Text classification examples

GLUE tasks

Based on the script `run_glue.py`.

Fine-tuning the library models for sequence classification on the GLUE benchmark: General Language Understanding Evaluation. This script can fine-tune any of the models on the hub and can also be used for a dataset hosted on our hub or your own data in a csv or a JSON file (the script might need some tweaks in that case, refer to the comments inside for help).

GLUE is made up of a total of 9 different tasks. Here is how to run the script on one of them:

```
export TASK_NAME=mrpc

python run_glue.py \
  --model_name_or_path bert-base-cased \
  --task_name $TASK_NAME \
  --do_train \
  --do_eval \
  --max_seq_length 128 \
  --per_device_train_batch_size 32 \
  --learning_rate 2e-5 \
  --num_train_epochs 3 \
  --output_dir /tmp/$TASK_NAME/
```

where task name can be one of cola, sst2, mrpc, stsb, qqp, mnli, qnli, rte, wnli.

We get the following results on the dev set of the benchmark with the previous commands (with an exception for MRPC and WNLI which are tiny and where we used 5 epochs instead of 3). Trainings are seeded so you should obtain the same results with PyTorch 1.6.0 (and close results with different versions), training times are given for information (a single Titan RTX was used):

Task	Metric	Result	Training time
CoLA	Matthews corr	56.53	3:17
SST-2	Accuracy	92.32	26:06
MRPC	F1/Accuracy	88.85/84.07	2:21
STS-B	Pearson/Spearman corr.	88.64/88.48	2:13
QQP	Accuracy/F1	90.71/87.49	2:22:26
MNLI	Matched acc./Mismatched acc.	83.91/84.10	2:35:23
QNLI	Accuracy	90.66	40:57
RTE	Accuracy	65.70	57
WNLI	Accuracy	56.34	24

Some of these results are significantly different from the ones reported on the test set of GLUE benchmark on the website. For QQP and WNLI, please refer to FAQ #12 on the website.

The following example fine-tunes BERT on the `imdb` dataset hosted on our hub:

```
python run_glue.py \
  --model_name_or_path bert-base-cased \
  --dataset_name imdb  \
  --do_train \
  --do_predict \
  --max_seq_length 128 \
  --per_device_train_batch_size 32 \
  --learning_rate 2e-5 \
  --num_train_epochs 3 \
  --output_dir /tmp/imdb/
```

Mixed precision training

If you have a GPU with mixed precision capabilities (architecture Pascal or more recent), you can use mixed precision training with PyTorch 1.6.0 or latest, or by installing the Apex library for previous versions. Just add the flag `--fp16` to your command launching one of the scripts mentioned above!

Using mixed precision training usually results in 2x-speedup for training with the same final results:

Task	Metric	Result	Training time	Result (FP16)	Training time (FP16)
CoLA	Matthews corr	56.53	3:17	56.78	1:41
SST-2	Accuracy	92.32	26:06	91.74	13:11
MRPC	F1/Accuracy	88.85/84.07	2:21	88.12/83.58	1:10
STS-B	Pearson/Spearman corr.	88.64/88.48	2:13	88.71/88.55	1:08
QQP	Accuracy/F1	90.71/87.49	2:22:26	90.67/87.43	1:11:54
MNLI	Matched acc./Mismatched acc.	83.91/84.10	2:35:23	84.04/84.06	1:17:06
QNLI	Accuracy	90.66	40:57	90.96	20:16
RTE	Accuracy	65.70	57	65.34	29
WNLI	Accuracy	56.34	24	56.34	12

PyTorch version, no Trainer

Based on the script `run_glue_no_trainer.py`.

Like `run_glue.py`, this script allows you to fine-tune any of the models on the hub on a text classification task, either a GLUE task or your own data in a csv or a JSON file. 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

```
export TASK_NAME=mrpc

python run_glue_no_trainer.py \
  --model_name_or_path bert-base-cased \
  --task_name $TASK_NAME \
  --max_length 128 \
  --per_device_train_batch_size 32 \
  --learning_rate 2e-5 \
  --num_train_epochs 3 \
  --output_dir /tmp/$TASK_NAME/
```

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

```
export TASK_NAME=mrpc

accelerate launch run_glue_no_trainer.py \
  --model_name_or_path bert-base-cased \
  --task_name $TASK_NAME \
  --max_length 128 \
  --per_device_train_batch_size 32 \
  --learning_rate 2e-5 \
  --num_train_epochs 3 \
  --output_dir /tmp/$TASK_NAME/
```

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.

XNLI

Based on the script `run_xnli.py`.

XNLI is a crowd-sourced dataset based on MultiNLI. It is an evaluation benchmark for cross-lingual text representations. Pairs of text are labeled with textual entailment annotations for 15 different languages (including both high-resource language such as English and low-resource languages such as Swahili).

Fine-tuning on XNLI

This example code fine-tunes mBERT (multi-lingual BERT) on the XNLI dataset. It runs in 106 mins on a single tesla V100 16GB.

```
python run_xnli.py \
  --model_name_or_path bert-base-multilingual-cased \
  --language de \
  --train_language en \
  --do_train \
  --do_eval \
  --per_device_train_batch_size 32 \
  --learning_rate 5e-5 \
  --num_train_epochs 2.0 \
  --max_seq_length 128 \
  --output_dir /tmp/debug_xnli/ \
  --save_steps -1
```

Training with the previously defined hyper-parameters yields the following results on the **test** set:

```
acc = 0.7093812375249501
```