Translation

This directory contains examples for finetuning and evaluating transformers on translation 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_translation.py` is a lightweight examples 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 of a translation fine-tuning with a MarianMT model:

```
python examples/pytorch/translation/run_translation.py \
    --model_name_or_path Helsinki-NLP/opus-mt-en-ro \
    --do_train \
    --do_eval \
    --source_lang en \
    --target_lang ro \
    --dataset_name wmt16 \
    --dataset_config_name ro-en \
    --output_dir /tmp/tst-translation \
    --per_device_train_batch_size=4 \
    --per_device_eval_batch_size=4 \
    --overwrite_output_dir \
    --predict_with_generate
```

MBart and some T5 models require special handling.

T5 models `t5-small`, `t5-base`, `t5-large`, `t5-3b` and `t5-11b` must use an additional argument: `--source_prefix "translate {source_lang} to {target_lang}"`. For example:

```
python examples/pytorch/translation/run_translation.py \
    --model_name_or_path t5-small \
    --do_train \
    --do_eval \
    --source_lang en \
    --target_lang ro \
    --source_prefix "translate English to Romanian: " \
    --dataset_name wmt16 \
    --dataset_config_name ro-en \
    --output_dir /tmp/tst-translation \
    --per_device_train_batch_size=4 \
    --per_device_eval_batch_size=4 \
    --overwrite_output_dir \
    --predict_with_generate
```

If you get a terrible BLEU score, make sure that you didn't forget to use the `--source_prefix` argument.

For the aforementioned group of T5 models it's important to remember that if you switch to a different language pair, make sure to adjust the source and target values in all 3 language-specific command line argument: `--source_lang`, `--target_lang` and `--source_prefix`.

MBart models require a different format for `--source_lang` and `--target_lang` values, e.g. instead of `en` it expects `en_XX`, for `ro` it expects `ro_RO`. The full MBart specification for language codes can be found here. For example:

```
python examples/pytorch/translation/run_translation.py \
    --model_name_or_path facebook/mbart-large-en-ro  \
    --do_train \
    --do_eval \
    --dataset_name wmt16 \
    --dataset_config_name ro-en \
    --source_lang en_XX \
    --target_lang ro_RO \
    --output_dir /tmp/tst-translation \
    --per_device_train_batch_size=4 \
    --per_device_eval_batch_size=4 \
    --overwrite_output_dir \
    --predict_with_generate
```

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

```
python examples/pytorch/translation/run_translation.py \
    --model_name_or_path t5-small \
    --do_train \
    --do_eval \
    --source_lang en \
    --target_lang ro \
    --source_prefix "translate English to Romanian: " \
    --dataset_name wmt16 \
    --dataset_config_name ro-en \
    --train_file path_to_jsonlines_file \
    --validation_file path_to_jsonlines_file \
    --output_dir /tmp/tst-translation \
    --per_device_train_batch_size=4 \
    --per_device_eval_batch_size=4 \
    --overwrite_output_dir \
    --predict_with_generate
```

The task of translation supports only custom JSONLINES files, with each line being a dictionary with a key `"translation"` and its value another dictionary whose keys is the language pair. For example:

```
{ "translation": { "en": "Others have dismissed him as a joke.", "ro": "Alții l-au numit o glumă." } }
{ "translation": { "en": "And some are holding out for an implosion.", "ro": "Iar alții așteaptă implozia." } }
```

Here the languages are Romanian (`ro`) and English (`en`).

If you want to use a pre-processed dataset that leads to high BLEU scores, but for the `en-de` language pair, you can use `--dataset_name stas/wmt14-en-de-pre-processed` , as following:

```
python examples/pytorch/translation/run_translation.py \
    --model_name_or_path t5-small \
    --do_train \
    --do_eval \
    --source_lang en \
    --target_lang de \
    --source_prefix "translate English to German: " \
    --dataset_name stas/wmt14-en-de-pre-processed \
    --output_dir /tmp/tst-translation \
    --per_device_train_batch_size=4 \
    --per_device_eval_batch_size=4 \
    --overwrite_output_dir \
    --predict_with_generate
```

With Accelerate

Based on the script `run_translation_no_trainer.py` .

Like `run_translation.py` , this script allows you to fine-tune any of the models supported on a translation 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_translation_no_trainer.py \
    --model_name_or_path Helsinki-NLP/opus-mt-en-ro \
    --source_lang en \
    --target_lang ro \
    --dataset_name wmt16 \
    --dataset_config_name ro-en \
    --output_dir ~/tmp/tst-translation
```

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_translation_no_trainer.py \
    --model_name_or_path Helsinki-NLP/opus-mt-en-ro \
    --source_lang en \
    --target_lang ro \
    --dataset_name wmt16 \
    --dataset_config_name ro-en \
    --output_dir ~/tmp/tst-translation
```

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.