Sequence-to-Sequence Training and Evaluation

This directory contains examples for finetuning and evaluating transformers on summarization and translation tasks. For deprecated `bertabs` instructions, see `bertabs/README.md`.

Supported Architectures

- `BartForConditionalGeneration`
- `MarianMTModel`
- `PegasusForConditionalGeneration`
- `MBartForConditionalGeneration`
- `FSMTForConditionalGeneration`
- `T5ForConditionalGeneration`

Download the Datasets

XSUM

```
cd examples/legacy/seq2seq
wget https://cdn-datasets.huggingface.co/summarization/xsum.tar.gz
tar -xzvf xsum.tar.gz
export XSUM_DIR=${PWD}/xsum
```

this should make a directory called `xsum/` with files like `test.source`. To use your own data, copy that files format. Each article to be summarized is on its own line.

CNN/DailyMail

```
cd examples/legacy/seq2seq
wget https://cdn-datasets.huggingface.co/summarization/cnn_dm_v2.tgz
tar -xzvf cnn_dm_v2.tgz  # empty lines removed
mv cnn_cln cnn_dm
export CNN_DIR=${PWD}/cnn_dm
```

this should make a directory called `cnn_dm/` with 6 files.

WMT16 English-Romanian Translation Data download with this command:

```
wget https://cdn-datasets.huggingface.co/translation/wmt_en_ro.tar.gz
tar -xzvf wmt_en_ro.tar.gz
export ENRO_DIR=${PWD}/wmt_en_ro
```

this should make a directory called `wmt_en_ro/` with 6 files.

WMT English-German

```
wget https://cdn-datasets.huggingface.co/translation/wmt_en_de.tgz
tar -xzvf wmt_en_de.tgz
export DATA_DIR=${PWD}/wmt_en_de
```

FSMT datasets (wmt) Refer to the scripts starting with `eval_` under: https://github.com/huggingface/transformers/tree/main/scripts/fsmt

Pegasus (multiple datasets) Multiple eval datasets are available for download from: https://github.com/stas00/porting/tree/master/datasets/pegasus

Your Data If you are using your own data, it must be formatted as one directory with 6 files:

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

The `.source` files are the input, the `.target` files are the desired output.

Potential issues

- native AMP (`--fp16` and no apex) may lead to a huge memory leak and require 10x gpu memory. This has been fixed in pytorch-nightly and the minimal official version to have this fix will be pytorch-1.7.1. Until then if you have to use mixed precision please use AMP only with pytorch-nightly or NVIDIA's apex. Reference: https://github.com/huggingface/transformers/issues/8403

Tips and Tricks

General Tips: - since you need to run from `examples/legacy/seq2seq`, and likely need to modify code, the easiest workflow is fork transformers, clone your fork, and run `pip install -e .` before you get started. - try `--freeze_encoder` or `--freeze_embeds` for faster training/larger batch size. (3hr per epoch with bs=8, see the "xsum_shared_task" command below) - `fp16_opt_level=O1` (the default works best). - In addition to the pytorch-lightning .ckpt checkpoint, a transformers checkpoint will be saved. Load it with `BartForConditionalGeneration.from_pretrained(f'{output_dir}/best_tfmr)`. - At the moment, `--do_predict` does not work in a multi-gpu setting. You need to use `evaluate_checkpoint` or the `run_eval.py` code. - This warning can be safely ignored: > "Some weights of BartForConditionalGeneration were not initialized from the model checkpoint at facebook/bart-large-xsum and are

newly initialized: ['final_logits_bias']" - Both finetuning and eval are 30% faster with `--fp16`. For that you need to install apex. - Read scripts before you run them!

Summarization Tips: - (summ) 1 epoch at batch size 1 for bart-large takes 24 hours and requires 13GB GPU RAM with fp16 on an NVIDIA-V100. - If you want to run experiments on improving the summarization finetuning process, try the XSUM Shared Task (below). It's faster to train than CNNDM because the summaries are shorter. - For CNN/DailyMail, the default `val_max_target_length` and `test_max_target_length` will truncate the ground truth labels, resulting in slightly higher rouge scores. To get accurate rouge scores, you should rerun calculate_rouge on the `{output_dir}/test_generations.txt` file saved by `trainer.test()` - `--max_target_length=60 --val_max_target_length=60 --test_max_target_length=100` is a reasonable setting for XSUM. - `wandb` can be used by specifying `--logger_name wandb`. It is useful for reproducibility. Specify the environment variable `WANDB_PROJECT='hf_xsum'` to do the XSUM shared task. - If you are finetuning on your own dataset, start from `distilbart-cnn-12-6` if you want long summaries and `distilbart-xsum-12-6` if you want short summaries. (It rarely makes sense to start from `bart-large` unless you are a researching finetuning methods).

Update 2018-07-18 Datasets: `LegacySeq2SeqDataset` will be used for all tokenizers without a `prepare_seq2seq_batch` method. Otherwise, `Seq2SeqDataset` will be used. Future work/help wanted: A new dataset to support multilingual tasks.

Fine-tuning using Seq2SeqTrainer

To use `Seq2SeqTrainer` for fine-tuning you should use the `finetune_trainer.py` script. It subclasses `Trainer` to extend it for seq2seq training. Except the `Trainer`-related `TrainingArguments`, it shares the same argument names as that of `finetune.py` file. One notable difference is that calculating generative metrics (BLEU, ROUGE) is optional and is controlled using the `--predict_with_generate` argument.

With PyTorch 1.6+ it'll automatically use `native AMP` when `--fp16` is set.

To see all the possible command line options, run:

```
python finetune_trainer.py --help
```

For multi-gpu training use `torch.distributed.launch`, e.g. with 2 gpus:

```
python -m torch.distributed.launch --nproc_per_node=2  finetune_trainer.py ...
```

At the moment, `Seq2SeqTrainer` does not support *with teacher* distillation.

All `Seq2SeqTrainer`-based fine-tuning scripts are included in the `builtin_trainer` directory.

TPU Training `Seq2SeqTrainer` supports TPU training with few caveats 1. As `generate` method does not work on TPU at the moment, `predict_with_generate` cannot be used. You should use `--prediction_loss_only` to only calculate loss, and do not set `--do_predict` and `--predict_with_generate`. 2. All sequences should be padded to be of equal length to avoid extremely slow training. (`finetune_trainer.py` does this automatically when running on TPU.)

We provide a very simple launcher script named `xla_spawn.py` that lets you run our example scripts on multiple TPU cores without any boilerplate. Just pass a `--num_cores` flag to this script, then your regular training script with its arguments (this is similar to the `torch.distributed.launch` helper for `torch.distributed`).

`builtin_trainer/finetune_tpu.sh` script provides minimal arguments needed for TPU training.

The following command fine-tunes `sshleifer/student_marian_en_ro_6_3` on TPU V3-8 and should complete one epoch in ~5-6 mins.

```
./builtin_trainer/train_distil_marian_enro_tpu.sh
```

Evaluation Commands

To create summaries for each article in dataset, we use `run_eval.py`, here are a few commands that run eval for different tasks and models. If 'translation' is in your task name, the computed metric will be BLEU. Otherwise, ROUGE will be used.

For t5, you need to specify –task translation_{src}*to*{tgt} as follows:

```
export DATA_DIR=wmt_en_ro
./run_eval.py t5-base \
    $DATA_DIR/val.source t5_val_generations.txt \
    --reference_path $DATA_DIR/val.target \
    --score_path enro_bleu.json \
    --task translation_en_to_ro \
    --n_obs 100 \
    --device cuda \
    --fp16 \
    --bs 32
```

This command works for MBART, although the BLEU score is suspiciously low.

```
export DATA_DIR=wmt_en_ro
./run_eval.py facebook/mbart-large-en-ro $DATA_DIR/val.source mbart_val_generations.txt \
    --reference_path $DATA_DIR/val.target \
    --score_path enro_bleu.json \
    --task translation \
    --n_obs 100 \
```

```
    --device cuda \
    --fp16 \
    --bs 32
```

Summarization (xsum will be very similar):

```
export DATA_DIR=cnn_dm
./run_eval.py sshleifer/distilbart-cnn-12-6 $DATA_DIR/val.source dbart_val_generations.txt \
    --reference_path $DATA_DIR/val.target \
    --score_path cnn_rouge.json \
    --task summarization \
    --n_obs 100 \

th 56 \
    --fp16 \
    --bs 32
```

Multi-GPU Evaluation

here is a command to run xsum evaluation on 8 GPUS. It is more than linearly faster than run_eval.py in some cases because it uses SortishSampler to minimize padding. You can also use it on 1 GPU. `data_dir` must have `{type_path}.source` and `{type_path}.target`. Run `./run_distributed_eval.py --help` for all clargs.

```
python -m torch.distributed.launch --nproc_per_node=8  run_distributed_eval.py \
    --model_name sshleifer/distilbart-large-xsum-12-3  \
    --save_dir xsum_generations \
    --data_dir xsum \
    --fp16  # you can pass generate kwargs like num_beams here, just like run_eval.py
```

Contributions that implement this command for other distributed hardware setups are welcome!

Single-GPU Eval: Tips and Tricks When using `run_eval.py`, the following features can be useful:

- if you running the script multiple times and want to make it easier to track what arguments produced that output, use `--dump-args`. Along with the results it will also dump any custom params that were passed to the script. For example if you used: `--num_beams 8 --early_stopping true`, the output will be: `{'bleu': 26.887, 'n_obs': 10, 'runtime': 1, 'seconds_per_sample': 0.1, 'num_beams': 8, 'early_stopping': True}`

  `--info` is an additional argument available for the same purpose of tracking the conditions of the experiment. It's useful to pass things that weren't in the argument list, e.g. a language pair `--info "lang:en-ru"`. But also if

you pass `--info` without a value it will fallback to the current date/time string, e.g. `2020-09-13 18:44:43`.

If using `--dump-args --info`, the output will be:

```
{'bleu': 26.887, 'n_obs': 10, 'runtime': 1, 'seconds_per_sample': 0.1, 'num_beams': 8,
```

If using `--dump-args --info "pair:en-ru chkpt=best`, the output will be:

```
{'bleu': 26.887, 'n_obs': 10, 'runtime': 1, 'seconds_per_sample': 0.1, 'num_beams': 8,
```

- if you need to perform a parametric search in order to find the best ones that lead to the highest BLEU score, let `run_eval_search.py` to do the searching for you.

  The script accepts the exact same arguments as `run_eval.py`, plus an additional argument `--search`. The value of `--search` is parsed, reformatted and fed to `run_eval.py` as additional args.

  The format for the `--search` value is a simple string with hparams and colon separated values to try, e.g.: `--search "num_beams=5:10 length_penalty=0.8:1.0:1.2 early_stopping=true:false"` which will generate `12 (2*3*2)` searches for a product of each hparam. For example the example that was just used will invoke `run_eval.py` repeatedly with:

  ```
  --num_beams 5 --length_penalty 0.8 --early_stopping true
  --num_beams 5 --length_penalty 0.8 --early_stopping false
  [...]
  --num_beams 10 --length_penalty 1.2 --early_stopping false
  ```

  On completion, this function prints a markdown table of the results sorted by the best BLEU score and the winning arguments.

```
bleu  | num_beams | length_penalty | early_stopping
----- | --------- | -------------- | --------------
26.71 |         5 |            1.1 |              1
26.66 |         5 |            0.9 |              1
26.66 |         5 |            0.9 |              0
26.41 |         5 |            1.1 |              0
21.94 |         1 |            0.9 |              1
21.94 |         1 |            0.9 |              0
21.94 |         1 |            1.1 |              1
21.94 |         1 |            1.1 |              0

Best score args:
stas/wmt19-en-ru data/en-ru/val.source data/en-ru/test_translations.txt --reference_path dat
```

If you pass `--info "some experiment-specific info"` it will get printed before the results table - this is useful for scripting and multiple runs, so one can

tell the different sets of results from each other.

Contributing

- follow the standard contributing guidelines and code of conduct.
- add tests to `test_seq2seq_examples.py`
- To run only the seq2seq tests, you must be in the root of the repository and run:

```
pytest examples/seq2seq/
```

Converting pytorch-lightning checkpoints

pytorch lightning `-do_predict` often fails, after you are done training, the best way to evaluate your model is to convert it.

This should be done for you, with a file called `{save_dir}/best_tfmr`.

If that file doesn't exist but you have a lightning `.ckpt` file, you can run

```
python convert_pl_checkpoint_to_hf.py PATH_TO_CKPT  randomly_initialized_hf_model_path save_
```

Then either `run_eval` or `run_distributed_eval` with `save_dir/best_tfmr` (see previous sections)

Experimental Features

These features are harder to use and not always useful.

Dynamic Batch Size for MT

`finetune.py` has a command line arg `--max_tokens_per_batch` that allows batches to be dynamically sized. This feature can only be used: - with fairseq installed - on 1 GPU - without sortish sampler - after calling `./save_len_file.py $tok $data_dir`

For example,

```
./save_len_file.py Helsinki-NLP/opus-mt-en-ro  wmt_en_ro
./dynamic_bs_example.sh --max_tokens_per_batch=2000 --output_dir benchmark_dynamic_bs
```

splits `wmt_en_ro/train` into 11,197 uneven lengthed batches and can finish 1 epoch in 8 minutes on a v100.

For comparison,

```
./dynamic_bs_example.sh --sortish_sampler --train_batch_size 48
```

uses 12,723 batches of length 48 and takes slightly more time 9.5 minutes.

The feature is still experimental, because: + we can make it much more robust if we have memory mapped/preprocessed datasets. + The speedup over sortish sampler is not that large at the moment.