google / sentencepiece

Code Issues 53 Pull requests 11 Actions Projects Security Insights

Unsupervised text tokenizer for Neural Network-based text generation.

Apache-2.0 license
Code of conduct
Security policy

11k stars 1.2k forks 126 watching Branches Activity Custom properties
Tags

Public repository

master 7 Branches 28 Tags Go to file t Go to file Add file Code

taku910 Merge pull request #1088 from pilleye/pilleye/fix-unigram-crash 2734490 · 4 months ago

.github	Upgrade cibuildwheel to v2.20.0	4 months ago
cmake	Enable iOS builds	3 years ago
data	add nfc, nfd normalization tsv files	last year
doc	fixing minor typos in the API.md	last year
python	AIX porting changes	4 months ago
src	Merge pull request #1088 from pilleye/pilleye/fi...	4 months ago
third_party	makes the return value of --help same as official...	last year
.gitignore	includes the sentencepiece source files in pytho...	2 years ago
CMakeLists.txt	bump CMake minimum required version to avoi...	last year
CONTRIBUTING.md	Update and rename CONTRIBUTING to CONTRI...	8 years ago
LICENSE	Initialize repository	8 years ago
README.md	Update README.md	2 years ago
VERSION.txt	increment version v0.2.1	last year
config.h.in	Switched to cmake	7 years ago
sentencepiece.pc.in	Fix pkg-config file to avoid overlinking	2 years ago

README Code of conduct Apache-2.0 license Security

SentencePiece

SentencePiece is an unsupervised text tokenizer and detokenizer mainly for Neural Network-based text generation systems where the vocabulary size is predetermined prior to the neural model training. SentencePiece implements **subword units** (e.g., **byte-pair-encoding (BPE)** [Sennrich et al.]) and **unigram language model** [Kudo.]) with the extension of direct training from raw sentences. SentencePiece allows us to make a purely end-to-end system that does not depend on language-specific pre/postprocessing.

This is not an official Google product.

Technical highlights

- **Purely data driven**: SentencePiece trains tokenization and detokenization models from sentences. Pre-tokenization (Moses tokenizer/MeCab/KyTea) is not always required.
- **Language independent**: SentencePiece treats the sentences just as sequences of Unicode characters. There is no language-dependent logic.
- **Multiple subword algorithms**: **BPE** [Sennrich et al.] and **unigram language model** [Kudo.] are supported.
- **Subword regularization**: SentencePiece implements subword sampling for subword regularization and BPE-dropout which help to improve the robustness and accuracy of NMT models.
- **Fast and lightweight**: Segmentation speed is around 50k sentences/sec, and memory footprint is around 6MB.
- **Self-contained**: The same tokenization/detokenization is obtained as long as the same model file is used.
- **Direct vocabulary id generation**: SentencePiece manages vocabulary to id mapping and can directly generate vocabulary id sequences from raw sentences.
- **NFKC-based normalization**: SentencePiece performs NFKC-based text normalization.

For those unfamiliar with SentencePiece as a software/algorithm, one can read a gentle introduction here.

Comparisons with other implementations

Feature	SentencePiece	subword-nmt	WordPiece
Supported algorithm	BPE, unigram, char, word	BPE	BPE*
OSS?	Yes	Yes	Google internal
Subword regularization	Yes	No	No
Python Library (pip)	Yes	No	N/A
C++ Library	Yes	No	N/A
Pre-segmentation required?	No	Yes	Yes
Customizable normalization (e.g., NFKC)	Yes	No	N/A
Direct id generation	Yes	No	N/A

Note that BPE algorithm used in WordPiece is slightly different from the original BPE.

Overview

What is SentencePiece?

SentencePiece is a re-implementation of **sub-word units**, an effective way to alleviate the open vocabulary problems in neural machine translation. SentencePiece supports two segmentation algorithms, **byte-pair-encoding (BPE)** [Sennrich et al.] and **unigram language model** [Kudo.]. Here are the high level differences from other implementations.

The number of unique tokens is predetermined

Neural Machine Translation models typically operate with a fixed vocabulary. Unlike most unsupervised word segmentation algorithms, which assume an infinite vocabulary, SentencePiece trains the segmentation model such that the final vocabulary size is fixed, e.g., 8k, 16k, or 32k.

Note that SentencePiece specifies the final vocabulary size for training, which is different from subword-nmt that uses the number of merge operations. The number of merge operations is a BPE-specific parameter and not applicable to other segmentation algorithms, including unigram, word and character.

Trains from raw sentences

Previous sub-word implementations assume that the input sentences are pre-tokenized. This constraint was required for efficient training, but makes the preprocessing complicated as we have to run language dependent tokenizers in advance. The implementation of SentencePiece is fast enough to train the model from raw sentences. This is useful for training the tokenizer and detokenizer for Chinese and Japanese where no explicit spaces exist between words.

Whitespace is treated as a basic symbol

The first step of Natural Language processing is text tokenization. For example, a standard English tokenizer would segment the text "Hello world." into the following three tokens.

> [Hello] [World] [.]

One observation is that the original input and tokenized sequence are **NOT reversibly convertible**. For instance, the information that is no space between "World" and "." is dropped from the tokenized sequence, since e.g., `Tokenize("World.") == Tokenize("World .")`

SentencePiece treats the input text just as a sequence of Unicode characters. Whitespace is also handled as a normal symbol. To handle the whitespace as a basic token explicitly, SentencePiece first escapes the whitespace with a meta symbol "▁" (U+2581) as follows.

> Hello▁World.

Then, this text is segmented into small pieces, for example:

> [Hello] [▁Wor] [ld] [.]

Since the whitespace is preserved in the segmented text, we can detokenize the text without any ambiguities.

```
detokenized = ''.join(pieces).replace('▁', ' ')
```

This feature makes it possible to perform detokenization without relying on language-specific resources.

Note that we cannot apply the same lossless conversions when splitting the sentence with standard word segmenters, since they treat the whitespace as a special symbol. Tokenized sequences do not preserve the necessary information to restore the original sentence.

- (en) Hello world. → [Hello] [World] [.] (A space between Hello and World)
- (ja) こんにちは世界。 → [こんにちは] [世界] [。] (No space between こんにちは and 世界)

Subword regularization and BPE-dropout

Subword regularization [Kudo.] and BPE-dropout Provilkov et al are simple regularization methods that virtually augment training data with on-the-fly subword sampling, which helps to improve the accuracy as well as robustness of NMT models.

To enable subword regularization, you would like to integrate SentencePiece library (C++/Python) into the NMT system to sample one segmentation for each parameter update, which is different from the standard off-line data preparations. Here's the example of Python library. You can find that 'New York' is segmented differently on each `SampleEncode (C++)` or `encode with enable_sampling=True (Python)` calls. The details of sampling parameters are found in sentencepiece_processor.h.

```
>>> import sentencepiece as spm
>>> s = spm.SentencePieceProcessor(model_file='spm.model')
>>> for n in range(5):
...     s.encode('New York', out_type=str, enable_sampling=True, alpha=0.1, nbest_size=-1)
...
['▁', 'N', 'e', 'w', '▁York']
['▁', 'New', '▁York']
['▁', 'New', '▁Y', 'o', 'r', 'k']
['▁', 'New', '▁York']
['▁', 'New', '▁York']
```

Installation

Python module

SentencePiece provides Python wrapper that supports both SentencePiece training and segmentation. You can install Python binary package of SentencePiece with.

```
pip install sentencepiece
```

For more detail, see Python module

Build and install SentencePiece command line tools from C++ source

The following tools and libraries are required to build SentencePiece:

- cmake
- C++11 compiler
- gperftools library (optional, 10-40% performance improvement can be obtained.)

On Ubuntu, the build tools can be installed with apt-get:

```
% sudo apt-get install cmake build-essential pkg-config libgoogle-perftools-dev
```

Then, you can build and install command line tools as follows.

```
% git clone https://github.com/google/sentencepiece.git
% cd sentencepiece
% mkdir build
% cd build
% cmake ..
% make -j $(nproc)
% sudo make install
% sudo ldconfig -v
```

On OSX/macOS, replace the last command with `sudo update_dyld_shared_cache`

Build and install using vcpkg

You can download and install sentencepiece using the vcpkg dependency manager:

```
git clone https://github.com/Microsoft/vcpkg.git
cd vcpkg
./bootstrap-vcpkg.sh
./vcpkg integrate install
./vcpkg install sentencepiece
```

The sentencepiece port in vcpkg is kept up to date by Microsoft team members and community contributors. If the version is out of date, please create an issue or pull request on the vcpkg repository.

Download and install SentencePiece from signed released wheels

You can download the wheel from the GitHub releases page. We generate SLSA3 signatures using the OpenSSF's slsa-framework/slsa-github-generator during the release process. To verify a release binary:

1. Install the verification tool from slsa-framework/slsa-verifier#installation.
2. Download the provenance file `attestation.intoto.jsonl` from the GitHub releases page.
3. Run the verifier:

```
slsa-verifier -artifact-path <the-wheel> -provenance attestation.intoto.jsonl -source github.com/google/sentencepiece -tag <the
```

pip install wheel_file.whl

Usage instructions

Train SentencePiece Model

```
% spm_train --input=<input> --model_prefix=<model_name> --vocab_size=8000 --character_coverage=1.0 --model_type=<type>
```

- `--input` : one-sentence-per-line **raw** corpus file. No need to run tokenizer, normalizer or preprocessor. By default, SentencePiece normalizes the input with Unicode NFKC. You can pass a comma-separated list of files.
- `--model_prefix` : output model name prefix. `<model_name>.model` and `<model_name>.vocab` are generated.
- `--vocab_size` : vocabulary size, e.g., 8000, 16000, or 32000
- `--character_coverage` : amount of characters covered by the model, good defaults are: `0.9995` for languages with rich character set like Japanese or Chinese and `1.0` for other languages with small character set.
- `--model_type` : model type. Choose from `unigram` (default), `bpe` , `char` , or `word` . The input sentence must be pretokenized when using `word` type.

Use `--help` flag to display all parameters for training, or see here for an overview.

Encode raw text into sentence pieces/ids

```
% spm_encode --model=<model_file> --output_format=piece < input > output
% spm_encode --model=<model_file> --output_format=id < input > output
```

Use `--extra_options` flag to insert the BOS/EOS markers or reverse the input sequence.

```
% spm_encode --extra_options=eos (add </s> only)
% spm_encode --extra_options=bos:eos (add <s> and </s>)
% spm_encode --extra_options=reverse:bos:eos (reverse input and add <s> and </s>)
```

SentencePiece supports nbest segmentation and segmentation sampling with `--output_format=(nbest|sample)_(piece|id)` flags.

```
% spm_encode --model=<model_file> --output_format=sample_piece --nbest_size=-1 --alpha=0.5 < input > output
% spm_encode --model=<model_file> --output_format=nbest_id --nbest_size=10 < input > output
```

Decode sentence pieces/ids into raw text

```
% spm_decode --model=<model_file> --input_format=piece < input > output
% spm_decode --model=<model_file> --input_format=id < input > output
```

Use `--extra_options` flag to decode the text in reverse order.

```
% spm_decode --extra_options=reverse < input > output
```

End-to-End Example

```
% spm_train --input=data/botchan.txt --model_prefix=m --vocab_size=1000
unigram_model_trainer.cc(494) LOG(INFO) Starts training with :
input: "../data/botchan.txt"
... <snip>
unigram_model_trainer.cc(529) LOG(INFO) EM sub_iter=1 size=1100 obj=10.4973 num_tokens=37630 num_tokens/piece=34.2091
trainer_interface.cc(272) LOG(INFO) Saving model: m.model
trainer_interface.cc(281) LOG(INFO) Saving vocabs: m.vocab

% echo "I saw a girl with a telescope." | spm_encode --model=m.model
▁I ▁saw ▁a ▁girl ▁with ▁a ▁ te le s c o pe .

% echo "I saw a girl with a telescope." | spm_encode --model=m.model --output_format=id
9 459 11 939 44 11 4 142 82 8 28 21 132 6

% echo "9 459 11 939 44 11 4 142 82 8 28 21 132 6" | spm_decode --model=m.model --input_format=id
I saw a girl with a telescope.
```

You can find that the original input sentence is restored from the vocabulary id sequence.

Export vocabulary list

```
% spm_export_vocab --model=<model_file> --output=<output file>
```

`<output file>` stores a list of vocabulary and emission log probabilities. The vocabulary id corresponds to the line number in this file.

Redefine special meta tokens

By default, SentencePiece uses Unknown (<unk>), BOS (<s>) and EOS (</s>) tokens which have the ids of 0, 1, and 2 respectively. We can redefine this mapping in the training phase as follows.

```
% spm_train --bos_id=0 --eos_id=1 --unk_id=5 --input=... --model_prefix=... --character_coverage=...
```

When setting -1 id e.g., `bos_id=-1` , this special token is disabled. Note that the unknown id cannot be disabled. We can define an id for padding (<pad>) as `--pad_id=3` .

If you want to assign another special tokens, please see Use custom symbols.

Vocabulary restriction

`spm_encode` accepts a `--vocabulary` and a `--vocabulary_threshold` option so that `spm_encode` will only produce symbols which also appear in the vocabulary (with at least some frequency). The background of this feature is described in subword-nmt page.

The usage is basically the same as that of `subword-nmt` . Assuming that L1 and L2 are the two languages (source/target languages), train the shared spm model, and get resulting vocabulary for each:

```
% cat {train_file}.L1 {train_file}.L2 | shuffle > train
% spm_train --input=train --model_prefix=spm --vocab_size=8000 --character_coverage=0.9995
% spm_encode --model=spm.model --generate_vocabulary < {train_file}.L1 > {vocab_file}.L1
% spm_encode --model=spm.model --generate_vocabulary < {train_file}.L2 > {vocab_file}.L2
```

`shuffle` command is used just in case because `spm_train` loads the first 10M lines of corpus by default.

Then segment train/test corpus with `--vocabulary` option

```
% spm_encode --model=spm.model --vocabulary={vocab_file}.L1 --vocabulary_threshold=50 < {test_file}.L1 > {test_file}.seg.L1
% spm_encode --model=spm.model --vocabulary={vocab_file}.L2 --vocabulary_threshold=50 < {test_file}.L2 > {test_file}.seg.L2
```

Advanced topics

- SentencePiece Experiments
- SentencePieceProcessor C++ API
- Use custom text normalization rules

Releases 27

v0.2.0 Latest
on Feb 19, 2024

+ 26 releases

Packages

No packages published

Used by 115k

+ 114,737

Contributors 88

+ 74 contributors

Languages

C++ 96.7% Python 1.0% SWIG 0.8% CMake 0.8% Jupyter Notebook 0.6% Perl 0.1%