Token classification

Based on the scripts `run_ner.py`.

The following examples are covered in this section:

- NER on the GermEval 2014 (German NER) dataset
- Emerging and Rare Entities task: WNUT'17 (English NER) dataset

Details and results for the fine-tuning provided by @stefan-it.

GermEval 2014 (German NER) dataset

Data (Download and pre-processing steps)

Data can be obtained from the GermEval 2014 shared task page.

Here are the commands for downloading and pre-processing train, dev and test datasets. The original data format has four (tab-separated) columns, in a pre-processing step only the two relevant columns (token and outer span NER annotation) are extracted:

```
curl -L 'https://drive.google.com/uc?
export=download&id=1Jjhbal535VVz2ap4v4r_rN1UEHTdLK5P' \
| grep -v "^#" | cut -f 2,3 | tr '\t' ' ' > train.txt.tmp
curl -L 'https://drive.google.com/uc?
export=download&id=1ZfRcQThdtAR5PPRjIDtrVP7BtXSCUBbm' \
| grep -v "^#" | cut -f 2,3 | tr '\t' ' ' > dev.txt.tmp
curl -L 'https://drive.google.com/uc?
export=download&id=1u9mb7kNJHWQCWyweMDRMuTFoOHOfeBTH' \
| grep -v "^#" | cut -f 2,3 | tr '\t' ' ' > test.txt.tmp
```

The GermEval 2014 dataset contains some strange "control character" tokens like `'\x96', '\u200e', '\x95', '\xad' or '\x80'`. One problem with these tokens is, that `BertTokenizer` returns an empty token for them, resulting in misaligned `InputExample`s. The `preprocess.py` script located in the `scripts` folder a) filters these tokens and b) splits longer sentences into smaller ones (once the max. subtoken length is reached).

Let's define some variables that we need for further pre-processing steps and training the model:

```
export MAX_LENGTH=128
export BERT_MODEL=bert-base-multilingual-cased
```

Run the pre-processing script on training, dev and test datasets:

```
python3 scripts/preprocess.py train.txt.tmp $BERT_MODEL $MAX_LENGTH > train.txt
python3 scripts/preprocess.py dev.txt.tmp $BERT_MODEL $MAX_LENGTH > dev.txt
python3 scripts/preprocess.py test.txt.tmp $BERT_MODEL $MAX_LENGTH > test.txt
```

The GermEval 2014 dataset has much more labels than CoNLL-2002/2003 datasets, so an own set of labels must be used:

```
cat train.txt dev.txt test.txt | cut -d " " -f 2 | grep -v "^$"| sort | uniq >
labels.txt
```

Prepare the run

Additional environment variables must be set:

```
export OUTPUT_DIR=germeval-model
export BATCH_SIZE=32
export NUM_EPOCHS=3
export SAVE_STEPS=750
export SEED=1
```

Run the Pytorch version

To start training, just run:

```
python3 run_ner.py --data_dir ./ \
--labels ./labels.txt \
--model_name_or_path $BERT_MODEL \
--output_dir $OUTPUT_DIR \
--max_seq_length  $MAX_LENGTH \
--num_train_epochs $NUM_EPOCHS \
--per_device_train_batch_size $BATCH_SIZE \
--save_steps $SAVE_STEPS \
--seed $SEED \
--do_train \
--do_eval \
--do_predict
```

If your GPU supports half-precision training, just add the `--fp16` flag. After training, the model will be both evaluated on development and test datasets.

JSON-based configuration file

Instead of passing all parameters via commandline arguments, the `run_ner.py` script also supports reading parameters from a json-based configuration file:

```
{
    "data_dir": ".",
    "labels": "./labels.txt",
    "model_name_or_path": "bert-base-multilingual-cased",
    "output_dir": "germeval-model",
    "max_seq_length": 128,
    "num_train_epochs": 3,
    "per_device_train_batch_size": 32,
    "save_steps": 750,
    "seed": 1,
    "do_train": true,
    "do_eval": true,
    "do_predict": true
}
```

It must be saved with a `.json` extension and can be used by running `python3 run_ner.py config.json`.

Evaluation

Evaluation on development dataset outputs the following for our example:

```
10/04/2019 00:42:06 - INFO - __main__ -   ***** Eval results  *****
10/04/2019 00:42:06 - INFO - __main__ -     f1 = 0.8623348017621146
10/04/2019 00:42:06 - INFO - __main__ -     loss = 0.07183869666975543
10/04/2019 00:42:06 - INFO - __main__ -     precision = 0.8467916366258111
10/04/2019 00:42:06 - INFO - __main__ -     recall = 0.8784592370979806
```

On the test dataset the following results could be achieved:

```
10/04/2019 00:42:42 - INFO - __main__ -   ***** Eval results  *****
10/04/2019 00:42:42 - INFO - __main__ -     f1 = 0.8614389652384803
10/04/2019 00:42:42 - INFO - __main__ -     loss = 0.07064602487454782
10/04/2019 00:42:42 - INFO - __main__ -     precision = 0.8604651162790697
10/04/2019 00:42:42 - INFO - __main__ -     recall = 0.8624150210424085
```

Run the Tensorflow 2 version

To start training, just run:

```
python3 run_tf_ner.py --data_dir ./ \
--labels ./labels.txt \
--model_name_or_path $BERT_MODEL \
--output_dir $OUTPUT_DIR \
--max_seq_length  $MAX_LENGTH \
--num_train_epochs $NUM_EPOCHS \
--per_device_train_batch_size $BATCH_SIZE \
--save_steps $SAVE_STEPS \
--seed $SEED \
--do_train \
--do_eval \
--do_predict
```

Such as the Pytorch version, if your GPU supports half-precision training, just add the `--fp16` flag. After training, the model will be both evaluated on development and test datasets.

Evaluation

Evaluation on development dataset outputs the following for our example:

```
           precision    recall  f1-score   support

 LOCderiv     0.7619    0.6154    0.6809        52
  PERpart     0.8724    0.8997    0.8858      4057
  OTHpart     0.9360    0.9466    0.9413       711
  ORGpart     0.7015    0.6989    0.7002       269
  LOCpart     0.7668    0.8488    0.8057       496
```

```
      LOC     0.8745    0.9191    0.8963       235
 ORGderiv     0.7723    0.8571    0.8125        91
 OTHderiv     0.4800    0.6667    0.5581        18
      OTH     0.5789    0.6875    0.6286        16
 PERderiv     0.5385    0.3889    0.4516        18
      PER     0.5000    0.5000    0.5000         2
      ORG     0.0000    0.0000    0.0000         3

micro avg     0.8574    0.8862    0.8715      5968
macro avg     0.8575    0.8862    0.8713      5968
```

On the test dataset the following results could be achieved:

```
           precision    recall  f1-score   support

  PERpart     0.8847    0.8944    0.8896      9397
  OTHpart     0.9376    0.9353    0.9365      1639
  ORGpart     0.7307    0.7044    0.7173       697
      LOC     0.9133    0.9394    0.9262       561
  LOCpart     0.8058    0.8157    0.8107      1150
      ORG     0.0000    0.0000    0.0000         8
 OTHderiv     0.5882    0.4762    0.5263        42
 PERderiv     0.6571    0.5227    0.5823        44
      OTH     0.4906    0.6667    0.5652        39
 ORGderiv     0.7016    0.7791    0.7383       172
 LOCderiv     0.8256    0.6514    0.7282       109
      PER     0.0000    0.0000    0.0000        11

micro avg     0.8722    0.8774    0.8748     13869
macro avg     0.8712    0.8774    0.8740     13869
```

Emerging and Rare Entities task: WNUT'17 (English NER) dataset

Description of the WNUT'17 task from the shared task website:

> *The WNUT'17 shared task focuses on identifying unusual, previously-unseen entities in the context of emerging discussions. Named entities form the basis of many modern approaches to other tasks (like event clustering and summarization), but recall on them is a real problem in noisy text - even among annotators. This drop tends to be due to novel entities and surface forms.*

Six labels are available in the dataset. An overview can be found on this page.

Data (Download and pre-processing steps)

The dataset can be downloaded from the official GitHub repository.

The following commands show how to prepare the dataset for fine-tuning:

```
mkdir -p data_wnut_17

curl -L
'https://github.com/leondz/emerging_entities_17/raw/master/wnut17train.conll'  | tr
```

```
'\t' ' ' > data_wnut_17/train.txt.tmp
curl -L
'https://github.com/leondz/emerging_entities_17/raw/master/emerging.dev.conll' | tr
'\t' ' ' > data_wnut_17/dev.txt.tmp
curl -L
'https://raw.githubusercontent.com/leondz/emerging_entities_17/master/emerging.test.ani
 | tr '\t' ' ' > data_wnut_17/test.txt.tmp
```

Let's define some variables that we need for further pre-processing steps:

```
export MAX_LENGTH=128
export BERT_MODEL=bert-large-cased
```

Here we use the English BERT large model for fine-tuning. The `preprocess.py` scripts splits longer sentences into smaller ones (once the max. subtoken length is reached):

```
python3 scripts/preprocess.py data_wnut_17/train.txt.tmp $BERT_MODEL $MAX_LENGTH >
data_wnut_17/train.txt
python3 scripts/preprocess.py data_wnut_17/dev.txt.tmp $BERT_MODEL $MAX_LENGTH >
data_wnut_17/dev.txt
python3 scripts/preprocess.py data_wnut_17/test.txt.tmp $BERT_MODEL $MAX_LENGTH >
data_wnut_17/test.txt
```

In the last pre-processing step, the `labels.txt` file needs to be generated. This file contains all available labels:

```
cat data_wnut_17/train.txt data_wnut_17/dev.txt data_wnut_17/test.txt | cut -d " " -
f 2 | grep -v "^$"| sort | uniq > data_wnut_17/labels.txt
```

Run the Pytorch version

Fine-tuning with the PyTorch version can be started using the `run_ner.py` script. In this example we use a JSON-based configuration file.

This configuration file looks like:

```
{
    "data_dir": "./data_wnut_17",
    "labels": "./data_wnut_17/labels.txt",
    "model_name_or_path": "bert-large-cased",
    "output_dir": "wnut-17-model-1",
    "max_seq_length": 128,
    "num_train_epochs": 3,
    "per_device_train_batch_size": 32,
    "save_steps": 425,
    "seed": 1,
    "do_train": true,
    "do_eval": true,
    "do_predict": true,
    "fp16": false
}
```

If your GPU supports half-precision training, please set `fp16` to `true`.

Save this JSON-based configuration under `wnut_17.json`. The fine-tuning can be started with `python3 run_ner_old.py wnut_17.json`.

Evaluation

Evaluation on development dataset outputs the following:

```
05/29/2020 23:33:44 - INFO - __main__ -   ***** Eval results *****
05/29/2020 23:33:44 - INFO - __main__ -     eval_loss = 0.26505235286212275
05/29/2020 23:33:44 - INFO - __main__ -     eval_precision = 0.7008264462809918
05/29/2020 23:33:44 - INFO - __main__ -     eval_recall = 0.507177033492823
05/29/2020 23:33:44 - INFO - __main__ -     eval_f1 = 0.5884802220680084
05/29/2020 23:33:44 - INFO - __main__ -     epoch = 3.0
```

On the test dataset the following results could be achieved:

```
05/29/2020 23:33:44 - INFO - transformers.trainer -   ***** Running Prediction *****
05/29/2020 23:34:02 - INFO - __main__ -     eval_loss = 0.30948806500973547
05/29/2020 23:34:02 - INFO - __main__ -     eval_precision = 0.5840108401084011
05/29/2020 23:34:02 - INFO - __main__ -     eval_recall = 0.3994439295644115
05/29/2020 23:34:02 - INFO - __main__ -     eval_f1 = 0.47440836543753434
```

WNUT'17 is a very difficult task. Current state-of-the-art results on this dataset can be found here.