, where $K = \lceil \frac{|R|}{|Q|} \rceil$, and then take the $|R|$ of them when $K|Q| > |R|$. Our results have shown that this method is more effective than other strategies, such as combining the distance scores generated for each query example. Note that by retrieving the top-K examples, we may repeat examples that are close to multiple query vectors. This effect is desirable because it allows us to naturally over-weight especially relevant upstream examples for re-learning.

3.3 Reranking Module

The dense retrieval module allows us to efficiently retrieve a subset of possibly-relevant examples from the upstream data D, at the cost of not permitting expensive processing of all possible query/candidate pairs, which would be $|Q| * |D|$ and too huge to compute. However, such an efficient dense-retrieval method might not benefit from a more careful assessment. To mitigate the lack of expensive processing, we present a re-ranking module that is based on a cross-encoder and learns to score a pair of query examples and candidate upstream examples. We leave the details of creating data for training such a reranker in Sec. A.

Encoding example pairs. The cross-encoder architecture has been widely used in sentence-pair classification tasks such as natural language inference and paraphrase detection. We here want to use a cross-encoder to encode the concatenation of a query example and a candidate upstream example, which is retrieved by the dense retriever in the first stage. Specifically, we fine-tune a RoBERTa (Liu et al., 2019) to classify whether a pair is a positive or negative match. The confidence of classifying such a pair to be positive is thus score of the candidate upstream example for this query example. On top of this, we can then develop a reranking module for further improvement.

Scoring paired data. To rerank the initially retrieved data from the dense retriever, we propose to apply the cross-encoder over all pairs of query examples Q and candidate retrieved examples R, thus producing scores for all $|Q| * |R|$ example-pairs. Then, we use the average of all scores involving a candidate retrieved example as its new score. Finally, we can take the new top-K examples based on this new ranking of the examples in R as the final retrieved data R'. Note that we call the ratio between R' and R the upsampling ratio μ.

3.4 Fine-Tuning with Retrieved Data

When we have the final retrieved data R_i for a certain query set Q_i, we can now enhance the upstream model $\mathcal{M}$ for the unseen task $\mathcal{U}_i$. To this end, we simply use a smaller learning rate to continually fine-tune $\mathcal{M}$ on the R_i for a few epochs.

4 Evaluation

4.1 Task Distribution

We follow Sanh et al. (2021) to use the templates from PromptSource (Bach et al., 2022) for converting data of different types of NLP tasks. In total, we have 36 upstream tasks and 10 target unseen tasks, where the upstream tasks are the same as the ones that the T0 model used. When we apply the natural-language templates for the test examples, we only keep the templates that can be evaluated with exact-match (e.g., classification, question answering, etc.) so that it is easier to use exact match (EM) and its relaxed version that allows mutual inclusion, SoftEM, as the metrics.

4.2 Upstream Learning

The released T0 model checkpoints are too large to be effectively fine-tuned on popular affordable GPUs. To improve the efficiency and keep the generality of our study on the retrieval augmentation methods, we propose to train a parameter-efficient alternative. Thus, we fine-tune a BART-large (0.4 billion parameters) following the recipe of T0 models. Specifically, we sample 50k examples at most from each upstream task to build a large upstream dataset consisting of 1.7 million examples (i.e., $|D| = 1.7$m), then use it to fine-tune a BART-large as $\mathcal{M}$, and call it **BART0**.

4.3 Setup and Configurations

In our main experiments, we use $|Q_i|$ = 16 query examples for each unseen task $\mathcal{U}_i$ and retrieve $|R_i|$ = 512 examples for augmenting BART0 to achieve better zero-shot performance. In the retrieval-augmentation stage, we use a learning rate of 1e-6 and a batch size of 4 to continually fine-tune all layers of BART0 for 2 epochs. As for re-ranking,

Task	T0-3B	BART0	Random	+ Rerank	SBERT	+ Rerank	BART0$_{\text{ENC}}$	+ Rerank
arc-challenge	41.30	35.70	$33.28_{\pm 1.50}$	$32.34_{\pm 4.34}$	$37.90_{\pm 1.22}$	$36.78_{\pm 2.05}$	$37.78_{\pm 0.73}$	$38.56_{\pm 0.91}$
anli_r3	26.00	30.50	$35.34_{\pm 1.52}$	$34.58_{\pm 1.47}$	$32.64_{\pm 2.53}$	$35.08_{\pm 2.44}$	$36.70_{\pm 0.53}$	$37.08_{\pm 0.54}$
hellaswag	34.40	39.40	$33.84_{\pm 5.59}$	$34.72_{\pm 7.36}$	$30.92_{\pm 7.82}$	$36.12_{\pm 5.05}$	$44.36_{\pm 3.07}$	$47.06_{\pm 3.94}$
openbookqa	38.50	34.40	$28.72_{\pm 2.46}$	$33.16_{\pm 3.07}$	$33.28_{\pm 1.24}$	$36.34_{\pm 1.04}$	$36.98_{\pm 1.55}$	$36.06_{\pm 1.90}$
piqa	45.30	36.10	$37.00_{\pm 2.71}$	$40.14_{\pm 2.25}$	$38.54_{\pm 2.17}$	$39.34_{\pm 1.90}$	$41.34_{\pm 1.75}$	$41.42_{\pm 2.85}$
squad_v2	30.60	32.40	$29.86_{\pm 5.46}$	$30.52_{\pm 2.82}$	$29.46_{\pm 0.84}$	$29.82_{\pm 0.77}$	$30.26_{\pm 1.54}$	$30.64_{\pm 1.45}$
cb	53.93	39.64	$47.07_{\pm 1.25}$	$44.57_{\pm 2.49}$	$48.00_{\pm 3.28}$	$47.50_{\pm 3.08}$	$44.50_{\pm 4.20}$	$41.57_{\pm 2.87}$
wic	45.70	46.70	$41.04_{\pm 2.18}$	$42.38_{\pm 8.76}$	$46.78_{\pm 2.22}$	$45.64_{\pm 2.03}$	$49.90_{\pm 0.50}$	$49.72_{\pm 1.34}$
wsc	50.00	57.88	$52.50_{\pm 2.29}$	$51.54_{\pm 4.01}$	$52.69_{\pm 6.13}$	$54.73_{\pm 2.55}$	$59.27_{\pm 1.96}$	$60.12_{\pm 1.19}$
winogrande_xl	47.60	51.10	$52.68_{\pm 0.83}$	$54.34_{\pm 2.30}$	$52.18_{\pm 3.20}$	$52.94_{\pm 1.94}$	$54.60_{\pm 1.35}$	$54.34_{\pm 1.83}$
Overall (**mean$_{\pm\text{std}}$**)	*41.33*	*40.38*	$39.13_{\pm 2.06}$	$39.83_{\pm 1.35}$	$40.24_{\pm 1.61}$	*41.43*$_{\pm 0.72}$	*43.57*$_{\pm 0.68}$	***43.66***$_{\pm 0.56}$
Overall (**max**)	↑	↑	*40.59*	*41.41*	*41.76*	*42.30*	***44.51***	*44.34*
Overall (**min**)	↑	↑	35.66	37.76	38.28	*40.40*	*42.65*	***42.95***
Overall (**median**)	↑	↑	39.93	40.25	*40.91*	*41.28*	*43.43*	***43.72***

Table 1: The experimental results (in SoftEM %) for comparing retrieval augmentation methods. T0-3B is about 18x larger than our BART0 (the base model for other methods), while their results are comparable. We highlight the base performance (*40.38*) and the results that are better than it. Each result here in the upper section is the average (and the std) performance of using 5 different query sets for a task. The lower section reports the mean, max, min, and median of the overall performance.

we set the upsampling ratio $\mu = 2$, meaning that we will first retrieve 1024 examples for reranking and use the top 512 as the final retrieved data. For each task $\mathcal{U}_i$, we use 5 different query sets, i.e., $\{Q_i^{(1)}, \ldots, Q_i^{(5)}\}$, and report their average as well as the std. Then, we take the average of all tasks as the overall performance. Finally, we report the mean, max, min, and median over such 5 rounds for comprehensively assessing different retrieval-augmentation methods. These experimental results are summarized in Table 1.

4.4 Experimental Results

BART0 vs T0-3B. As we mentioned before, we find that BART0 is surprisingly comparable with the much larger T0-3B in terms of their zero-shot performance on these unseen tasks. We use BART0 as our base model for testing different retrieval-augmentation methods and thus its overall performance *40.38* is what we want to compare with.

Random Retrieval. The *Random* column shows the results when we randomly sample R_i from the upstream data D without using any information from Q_i. From the mean and median, we can see that such random retrieval does not produce better performance, which is expected. However, the max performance (from the 5 rounds) is better than the base model, especially with reranking (*41.41*). This suggests that it is promising to study better retrieval methods for more consistent improvement.

SBERT & BART0$_{\text{ENC}}$. We here use Sentence-BERT (Reimers and Gurevych, 2019) (SBERT) as a strong baseline method to create a dense index of the upstream data, compared with our proposed method (BART0$_{\text{ENC}}$) — using the encoder of the upstream model (i.e., BART0) to induce the upstream index. We can see that BART0$_{\text{ENC}}$ always outperforms other methods, no matter if we employ the reranking module or not. Even its minimum performance in the five rounds (*42.65*) is better than the maximum of the SBERT+Rerank (*42.30*). Besides, the standard deviation also becomes smaller which means that BART0$_{\text{ENC}}$ produces more consistently better results across different query sets. The SBERT indexing relies mainly on the semantic similarities between the query example and upstream examples. On the other hand, our proposed BART0$_{\text{ENC}}$ uses the hidden representations inside the upstream model $\mathcal{M}$ for representing examples. We believe using such a method can help us find the examples that share similar reasoning skills that are learned by the upstream model.

Reranking. Table 1 presents the results by adding the reranking module in the three groups with different retrievers. We find that reranking can indeed further improve the performance on multiple dimensions (mean, min, median, max), where the only exception is that the maximum performance of BART0$_{\text{ENC}}$ is slightly worse with the reranking. Also, we find that the standard deviations across different $Q_i^{(j)}$ are significantly reduced when we add the reranking step, e.g., 2.06→1.35 for Random and 1.61→0.72 for SBERT.

5 Conclusion

We show that it is promising to develop retrieval-augmentation methods for improving natural language-instructed NLP models to achieve better cross-task generalization without any training data. Our proposed dense retrieval with a reranking pipeline can produce a significant improvement, even with simple indexing methods. We look forward to more exciting future work in this direction.

References

Stephen H. Bach, Victor Sanh, Zheng-Xin Yong, Albert Webson, Colin Raffel, Nihal V. Nayak, Abheesht Sharma, Taewoon Kim, M Saiful Bari, Thibault Fevry, Zaid Alyafeai, Manan Dey, Andrea Santilli, Zhiqing Sun, Srulik Ben-David, Canwen Xu, Gunjan Chhablani, Han Wang, Jason Alan Fries, Maged S. Al-shaibani, Shanya Sharma, Urmish Thakker, Khalid Almubarak, Xiangru Tang, Xiangru Tang, Mike Tian-Jian Jiang, and Alexander M. Rush. 2022. Promptsource: An integrated development environment and repository for natural language prompts.

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

Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Veselin Stoyanov, and Luke Zettlemoyer. 2020. BART: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 7871–7880, Online. Association for Computational Linguistics.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. Roberta: A robustly optimized bert pretraining approach. *ArXiv preprint*, abs/1907.11692.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research*, 21(140):1–67.

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 3982–3992, Hong Kong, China. Association for Computational Linguistics.

Victor Sanh, Albert Webson, Colin Raffel, Stephen H. Bach, Lintang Sutawika, Zaid Alyafeai, Antoine Chaffin, Arnaud Stiegler, Teven Le Scao, Arun Raja, Manan Dey, M Saiful Bari, Canwen Xu, Urmish Thakker, Shanya Sharma Sharma, Eliza Szczechla, Taewoon Kim, Gunjan Chhablani, Nihal Nayak, Debajyoti Datta, Jonathan Chang, Mike Tian-Jian Jiang, Han Wang, Matteo Manica, Sheng Shen, Zheng Xin Yong, Harshit Pandey, Rachel Bawden, Thomas Wang, Trishala Neeraj, Jos Rozen, Abheesht Sharma, Andrea Santilli, Thibault Fevry, Jason Alan Fries, Ryan Teehan, Stella Biderman, Leo Gao, Tali Bers, Thomas Wolf, and Alexander M. Rush. 2021. Multitask prompted training enables zero-shot task generalization.

Jason Wei, Maarten Bosma, Vincent Zhao, Kelvin Guu, Adams Wei Yu, Brian Lester, Nan Du, Andrew M. Dai, and Quoc V. Le. 2021. Finetuned language models are zero-shot learners. *ArXiv preprint*, abs/2109.01652.

Qinyuan Ye, Bill Yuchen Lin, and Xiang Ren. 2021. CrossFit: A few-shot learning challenge for cross-task generalization in NLP. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 7163–7189, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

A Distant Supervision for Reranking

To train a reranker scoring a candidate upstream example for a given query example, we have to get some kind of supervision. Note that, in this stage, we know nothing about the unseen tasks and what we can access is only the upstream data D. To learn such a reranker only from D, we propose to create distant supervision based on the dense retriever (Sec. 3.2). We define a data point of distant supervision as a tuple $Z = (Z_q, Z_p, Z_n)$: 1) Z_q is a set of query examples; 2) Z_p is the set of positive examples; 3) Z_n is the set of negative examples. We expect that Z_p are more suitable than Z_n as the retrieved data if Z_q would be a query set.

To this end, we first randomly sample an upstream task $\mathcal{T}_i$ and use a small subset of its training data as the Z_q. Then, we follow the procedure introduced before to apply the dense retriever using Z_q as the query examples, and get its initial retrieved data dubbed as R_Z. This R_Z will thus be the candidate pool where we create Z_p and Z_n. That is, $Z_p \subset R_Z$ and $Z_n \subset R_Z$. Note that we need to discard the examples in R_Z that are from the tasks which are similar to the selected $\mathcal{T}_i$, so that the generated tuples are closer to the scenarios where we use the reranker for the query sets of unseen tasks.

Our criteria to select Z_p and Z_n from R_Z is motivated by the hypothesis that a more suitable set retrieved examples should improve the performance $\mathcal{M}$ on Z_q after fine-tuning with it. Therefore, we keep sampling a small subset of R_Z and fine-tune $\mathcal{M}$ on it, and then use the fine-tuned model to evaluate on Z_q. The performance of such a temporarily fine-tuned model can be seen as a score telling us how well this subset can benefit Z_q. Through multiple rounds of such sample-train-test procedures, we can thus score each example in R_Z by taking the average of all test results that it has been involved. With such a new ranking of examples in R_Z, we take the top ones as Z_p and bottom ones as the Z_n, because the ones with higher scores are more likely to benefit $\mathcal{M}$ for the task represented by Z_q.