token2index Documentation

Release 1.0.2

Dennis Ulmer

Oct 28, 2020

Contents

`token2index` is a small yet powerful library facilitating the fast and easy creation of a data structure mapping tokens to indices, primarily aimed at applications for Natural Language Processing. The library is fully tested, and does not require any additional requirements. The documentation can be found here, some feature highlights are shown below.

Who / what is this for?

This class is written to be used for NLP applications where we want to assign an index to every word in a sequence e.g. to be later used to look up corresponding word embeddings. Building an index and indexing batches of sequences for Deep Learning models using frameworks like PyTorch or Tensorflow are common steps but are often written from scratch every time. This package provides a ready-made package combining many useful features, like reading vocabulary files, building indices from a corpus or indexing entire batches in one single function call, all while being fully tested.

CHAPTER 1

Feature Highlights

- **Building and extending vocab**

  One way to build the index from a corpus is using the build() function:

```
   >>> from t2i import T2I
   >>> t2i = T2I.build(["colorless green ideas dream furiously", "the horse raced␣
↪past the barn fell"])
   >>> t2i
   T2I(Size: 13, unk_token: <unk>, eos_token: <eos>, pad_token: <pad>, {'colorless
↪': 0, 'green': 1, 'ideas': 2, 'dream': 3, 'furiously': 4, 'the': 5, 'horse': 6,
↪'raced': 7, 'past': 8, 'parn': 9, 'fell': 10, '<unk>': 11, '<eos>': 12, '<pad>
↪': 13})

The index can always be extended again later using ``extend()``\ :
```

```
>>> t2i = t2i.extend("completely new words")
T2I(Size: 16, unk_token: <unk>, eos_token: <eos>, pad_token: <pad>, {'colorless':␣
↪0, 'green': 1, 'ideas': 2, 'dream': 3, 'furiously': 4, 'the': 5, 'horse': 6,
↪'raced': 7, 'past': 8, 'barn': 9, 'fell': 10, 'completely': 13, 'new': 14,
↪'words': 15, '<unk>': 16, '<eos>': 17, '<pad>': 18})
```

- **Easy indexing (of batches)**

  Index multiple sentences at once in a single function call!

```
   >>> t2i.index(["the green horse raced <eos>", "ideas are a dream <eos>"])
   [[5, 1, 6, 7, 12], [2, 11, 11, 3, 12]]

where unknown tokens are always mapped to ``unk_token``.
```

- **Easy conversion back to strings**

  Reverting indices back to strings is equally as easy:

```
>>> t2i.unindex([5, 14, 16, 3, 6])
'the new <unk> dream horse'
```

- **Automatic padding**

  You are indexing multiple sentences of different length and want to add padding? No problem! `index()` has two options available via the `pad_to` argument. The first is padding to the maximum length of all the sentences:

```
   >>> padded_sents = t2i.index(["the green horse raced <eos>", "ideas <eos>"],␣
↪pad_to="max")
   >>> padded_sents
   [[5, 1, 6, 7, 12], [2, 12, 13, 13, 13]]
   >>> t2i.unindex(padded_sents)
   [['the green horse raced <eos>', 'ideas <eos> <pad> <pad> <pad>']]

Alternatively, you can also pad to a pre-defined length:
```

```
>>> padded_sents = t2i.index(["the green horse <eos>", "past ideas <eos>"], pad_
↪to=5)
>>> padded_sents
[[5, 1, 6, 12, 13], [8, 2, 12, 13, 13]]
>>> t2i.unindex(padded_sents)
[['the green horse <eos> <pad>', 'past ideas <eos> <pad> <pad>']]
```

- **Vocab from file**

  Using `T2I.from_file()`, the index can be created directly by reading from an existing vocab file. Refer to its documentation here for more info.

- **Fixed memory size**

  Although the `defaultdict` class from Python's `collections` package also posses the functionality to map unknown keys to a certain value, it grows in size for every new key. `T2I` memory size stays fixed after the index is built.

- **Support for special tokens**

  To enable flexibility in modern NLP applications, `T2I` allows for an arbitrary number of special tokens (like a masking or a padding token) during init!

```
>>> t2i = T2I(special_tokens=["<mask>"])
>>> t2i
T2I(Size: 3, unk_token: <unk>, eos_token: <eos>, {'<unk>': 0, '<eos>': 1, '<mask>
↪': 2})
```

- **Explicitly supported programmer laziness**

  Too lazy to type? The library saves you a few keystrokes here and there. instead of calling `t2i.index(...)` you can directly call `t2i(...)` to index one or multiple sequences. Furthermore, key functions like `index()`, `unindex()`, `build()` and `extend()` support strings or iterables of strings as arguments alike.

CHAPTER 2

Compatibility with other frameworks (Numpy, PyTorch, Tensorflow)

It is also ensured that `T2I` is easily compatible with frameworks like Numpy, PyTorch and Tensorflow, without needing them as requirements:

Numpy

```
>>> import numpy as np
>>> t = np.array(t2i.index(["the new words are ideas <eos>", "the green horse <eos>
↪<pad> <pad>"]))
>>> t
array([[ 5, 15, 16, 17,  2, 18],
   [ 5,  1,  6, 18, 19, 19]])
>>> t2i.unindex(t)
['the new words <unk> ideas <eos>', 'the green horse <eos> <pad> <pad>']
```

PyTorch

```
>>> import torch
>>> t = torch.LongTensor(t2i.index(["the new words are ideas <eos>", "the green horse
↪<eos> <pad> <pad>"]))
>>> t
tensor([[ 5, 15, 16, 17,  2, 18],
    [ 5,  1,  6, 18, 19, 19]])
>>> t2i.unindex(t)
['the new words <unk> ideas <eos>', 'the green horse <eos> <pad> <pad>']
```

Tensorflow

```
>>> import tensorflow as tf
>>> t = tf.convert_to_tensor(t2i.index(["the new words are ideas <eos>", "the green␣
↪horse <eos> <pad> <pad>"]), dtype=tf.int32)
>>> t
tensor([[ 5, 15, 16, 17,  2, 18],
    [ 5,  1,  6, 18, 19, 19]])
>>> t2i.unindex(t)
['the new words <unk> ideas <eos>', 'the green horse <eos> <pad> <pad>']
```

CHAPTER 3

Installation

Installation can simply be done using `pip`:

CHAPTER 4

Citing

If you use `token2index` for research purposes, please cite the library using the following citation info:

CHAPTER 5

Documentation

Define a lightweight data structure to store and look up the indices belonging to arbitrary tokens. Originally based on the diagnnose W2I class.

class t2i.**T2I**(*index: Union[Dict[str, int], t2i.Index, None] = None, counter: Optional[collections.Counter] = None, max_size: Optional[int] = None, min_freq: int = 1, unk_token: str = '<unk>', eos_token: str = '<eos>', pad_token: str = '<pad>', special_tokens: Iterable[str] = ()*)

Provides vocab functionality mapping tokens to indices. After building an index, sentences or a corpus of sentences can be mapped to the tokens' assigned indices. There are special tokens for the end of a sentence (eos_token) and for tokens that were not added to the index during the build phase (unk_token).

static build(*corpus: Union[str, Iterable[str], Iterable[Iterable[str]]], delimiter: str = ' ', counter: Optional[collections.Counter] = None, max_size: Optional[int] = None, min_freq: int = 1, unk_token: str = '<unk>', eos_token: str = '<eos>', pad_token: str = '<pad>', special_tokens: Iterable[str] = ()*)

Build token index from scratch on a corpus.

corpus: Corpus Corpus that is being used to build the index.

delimiter: str Delimiter between tokens. Default is a whitespace ' '.

counter: Optional[Counter] Counter with token frequencies in corpus. Default is None.

max_size: Optional[int] Maximum size of T2I index. Default is None, which means no maximum size.

min_freq: int Minimum frequency of a token for it to be included in the index. Default is 1.

unk_token: str Token that should be used for unknown words. Default is 'STD_UNK'.

eos_token: str Token that marks the end of a sequence. Default is '<eos>'.

pad_token: str Padding token. Default is '<pad>'.

special_tokens: Iterable[str] An arbitrary number of additional special tokens, given as unnamed arguments.

t2i: T2I New T2I object.

extend(*corpus: Union[str, Iterable[str], Iterable[Iterable[str]]], delimiter: str = ' '*)
Extend an existing T2I with tokens from a new tokens and build indices for them.

corpus: Corpus Corpus that is being used to extend the index.

delimiter: str Delimiter between tokens. Default is a whitespace ' '.

t2i: T2I New T2I object.

static from_file(*vocab_path: str, encoding: str = 'utf-8', delimiter: str = '\t', counter: Optional[collections.Counter] = None, max_size: Optional[int] = None, min_freq: int = 1, unk_token: str = '<unk>', eos_token: str = '<eos>', pad_token: str = '<pad>', special_tokens: Iterable[str] = ()*)
Generate a T2I object from a file. This file can have two possible formats:

1. One token per line (in which case the index is the line number)
2. A token and its corresponding index, separated by some delimiter (default is " "):

vocab_path: str Path to vocabulary file.

encoding: str Encoding of vocabulary file (default is 'utf-8').

delimiter: str Delimiter in case the format is token <delimiter> index. Default is ' '.

counter: Optional[Counter] Counter with token frequencies in corpus. Default is None.

max_size: Optional[int] Maximum size of T2I index. Default is None, which means no maximum size.

min_freq: int Minimum frequency of a token for it to be included in the index. Default is 1.

unk_token: str Token that should be used for unknown words. Default is 'STD_UNK'.

eos_token: str Token that marks the end of a sequence. Default is '<eos>'.

pad_token: str Padding token. Default is '<pad>'.

special_tokens: Iterable[str] An arbitrary number of additional special tokens.

t2i: T2I T2I object built from vocab file.

index(*corpus: Union[str, Iterable[str], Iterable[Iterable[str]]], delimiter: str = ' ', pad_to: Union[str, int, None] = None*) → [typing.Iterable[int], typing.Iterable[typing.Iterable[int]]]
Assign indices to a sentence or a series of sentences.

corpus: Corpus Corpus that is being indexed.

delimiter: str Delimiter between tokens. Default is a whitespace ' '.

pad_to: Optional[Union[str, int]] Indicate whether shorter sequences in this corpus should be padded up to the length of the longest sequence ('max') or to a fixed length (any positive integer) or not not at all (None). Default is None.

indexed_corpus: IndexedCorpus Indexed corpus.

indices() → Tuple[int, ...]
Return all indices in this T2I object.

static load(*path: str*)
Load serialized T2I object.

save(*path: str*) → None
Save T2I object as pickle.

t2i
Return the dictionary mapping tokens to unique indices.

t2i: Index Dictionary mapping from tokens to indices.

tokens() → Tuple[str, ...]
Return all token in this T2I object.

unindex(*indexed_corpus: [typing.Iterable[int], typing.Iterable[typing.Iterable[int]]], joiner: Optional[str] = ' '*) → Union[str, Iterable[str], Iterable[Iterable[str]]]
Convert indices back to their original tokens. A joiner can be specified to determine how tokens are pieced back together. If the joiner is None, the tokens are not joined and are simply returned as a list.

indexed_corpus: IndexedCorpus An indexed corpus.

joiner: Optional[str] String used to join tokens. Default is a whitespace ' '. If the value is None, tokens are not joined and a list of tokens is returned.

corpus: Corpus Un-indexed corpus.

class t2i.**Index**
(Technically) A defaultdict where the value return value for an unknown key is the number of entries. However, it doesn't inherit from defaultdict, because functions returning the value for missing keys can only return a constant value. In this case, after every lookup of a new token, this value for an unknown is incremented by one.

highest_idx
Return the currently highest index in the index. Return -1 if the index is empty.

items() → Iterable[Tuple[str, int]]
The same as a usual dict items(), except that the entries are sorted by index (this has otherwise proven to create problems in Python < 3.6).

Python Module Index

t

Index