OpenMatch

Release v1.0

OpenMatch Developers

Sep 04, 2021

GET STARTED

CHAPTER

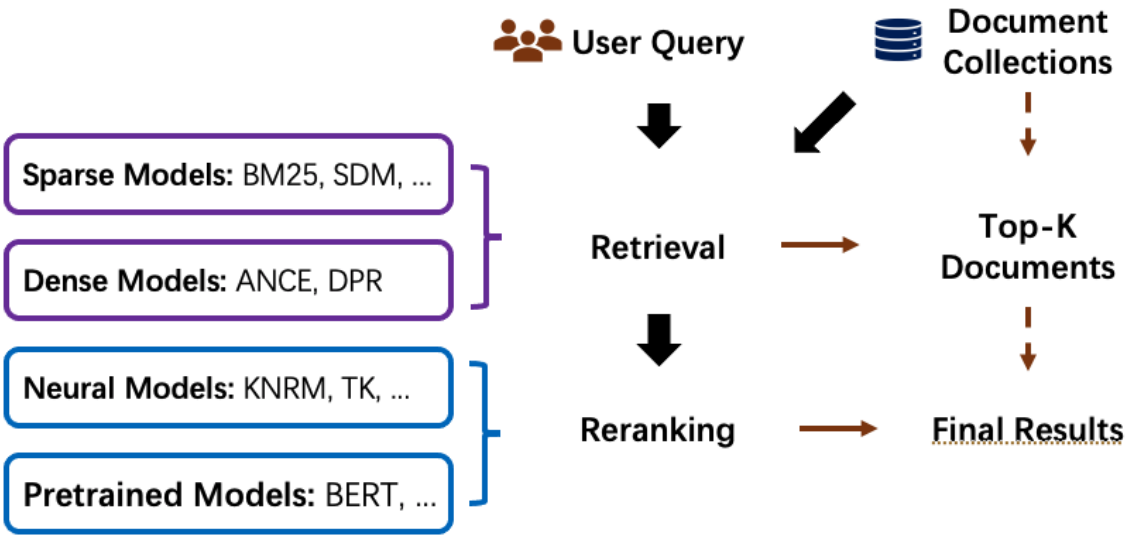
ONE

INTRODUCTION

OpenMatch (https://github.com/thunlp/OpenMatch) is an open-source and extensible library that provides a unified framework to implement information retrieval (IR) models.

1.1 IR Pipelines

Information Retrieval (IR) usually employs a two-step pipeline to retrieve documents from a large-scale document collection. OpenMatch provides several pre-defined modules and helps users build an effective IR system with retrieval and reranking steps.



In the retrieval stage, we can use **sparse retrieval models** or **dense retrieval models** to retrieve candidate documents from a large-scale document collection. Then, in the reranking stage, we can use more sophisticated **Neural IR models** to rerank the retrieved document candidates to achieve better ranking performance.

Sparse Retrieval is defined as a sparse bag-of-words retrieval model. These models match terms or phrases of queries and documents with exact matches.

Dense Retrieval conducts retrieval by encoding documents and queries into low-dimensional dense vectors. Then dense retrieval models effectively retrieve the candidate documents that are nearest to the query embedding. They usually use Faiss, a library for efficient similarity search and clustering of dense vectors, uring retrieval. Dense retrieval models are also representation-based Neural IR models and they usually use the pre-trained language models to encode both queries and documents.

Neural IR Models uses neural networks as rankers to rerank documents. Neural Information Retrieval (Neu-IR) models leverage neural networks to conduct semantic matches and can be categorized as representation-based ones and interaction-based ones. Representation-based methods encode the representation of query and document separately into two distributed representations. Interaction-based methods model the fine-grained interactions between query and documents, often by the translation matrices between all query and document term pairs. The interaction-based Neu-IR models are usually used in the reranking stage. Recently, the pre-trained language models are also widely used to implement pre-trained IR models, which are also interaction-based ones.

1.2 OpenMatch Solutions

OpenMatch first inherits Anserini and ANCE to build the document index for sparse retrieval and dense retrieval to efficiently search candidate documents from a large-scale document collection.

Then OpenMatch provides the following models to calculate ranking features for reranking and IR baseline implementation.

Components	Models
Sparse Retriever	Boolean AND/OR, LM, BM25, SDM, TF-IDF, Cosine Similarity, Coordinate match, Dirichlet LM ...
Dense Retriever	DPR, ANCE, ConvDR
Neural Ranker	K-NRM, Conv-KNRM, TK, BERT, RoBERTa, ELECTRA ...
LeToR	Coordinate Ascent, RankNet, LambdaMART, Random Forests ...

Finally, OpenMatch uses **Feature Ensemble** (Learning to rank, LeToR) to fuse ranking features from ranking models of both retrieval and reranking stages to further improve model performance. We can use RankLib and RankSVM toolkits to implement ranking feature ensemble.

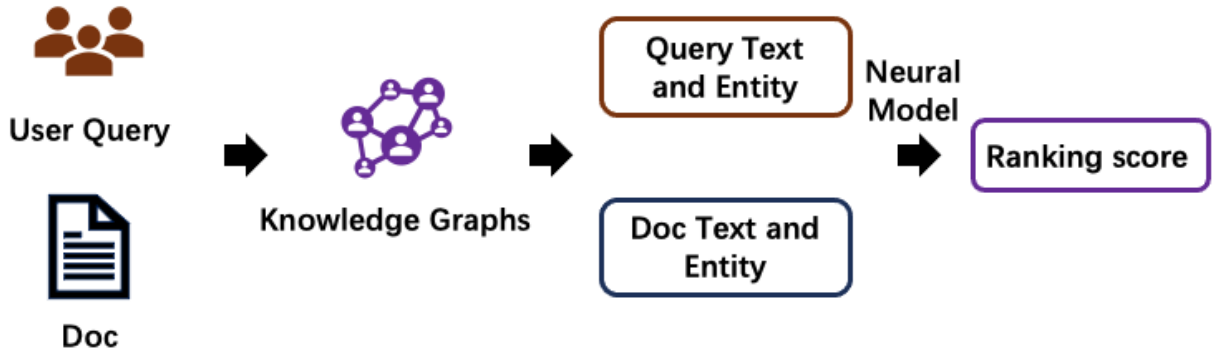
Moreover, OpenMatch incorporates advanced IR technologies to guarantee the Neu-IR performance on **few-shot ranking** and **conversational search**.

1.3 Few-Shot Ranking

Domain Transfer Learning can leverage external knowledge graphs or weak supervision data to guide and help ranker to overcome data scarcity.

1.3.1 Knowledge Enhanced IR Models

Knowledge Enhancement incorporates entity semantics of external knowledge graphs to enhance neural ranker. The entity semantics come from both structural (knowledge graph) and non-structural knowledge (entity description).



1.3.2 Data Augmentation for Neural IR Models

Pre-trained language models, such as BERT, conduct significant improvement on natural language tasks. However, for information retrieval, the performance of such pre-trained language models shows less effectiveness. On the contrary, when these models are fine-tuned with user searching logs, e.g. Bing log, their ranking accuracy achieves better. It demonstrates that the relevance supervisions of query and document are the bottleneck of neural information retrieval models.

Data Augmentation leverages weak supervision data to improve the ranking accuracy in certain areas that lacks large-scale relevance labels, such as the Biomedical domain and legal domain. In this part, OpenMatch provides some reproduced models to **generate weak supervision data** and **select weak supervision data with few labeled data**.

OpenMatch incorporates several advanced training methods to further broaden the advance of Neu-IR models in few-shot ranking scenarios. These few-shot training methods mainly focus on alleviating data scarcity in ranking scenarios by leveraging domain knowledge and data augmentation.

Weak supervision generation. The query-document relevance is usually the core of Neu-IR models. Nevertheless, these relevant labels are not easy to get. For example, the massive search log is a privilege and not a commodity, even in search engine companies; Some special areas, such as medicine and minority language, we couldn't even find enough experts for relevant labeling. As a substitute, some weak supervisions, such as the anchor texts and linked web pages on Internet, can serve as weak supervision text relevance labels. Besides, we can also generate queries according to the documents for synthesizing query-document relevance.

Weak supervision selection/reweighting. Nevertheless, the pseudo queries can be too general, even not informative at all, or matched with multiple documents. To solve this problem, we can use some data selection or reweighting methods, such as ReInfoSelect and MetaAdaptRank tries to select weak supervision data during training according to the model performance on the target data.

1.4 Conversational Search

With the recent development of information retrieval, information retrieval systems have been seeking more conversational interactions with users to improve user searching experiments. Conversational IR proposes a good way to better satisfying user searching intentions, which further considers previous searching sessions. Thus, how to model the searching history is the core of conversational IR.

A natural direction to tackle this challenge is to rewrite the conversational queries according to the previous searching queries, which helps include necessary information for a standard IR system. Various approaches have been developed to rewrite queries, including IR-style query expansion/term reweighting, NLP-style coreference resolution, and neural-based query rewriting. OpenMatch reproduces some advanced models in conversational search for users, such as ConvDR.

CHAPTER

TWO

INSTALLATION

2.1 Using PyPI

```
pip install git+https://github.com/thunlp/OpenMatch.git
```

2.2 Using Git Repository

```
git clone https://github.com/thunlp/OpenMatch.git
cd OpenMatch
python setup.py install
```

2.3 Using Docker

To build an OpenMatch docker image from Dockerfile

```
docker build -t <image_name> .
```

To run your docker image just built above as a container

```
docker run --gpus all --name=<container_name> -it -v /:/all/ --rm <image_name>:<TAG>
```

CHAPTER

THREE

SETTINGS AND BENCHMARK

Information Retrieval (IR) aims to retrieve relevant documents from a large-scale document collection to satisfy user needs, which have been used in many real-world applications, such as digital libraries, expert findings. A good IR system can benefit lots of NLP tasks, such as question answering, fact verification, and entity search. With the increasing of online documents and searching demand, achieving an effective and efficient IR system is very crucial for our society.

3.1 Ranking Scenarios in IR

According to different application scenarios, IR tasks can be divided into the following categories, ad hoc retrieval, and question answering. For example, for two related questions about Obama's family members, for document retrieval, we look forward to returning a document or paragraph related to input keywords. For question answering, we usually input a natural language sentence, so we need to understand the underlying semantic information of the given query and return the corresponding search results according to the semantic information described in the query.

- Ad hoc Retrieval
  - Query: Obama family tree
  - Document:
    - **Family of Barack Obama** - Wikipedia
    - Barack **Obama Family Tree** along with family connections to other famous kin. Genealogy charts for Barack Obama may include up to 30 generations of ...
- Question Answering
  - Query: Who is Barack Obama's sister?
  - Answer:

Maya Soetoro-Ng

Auma Obama

During the past decades, many ranking models (Traditional Information Retrieval Models) have been proposed to retrieve related documents, including vector space models, probabilistic models, and learning to rank (LTR) models. Such ranking techniques, especially the LTR models, have already achieved great success in many IR applications,

such as commercial web search engines, Google, Bing, and Baidu. Nevertheless, the above models usually pay more attention to keyword-oriented searching with exact matches and conduct shallow comprehension for both query and document because of the vocabulary mismatch problem.

In recent years, deep neural networks have led to a new tendency in the Information Retrieval area. Existing LTR models usually derive from hand-crafted features, which are usually time-consuming to design and show less generalize ability to other ranking scenarios. Compared to LTR models, neural models can be optimized end-to-end with relevance supervisions, and do not need these handcraft features.

In addition, the neural models also show their capability to recognize the potential matching patterns with the strong capability of the neural network, which helps ranking models comprehend the semantic information of query and candidate document. Due to these potential benefits, the neural models show their effectiveness in understanding the user intents, which thrives on searching in the question answering and fact verification tasks. The work of neural models has substantial grown in applying neural networks for ranking models in both academia and industry in recent years.

3.2 IR Benchmarks

The statistics of some IR datasets are listed as follows:

Dataset	Query	Query-Doc Pair
ClueWeb09-B	200	50M
ClueWeb12-B13	100	50M
Robust04	249	500K
MS MARCO	1.01M	485M
TREC COVID	50	191k

CHAPTER

FOUR

QUICK START

4.1 Preliminary

Given the query and document, ranking models aim to calculate the matching score between them.

```
import torch
import OpenMatch as om

query = "Classification treatment COVID-19"
doc = "By retrospectively tracking the dynamic changes of LYM% in death cases and cured␣
↪cases, this study suggests that lymphocyte count is an effective and reliable␣
↪indicator for disease classification and prognosis in COVID-19 patients."
```

4.2 Traditional IR models

We extract several classic IR features, and train learning-to-rank models, such as RankSVM, Coor-Ascent, on ClueWeb09-B, Robust04 and TREC-COVID datasets with 5 fold cross-validation. All the results can be found in our paper of ACL 2021.

The features consists of Boolean AND; Boolean OR; Coordinate match; Cosine similarity of bag-of-words vectors; TF-IDF; BM25; language models with no smoothing, Dirichlet smoothing, JM smoothing, and two-way smoothing. More details are available at classic_extractor.

OpenMatch provides several classic IR feature extractors. Document collection is needed for the creation of inverted dict. The parameter "docs" is a dict for all documents: "docs[docid] = doc".

```
corpus = om.Corpus(docs)
docs_terms, df, total_df, avg_doc_len = corpus.cnt_corpus()
query_terms, query_len = corpus.text2lm(query)
doc_terms, doc_len = corpus.text2lm(doc)
extractor = om.ClassicExtractor(query_terms, doc_terms, df, total_df, avg_doc_len)
features = extractor.get_feature()
```

4.3 Pretrained IR models

OpenMatch inherients parameters of pretrained language models from hugginface's trasnformers.

```
from transformers import AutoTokenizer

tokenizer = AutoTokenizer.from_pretrained("allenai/scibert_scivocab_uncased")
input_ids = tokenizer.encode(query, doc)
model = om.models.Bert("allenai/scibert_scivocab_uncased")
ranking_score, ranking_features = model(torch.tensor(input_ids).unsqueeze(0))
```

4.4 Neural IR models

```
tokenizer = om.data.tokenizers.WordTokenizer(pretrained="./data/glove.6B.300d.txt")
query_ids, query_masks = tokenizer.process(query, max_len=16)
doc_ids, doc_masks = tokenizer.process(doc, max_len=128)
model = om.models.KNRM(vocab_size=tokenizer.get_vocab_size(),
                       embed_dim=tokenizer.get_embed_dim(),
                       embed_matrix=tokenizer.get_embed_matrix())
ranking_score, ranking_features = model(torch.tensor(query_ids).unsqueeze(0),
                                        torch.tensor(query_masks).unsqueeze(0),
                                        torch.tensor(doc_ids).unsqueeze(0),
                                        torch.tensor(doc_masks).unsqueeze(0))
```

The pretrained word embeddings (GloVe) can be downloaded using:

```
wget http://nlp.stanford.edu/data/glove.6B.zip -P ./data
unzip ./data/glove.6B.zip -d ./data
```

4.5 Evaluation

```
metric = om.Metric()
res = metric.get_metric(qrels, ranking_list, 'ndcg_cut_20')
res = metric.get_mrr(qrels, ranking_list, 'mrr_cut_10')
```

CHAPTER

FIVE

OPENMATCH GUIDELINES

5.1 Train

For bert-like models training

```
sh train_bert.sh
```

For edrm, cknrm, knrm or tk training

```
sh train.sh
```

5.1.1 Options

```
-task             'ranking': pair-wise, 'classification': query-doc.
-model            'bert', 'tk', 'edrm', 'cknrm' or 'knrm'.
-reinfoselect     use reinfoselect or not.
-resset           reset the model or not, used in reinfoselect setting.
-train            path to training dataset.
-max_input        max input of instances.
-save             path for saving model checkpoint.
-dev              path to dev dataset.
-qrels            path to qrels.
-vocab            path to glove or customized vocab.
-ent_vocab        path to entity vocab, for edrm.
-pretrain         path to pretrained bert model.
-checkpoint       path to checkpoint.
-res              path for saving result.
-metric           which metrics to use, e.g. ndcg_cut_10.
-mode             use cls or pooling as bert representation.
-n_kernels        kernel number, for tk, edrm, cknrm or knrm.
-max_query_len    max length of query tokens.
-max_doc_len      max length of document tokens.
-maxp             bert max passage.
-epoch            how many epoch.
-batch_size       batch size.
-lr               learning rate.
-n_warmup_steps   warmup steps.
-eval_every       e.g. 1000, every 1000 steps evaluate on dev data.
```

5.2 Inference

For bert-like models inference

```
sh inference_bert.sh
```

For edrm, cknrm, knrm or tk inference

```
sh inference.sh
```

5.2.1 Options

```
-task              'ranking': pair-wise, 'classification': query-doc.
-model             'bert', 'tk', 'edrm', 'cknrm' or 'knrm'.
-max_input         max input of instances.
-test              path to test dataset.
-vocab             path to glove or customized vocab.
-ent_vocab         path to entity vocab.
-pretrain          path to pretrained bert model.
-checkpoint        path to checkpoint.
-res               path for saving result.
-n_kernels         kernel number, for tk, edrm, cknrm or knrm.
-max_query_len     max length of query tokens.
-max_doc_len       max length of document tokens.
-batch_size        batch size.
```

5.3 Data Format

5.3.1 Ranking Task

For bert, tk, cknrm or knrm:

file	format
train	{"query": str, "doc_pos": str, "doc_neg": str}
dev	{"query": str, "doc": str, "label": int, "query_id": str, "doc_id": str, "retrieval_score": float}
test	{"query": str, "doc": str, "query_id": str, "doc_id": str, "retrieval_score": float}

For edrm:

file	format
train	+{"query_ent": list, "doc_pos_ent": list, "doc_neg_ent": list, "query_des": list, "doc_pos_des": list, "doc_neg_des": list}
dev	+{"query_ent": list, "doc_ent": list, "query_des": list, "doc_des": list}
test	+{"query_ent": list, "doc_ent": list, "query_des": list, "doc_des": list}

The *query_ent*, *doc_ent* is a list of entities relevant to the query or document, *query_des* is a list of entity descriptions.

5.3.2 Classification Task

Only train file format different with ranking task.

For bert, tk, cknrm or knrm:

file	format
train	{"query": str, "doc": str, "label": int}

For edrm:

file	format
train	+{"query_ent": list, "doc_ent": list, "query_des": list, "doc_des": list}

5.3.3 Others

The dev and test files can be set as:

```
-dev queries={path to queries},docs={path to docs},qrels={path to qrels},trec={path to␣
↪trec}
-test queries={path to queries},docs={path to docs},trec={path to trec}
```

file	format
queries	{"query_id":, "query":}
docs	{"doc_id":, "doc":}
qrels	query_id iteration doc_id label
trec	query_id Q0 doc_id rank score run-tag

For edrm, the queries and docs are a little different:

file	format
queries	+{"query_ent": list, "query_des": list}
docs	+{"doc_ent": list, "doc_des": list}

Other bert-like models are also available, e.g. electra, scibert. You just need to change the path to the vocab and the pretrained model.

You can also train bert for masked language model with *train_bertmlm.py*. The train file format is as follows:

file	format
train	{'doc': str}

If you want to concatenate the neural features with retrieval scores (SDM/BM25), and run coor-ascent, you need to generate a features file using *gen_feature.py*, and run

```
sh coor_ascent.sh
```

CHAPTER

SIX

AD-HOC SEARCH

All experiments are measured on ndcg@20 with 5 fold cross-validation.

6.1 Data Statistics

Data can be downloaded from Datasets. Note that we followed the settings of Zhuyun Dai's work. We only use the subset of each dataset which was retrievaled by Indri's SDM for experiments.

Datasets	Queries/Anchors	Query/Anchor-Doc Pairs	Released Files
ClueWeb09-B	200	47.1K	Queries, Q-D Relations, SDM scores
Robust04	249	311K	Queries, Q-D Relations, SDM scores
ClueWeb12-B13	100	28.9K	Queries, Q-D Relations, SDM scores

As we cannot release the document contents, the document IDs are used instead.

6.2 Tasks

* ClueWeb09. **Domain: Web Pages**. The ClueWeb09 dataset was created to support research on information retrieval and related human language technologies. It consists of about 1 billion web pages in ten languages that were collected in January and February 2009. The dataset is used by several tracks of the TREC conference.

* ClueWeb12. **Domain: Web Pages**. The ClueWeb12 dataset was created to support research on information retrieval and related human language technologies. The dataset consists of 733,019,372 English web pages, collected between February 10, 2012 and May 10, 2012. ClueWeb12 is a companion or successor to the ClueWeb09 web dataset. Distribution of ClueWeb12 began in January 2013.

* Robust04. **Domain: News Articles**. The goal of the Robust track is to improve the consistency of retrieval technology by focusing on poorly performing topics. In addition, the track brings back a classic, ad hoc retrieval task in TREC that provides a natural home for new participants. An ad hoc task in TREC investigates the performance of systems that search a static set of documents using previously-unseen topics. For each topic, participants create a query and submit a ranking of the top 1000 documents for that topic.

6.3 Models

In ad-hoc search experiments, we use KNRM, Conv-KNRM, EDRM and TK as neural IR models, BERT and ELECTRA as pretrained models.

6.3.1 Training

For training neural ranking models, we use the KNRM model for example. The **-model** parameter can be set to any neural ranking model, such as **tk**, **cknrm** and **edrm**.

```
CUDA_VISIBLE_DEVICES=0 \
python train.py \
        -task ranking \
        -model knrm \
        -train queries=./data/query.jsonl,docs=./data/doc.jsonl,qrels=./data/qrels,trec=.
↪/data/fold0/train.trec \
        -max_input 12800000 \
        -save ./checkpoints/knrm.bin \
        -dev queries=./data/query.jsonl,docs=./data/doc.jsonl,qrels=./data/qrels,trec=./
↪data/fold0/dev.trec \
        -qrels ./data/qrels \
        -vocab ./data/glove.6B.100d.txt \
        -res ./results/knrm.trec \
        -metric ndcg_cut_20 \
        -n_kernels 21 \
        -max_query_len 16 \
        -max_doc_len 256 \
        -epoch 1 \
        -batch_size 32 \
        -lr 1e-3 \
        -n_warmup_steps 1000 \
        -eval_every 100
```

For training pretrained models, we use the BERT model for example. We can also use any pretrained model.

```
CUDA_VISIBLE_DEVICES=0 \
python train.py \
        -task ranking \
        -model bert \
        -train queries=./data/query.jsonl,docs=./data/doc.jsonl,qrels=./data/qrels,trec=.
↪/data/fold0/train.trec \
        -max_input 12800000 \
        -save ./checkpoints/bert.bin \
        -dev queries=./data/query.jsonl,docs=./data/doc.jsonl,qrels=./data/qrels,trec=./
↪data/fold0/dev.trec \
        -qrels ./data/qrels \
        -vocab bert-base-uncased \
        -pretrain bert-base-uncased \
        -res ./results/bert.trec \
        -metric ndcg_cut_20 \
        -max_query_len 32 \
        -max_doc_len 221 \
```

(continues on next page)

(continued from previous page)

```
        -epoch 1 \
        -batch_size 16 \
        -lr 3e-6 \
        -n_warmup_steps 1000 \
        -eval_every 100
```

For getting classic IR features (e.g. boolean, language model, tfidf, bm25 ...), we need first read the document file to a dict (docs[docid] = doc), then for each given query-doc pair, classic features can be compute as follows:

```
import OpenMatch as om

corpus = om.Corpus(docs)
docs_terms, df, total_df, avg_doc_len = corpus.cnt_corpus()
query_terms, query_len = corpus.text2lm(query)
doc_terms, doc_len = corpus.text2lm(doc)
extractor = om.ClassicExtractor(query_terms, doc_terms, df, total_df, avg_doc_len)
features = extractor.get_feature()
```

6.3.2 Inference

For neural ranking models:

```
CUDA_VISIBLE_DEVICES=0 \
python inference.py \
    -task ranking \
    -model knrm \
    -max_input 1280000 \
    -vocab ./data/glove.6B.300d.txt \
    -checkpoint ./checkpoints/knrm.bin \
    -test queries=./data/query.jsonl,docs=./data/doc.jsonl,qrels=./data/qrels,trec=./
↪data/fold0/test.trec \
    -res ./results/knrm.trec \
    -max_query_len 16 \
    -max_doc_len 256 \
    -batch_size 512
```

For pretrained models:

```
CUDA_VISIBLE_DEVICES=0 \
python inference.py \
        -task ranking \
        -model bert \
        -max_input 12800000 \
        -test queries=./data/query.jsonl,docs=./data/doc.jsonl,qrels=./data/qrels,trec=./
↪data/fold0/test.trec \
        -vocab bert-base-uncased \
        -pretrain bert-base-uncased \
        -checkpoint ./checkpoints/bert.bin \
        -res ./results/bert.trec \
        -max_query_len 32 \
        -max_doc_len 221 \
        -batch_size 256
```

6.4 Results

* Ad-hoc Search

Retriever	Reranker	Coor-Ascent	ClueWeb09	Robust04	ClueWeb12
SDM	KNRM	-	0.1880	0.3016	0.0968
SDM	Conv-KNRM	-	0.1894	0.2907	0.0896
SDM	EDRM	-	0.2015	0.2993	0.0937
SDM	TK	-	0.2306	0.2822	0.0966
SDM	BERT Base	-	0.2701	0.4168	0.1183
SDM	ELECTRA Base	-	0.2861	0.4668	0.1078

6.5 MetaAdaptRank

Here provides the guiding code for the method of meta-learning to reweight synthetic weak supervision data, which uses target data to reweight contrastive synthetic data (CTSyncSup) during the learning to rank process. A detailed introduction to the technology can be found in the paper Few-Shot Text Ranking with Meta Adapted Synthetic Weak Supervision. This method contains two parts:

1. Contrastive Supervision Synthesis
2. Meta Learning to Reweight

6.5.1 Contrastive Supervision Synthesis

Source-domain NLG training. We train two query generators (QG & ContrastQG) with the MS MARCO dataset using `train_nlg.sh`:

```
bash train_nlg.sh
```

Optional arguments:

```
--generator_mode            choices=['qg', 'contrastqg']
--pretrain_generator_type   choices=['t5-small', 't5-base']
--train_file                The path to the source-domain nlg training dataset
--save_dir                  The path to save the checkpoints data
```

Target-domain NLG inference. The whole nlg inference pipline contains five steps:

- 1/ Data preprocess
- 2/ Seed query generation
- 3/ BM25 subset retrieval
- 4/ Contrastive doc pairs sampling
- 5/ Contrastive query generation

1/ Data preprocess. convert target-domain documents into the nlg format using `prepro_dataset.sh` in the folder `preprocess`:

```
bash prepro_dataset.sh
```

Optional arguments:

```
--dataset_name          The name of the target dataset
--input_path            The path to the target dataset
--output_path           The path to save the preprocess data
```

2/ Seed query generation. utilize the trained QG model to generate seed queries for each target documents using `qg_inference.sh` in the folder `run_shell`:

```
bash qg_inference.sh
```

Optional arguments:

```
--generator_mode            choices='qg'
--pretrain_generator_type   choices=['t5-small', 't5-base']
--target_dataset_name       The name of the target dataset
--generator_load_dir        The path to the pretrained QG checkpoints
```

3/ BM25 subset retrieval. utilize BM25 to retrieve document subset according to the seed queries using the following shell commands in the folder `bm25_retriever`:

```
bash build_index.sh
bash retrieve.sh
```

Optional arguments:

```
--dataset_name          The name of the target dataset
--data_path             The path to the target dataset
```

4/ Contrastive doc pairs sampling. pairwise sample contrastive doc pairs from the BM25 retrieved subset using `sample_contrast_pairs.sh` in the folder `preprocess`:

```
bash sample_contrast_pairs.sh
```

Optional arguments:

```
--dataset_name          choices=['clueweb09', 'robust04', 'trec-covid']
--input_path            The path to the target dataset
--generator_folder      The path to the sampled data
```

5/ Contrastive query generation. utilize the trained ContrastQG model to generate new queries based on contrastive document pairs using `cqg_inference.sh` in the folder `run_shell`:

```
bash cqg_inference.sh
```

Optional arguments:

```
--generator_mode            choices='contrastqg'
--pretrain_generator_type   choices=['t5-small', 't5-base']
--target_dataset_name       The name of the target dataset
--generator_load_dir        The path to the pretrained ContrastQG checkpoints
```

6.5.2 Meta Learning to Reweight

The code to run meta-learning-to-reweight is in the shell file

```
bash meta_dist_train.sh
```

In the shell file, the code is written as

```
export gpu_num=4 ## GPU Number
export master_port=23900
export job_name=MetaBERT

## ************************************
export DATA_DIR= ## please set your dataset path here.
export SAVE_DIR= ## please set your saving path here.

## ************************************
CUDA_VISIBLE_DEVICES=0,1,2,3 OMP_NUM_THREADS=1 python -u -m torch.distributed.launch --
↪nproc_per_node=$gpu_num --master_port $master_port meta_dist_train.py \
-job_name $job_name \
-save_folder $SAVE_DIR/results \
-model bert \
-task ranking \
-max_input 12800000 \
-train queries=$DATA_DIR/queries.train.tsv,docs=$DATA_DIR/collection.tsv,qrels=$DATA_DIR/
↪qrels.train.tsv,trec=$DATA_DIR/trids_bm25_marco-10.tsv \
-dev queries=$DATA_DIR/queries.dev.small.tsv,docs=$DATA_DIR/collection.tsv,qrels=$DATA_
↪DIR/qrels.dev.small.tsv,trec=$DATA_DIR/run.msmarco-passage.dev.small.100.trec \
-target trec=$DATA_DIR/devids_bm25_marco.tsv \
-qrels $DATA_DIR/qrels.dev.small.tsv \
-vocab bert-base-uncased \
-pretrain bert-base-uncased \
-metric mrr_cut_10 \
-max_query_len 32 \
-max_doc_len 221 \
-epoch 3 \
-train_batch_size 8 \
-target_batch_size 16 \
-gradient_accumulation_steps 2 \
-dev_eval_batch_size 1024 \
-lr 3e-6 \
-n_warmup_steps 160000 \
-logging_step 2000 \
-eval_every 10000 \
-eval_during_train \
```

The tsv format of `-target` data is totally the same with the `-train` data.

```
query_id \t pos_docid \t neg_docid
```

Retriever	Reranker	Augment Data	Coor-Ascent	ClueWeb09	Robust04	TREC-COVID
SDM	MetaAdaptRank	CTSyncSup	+	0.3416	0.4916	0.8378

CHAPTER

SEVEN

MS MARCO PASSAGE RANKING

Given a query q and a the 1000 most relevant passages P = p1, p2, p3,... p1000, as retrieved by BM25 a successful system is expected to rerank the most relevant passage as high as possible. For this task not all 1000 relevant items have a human labeled relevant passage. Evaluation will be done using MRR.

7.1 Tasks

* MSMARCO. **Domain: Web Pages**.

* MSMARCO Passage Ranking. **Domain: Web Pages**.

7.2 Models

We use BERT base model, RoBERTa large model, and ELECTRA base and large models for experiments.

7.2.1 Training

Train.

```
CUDA_VISIBLE_DEVICES=0 \
python train.py \
        -task ranking \
        -model bert \
        -train  queries=./data/queries.train.small.tsv,docs=./data/collection.tsv,qrels=.
↪/data/qrels.train.tsv,trec=./data/trids_bm25_marco-10.tsv \
        -max_input 12800000 \
        -save ./checkpoints/bert.bin \
        -dev queries=./data/queries.dev.small.tsv,docs=./data/collection.tsv,qrels=./
↪data/qrels.dev.small.tsv,trec=./data/run.msmarco-passage.dev.small.100.trec \
        -qrels ./data/qrels.dev.small.tsv \
        -vocab bert-base-uncased \
        -pretrain bert-base-uncased \
        -res ./results/bert.trec \
        -metric mrr_cut_10 \
        -max_query_len 32 \
        -max_doc_len 221 \
        -epoch 3 \
        -batch_size 16 \
```

(continues on next page)

(continued from previous page)

```
        -lr 3e-6 \
        -n_warmup_steps 160000 \
        -eval_every 10000
```

To train electra-large and roberta-large

First convert training data to jsonl version vis data_process.py

```
import json
import codecs

def main():
    f_train_tsv = codecs.open('./data/triples.train.small.tsv','r')
    f_train_jsonl = codecs.open('./data/train.jsonl', 'w')
    cnt = 0
    for line in f_train_tsv:
        s = line.strip().split('\t')
        f_train_jsonl.write(json.dumps({'query':s[0],'doc_pos':s[1],'doc_neg':s[2]}) + '\
↪n')
        cnt += 1
        if cnt > 3000000:
            break
    f_train_jsonl.close()
    f_train_tsv.close()
    print(cnt)

if __name__ == "__main__":
    main()
```

```
python3 data_process.py
```

Train electra-large

```
CUDA_VISIBLE_DEVICES=0 \
python train.py\
        -task ranking \
        -model bert \
        -train ./data/train.jsonl \
        -max_input 3000000 \
        -save ./checkpoints/electra_large.bin \
        -dev queries=./data/queries.dev.small.tsv,docs=./data/collection.tsv,qrels=./
↪data/qrels.dev.small.tsv,trec=./data/run.msmarco-passage.dev.small.100.trec \
        -qrels ./data/qrels.dev.small.tsv \
        -vocab google/electra-large-discriminator \
        -pretrain google/electra-large-discriminator \
        -res ./results/electra_large.trec \
        -metric mrr_cut_10 \
        -max_query_len 32 \
        -max_doc_len 256 \
        -epoch 1 \
        -batch_size 2 \
        -lr 5e-6 \
        -eval_every 10000
```

Train roberta-large

```
CUDA_VISIBLE_DEVICES=0 \
python train.py \
        -task ranking \
        -model roberta \
        -train ./data/train.jsonl \
        -max_input 3000000 \
        -save ./checkpoints/roberta_large.bin \
        -dev queries=./data/queries.dev.small.tsv,docs=./data/collection.tsv,qrels=./
↪data/qrels.dev.small.tsv,trec=./data/run.msmarco-passage.dev.small.100.trec \
        -qrels ./data/qrels.dev.small.tsv \
        -vocab roberta-large \
        -pretrain roberta-large \
        -res ./results/roberta_large.trec \
        -metric mrr_cut_10 \
        -max_query_len 32 \
        -max_doc_len 256 \
        -epoch 1 \
        -batch_size 1 \
        -lr 5e-7 \
        -eval_every 20000
```

Since the whole dev dataset is too large, we only evaluate on top100 when training, and inference on whole dataset.

7.2.2 Inference

Get data and checkpoint from Google Drive

Get checkpoints of electra-large and roberta-large from electra-large roberta-large

Get MS MARCO collection.

```
wget https://msmarco.blob.core.windows.net/msmarcoranking/collection.tar.gz -P ./data
tar -zxvf ./data/collection.tar.gz -C ./data/
```

Reproduce bert-base, MRR@10(dev): 0.3494.

```
CUDA_VISIBLE_DEVICES=0 \
python inference.py \
        -task ranking \
        -model bert \
        -max_input 12800000 \
        -test queries=./data/queries.dev.small.tsv,docs=./data/collection.tsv,trec=./
↪data/run.msmarco-passage.dev.small.trec \
        -vocab bert-base-uncased \
        -pretrain bert-base-uncased \
        -checkpoint ./checkpoints/bert-base.bin \
        -res ./results/bert-base_msmarco-dev.trec \
        -max_query_len 32 \
        -max_doc_len 221 \
        -batch_size 256
```

Reproduce electra-base, MRR@10(dev): 0.3518.

```
CUDA_VISIBLE_DEVICES=0 \
python inference.py \
        -task ranking \
        -model bert \
        -max_input 12800000 \
        -test queries=./data/queries.dev.small.tsv,docs=./data/collection.tsv,trec=./
↪data/run.msmarco-passage.dev.small.trec \
        -vocab google/electra-base-discriminator \
        -pretrain google/electra-base-discriminator \
        -checkpoint ./checkpoints/electra-base.bin \
        -res ./results/electra-base_msmarco-dev.trec \
        -max_query_len 32 \
        -max_doc_len 221 \
        -batch_size 256
```

Reproduce electra-large, MRR@10(dev): 0.388

```
CUDA_VISIBLE_DEVICES=0 \
python inference.py \
        -task ranking \
        -model bert \
        -max_input 12800000 \
        -test queries=./data/queries.dev.small.tsv,docs=./data/collection.tsv,trec=./
↪data/run.msmarco-passage.dev.small.trec \
        -vocab google/electra-large-discriminator \
        -pretrain google/electra-large-discriminator \
        -checkpoint ./checkpoints/electra_large.bin \
        -res ./results/electra-large_msmarco-dev.trec \
        -max_query_len 32 \
        -max_doc_len 221 \
        -batch_size 256
```

Reproduce roberta-large, MRR@10(dev): 0.386

```
CUDA_VISIBLE_DEVICES=0 \
python inference.py \
        -task ranking \
        -model roberta \
        -max_input 12800000 \
        -test queries=./data/queries.dev.small.tsv,docs=./data/collection.tsv,trec=./
↪data/run.msmarco-passage.dev.small.trec \
        -vocab roberta-large \
        -pretrain roberta-large \
        -checkpoint ./checkpoints/roberta_large.bin \
        -res ./results/roberta-large_msmarco-dev.trec \
        -max_query_len 32 \
        -max_doc_len 221 \
        -batch_size 256
```

The checkpoints of roberta-large and electra-large are trained on MS-MARCO training data

```
wget https://msmarco.blob.core.windows.net/msmarcoranking/triples.train.small.tar.gz -P .
↪/data
tar -zxvf ./data/triples.train.small.tar.gz -C ./data/
```

For eval dataset inference, just change the trec file to *./data/run.msmarco-passage.eval.small.trec*. The top1000 trec files for dev and eval queries are generated following anserini.

7.3 Results

Results of the runs we submitted.

Retriever	Reranker	Coor-Ascent	dev	eval
BM25	BERT Base	-	0.349	0.345
BM25	ELECTRA Base	-	0.352	0.344
BM25	RoBERTa Large	-	0.386	0.375
BM25	ELECTRA Large	+	0.388	0.376

CHAPTER

EIGHT

MS MARCO DOCUMENT RANKING

MS MARCO (Microsoft Machine Reading Comprehension) is a large scale dataset, the current dataset has 1,010,916 unique real queries that were generated by sampling and anonymizing Bing usage logs. The corpus of document ranking task has 3.2 million documents and the training set has 367,013 queries.

8.1 Tasks

* MSMARCO. **Domain: Web Pages**.

* MSMARCO Document Ranking. **Domain: Web Pages**.

8.2 Datasets & Checkpoints

For BERT FirstP, we concatenate the title and content of each document by a '[SEP]'. For BERT MaxP, we only use the content of each document. To reproduce our runs, we need to preprocess the official document file to the format: *doc_id doc*.

Type	File	Records	Format	Description
Corpus	msmarco-docs.tsv	3,213,835	tsv: docid, url, title, body	Document Collections
Train	msmarco-doctrain-queries.tsv	367,013	tsv: qid, query	Training Queries
Train	msmarco-doctrain-qrels.tsv	384,597	TREC qrels	Training Query-Doc Relevance Labels
Train	Training-Data-FirstP	7,340,240	tsv: qid, docid, label	ANCE FirstP training data
Train	Training-Data-MaxP	7,340,240	tsv: qid, docid, label	ANCE MaxP training data
Dev	msmarco-docdev-queries.tsv	5,193	tsv: qid, query	Dev Queries
Dev	msmarco-docdev-qrels.tsv	5,478	TREC qrels	Dev Query-Doc Relevance Labels
Dev	ANCE-FirstP-dev-top100	519,300	TREC submission	ANCE FirstP dev top100
Dev	ANCE-MaxP-dev-top100	519,300	TREC submission	ANCE MaxP dev top100
Test	docleaderboard-queries.tsv	5,793	tsv: qid, query	Test Queries
Test	ANCE-FirstP-eval-top100	579,300	TREC submission	ANCE FirstP eval top100
Test	ANCE-MaxP-eval-top100	579,300	TREC submission	ANCE MaxP eval top100
Model	BERT-Base-ANCE-FirstP	-	-	BERT Base ANCE FirstP checkpoint
Model	BERT-Base-ANCE-MaxP	-	-	BERT Base ANCE MaxP checkpoint
Model	F-MaxP	-	-	BERT Base ANCE MaxP Coor-Ascent weights

8.3 Models

We use ANCE FirstP and MaxP as retrieval models, BERT FirstP and MaxP as reranking models. The FirstP and MaxP settings are same as paper.

8.3.1 Training

You can also finetune BERT yourself instead of using our checkpoints.

* BERT FirstP

We provide our training data (qid did label): Training-Data-FirstP. 10 negative documents are randomly sampled for each training query from ANCE FirstP top-100 documents. Since the dev dataset is too large to evaluate every 10000 steps, we only evaluate the top-100 documents of the first 50 dev queries: *msmarco-doc_dev_firstp-50.jsonl*.

```
CUDA_VISIBLE_DEVICES=0 \
python train.py \
        -task classification \
        -model bert \
        -train queries=./data/msmarco-doctrain-queries.tsv,docs=./data/msmarco-docs-
↪firstp.tsv,qrels=./data/msmarco-doctrain-qrels.tsv,trec=./data/bids_marco-doc_ance-
↪firstp-10.tsv \
```

(continues on next page)

(continued from previous page)

```
        -max_input 12800000 \
        -save ./checkpoints/bert-base-firstp.bin \
        -dev ./data/msmarco-doc_dev_firstp-50.jsonl \
        -qrels ./data/msmarco-docdev-qrels.tsv \
        -vocab bert-base-uncased \
        -pretrain bert-base-uncased \
        -res ./results/bert.trec \
        -metric mrr_cut_100 \
        -max_query_len 64 \
        -max_doc_len 445 \
        -epoch 1 \
        -batch_size 4 \
        -lr 3e-6 \
        -n_warmup_steps 100000 \
        -eval_every 10000
```

After BERT finetuning, we choose the best checkpoint on dev dataset to generate BERT features.

```
CUDA_VISIBLE_DEVICES=0 \
python gen_feature.py \
        -task classification \
        -model bert \
        -max_input 12800000 \
        -dev ./data/msmarco-doc_dev_firstp.jsonl \
        -vocab bert-base-uncased \
        -pretrain bert-base-uncased \
        -checkpoint ./checkpoints/bert-base-firstp.bin \
        -res ./features/bert-base_ance_dev_firstp_features \
        -max_query_len 64 \
        -max_doc_len 445 \
        -batch_size 256
```

Then, we run Coor-Ascent on these features using RankLib to learned the weight of each feature.

```
java -jar LeToR/RankLib-2.1-patched.jar -train features/bert-base_ance_dev_firstp_
↪features -ranker 4 -metric2t RR@100 -save checkpoints/f_firstp.ca
```

Finally, we can generate the features of eval dataset, and compute the ranking scores using the feature weights, which is the same as that in the *inference* section.

* BERT MaxP

We provde our training data (qid did label): Training-Data-MaxP. 10 negative documents are randomly sampled for each training query from ANCE MaxP top-100 documents. Since the dev dataset is too large to evaluate every 10000 steps, we only evaluate the top-100 documents of the first 50 dev queries: *msmarco-doc_dev_maxp-50.jsonl*.

Train.

```
CUDA_VISIBLE_DEVICES=0,1,2,3 \
python train.py \
        -task classification \
        -model bert \
        -train queries=./data/msmarco-doctrain-queries.tsv,docs=./data/msmarco-docs-maxp.
↪tsv,qrels=./data/msmarco-doctrain-qrels.tsv,trec=./data/bids_marco-doc_ance-maxp-10.
↪tsv \
```

(continues on next page)

(continued from previous page)

```
        -max_input 12800000 \
        -save ./checkpoints/bert-base-maxp.bin \
        -dev ./data/msmarco-doc_dev_maxp-50.jsonl \
        -qrels ./data/msmarco-docdev-qrels.tsv \
        -vocab bert-base-uncased \
        -pretrain bert-base-uncased \
        -res ./results/bert.trec \
        -metric mrr_cut_100 \
        -max_query_len 64 \
        -max_doc_len 445 \
        -maxp \
        -epoch 1 \
        -batch_size 8 \
        -lr 2e-5 \
        -n_warmup_steps 50000 \
        -eval_every 10000
```

After BERT finetuning, we choose the best checkpoint on dev dataset to generate BERT features.

```
CUDA_VISIBLE_DEVICES=0 \
python gen_feature.py \
        -task classification \
        -model bert \
        -max_input 12800000 \
        -dev ./data/msmarco-doc_dev_maxp.jsonl \
        -vocab bert-base-uncased \
        -pretrain bert-base-uncased \
        -checkpoint ./checkpoints/bert-base-maxp.bin \
        -res ./features/bert-base_ance_dev_maxp_features \
        -max_query_len 64 \
        -max_doc_len 445 \
        -maxp \
        -batch_size 64
```

Then, we run Coor-Ascent on these features using RankLib to learned the weight of each feature.

```
java -jar LeToR/RankLib-2.1-patched.jar -train features/bert-base_ance_dev_maxp_features␣
↪-ranker 4 -metric2t RR@100 -save checkpoints/f_maxp.ca
```

Finally, we can generate the features of eval dataset, and compute the ranking scores using the feature weights, which is the same as that in the *inference* section.

8.3.2 Inference

* BERT FirstP

We provide the ANCE FirstP top-100 documents of dev and docleaderboard queries in aliyun in standard TREC format. You can click to download these data.

Preprocess dev and eval dataset, *msmarco-docs-firstp.tsv* is the preprocessed document file, each line is *doc_id title [SEP] content*:

```
python data/preprocess.py -input_trec data/ANCE_FirstP_dev.trec -input_qrels data/
↪msmarco-docdev-qrels.tsv -input_queries data/msmarco-docdev-queries.tsv -input_docs␣
↪data/msmarco-docs-firstp.tsv -output data/msmarco-doc_dev_firstp.jsonl
python data/preprocess.py -input_trec data/ANCE_FirstP_eval.trec -input_queries data/
↪docleaderboard-queries.tsv -input_docs data/msmarco-docs-firstp.tsv -output data/
↪msmarco-doc_eval_firstp.jsonl
```

The checkpoint of BERT Base FirstP is available at BERT-Base-ANCE-FirstP. Now you can reproduce *ANCE FirstP + BERT Base FirstP*, MRR@100(dev): 0.4079.

```
CUDA_VISIBLE_DEVICES=0 \
python inference.py \
        -task classification \
        -model bert \
        -max_input 12800000 \
        -test ./data/msmarco-doc_dev_firstp.jsonl \
        -vocab bert-base-uncased \
        -pretrain bert-base-uncased \
        -checkpoint ./checkpoints/bert-base_ance_firstp.bin \
        -res ./results/bert-base_ance_dev_firstp.trec \
        -max_query_len 64 \
        -max_doc_len 445 \
        -batch_size 256
```

* BERT MaxP

ANCE MaxP top-100 documents of dev and docleaderboard queries are also provided.

Preprocess dev dataset, *msmarco-docs-maxp.tsv* is the preprocessed document file, each line is *doc_id content*:

```
python data/preprocess.py -input_trec data/ANCE_FirstP_dev.trec -input_qrels data/
↪msmarco-docdev-qrels.tsv -input_queries data/msmarco-docdev-queries.tsv -input_docs␣
↪data/msmarco-docs-firstp.tsv -output data/msmarco-doc_dev_maxp.jsonl
python data/preprocess.py -input_trec data/ANCE_FirstP_eval.trec -input_queries data/
↪docleaderboard-queries.tsv -input_docs data/msmarco-docs-firstp.tsv -output data/
↪msmarco-doc_eval_maxp.jsonl
```

The checkpoint of BERT Base MaxP is available at BERT-Base-ANCE-MaxP. Now you can reproduce *ANCE MaxP + BERT Base MaxP*, MRR@100(dev): 0.4094.

```
CUDA_VISIBLE_DEVICES=0 \
python inference.py \
        -task classification \
        -model bert \
        -max_input 12800000 \
        -test ./data/msmarco-doc_dev_maxp.jsonl \
        -vocab bert-base-uncased \
        -pretrain bert-base-uncased \
        -checkpoint ./checkpoints/bert-base_ance_maxp.bin \
        -res ./results/bert-base_ance_dev_maxp.trec \
        -max_query_len 64 \
        -max_doc_len 445 \
        -maxp \
        -batch_size 64
```

We also provide the weights of BERT Base MaxP features learned by Coor-Ascent: F-MaxP. First, generate the BERT Base MaxP features of eval dataset.

```
CUDA_VISIBLE_DEVICES=0 \
python gen_feature.py \
        -task classification \
        -model bert \
        -max_input 12800000 \
        -dev ./data/msmarco-doc_eval_maxp.jsonl \
        -vocab bert-base-uncased \
        -pretrain bert-base-uncased \
        -checkpoint ./checkpoints/bert-base_ance_maxp.bin \
        -res ./features/bert-base_ance_eval_maxp_features \
        -max_query_len 64 \
        -max_doc_len 445 \
        -maxp \
        -batch_size 64
```

Then, we compute the ranking score using the weights.

```
java -jar LeToR/RankLib-2.1-patched.jar -load checkpoints/f_maxp.ca -rank features/bert-
↪base_ance_eval_maxp_features -score f0.score
python LeToR/gen_trec.py -dev data/msmarco-doc_eval_maxp.jsonl -res results/bert-base_
↪ance_eval_maxp_ca.trec -k -1
```

8.3.3 Dense Retriever Inference

We provide dense retriever codes (optimized ANCE) in OpenMatch for MSMARCO Document and MSMARCO-like datasets.

In such dataset, several files must be provided: - documents - `[document collection.tsv]`: each line contains `[passage id]\t[text]\n` for the document texts. `[passage id]`s are in format "Dxxxxx", where "xxxxx" are integers. - files need be provided for each query set. (training, dev, eval, etc.) - `[custom queries.tsv]`: `[query id]\t[text]\n` for lines. `[query id]` is also integers. - `[custom qrels.tsv]`: `[query id] 0 [passage id] 1\n` for lines. This is optional because we may not have answers for the testset queries.

Let's go to the retriever folder to do futher operation `cd ./retrievers/DANCE/`.

Pre-processing:

```
python data/custom_data.py \
--data_dir [raw tsv data folder] \
--out_data_dir [processed data folder] \
--model_type rdot_nll \
--model_name_or_path roberta-base \
--max_seq_length 512 \
--data_type 0 \
--doc_collection_tsv [doc text path] \
--save_prefix [query saving name] \
--query_collection_tsv [query text path] \
--qrel_tsv [optional qrel tsv] \
```

You can specify a pytorch checkpoint and use it to inference the embeddings of the documents or queries.

```
python -u -m torch.distributed.launch \
--nproc_per_node=[num GPU] --master_port=57135 ./drivers/run_ann_emb_inference.py \
--model_type rdot_nll \
--inference_type query --save_prefix [prefix of the query preprocessed file. eg., train] \
--split_ann_search --gpu_index \
--init_model_dir [checkpoint folder] \
--data_dir [processed data folder] \
--training_dir [task folder] \
--output_dir [task folder]/ann_data/ \
--cache_dir [task folder]/ann_data/cache/ \
--max_query_length 64 --max_seq_length 512 --per_gpu_eval_batch_size 256
```

With using parameters `--inference_type query --save_prefix [prefix of the query preprocessed file. eg., train] \`, you can inference different sets of queries. With using parameters `--inference_type document` and removing `--save_prefix`, you can inference the document embeddings.

Next, you can use the following code to produce the trec format retrieval results of different query sets. Note that the embedding files will be matched by `emb_file_pattern = os.path.join(emb_dir, f'{emb_prefix}{checkpoint_postfix}__emb_p__data_obj_*.pb')`, check out how your embeddings are saved and specify the `checkpoint_postfix` for the program to load the embeddings.

```
python ./evaluation/retrieval.py \
--trec_path [output trec path] \
--emb_dir [folder dumpped query and passage/document embeddings which is output_dir, same as --output_dir above] \
--checkpoint_postfix [checkpoint custom name] \
--processed_data_dir [processed data folder] ] \
--queryset_prefix [query saving name] \
--gpu_index True --topN 100 --data_type 0
```

Now you can calculate different metrics and play with the trec files for further reranking experiments with OpenMatch.

8.4 Results

Results of the runs we submitted.

Retriever	Reranker	Coor-Ascent	dev	eval
ANCE FirstP	-	-	0.373	0.334
ANCE MaxP	-	-	0.383	0.342
ANCE FirstP+BM25	BERT Base FirstP	-	0.407	-
ANCE FirstP+BM25	BERT Base FirstP	+	0.431	0.380
ANCE MaxP	BERT Base MaxP	-	0.409	-
ANCE MaxP	BERT Base MaxP	+	0.432	0.391

CHAPTER

NINE

TREC COVID

Top Spot on TREC-COVID Challenge (May 2020, Round2).

The twin goals of the challenge are to evaluate search algorithms and systems for helping scientists, clinicians, policy makers, and others manage the existing and rapidly growing corpus of scientific literature related to COVID-19, and to discover methods that will assist with managing scientific information in future global biomedical crises.

>> Reproduce Our Submit >> About COVID-19 Dataset >> Our Paper

9.1 Data Statistics

Data can be downloaded from Datasets. Each round (except Round1) only contains 5 new queries.

Datasets	Queries	Valid Documents
Round1	30	51.1K
Round2	35	59.9K
Round3	40	128.5K
Round4	45	157.8K
Round5	50	191.2K

9.2 Tasks

TREC-COVID. **Domain: BioMed Papers**.

9.3 Models

We use SciBERT base model for TREC-COVID experiments. ReInfoSelect and MetaAdaptRank frameworks are used to select more adaptive data for better performance of ranking models.

9.4 Results

Round	Method	Pre-trained Model	NDCG@20	P@20
5	MetaAdaptRank	PudMedBERT	0.7904	0.9400

9.4.1 training

Get training data from google drive.

Preprocess round1 data.

```
python ./data/preprocess.py \
  -input_trec anserini.covid-r1.fusion1.txt \
  -input_qrels qrels-cord19-round1.txt \
  -input_queries questions_cord19-rnd1.txt \
  -input_docs cord19_0501_titabs.jsonl \
  -output ./data/dev_trec-covid-round1.jsonl
```

Train.

```
CUDA_VISIBLE_DEVICES=0 \
python train.py \
        -task classification \
        -model bert \
        -train ./data/seanmed.train.320K-pairs.jsonl \
        -max_input 1280000 \
        -save ./checkpoints/scibert.bin \
        -dev ./data/dev_trec-covid-round1.jsonl \
        -qrels qrels-cord19-round1.txt \
        -vocab allenai/scibert_scivocab_uncased \
        -pretrain allenai/scibert_scivocab_uncased \
        -res ./results/scibert.trec \
        -metric ndcg_cut_10 \
        -max_query_len 32 \
        -max_doc_len 256 \
        -epoch 5 \
        -batch_size 16 \
        -lr 2e-5 \
        -n_warmup_steps 4000 \
        -eval_every 200
```

For ReInfoSelect training:

```
CUDA_VISIBLE_DEVICES=0 \
python train.py \
        -task classification \
        -model bert \
        -reinfoselect \
        -reset \
        -train ./data/seanmed.train.320K-pairs.jsonl \
        -max_input 1280000 \
        -save ./checkpoints/scibert_rl.bin \
```

(continues on next page)

(continued from previous page)

```
        -dev ./data/dev_trec-covid-round1.jsonl \
        -qrels qrels-cord19-round1.txt \
        -vocab allenai/scibert_scivocab_uncased \
        -pretrain allenai/scibert_scivocab_uncased \
        -checkpoint ./checkpoints/scibert.bin \
        -res ./results/scibert_rl.trec \
        -metric ndcg_cut_10 \
        -max_query_len 32 \
        -max_doc_len 256 \
        -epoch 5 \
        -batch_size 8 \
        -lr 2e-5 \
        -tau 1 \
        -n_warmup_steps 5000 \
        -eval_every 1
```

9.4.2 Inference

Get checkpoint. checkpoints

Get data from Google Drive. round1 and round2

Filter round1 data from round2 data.

```
python data/filter.py \
  -input_qrels qrels-cord19-round1.txt \
  -input_trec anserini.covid-r2.fusion2.txt \
  -output_topk 50 \
  -output_trec anserini.covid-r2.fusion2-filtered.txt
```

Preprocess round2 data.

```
python ./data/preprocess.py \
  -input_trec anserini.covid-r2.fusion2-filtered.txt \
  -input_queries questions_cord19-rnd2.txt \
  -input_docs cord19_0501_titabs.jsonl \
  -output ./data/test_trec-covid-round2.jsonl
```

Reproduce scibert.

```
CUDA_VISIBLE_DEVICES=0 \
python inference.py \
        -task classification \
        -model bert \
        -max_input 1280000 \
        -test ./data/test_trec-covid-round2.jsonl \
        -vocab allenai/scibert_scivocab_uncased \
        -pretrain allenai/scibert_scivocab_uncased \
        -checkpoint ./checkpoints/scibert.bin \
        -res ./results/scibert.trec \
        -mode cls \
        -max_query_len 32 \
```

(continues on next page)

(continued from previous page)

```
        -max_doc_len 256 \
        -batch_size 32
```

Reproduce reinfoselect scibert.

```
CUDA_VISIBLE_DEVICES=0 \
python inference.py \
        -task classification \
        -model bert \
        -max_input 1280000 \
        -test ./data/test_trec-covid-round2.jsonl \
        -vocab allenai/scibert_scivocab_uncased \
        -pretrain allenai/scibert_scivocab_uncased \
        -checkpoint ./checkpoints/reinfoselect_scibert.bin \
        -res ./results/reinfoselect_scibert.trec \
        -mode pooling \
        -max_query_len 32 \
        -max_doc_len 256 \
        -batch_size 32
```