Intro

Authors: @patrickvonplaten and @lhoestq

Aimed at tackling the knowledge-intensive NLP tasks (think tasks a human wouldn't be expected to solve without access to external knowledge sources), RAG models are seq2seq models with access to a retrieval mechanism providing relevant context documents at training and evaluation time.

A RAG model encapsulates two core components: a question encoder and a generator. During a forward pass, we encode the input with the question encoder and pass it to the retriever to extract relevant context documents. The documents are then prepended to the input. Such contextualized inputs are passed to the generator.

Read more about RAG at https://arxiv.org/abs/2005.11401.

Note

⚠ This project should be run with pytorch-lightning==1.3.1 which has a potential security vulnerability

Finetuning

Our finetuning logic is based on scripts from `examples/seq2seq`. We accept training data in the same format as specified there - we expect a directory consisting of 6 text files:

```
train.source
train.target
val.source
val.target
test.source
test.target
```

A sample finetuning command (run `./examples/research_projects/rag/finetune_rag.py --help` to list all available options):

```
python examples/research_projects/rag/finetune_rag.py \
    --data_dir $DATA_DIR \
    --output_dir $OUTPUT_DIR \
    --model_name_or_path $MODEL_NAME_OR_PATH \
    --model_type rag_sequence \
    --fp16 \
    --gpus 8
```

We publish two `base` models which can serve as a starting point for finetuning on downstream tasks (use them as `model_name_or_path`):

- `facebook/rag-sequence-base` - a base for finetuning `RagSequenceForGeneration` models,
- `facebook/rag-token-base` - a base for finetuning `RagTokenForGeneration` models.

The `base` models initialize the question encoder with `facebook/dpr-question_encoder-single-nq-base` and the generator with `facebook/bart-large`.

If you would like to initialize finetuning with a base model using different question encoder and generator architectures, you can build it with a consolidation script, e.g.:

```
python examples/research_projects/rag/consolidate_rag_checkpoint.py \
    --model_type rag_sequence \
    --generator_name_or_path facebook/bart-large-cnn \
    --question_encoder_name_or_path facebook/dpr-question_encoder-single-nq-base \
    --dest path/to/checkpoint
```

You will then be able to pass `path/to/checkpoint` as `model_name_or_path` to the `finetune_rag.py` script.

Document Retrieval

When running distributed fine-tuning, each training worker needs to retrieve contextual documents for its input by querying a index loaded into memory. RAG provides two implementations for document retrieval, one with `torch.distributed` communication package and the other with `Ray`.

This option can be configured with the `--distributed_retriever` flag which can either be set to `pytorch` or `ray`. By default this flag is set to `pytorch`.

For the Pytorch implementation, only training worker 0 loads the index into CPU memory, and a gather/scatter pattern is used to collect the inputs from the other training workers and send back the corresponding document embeddings.

For the Ray implementation, the index is loaded in *separate* process(es). The training workers randomly select which retriever worker to query. To use Ray for distributed retrieval, you have to set the `--distributed_retriever` arg to `ray`. To configure the number of retrieval workers (the number of processes that load the index), you can set the `num_retrieval_workers` flag. Also make sure to start the Ray cluster before running fine-tuning.

```
# Start a single-node Ray cluster.
ray start --head

python examples/research_projects/rag/finetune_rag.py \
    --data_dir $DATA_DIR \
    --output_dir $OUTPUT_DIR \
    --model_name_or_path $MODEL_NAME_OR_PATH \
    --model_type rag_sequence \
    --fp16 \
    --gpus 8
    --distributed_retriever ray \
    --num_retrieval_workers 4

# Stop the ray cluster once fine-tuning has finished.
ray stop
```

Using Ray can lead to retrieval speedups on multi-GPU settings since multiple processes load the index rather than just the rank 0 training worker. Using Ray also allows you to load the index on GPU since the index is loaded on a separate processes than the model, while with pytorch distributed retrieval, both are loaded in the same process potentially leading to GPU OOM.

Evaluation

Our evaluation script enables two modes of evaluation (controlled by the `eval_mode` argument): `e2e` - end2end evaluation, returns EM (exact match) and F1 scores calculated for the downstream task and `retrieval` - which returns precision@k of the documents retrieved for provided inputs.

The evaluation script expects paths to two files:

- `evaluation_set` - a path to a file specifying the evaluation dataset, a single input per line.
- `gold_data_path` - a path to a file contaning ground truth answers for datapoints from the `evaluation_set`, a single output per line. Check below for expected formats of the gold data files.

Retrieval evaluation

For `retrieval` evaluation, we expect a gold data file where each line will consist of a tab-separated list of document titles constituting positive contexts for respective datapoints from the `evaluation_set`. E.g. given a question `who sings does he love me with reba` in the `evaluation_set`, a respective ground truth line could look as follows:

```
Does He Love You        Does He Love You        Red Sandy Spika dress of Reba McEntire
Greatest Hits Volume Two (Reba McEntire album)  Shoot for the Moon (album)
```

We demonstrate how to evaluate retrieval against DPR evaluation data. You can download respective files from links listed here.

1. Download and unzip the gold data file. We use the `biencoder-nq-dev` from https://dl.fbaipublicfiles.com/dpr/data/retriever/biencoder-nq-dev.json.gz.

   ```
   wget https://dl.fbaipublicfiles.com/dpr/data/retriever/biencoder-nq-dev.json.gz && gzip -d biencoder-nq-dev.json.gz
   ```

2. Parse the unziped file using the `parse_dpr_relevance_data.py`

   ```
   mkdir output # or wherever you want to save this
   python examples/research_projects/rag/parse_dpr_relevance_data.py \
       --src_path biencoder-nq-dev.json \
       --evaluation_set output/biencoder-nq-dev.questions \
       --gold_data_path output/biencoder-nq-dev.pages
   ```

3. Run evaluation:

   ```
   python examples/research_projects/rag/eval_rag.py \
       --model_name_or_path facebook/rag-sequence-nq \
       --model_type rag_sequence \
       --evaluation_set output/biencoder-nq-dev.questions \
       --gold_data_path output/biencoder-nq-dev.pages \
       --predictions_path output/retrieval_preds.tsv  \
       --eval_mode retrieval \
       --k 1
   ```

```
# EXPLANATION
 python examples/research_projects/rag/eval_rag.py \
     --model_name_or_path facebook/rag-sequence-nq \ # model name or path of
the model we're evaluating
     --model_type rag_sequence \ # RAG model type (rag_token or rag_sequence)
     --evaluation_set output/biencoder-nq-dev.questions \ # an input dataset
for evaluation
     --gold_data_path poutput/biencoder-nq-dev.pages \ # a dataset containing
ground truth answers for samples from the evaluation_set
     --predictions_path output/retrieval_preds.tsv  \ # name of file where
predictions will be stored
     --eval_mode retrieval \ # indicates whether we're performing retrieval
evaluation or e2e evaluation
     --k 1 # parameter k for the precision@k metric
```

End-to-end evaluation

We support two formats of the gold data file (controlled by the `gold_data_mode` parameter):

- `qa` - where a single line has the following format: `input [tab] output_list` , e.g.:

```
who is the owner of reading football club       ['Xiu Li Dai', 'Dai Yongge', 'Dai
Xiuli', 'Yongge Dai']
```

- `ans` - where a single line contains a single expected answer, e.g.:

```
Xiu Li Dai
```

Predictions of the model for the samples from the `evaluation_set` will be saved under the path specified by the `predictions_path` parameter. If this path already exists, the script will use saved predictions to calculate metrics. Add `--recalculate` parameter to force the script to perform inference from scratch.

An example e2e evaluation run could look as follows:

```
python examples/research_projects/rag/eval_rag.py \
    --model_name_or_path facebook/rag-sequence-nq \
    --model_type rag_sequence \
    --evaluation_set path/to/test.source \
    --gold_data_path path/to/gold_data \
    --predictions_path path/to/e2e_preds.txt \
    --eval_mode e2e \
    --gold_data_mode qa \
    --n_docs 5 \ # You can experiment with retrieving different number of documents
at evaluation time
    --print_predictions \
    --recalculate \ # adding this parameter will force recalculating predictions
even if predictions_path already exists
```

Use your own knowledge source

By default, RAG uses the English Wikipedia as a knowledge source, known as the 'wiki_dpr' dataset. With `use_custom_knowledge_dataset.py` you can build your own knowledge source, *e.g.* for RAG.

For instance, if documents are serialized as tab-separated csv files with the columns "title" and "text", one can use `use_own_knowledge_dataset.py` as follows:

```
python examples/research_projects/rag/use_own_knowledge_dataset.py \
    --csv_path path/to/my_csv \
    --output_dir path/to/my_knowledge_dataset \
```

The created outputs in `path/to/my_knowledge_dataset` can then be used to finetune RAG as follows:

```
python examples/research_projects/rag/finetune_rag.py \
    --data_dir $DATA_DIR \
    --output_dir $OUTPUT_DIR \
    --model_name_or_path $MODEL_NAME_OR_PATH \
    --model_type rag_sequence \
    --fp16 \
    --gpus 8
    --index_name custom
    --passages_path path/to/data/my_knowledge_dataset
    --index_path path/to/my_knowledge_dataset_hnsw_index.faiss
```