Text classification examples

This folder contains some scripts showing examples of *text classification* with the 🤗 Transformers library. For straightforward use-cases you may be able to use these scripts without modification, although we have also included comments in the code to indicate areas that you may need to adapt to your own projects.

run_text_classification.py

This script handles perhaps the single most common use-case for this entire library: Training an NLP classifier on your own training data. This can be whatever you want - you could classify text as abusive/hateful or allowable, or forum posts as spam or not-spam, or classify the genre of a headline as politics, sports or any number of other categories. Any task that involves classifying natural language into two or more different categories can work with this! You can even do regression, such as predicting the score on a 1-10 scale that a user gave, given the text of their review.

The preferred input format is either a CSV or newline-delimited JSON file that contains a `sentence1` and `label` field. If your task involves comparing two texts (for example, if your classifier is deciding whether two sentences are paraphrases of each other, or were written by the same author) then you should also include a `sentence2` field in each example. If you do not have a `sentence1` field then the script will assume the non-label fields are the input text, which may not always be what you want, especially if you have more than two fields!

Here is a snippet of a valid input JSON file, though note that your texts can be much longer than these, and are not constrained (despite the field name) to being single grammatical sentences:

```
{"sentence1": "COVID-19 vaccine updates: How is the rollout proceeding?", "label": "news"}
{"sentence1": "Manchester United celebrates Europa League success", "label": "sports"}
```

Usage notes

If your inputs are long (more than ~60-70 words), you may wish to increase the `--max_seq_length` argument beyond the default value of 128. The maximum supported value for most models is 512 (about 200-300 words), and some can handle even longer. This will come at a cost in runtime and memory use, however.

We assume that your labels represent *categories*, even if they are integers, since text classification is a much more common task than text regression. If your labels are floats, however, the script will assume you want to do regression. This is something you can edit yourself if your use-case requires it!

After training, the model will be saved to `--output_dir`. Once your model is trained, you can get predictions by calling the script without a `--train_file` or `--validation_file`; simply pass it the output_dir containing the trained model and a `--test_file` and it will write its predictions to a text file for you.

Multi-GPU and TPU usage

By default, the script uses a `MirroredStrategy` and will use multiple GPUs effectively if they are available. TPUs can also be used by passing the name of the TPU resource with the `--tpu` argument.

Memory usage and data loading

One thing to note is that all data is loaded into memory in this script. Most text classification datasets are small enough that this is not an issue, but if you have a very large dataset you will need to modify the script to handle data streaming. This is particularly challenging for TPUs, given the stricter requirements and the sheer volume of data required to keep them fed. A full explanation of all the possible pitfalls is a bit beyond this example script and README, but for more information you can see the 'Input Datasets' section of this document.

Example command

```
python run_text_classification.py \
--model_name_or_path distilbert-base-cased \
--train_file training_data.json \
--validation_file validation_data.json \
--output_dir output/ \
--test_file data_to_predict.json
```

run_glue.py

This script handles training on the GLUE dataset for various text classification and regression tasks. The GLUE datasets will be loaded automatically, so you only need to specify the task you want (with the `--task_name` argument). You can also supply your own files for prediction with the `--predict_file` argument, for example if you want to train a model on GLUE for e.g. paraphrase detection and then predict whether your own data contains paraphrases or not. Please ensure the names of your input fields match the names of the features in the relevant GLUE dataset - you can see a list of the column names in the `task_to_keys` dict in the `run_glue.py` file.

Usage notes

The `--do_train`, `--do_eval` and `--do_predict` arguments control whether training, evaluations or predictions are performed. After training, the model will be saved to `--output_dir`. Once your model is trained, you can call the script without the `--do_train` or `--do_eval` arguments to quickly get predictions from your saved model.

Multi-GPU and TPU usage

By default, the script uses a `MirroredStrategy` and will use multiple GPUs effectively if they are available. TPUs can also be used by passing the name of the TPU resource with the `--tpu` argument.

Memory usage and data loading

One thing to note is that all data is loaded into memory in this script. Most text classification datasets are small enough that this is not an issue, but if you have a very large dataset you will need to modify the script to handle data streaming. This is particularly challenging for TPUs, given the stricter requirements and the sheer volume of data required to keep them fed. A full explanation of all the possible pitfalls is a bit beyond this example script and README, but for more information you can see the 'Input Datasets' section of this document.

Example command

```
python run_glue.py \
--model_name_or_path distilbert-base-cased \
--task_name mnli \
--do_train \
--do_eval \
--do_predict \
--predict_file data_to_predict.json
```