Token classification

PyTorch version

Fine-tuning the library models for token classification task such as Named Entity Recognition (NER), Parts-of-speech tagging (POS) or phrase extraction (CHUNKS). The main scrip `run_ner.py` leverages the 🤗 Datasets library and the Trainer API. You can easily customize it to your needs if you need extra processing on your datasets.

It will either run on a datasets hosted on our hub or with your own text files for training and validation, you might just need to add some tweaks in the data preprocessing.

The following example fine-tunes BERT on CoNLL-2003:

```
python run_ner.py \
  --model_name_or_path bert-base-uncased \
  --dataset_name conll2003 \
  --output_dir /tmp/test-ner \
  --do_train \
  --do_eval
```

or just can just run the bash script `run.sh`.

To run on your own training and validation files, use the following command:

```
python run_ner.py \
  --model_name_or_path bert-base-uncased \
  --train_file path_to_train_file \
  --validation_file path_to_validation_file \
  --output_dir /tmp/test-ner \
  --do_train \
  --do_eval
```

Note: This script only works with models that have a fast tokenizer (backed by the 🤗 Tokenizers library) as it uses special features of those tokenizers. You can check if your favorite model has a fast tokenizer in this table, if it doesn't you can still use the old version of the script.

Old version of the script

You can find the old version of the PyTorch script here.

Pytorch version, no Trainer

Based on the script run_ner_no_trainer.py.

Like `run_ner.py`, this script allows you to fine-tune any of the models on the hub on a token classification task, either NER, POS or CHUNKS tasks 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=ner

python run_ner_no_trainer.py \
  --model_name_or_path bert-base-cased \
  --dataset_name conll2003 \
  --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=ner

accelerate launch run_ner_no_trainer.py \
  --model_name_or_path bert-base-cased \
  --dataset_name conll2003 \
  --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.