The OneIE Package

v0.4.8

October 9, 2021

Contents

1 Requirements

The following packages are required by OneIE.

- **Python 3.7**: Some packages (e.g., transformers 3.0.2) or interfaces are not compatible with other Python versions.
- Python packages:
  - **PyTorch 1.0**: Install the CPU version if you use this tool on a machine without GPUs.
  - **transformers 3.0.2**: Transformers 3.1+ may cause some model loading issue.
  - **tqdm**: TQDM is used to display progress bars.
  - **lxml**: LXML is used to read XML files.
  - **nltk**: NLTK is used by the pre-processing scripts.

☞ If you get a "Resource punkt not found" error, download the `punkt` data using:

```
$ conda activate <your_env_name> // If you run OneIE in a Conda environment
$ python

>>> import nltk
>>> nltk.download("punkt")
```

☞ The `transformers` version is wrong if you get this error:

```
RuntimeError: Error(s) in loading state_dict for OneIE:
    Missing key(s) in state_dict: "bert.embeddings.position_ids".
```

☞ The Python version is likely to be wrong if you see an error caused by some unexpected keyword argument for a function from the Python standard library, such as

```
Traceback (most recent call last):
  File "predict.py", line 15, in <module>
    from data import IEDatasetEval
  File "/home/oneie_v0.4.8/data.py", line 34, in <module>
    defaults=[None] * len(instance_fields))
TypeError: namedtuple() got an unexpected keyword argument 'defaults'
```

2 Data Format

OneIE uses different file formats for training and evaluation.

- To train a new model from scratch, you need to prepare your own data in the JSON format described in Section 2.1 Training Data Format.
- For prediction purposes, you need to prepare your test data in plain text or LTF format described in Section 2.2 Evaluation Data Format.

2.1 Training Data Format

☞ You may skip this section if you use preprocessing scripts provided in this package to generate ACE or ERE training data.

The training data is encoded in inline JSON format, where each line in the file is a single training instance. The following example shows a training instance. Note that this example is prettyprinted, and in the actual file it should be compressed into a single line.


```
{
    "doc_id": "DOC001",
    "sent_id": "DOC001-2",
    "tokens": ["Blackboard", "to", "merge", "with", "Anthology"],
    "pieces": ["Black", "##board", "to", "merge", "with", "Anthology"],
    "token_lens": [2, 1, 1, 1, 1],
    "sentence": "Blackboard to merge with Anthology",
    "entity_mentions": [
        {
            "id": "DOC001-2-E1",
            "text": "Blackboard",
            "entity_type": "ORG",
            "entity_subtype": "Company",
            "mention_type": "NAM",
            "start": 0,
            "end": 1
        },
        {
            "id": "DOC001-2-E2",
            "text": "Anthology",
            "entity_type": "ORG",
            "entity_subtype": "Company",
            "mention_type": "NAM",
            "start": 4,
            "end": 5
        }
    ],
    "relation_mentions": [],
    "event_mentions": [
        {
            "id": "DOC001-2-EV1",
            "event_type": "BUSINESS:MERGE-ORG",
            "trigger": {
                "text": "merge",
                "start": 2,
                "end": 3
            },
            "arguments": [
                {
                    "entity_id": "DOC001-2-E1",
                    "text": "Blackboard",
                    "role": "Org"
                },
                {
```

```
                    "entity_id": "DOC001-2-E2",
                    "text": "Anthology",
                    "role": "Org"
                }
            ]
        }
    ]
}
```

Each training example has the following fields:

- `doc_id`: Document ID.
- `sent_id`: Unique sentence ID.
- `tokens`: A list of tokens.
- `pieces`: A list of wordpieces. The tokenizer used to split tokens into wordpieces and the language model used by OneIE should match. For example, if `bert-base-cased` is used in OneIE, the training data should also be tokenized using `bert-base-cased`'s tokenizer. Using other tokenizers (e.g., `bert-base-uncased`, `roberta-large`) may degrade the performance or cause unexpected issues.
- `token_lens`: A list of token lengths. The length of a token is the number of wordpieces it contains.
- `sentence`: Raw text of the sentence.
- `entity_mentions`: A list of entity mentions.
  - `id`: Unique entity mention ID.
  - `text`: Surface form of the mention.
  - `entity_type`: Entity type.
  - `entity_subtype`: Entity subtype.
  - `mention_type`: Mention type (NAM, NOM, or PRO).
  - `start`: Start token index.
  - `end`: End token index + 1.
- `relation_mentions`: A list of relation mentions.
  - `id`: Unique relation ID.
  - `relation_type`: Relation type.
  - `relation_subtype`: Relation subtype.
  - `arguments`: A list of (two) relation arguments.
    - `entity_id`: Entity mention ID.
    - `text`: Surface form of the entity mention.
    - `role`: Role in the relation.
- `event_mentions`: A list of event mentions.
  - `id`: Unique event ID.
  - `event_type`: Event type.
  - `trigger`: Event trigger.
    - `text`: Text of the trigger.
    - `start`: Start token index.
    - `end`: End token index + 1.
  - `arguments`: A list of event arguments.
    - `entity_id`: Entity mention ID.
    - `text`: Surface form of the entity mention.

* `role`: Role of the event argument.

☞ To convert your own data to OneIE's training format, you may reuse some classes and functions defined in `preprocessing/process_ace.py`, such as `Entity`, `Relation`, `Event`, `Sentence`, `Document`, and `convert_to_oneie()`.

In this package, we also provide preprocessing scripts to generate training data of the following datasets mentioned in our paper.

- ACE05-R and ACE05-E: These two datasets are used in [1]. To generate them, you will need to run DyGIE++'s preprocessing script[1] on the raw ACE2005 dataset and then run `preprocessing/preprocess_dygiepp.py` in this package on the output files. Example:

```
python preprocessing/process_dygiepp.py
  -i train.json \
  -o train.oneie.json
```

- ACE05-E+ and ACE05-CN: To generate these two datasets, run `process_ace.py` in the `preprocessing` folder on the raw ACE2005 dataset.

```
python preprocessing/process_ace.py \
  -i <INPUT_DIR>/LDC2006T06/data \
  -o <OUTPUT_DIR> \
  -s resource/splits/ACE05-E \
  -b bert-large-cased \
  -c <BERT_CACHE_DIR> \
  -l english
```

  where `-s` is used to specify the path to document id files, which are provided in `resource/splits`, `-b` is the model name (`bert-base-cased`, `bert-base-uncased`, `bert-large-cased`, or `bert-large-uncased`)[2], `-c` is the path to BERT's cache directory, and `-l` is used to specify the language (`english` or `chinese`).

- ERE-EN and ERE-ES: Use the `process_ere.py` script to convert raw ERE datasets (LDC2015E29, LDC2015E68, LDC2015E78, LDC2015E107) to the OneIE's JSON format using

```
python preprocessing/process_ere.py \
  -i <INPUT_DIR>/data \
  -o <OUTPUT_DIR> \
  -b bert-large-cased \
  -c <BERT_CACHE_DIR> \
  -l english \
  -d normal
```

  where `-l` is used to specify the language (`english` or `spanish`), and `-d` is used to specify the dataset (`normal`: LDC2015E29, `r2v2`: LDC2015E68, `parallel`: LDC2015E78, or `spanish`: LDC2015E107).

2.2 Evaluation Data Format

In the evaluation mode, OneIE supports two input file formats:

- Plain text: Each file is a document. Sentence tokenization and word tokenization are performed during file reading using NLTK, and some languages are therefore not supported.
- LTF XML: LTF files are tokenized and preferred. Two LTF sample files can be found in the `input` folder.

[1] https://github.com/dwadden/dygiepp/tree/master/scripts/data
[2] https://huggingface.co/transformers/pretrained_models.html

2.3 Output Format

OneIE save results in JSON format, where each line is a JSON object representing the output of a sentence containing the following fields:

- `doc_id`: Document ID.
- `sent_id`: Sentence ID.
- `tokens`: A list of sentence tokens.
- `token_ids`: A list of token IDs in the format of `doc_id:start_offset-end_offset`.
- `graph`: The information graph predicted by the model.
  - `entities`: A list of predicted entities. Each item in the list has exactly five values: `start_token_index`, `end_token_index`, `entity_type`, `mention_type`, `score`. For example, `[3, 5, "GPE", "NAM", 1.0]` means the index of the start token is `3`, index of the end token is `4` (5 - 1), entity type is `GPE`, mention type is `NAM`, and local score is `1.0`.
  - `triggers`: A list of predicted triggers. It is similar to `entities`, while each item has four values: `start_token_index`, `end_token_index`, `event_type`, `score`.
  - `relations`: A list of predicted relations. Each item in the list has four values: `arg1_entity_index`, `arg2_entity_index`, `relation_type`, `score`. In the following example, `[1, 0, "ORG-AFF", 0.52]` means there is an `ORG-AFF` relation between entity `1` ("leader") and entity `0` ("North Korean") with a local score of `0.52`. The order of arg1 and arg2 can be ignored for `"SOC-PER"` as this relation is symmetric.
  - `roles`: A list of predicted argument roles. Each item has four values: `trigger_index`, `entity_index`, `role`, `score`. In the following example, `[0, 2, "Attacker", 0.8]` means entity `2` (Kim Jong Un) is the Attacker argument of event `0` (`"detonate": Conflict:Attack`), and the local score is `0.8`.

The following example shows the output of the sentence "*On Tuesday, North Korean leader Kim Jong Un threatened to detonate a more powerful H-bomb in the future and called for an expansion of the size and power of his country's nuclear arsenal, the state television agency KCNA reported.*"


```
{
    "doc_id": "HC0003PYD",
    "sent_id": "HC0003PYD-16",
    "token_ids": [
        "HC0003PYD:2295-2296",
        "HC0003PYD:2298-2304",
        "HC0003PYD:2305-2305",
        "HC0003PYD:2307-2311",
        "HC0003PYD:2313-2318",
        ...
    ],
    "tokens": ["On", "Tuesday", ",", "North", "Korean", "leader", "Kim", "Jong
    ", "Un", "threatened", "to", "detonate", "a", "more", "powerful", "H-bomb",
    "in", "the", "future", "and", "called", "for", "an", "expansion", "of", "the
    ", "size", "and", "power", "of", "his", "country's", "nuclear", "arsenal",
    ",", "the", "state", "television", "agency", "KCNA", "reported", "."],
    "graph": {
        "entities": [
            [3, 5, "GPE", "NAM", 1.0],
            [5, 6, "PER", "NOM", 0.2],
            [6, 9, "PER", "NAM", 0.5060472888322202],
            [15, 16, "WEA", "NOM", 0.5332313915378754],
            [30, 31, "PER", "PRO", 1.0],
            [32, 33, "WEA", "NOM", 1.0],
            [33, 34, "WEA", "NOM", 0.5212696155645499],
            [36, 37, "GPE", "NOM", 0.4998288792916457],
            [38, 39, "ORG", "NOM", 1.0],
            [39, 40, "ORG", "NAM", 0.5294904130032032]
```

```
        ],
        "triggers": [
            [11, 12, "Conflict:Attack", 1.0]
        ],
        "relations": [
            [1, 0, "ORG-AFF", 1.0]
        ],
        "roles": [
            [0, 2, "Attacker", 0.4597024700555278],
            [0, 3, "Instrument", 1.0]
        ]
    }
}
```

2.4 Valid Pattern Files

In addition to global constraints learned by the model, OneIE can also filter out some invalid combinations following pre-defined rules. These rules usually come from annotation guidelines. For example, "a `Personnel:Elect` event can only have arguments of these roles: `Place`, `Person`, and `Entity`". Such rules are defined in three files:

- `event_role.json` that defines roles allowed for each event type. Example:

```
"Movement:Transport": [
    "Vehicle",
    "Artifact",
    "Agent",
    "Origin",
    "Destination"
],
```

- `relation_entity.json` that defines entity types allowed for endpoint entities of each relation type. Example:

```
"ORG-AFF": ["ORG", "PER", "GPE", "FAC"]
```

- `role_entity.json` that defines entity types allowed for each event argument role. Example:

```
"Attacker": ["ORG", "PER", "GPE"]
```

☞ We provide example files for the ACE ontology in `resource/valid_patterns`.

3 Training

A training script (`train.py`) is provided in the OneIE package to train new IE models from scratch on ACE, ERE, or other datasets in the required format (See: Section 2.1 Training Data Format):

- `cd` to the root directory of the OneIE package
- Set the environment variable `PYTHONPATH` to the current directory. For example, if you unpack this package to `/home/oneie_v0.4.8`, run `export PYTHONPATH=/home/oneie_v0.4.8` or add `PYTHONPATH=/home/oneie_v0.4.8` before the following `python` command.
- Run this commandline to train a new model: `python train.py -c <CONFIG_FILE_PATH>`. We provide an example configuration file `config/example.json`. Fill in the following paths in the configuration file:
  - `BERT_CACHE_DIR`: Pre-trained BERT models, configs, and tokenizers will be downloaded to this directory.
  - `TRAIN_FILE_PATH`, `DEV_FILE_PATH`, `TEST_FILE_PATH`: Path to the training/dev/test files.
  - `OUTPUT_DIR`: The model will be saved to sub folders in this directory.
  - `VALID_PATTERN_DIR`: Valid patterns created based on the annotation guidelines or training set. Example files are provided in `resource/valid_patterns`.

☞ Valid pattern files provided in `resource/valid_patterns` are designed for the ACE ontology. Modify these files accordingly when training new models on other datasets.

4 Evaluation

The evaluation script (`predict.py`) is used to load a trained model and make predictions on new data.

☞ The English, Chinese, and Spanish models provided on our website (http://blender.cs.illinois.edu/software/oneie) are different from what are reported in the paper. To reproduce results reported in the paper, train new models from scratch using `train.py`.

- `cd` to the root directory of this package
- Set the environment variable `PYTHONPATH` to the current directory. For example, if you unpack this package to `/home/oneie_v0.4.8`, run `export PYTHONPATH=/home/oneie_v0.4.8`.
- Load and run a model using the following command:

```
python predict.py \
  -m best.role.mdl \
  -i input \
  -o output \
  -c output_cs \
  --format ltf
```

  Arguments of this command include:

  - `-m`, `-model_path`: Path to the trained model.
  - `-i`, `-input_dir`: Path to the input directory. LTF format sample files can be found in the `input` directory.
  - `-o`, `-output_dir`: Path to the output directory (JSON format). Output files are in the JSON format. Sample files can be found in the `output` directory.
  - `-c`, `-cs_dir`: (optional) Path to the output directory (CS format). Sample files can be found in the `output_cs` directory.
  - `-l`, `-log_path`: (optional) Path to the log file. A sample file `log.json` can be found in the `output` folder.

- `-gpu`: (optional) Use GPU.
- `-d`, `-device`: (optional) GPU device index (for multi-GPU machines).
- `-b`, `-batch_size`: (optional) Batch size. For a 16GB GPU, a batch size of 10 15 is a reasonable value.
- `-max_len`: (optional) Max sentence length. Sentences longer than this value will be ignored. You may need to decrease `batch_size` if you set `max_len` to a larger number.
- `-beam_size`: (optional) Beam set size of the decoder. Increasing this value may improve the results and make the decoding slower.
- `-lang`: (optional) Model language.
- `-format`: Input file format (`txt` or `ltf`).

References

[1] D. Wadden, U. Wennberg, Y. Luan, and H. Hajishirzi, "Entity, relation, and event extraction with contextualized span representations," *ArXiv*, vol. abs/1909.03546, 2019.