
CTC Word Beam Search Decoding Algorithm

- **Update 2024: Support Python versions 3.11 and 3.12**
- **Update 2021: Python package is the default way of installation**
- **Update 2020: installable Python package**

Connectionist Temporal Classification (CTC) decoder with dictionary and Language Model (LM).

Installation

- Go to the root level of the repository
- Execute `pip install .`
- Go to `tests/` and execute `pytest` to check if installation worked

Usage

The following toy example shows how to use word beam search. The hypothetical model (e.g. a text recognition model) is able to recognize 3 different characters: “a”, “b” and “ ” (whitespace). Words in that toy example can contain the characters “a” and “b” (but not “ ” which is the word separator). The language model is trained from a text corpus which contains only two words: “a” and “ba”.

In this code snippet an instance of word beam search is created, and a $T \times B \times (C+1)$ shaped numpy array is decoded:

```
1 import numpy as np
2 from word_beam_search import WordBeamSearch
3
4 corpus = 'a ba' # two words "a" and "ba", separated by whitespace
5 chars = 'ab ' # the characters that can be recognized (in this order)
6 word_chars = 'ab' # characters that form words
7
8 # RNN output
9 # 3 time-steps and 4 characters per time time ("a", "b", " ", CTC-blank
10 # )
11 mat = np.array([[[0.9, 0.1, 0.0, 0.0]],
12                 [[0.0, 0.0, 0.0, 1.0]],
13                 [[0.6, 0.4, 0.0, 0.0]])
14
15 # initialize word beam search (only do this once in your code)
16 wbs = WordBeamSearch(25, 'Words', 0.0, corpus.encode('utf8'), chars.
17                       encode('utf8'), word_chars.encode('utf8'))
18
19 # compute label string
20 label_str = wbs.compute(mat)
```

The decoder returns a list with a decoded label string for each batch element. To finally get the character strings, map each label to its corresponding character:

```
1 char_str = [] # decoded texts for batch
2 for curr_label_str in label_str:
3     s = ''.join([chars[label] for label in curr_label_str])
4     char_str.append(s)
```

Examples: * Both this toy example and a real text recognition example can be found in [tests/test_word_beam_search.py](#) * The SimpleHTR repository implements a handwritten text recognition system and optionally uses word beam search

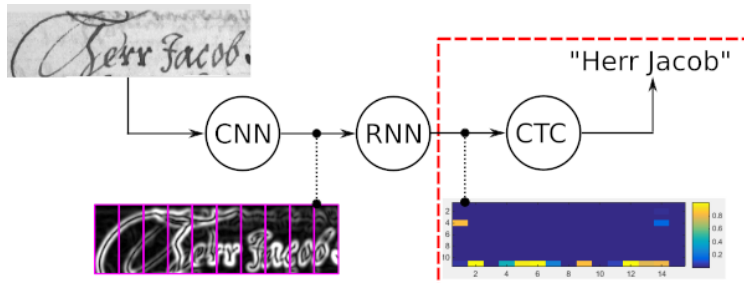
Documentation of parameters

Parameters of the constructor of the [WordBeamSearch](#) class: * Beam Width (beam_width): number of beams which are kept per time-step * Scoring mode (lm_type): pass one of the four strings (not case-sensitive). The runtime with respect to the dictionary size W is given. * “Words”: only use dictionary, no scoring: $O(1)$ * “NGrams”: use dictionary and score beams with LM: $O(\log(W))$ * “NGramsForecast”: forecast (possible) next words and apply LM to these words: $O(W \log(W))$ * “NGramsForecastAndSample”: restrict number of (possible) next words to at most 20 words: $O(W)$ * Smoothing (lm_smoothing): LM uses add-k smoothing to allow word pairs which are not known from the training text, i.e. for which the bigram probability is zero. Set to values between 0 and 1, e.g. 0.01. To disable smoothing, set to 0 * Text (corpus): is given as a UTF8 encoded string. The operation creates its dictionary and (optionally) LM from it * Characters (chars): is given as a UTF8 encoded string. If the number of characters is C , then the RNN output must have the size $T \times B \times (C+1)$ with the last entry representing the CTC-blank label. The ordering of the characters must correspond to the ordering in the RNN output, e.g. if the RNN outputs the probabilities for “a”, “b”, “ ” and CTC-blank in this order, then the string “ab” must be passed * Word characters (word_chars): is given as a UTF8 encoded string. Define how the algorithm extracts words from the text. If the word characters are “ab”, and the text “aa ab bbb a” is passed, then the words “aa”, “ab” and “bbb” will be extracted and used for the dictionary and the LM. To be able to recognize multiple words (e.g. a text-line), the word characters must be a subset of the characters recognized by the RNN (i.e. there must be at least one word-separating character like the space character): $0 < \text{len}(\text{wordChars}) < \text{len}(\text{chars})$. In case only single words have to be detected, there is no need for a separating character, therefore the two parameters may also be equal: $0 < \text{len}(\text{wordChars}) \leq \text{len}(\text{chars})$

Input to the [WordBeamSearch.compute](#) method: * Input matrix (mat) * numpy array * shape $T \times B \times (C+1)$ * T is the number of time-steps, B the number of batch elements and C the number of characters * softmax-function already applied * CTC-blank must be the last entry along the character dimension in the matrix

Algorithm

Word beam search is a CTC decoding algorithm. It is used for sequence recognition tasks like hand-written text recognition or automatic speech recognition.



The four main properties of word beam search are:

- Words constrained by dictionary
- Allows arbitrary number of non-word characters between words (numbers, punctuation marks)
- Optional word-level Language Model (LM)
- Faster than token passing

The following sample shows a typical use-case of word beam search along with the results given by five different decoders. Best path decoding and vanilla beam search get the words wrong as these decoders only use the noisy output of the optical model. Extending vanilla beam search by a character-level LM improves the result by only allowing likely character sequences. Token passing uses a dictionary and a word-level LM and therefore gets all words right. However, it is not able to recognize arbitrary character strings like numbers. Word beam search is able to recognize the words by using a dictionary, but it is also able to correctly identify the non-word characters.

A roindan number: 1234.

↓

Best path decoding	"A roindan nunibr: 1234."	✗
Vanilla beam search	"A roindan numbr: 1234."	✗
Vanilla beam search LM	"A randan number: 1234."	✗
Token passing	"A random number"	✗
Word beam search	"A random number: 1234."	✓

More information: * A short overview is given in the poster * More details can be found in the ICFHR 2018 paper

Extras

- Python prototype: [extras/prototype/](#)
- TensorFlow custom operation: [extras/tf/](#)

Citation

Please cite the following paper if you are using word beam search in your research work.

```
1 @inproceedings{scheidl2018wordbeamsearch,
2   title = {Word Beam Search: A Connectionist Temporal Classification
3     Decoding Algorithm},
4   author = {Scheidl, H. and Fiel, S. and Sablatnig, R.},
5   booktitle = {16th International Conference on Frontiers in
6     Handwriting Recognition},
7   pages = {253--258},
8   year = {2018},
9   organization = {IEEE}
```

References

- Word Beam Search: A CTC Decoding Algorithm
- Beam Search Decoding in CTC-trained Neural Networks
- Scheidl - Handwritten Text Recognition in Historical Documents
- Scheidl - Word Beam Search: A Connectionist Temporal Classification Decoding Algorithm