## Summarization

This directory contains examples for finetuning and evaluating transformers on summarization tasks. Please tag @patil-suraj with any issues/unexpected behaviors, or send a PR! For deprecated `bertabs` instructions, see `bertabs/README.md`. For the old `finetune_trainer.py` and related utils, see `examples/legacy/seq2seq`.

### Supported Architectures

- `BartForConditionalGeneration`
- `FSMTForConditionalGeneration` (translation only)
- `MBartForConditionalGeneration`
- `MarianMTModel`
- `PegasusForConditionalGeneration`
- `T5ForConditionalGeneration`
- `MT5ForConditionalGeneration`

`run_summarization.py` is a lightweight example of how to download and preprocess a dataset from the 🤗 Datasets library or use your own files (jsonlines or csv), then fine-tune one of the architectures above on it.

For custom datasets in `jsonlines` format please see: https://huggingface.co/docs/datasets/loading_datasets.html#json-files and you also will find examples of these below.

## With Trainer

Here is an example on a summarization task:

```
python examples/pytorch/summarization/run_summarization.py \
    --model_name_or_path t5-small \
    --do_train \
    --do_eval \
    --dataset_name cnn_dailymail \
    --dataset_config "3.0.0" \
    --source_prefix "summarize: " \
    --output_dir /tmp/tst-summarization \
    --per_device_train_batch_size=4 \
    --per_device_eval_batch_size=4 \
    --overwrite_output_dir \
    --predict_with_generate
```

Only T5 models `t5-small`, `t5-base`, `t5-large`, `t5-3b` and `t5-11b` must use an additional argument: `--source_prefix "summarize: "`.

We used CNN/DailyMail dataset in this example as `t5-small` was trained on it and one can get good scores even when pre-training with a very small sample.

Extreme Summarization (XSum) Dataset is another commonly used dataset for the task of summarization. To use it replace `--dataset_name cnn_dailymail --dataset_config "3.0.0"` with `--dataset_name xsum`.

And here is how you would use it on your own files, after adjusting the values for the arguments `--train_file`, `--validation_file`, `--text_column` and `--summary_column` to match your setup:

```
python examples/pytorch/summarization/run_summarization.py \
    --model_name_or_path t5-small \
    --do_train \
    --do_eval \
    --train_file path_to_csv_or_jsonlines_file \
    --validation_file path_to_csv_or_jsonlines_file \
    --source_prefix "summarize: " \
    --output_dir /tmp/tst-summarization \
    --overwrite_output_dir \
    --per_device_train_batch_size=4 \
    --per_device_eval_batch_size=4 \
    --predict_with_generate
```

The task of summarization supports custom CSV and JSONLINES formats.

**Custom CSV Files**

If it's a csv file the training and validation files should have a column for the inputs texts and a column for the summaries.

If the csv file has just two columns as in the following example:

```
text,summary
"I'm sitting here in a boring room. It's just another rainy Sunday afternoon. I'm
wasting my time I got nothing to do. I'm hanging around I'm waiting for you. But
nothing ever happens. And I wonder","I'm sitting in a room where I'm waiting for
something to happen"
"I see trees so green, red roses too. I see them bloom for me and you. And I think
to myself what a wonderful world. I see skies so blue and clouds so white. The
bright blessed day, the dark sacred night. And I think to myself what a wonderful
world.","I'm a gardener and I'm a big fan of flowers."
"Christmas time is here. Happiness and cheer. Fun for all that children call. Their
favorite time of the year. Snowflakes in the air. Carols everywhere. Olden times and
ancient rhymes. Of love and dreams to share","It's that time of year again."
```

The first column is assumed to be for `text` and the second is for summary.

If the csv file has multiple columns, you can then specify the names of the columns to use:

```
    --text_column text_column_name \
    --summary_column summary_column_name \
```

For example if the columns were:

```
id,date,text,summary
```

and you wanted to select only `text` and `summary`, then you'd pass these additional arguments:

```
    --text_column text \
    --summary_column summary \
```

**Custom JSONLINES Files**

The second supported format is jsonlines. Here is an example of a jsonlines custom data file.

```
{"text": "I'm sitting here in a boring room. It's just another rainy Sunday
afternoon. I'm wasting my time I got nothing to do. I'm hanging around I'm waiting
for you. But nothing ever happens. And I wonder", "summary": "I'm sitting in a room
where I'm waiting for something to happen"}
{"text": "I see trees so green, red roses too. I see them bloom for me and you. And
I think to myself what a wonderful world. I see skies so blue and clouds so white.
The bright blessed day, the dark sacred night. And I think to myself what a
wonderful world.", "summary": "I'm a gardener and I'm a big fan of flowers."}
{"text": "Christmas time is here. Happiness and cheer. Fun for all that children
call. Their favorite time of the year. Snowflakes in the air. Carols everywhere.
Olden times and ancient rhymes. Of love and dreams to share", "summary": "It's that
time of year again."}
```

Same as with the CSV files, by default the first value will be used as the text record and the second as the summary record. Therefore you can use any key names for the entries, in this example `text` and `summary` were used.

And as with the CSV files, you can specify which values to select from the file, by explicitly specifying the corresponding key names. In our example this again would be:

```
    --text_column text \
    --summary_column summary \
```

## With Accelerate

Based on the script `run_summarization_no_trainer.py` .

Like `run_summarization.py` , this script allows you to fine-tune any of the models supported on a summarization task, the main difference is that this script exposes the bare training loop, to allow you to quickly experiment and add any customization you would like.

It offers less options than the script with `Trainer` (for instance you can easily change the options for the optimizer or the dataloaders directly in the script) but still run in a distributed setup, on TPU and supports mixed precision by the mean of the 🤗 `Accelerate` library. You can use the script normally after installing it:

```
pip install accelerate
```

then

```
python run_summarization_no_trainer.py \
    --model_name_or_path t5-small \
    --dataset_name cnn_dailymail \
    --dataset_config "3.0.0" \
    --source_prefix "summarize: " \
    --output_dir ~/tmp/tst-summarization
```

You can then use your usual launchers to run in it in a distributed environment, but the easiest way is to run

```
accelerate config
```

and reply to the questions asked. Then

```
accelerate test
```

that will check everything is ready for training. Finally, you can launch training with

```
accelerate launch run_summarization_no_trainer.py \
    --model_name_or_path t5-small \
    --dataset_name cnn_dailymail \
    --dataset_config "3.0.0" \
    --source_prefix "summarize: " \
    --output_dir ~/tmp/tst-summarization
```

This command is the same and will work for:

- a CPU-only setup
- a setup with one GPU
- a distributed training with several GPUs (single or multi node)
- a training on TPUs

Note that this library is in alpha release so your feedback is more than welcome if you encounter any problem using it.