Word Beam Search: A Connectionist Temporal Classification Decoding Algorithm

Harald Scheidl, Stefan Fiel, Robert Sablatnig
Computer Vision Lab
TU Wien
1040 Vienna, Austria
harald_scheidl@hotmail.com, {fiel,sab}@cvl.tuwien.ac.at

***Abstract*—Recurrent Neural Networks (RNNs) are used for sequence recognition tasks such as Handwritten Text Recognition (HTR) or speech recognition. If trained with the Connectionist Temporal Classification (CTC) loss function, the output of such a RNN is a matrix containing character probabilities for each time-step. A CTC decoding algorithm maps these character probabilities to the final text. Token passing is such an algorithm and is able to constrain the recognized text to a sequence of dictionary words. However, the running time of token passing depends quadratically on the dictionary size and it is not able to decode arbitrary character strings like numbers. This paper proposes word beam search decoding, which is able to tackle these problems. It constrains words to those contained in a dictionary, allows arbitrary non-word character strings between words, optionally integrates a word-level language model and has a better running time than token passing. The proposed algorithm outperforms best path decoding, vanilla beam search decoding and token passing on the IAM and Bentham HTR datasets. An open-source implementation is provided.**

***Index Terms*—connectionist temporal classification, decoding, language model, recurrent neural network, speech recognition, handwritten text recognition**

I. Introduction

Sequence recognition is the task of transcribing sequences of data with sequences of labels [1]. Well known use-cases are Handwritten Text Recognition (HTR) and speech recognition. Graves et al. [2] introduce the Connectionist Temporal Classification (CTC) operation which enables neural network training from pairs of data and target labelings (text). The neural network is trained to output the labelings in a specific coding scheme. Decoding algorithms are used to calculate the final labeling. Hwang and Sung [3] present a beam search decoding algorithm which can be extended by a character-level Language Model (LM). Graves et al. [4] introduce the token passing algorithm, which constraints its output to a sequence of dictionary words and uses a word-level LM. The motivation to propose the Word Beam Search (WBS) decoding algorithm[1] is twofold:

- Vanilla Beam Search (VBS) decoding works on character-level and does not constrain its beams (text candidates) to dictionary words.
- The running time of token passing depends quadratically on the dictionary size [4], which is not feasible for large dictionaries as shown in Section V. Further, the algorithm does not handle non-word character strings. Punctuation-marks and large numbers occur in the IAM and Bentham datasets, however, putting all possible combinations of these into the dictionary would enlarge it unnecessarily.

WBS uses a prefix tree that is created from a dictionary to constrain the words in the recognized text. Four different methods to score the beams by a word-level LM are proposed: (1) only constrain the beams by the dictionary, (2) score when a word is completely recognized, (3) forecast the score by calculating possible next words (Ortmanns et. al [5] use this idea in the context of hidden Markov models) and (4) forecast the score with a random sample of possible next words. Further, there is an operating-state in which arbitrary non-word character strings are recognized. The proposed algorithm is able to outperform best path decoding, VBS and token passing on the IAM [6] and Bentham [7] datasets. Furthermore, the running time outperforms token passing.

The rest of the paper is organized as follows: in Section II, a brief introduction to CTC loss and CTC decoding is given. Then, prefix trees and LMs are discussed. Section IV presents the proposed algorithm. The evaluation compares the scoring-modes of the algorithm and further compares the results with other decoding algorithms. Finally, the conclusion summarizes the paper.

II. State of the art

First, the CTC operation is discussed. Afterwards, two state-of-the-art decoding algorithms, namely VBS and token passing, are presented.

A. Connectionist Temporal Classification

A Recurrent Neural Network (RNN) outputs a sequence of length T with $C + 1$ character probabilities per sequence element, where C denotes the number of characters [4]. An additional pseudo-character is added to the RNN output which is called *blank* and is denoted by "-" in this paper. Picking one character per time-step from the RNN output and concatenating them form a path π [4]. The probability of a path is defined as the product of all character probabilities on this path. A single character from a labeling is encoded by one or multiply adjacent occurrences of this character on the path, possibly followed by a sequence of blanks [4]. A way

[1] An open-source implementation is available at: https://github.com/githubharald/CTCWordBeamSearch

978-1-5386-5875-8/18/$31.00 ©2018 IEEE
DOI 10.1109/ICFHR-2018.2018.00052

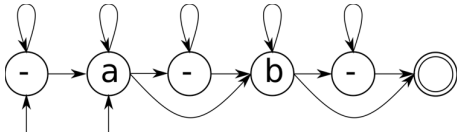

Fig. 1. FSM which produces valid paths (encodings) for the labeling "ab". The two left-most states are the initial states while the right-most state is the final state.

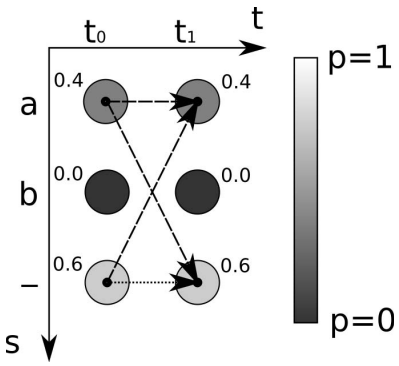

Fig. 2. An example of a RNN output. The sequence has a length of 2 with time-steps t_0 and t_1. For each time-step a probability distribution over the possible characters ("a" and "b" and blank "-") is outputted by the RNN. Lines indicate four different paths through the RNN output: the dotted line indicates the best path yielding the labeling "" while the dashed lines indicate paths yielding the labeling "a".

to model this encoding is by using a Finite State Machine (FSM) [3]. The FSM is created as follows: the labeling is first extended by inserting blanks and then a state is created for each character and blank. Consecutive states are connected by a transition and a self-loop is added to each state. Further, a direct transition skipping the blank is added for consecutive but different characters. Figure 1 shows a FSM which produces valid paths for the labeling "ab" by proceeding from a start state to the final state on an arbitrary path, e.g. " - a a - - b -" or "a b" among others.

To decode a path into a labeling, the encoding operation implemented by the FSM has to be inverted which is done by a collapsing function B [3]. It is applied to a path π and yields a labeling l by first removing repeated characters and then removing blanks on the path. To give an example, the path "a - b b - -" is collapsed to B("a - b b - -") ="ab". The loss is calculated by taking all paths yielding the target labeling l (i.e. all paths π for which $B(\pi) = l$ holds) and summing over their probabilities. This enables training the RNN without knowing the character-positions of the target labeling in the input.

After the RNN is trained by the CTC loss, new samples are presented to the neural network to recognize the handwritten text. A first approximation of decoding the RNN output is to take the most probable character per time-step forming the so called best path and then apply the collapsing function B to this path [4]. This approximation algorithm is called best path decoding [4]. However, there are situations for which this yields the wrong result as illustrated in Figure 2. The given RNN output has a length of 2 and has 2 possible characters "a" and "b" and further the blank "-". Taking the most probable characters yields the best path "- -" and therefore the empty labeling B("- -")="" with probability $0.6 \cdot 0.6 = 0.36$. However, the correct answer is "a", this can be seen by summing up the probabilities of all paths yielding this labeling: "a -", "- a" and "a a" with probability $2 \cdot 0.6 \cdot 0.4 + 0.4 \cdot 0.4 = 0.64$. The other decoding algorithms presented in this paper are able to correctly handle such situations.

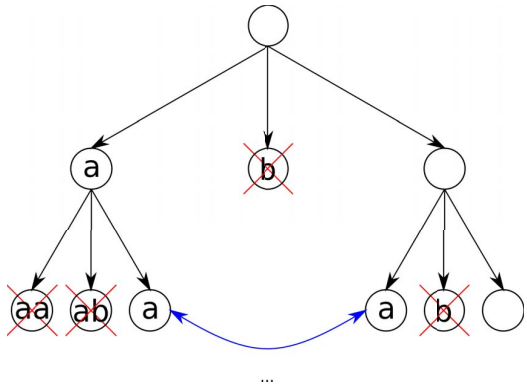

Fig. 3. Iteratively extending beam-labelings (from top to bottom) forms this tree. Only the two best-scoring beams are kept per time-step (i.e. $BW = 2$), all others are removed (red). Equal labelings get merged (blue).

B. Vanilla Beam Search Decoding

The VBS decoding algorithm is described in the paper of Hwang and Sung [3]. An illustration is shown in Figure 3. The RNN output is a matrix of size $T \times (C+1)$ and is fed into the decoding algorithm. Multiple candidates for the final labeling are iteratively calculated and are called *beams*. At each time-step, each beam-labeling is extended by all possible characters. Additionally, the original beam is also copied to the next time-step. This forms a tree as shown in Figure 3. To avoid exponential growth of the tree, only the best beams are kept at each time-step: the Beam Width (BW) governs the number of beams to keep. If two beam-labelings at a given time-step are equal, they get merged by first summing up the probabilities and then removing one of them. A character-level LM can optionally be used to score the extension of a beam-labeling by a character. Constrained decoding is possible by removing a beam as soon as an Out-Of-Vocabulary (OOV) word occurs [8]. However, there is the chance that each beam contains an OOV word which limits the usage of this constrained decoding approach.

The time-complexity can be derived from the pseudo-code of Algorithm 1. At each time-step, the beams are sorted according to their score. In the previous time-step, each of the BW beams is extended by C characters, therefore $BW \cdot C$ beams have to be sorted which accounts for $\mathcal{O}(BW \cdot C \cdot log(BW \cdot C))$. As this sorting happens for each of the T time-steps, the overall time-complexity is

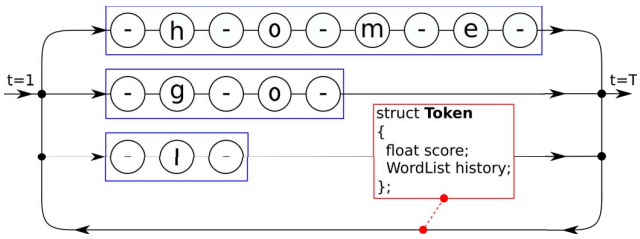

Fig. 4. Three word models (blue) are put in parallel. Information flow is implemented by tokens (red) which are passed through the states and between the words.

$\mathcal{O}(T \cdot BW \cdot C \cdot log(BW \cdot C))$. The two inner loops can be ignored as they only account for $\mathcal{O}(BW \cdot C)$.

C. Token Passing

This algorithm is proposed by Graves et al. [4], however, the following discussion is based on another publication from Graves [1]. A dictionary, a LM and a RNN output are given and the algorithm outputs a sequence of dictionary words. For each word a word-model is created, which essentially is a state machine connecting consecutive characters with respect to the already discussed CTC coding scheme. A word sequence is modeled by putting multiple word-models in parallel, connecting all end-states with all begin-states. The information flow is implemented by tokens, which are passed from state to state. Each token holds the score and the history of already visited words. The algorithms searches for the most likely sequence of dictionary words by aligning them with the RNN output and scoring word-transitions with a word-level LM. Figure 4 shows three word-models which are put in parallel. The time-complexity of this algorithm is $\mathcal{O}(T \cdot W^2)$, where T denotes the sequence length and W the dictionary size [1].

III. Methodology

The proposed WBS decoding algorithm uses a prefix tree to query the characters that can extend the current beam-labeling. Further, words which have the current beam-labeling as prefix can also be queried. A LM is used to score the beams on word-level.

A. Prefix Tree

A prefix tree or trie (from re*trie*val) is a basic tool in the domain of string processing [9]. Figure 5 shows a prefix tree containing 5 words. It is a tree-datastructure and therefore consists of edges and nodes. Each edge is labeled by a character and points to the next node. A node has a word-flag which indicates if this node represents a word. A word is encoded in the tree by a path starting at the root node and following the edges labeled with the corresponding characters of the word. Querying characters which follow a given prefix is easy: the node corresponding to the prefix is identified and the outgoing edge labels determine the characters which can follow the prefix. It is also possible to identify all words which contain a given prefix: starting from the corresponding node

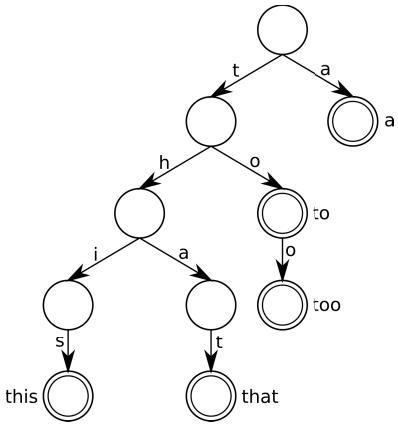

Fig. 5. Prefix tree containing the words “a”, “to”, “too”, “this” and “that”. Double circles indicate that the word-flag is set.

of the prefix, all descendant nodes are collected which have the word-flag set. As an example, the characters and words following the prefix “th” in Figure 5 are determined. The node is found by starting at the root node and following the edges “t” and “h”. Possible following characters are the outgoing edges “i” and “a” of this node. Words starting with the given prefix are “this” and “that”.

Aoe et al. [10] show an efficient implementation of this datastructure. Finding the node for a prefix with length L needs $\mathcal{O}(L)$ time in their implementation. The time to find all words containing a given prefix depends on the number of nodes of the tree. An upper bound for the number of nodes is the number of words W times the maximum number of characters M of a word, therefore the time to find all words is $\mathcal{O}(W \cdot M)$.

B. Language Model

A LM is able to predict upcoming words given previous words and it is also able to assign probabilities to given sequences of words [11]. It can be queried to give the probability $P(w|h)$ that a word sequence (history) h is followed by the word w. Such a model is trained from a text by counting how often w follows h. The probability of a sequence is then $P(h) = P(w_1) \cdot P(w_2|w_1) \cdot P(w_3|w_1, w_2) \cdot ... \cdot P(w_n|w_1, w_2, ..., w_{n-1})$ [11].

It is not feasible to learn all possible word sequences, therefore an approximation called N-gram is used [11]. Instead of using the complete history, only a few words from the past are used to predict the next word. N-grams with N=2 are called bigrams. Bigrams only take the last word into account, i.e. they approximate $P(w_n|h)$ by $P(w_n|w_{n-1})$. The probability of a sequence is then given by $P(h) = P(w_1) \cdot \prod_{n=2}^{|h|} P(w_n|w_{n-1})$. Another special case is the unigram LM, which does not consider the history at all but only the relative frequency of a word in the training-text, i.e. $P(h) = \prod_{n=1}^{|h|} P(w_n)$.

The N-gram distributions are learned from a training-text. For the unigram distribution, the number of occurrences of a word is counted and normalized by the total number of words in the text. The bigram distribution is calculated by

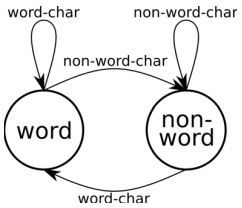

Fig. 6. A beam can be in one of two states.

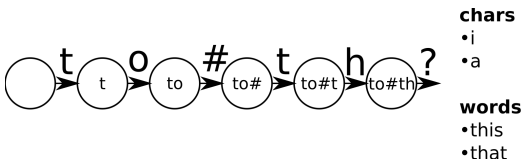

Fig. 7. A beam currently in the word-state. The "#" character represents a non-word character. The current prefix "th" can be extended by "i" and "a" and can be extended to form the words "this" and "that". The prefix tree from Figure 5 is assumed in this example.

first counting how often a word w_1 is followed by a word w_2 and then normalizing by the total number of words which follow w_1. If a word is contained in the test-text but not in the training-text, it is called OOV word. In this case a zero probability is assigned to the sequence, even if only one OOV word occurs. To overcome this problem *smoothing* can be applied to the N-gram distribution, more details are available in Jurafsky [11].

IV. Proposed algorithm

WBS decoding is a modification of VBS decoding and has the following properties:

- Words are constrained to dictionary words.
- Any number of non-word characters is allowed between words.
- A word-level bigram LM can optionally be integrated.
- Better running time than token passing (regarding time-complexity and real running time on a computer).

To constrain words to dictionary words and also allow arbitrary non-word characters between words, each beam is in one of two states. If a beam gets extended by a word-character (typically "a", "b", ...), then the beam is in the word-state, otherwise it is in the non-word-state. Figure 6 shows the two beam-states and the transitions. A beam is extended by a set of characters which depends on the beam-state. If the beam is in the non-word-state, the beam-labeling can be extended by all possible non-word-characters (typically " ", ".", ...). Furthermore, it can be extended by each character which occurs as the first character of a word. These characters are retrieved from the edges which leave the root node of the prefix tree. If the beam is in word-state, the prefix tree is queried for a list of possible next characters. Figure 7 shows an example of a beam currently in the word-state. The last word-characters form the prefix "th", the corresponding node in the prefix tree is found by following the edges "t" and "h". The outgoing edges of this node determine the next characters, which are "i" and "a" in this example. In addition, if the current prefix represents a complete word (e.g. "to"), then the next characters also include all non-word-characters.

Optionally, a word-level LM can be integrated. A bigram LM is assumed in the following text. The more words a beam contains, the more often it gets scored by the LM. To account for this, the score gets normalized by the number of words. Four possible LM scoring-modes exist and names are assigned to them which are used throughout the paper:

- Words: only a dictionary but no LM is used.
- N-grams: each time a beam makes a transition from the word-state to the non-word-state, the beam-labeling gets scored by the LM.
- N-grams + Forecast: each time a beam is extended by a word-character, all possible next words are queried from the prefix tree. Figure 7 shows an example for the prefix "th" which can be extended to the words "this" and "that". All beam-extensions by possible next words are scored by the LM and the scores are summed up. This scoring scheme can be regarded as a LM forecast.
- N-grams + Forecast + Sample: in the worst case, all words of the dictionary have to be taken into account for the forecast. To limit the number of possible next words, these are randomly sampled before calculating the LM score. The sum of the scores must be corrected to account for the sampling process.

Algorithm 1 shows the pseudo-code for WBS decoding. The set B holds the beams of the current time-step, and P holds the probabilities for the beams. P_b is the probability that the paths of a beam end with a blank, P_{nb} that they end with a non-blank, and P_{txt} is the probability assigned by the LM. P_{tot} is an abbreviation for $P_b + P_{nb}$. The algorithm iterates from $t = 1$ through $t = T$ and creates a tree of beam-labelings as shown in Figure 3. An empty beam is denoted by $\varnothing$ and the last character of a beam is indexed by -1. The best beams are obtained by sorting them with regard to $P_{tot} \cdot P_{txt}$ and only keep the BW best ones. For each of the beams, the probability of seeing the beam-labeling at the current time-step is calculated. Separately book-keeping for paths ending with a blank and paths ending with a non-blank accounts for the CTC coding scheme. Each beam gets extended by a set of possible next characters, depending on the beam-state. When extending a beam, the LM calculates a score P_{txt} depending on the scoring-mode. The normalization of the LM score is achieved by taking P_{txt} to the power of $1/numWords(b)$, where $numWords(b)$ is the number of words contained in the beam b. After the algorithm finished its iteration through time, the beam-labelings get completed if necessary: if a beam-labeling ends with a prefix not representing a complete word, the prefix tree is queried to give a list of possible words which contain the prefix. Two different ways to implement the completion exist: either the beam-labeling is extended by the most likely word (according to the LM), or the beam-labeling is only completed if the list of possible words contains exactly one entry.

Algorithm 1: Word Beam Search

Data: RNN output matrix mat, BW and LM
Result: most probable labeling

$B = \{\varnothing\}$;
$P_b(\varnothing, 0) = 1$;
for $t = 1...T$ **do**
  $bestBeams = bestBeams(B, BW)$;
  B={};
  for $b \in bestBeams$ **do**
    if b != $\varnothing$ **then**
      $P_{nb}(b,t)$ += $P_{nb}(b,t-1) \cdot mat(b(-1),t)$;
    end
    $P_b(b,t)$ += $P_{tot}(b,t-1) \cdot mat(blank,t)$;
    $B = B \cup b$;
    $nextChars = nextChars(b)$;
    for $c \in nextChars$ **do**
      $b' = b + c$;
      $P_{txt}(b') = scoreBeam(LM, b, c)$;
      if $b(t) == c$ **then**
        $P_{nb}(b',t)$ += $mat(c,t) \cdot P_b(b,t-1)$;
      else
        $P_{nb}(b',t)$ += $mat(c,t) \cdot P_{tot}(b,t-1)$;
      end
      $B = B \cup b'$;
    end
  end
end
$B = completeBeams(B)$;
return $bestBeams(B, 1)$;

The time-complexity depends on the scoring-mode used. If no LM is used, the only difference to VBS is to query the next possible characters. This takes $\mathcal{O}(M \cdot C)$ time, where M is the maximum length of a word and C is the number of unique characters. The overall running time therefore is $\mathcal{O}(T \cdot BW \cdot C \cdot (log(BW \cdot C) + M))$. If a LM is used, then a lookup in a unigram and/or bigram table is performed when extending a beam. Searching such a table takes $\mathcal{O}(log(W))$. This sums to the overall running time of $\mathcal{O}(T \cdot BW \cdot C \cdot (log(BW \cdot C) + M + log(W)))$. In the case of LM forecasting, the next words have to be searched which takes $\mathcal{O}(M \cdot W)$. The LM is queried S times, where S is the size of the word sample. If no sampling is used, then $S = W$. The running time sums to $\mathcal{O}(T \cdot BW \cdot C \cdot (log(BW \cdot C) + S \cdot log(W) + W \cdot M))$.

V. Results

Evaluation is done using the IAM and Bentham HTR datasets. The goal of the evaluation is to compare the performance of different decoding algorithms given the same neural network output. It is not about achieving or outperforming state-of-the-art results on the mentioned datasets. Character Error Rate (CER) and Word Error Rate (WER) are chosen as error measures [12]. The neural network is inspired by the CRNN model proposed by Shi et al. [13] and is implemented using the TensorFlow framework. It consists of seven convolutional layers, two RNN layers and a final CTC layer. The output sequence of the neural network has a length of 100 time-steps. IAM consists of 79 different characters while Bentham has 93 characters, therefore the output of the RNN is a matrix of size $T \times (C + 1) = 100 \times 80$ and 100×94 respectively. The tested algorithms are: best path decoding, token passing, VBS and WBS. The last three algorithms are implemented as custom TensorFlow operations in C++. For WBS decoding, the four scoring modes are evaluated. Experiments for the BW values 15, 30 and 50 are conducted. The LM is either trained with the text of the test-set (denoted as *Te*) or the text of the training-set concatenated with a word list[2] (denoted as *Tr+L*) which consists of 370,099 words. The resulting dictionary sizes are as follows: 3,707 and 373,412 unique words for IAM and 1,911 and 372,933 unique words for Bentham. Training the LM with the text that has to be recognized can be seen as the best case and is of course a simplification (LM has zero OOV words). Using the training-set concatenated with the word list, on the other hand, is a very rudimentary training-text for the LM. In practice, results can be expected to be in between these two extreme cases. The LM uses add-k smoothing with a smoothing value of $k = 0.01$.

The results of the experiments are shown in Table I and are given in the format CER (%) / WER (%) / time per sample (ms). The latter value includes the time needed to evaluate the neural network. WBS is always able to outperform best path decoding and token passing in at least one of its scoring-modes. Regarding the BW, increasing this value for VBS only marginally changes the results. The CER does not change at all except when using the *Tr+L* training-text for IAM, while the WER varies by around 0.1%. This suggests that a BW of 15 is large enough for this algorithm. In contrast, both error measures improve when increasing the BW of WBS. CER gets improved by up to 0.76% and WER by up to 0.86%. WBS using *Words* mode outperforms VBS as long as the BW is large enough (30 or 50). A possible explanation for the different impact of this parameter can be derived from the variability of the words contained in the beam-labelings of a single time-step. The word-variability of VBS is greater than the one of WBS and the beam-labelings of the latter algorithm mainly differ in the punctuation. Therefore WBS needs a larger number of beams to allow recovering from a wrong word-hypothesis. Increasing the BW from 15 to 50 increases the running time by a factor of around 3. Regarding the scoring modes of WBS (BW fixed to 15 from now on), the WER achieved by the *Words* mode can always be outperformed by at least one of the other modes which incorporate N-gram probabilities. For IAM the best CER and WER is obtained by *N-grams + Forecast* mode for the *Tr+L* training-text and *N-grams + Forecast + Sample* mode for the *Te* training-text. For both training-texts of Bentham *N-grams* mode yields the best WER. The CER also benefits from considering N-gram probabilities while decoding. The only exception is when using

[2]Taken from https://github.com/dwyl/english-words

TABLE I
EXPERIMENTAL RESULTS GIVEN AS CER / WER / TIME PER SAMPLE (MS). LM TRAINING TEXTS: TRAINING-SET CONCATENATED WITH WORD-LIST (TR+L), TEST-SET (TE). WBS SCORING MODES: WORDS (W), N-GRAMS (N), N-GRAMS + FORECAST (N+F), N-GRAMS + FORECAST + SAMPLE (N+F+S).

	IAM		Bentham	
Algorithm	**Tr+L**	**Te**	**Tr+L**	**Te**
Best Path Decoding	8.77 / 29.07 / 12	8.77 / 29.07 / 12	5.60 / 17.06 / 15	5.60 / 17.06 / 15
Token Passing	(not feasible)	10.46 / 12.37 / 762	(not feasible)	8.16 / 9.24 / 1250
VBS, BW=15	8.48 / 28.24 / 56	8.27 / 27.34 / 64	5.55 / 16.39 / 69	5.35 / 16.02 / 63
VBS, BW=30	8.49 / 28.27 / 101	8.27 / 27.36 / 108	5.55 / 16.45 / 125	5.35 / 15.96 / 124
VBS, BW=50	8.49 / 28.27 / 168	8.27 / 27.32 / 184	5.55 / 16.55 / 210	5.35 / 16.06 / 202
WBS, mode=W, BW=15	8.95 / 24.19 / 90	5.62 / 11.01 / 85	5.47 / 14.09 / 77	4.22 / 7.90 / 56
WBS, mode=W, BW=30	8.44 / 23.77 / 145	5.08 / 10.42 / 140	5.18 / 13.92 / 156	3.90 / 7.60 / 104
WBS, mode=W, BW=50	8.25 / 23.67 / 229	4.86 / 10.15 / 217	5.12 / 13.87 / 289	3.73 / 7.50 / 182
WBS, mode=N, BW=15	10.00 / 23.88 / 83	5.33 / 9.77 / 56	6.15 / 13.85 / 99	4.07 / 7.08 / 74
WBS, mode=N+F, BW=15	8.61 / 22.86 / 16388	5.23 / 9.82 / 1040	6.76 / 18.00 / 24465	4.05 / 7.36 / 274
WBS, mode=N+F+S, BW=15	8.62 / 22.91 / 12226	5.21 / 9.78 / 786	6.75 / 18.06 / 18349	4.06 / 7.39 / 223

the *Tr+L* training-text for Bentham, in which case *Words* mode achieves the best CER. Best path decoding is the fastest algorithm which needs at most $15ms$ per sample. VBS is around 5 times slower than best path decoding. The running time of WBS mainly depends on the scoring mode and the dictionary size. It stays below $100ms$ for the *Words* and *N-grams* modes. Increasing the dictionary size by a factor of 100 increases the running time by a factor of 1.5 for *N-grams* mode on the IAM dataset. However, when using *N-grams + Forecast* mode on the same dataset the running time increases by a factor of 15.7. Therefore the forecasting-modes are only feasible for small dictionaries, while the *Words* and *N-grams* modes can also be used with large dictionaries. Token passing is only evaluated for the smaller dictionary (because of its quadratic dependence on the dictionary size) created from the *Te* training-text, for which the algorithm takes $762ms$ per sample for IAM and $1250ms$ for Bentham. This proves the claim that WBS is faster than token passing when used with *N-grams* mode (which matches the type of LM integrated into token passing).

VI. CONCLUSION

A decoding algorithm for CTC-trained neural networks was proposed which restricts words to dictionary words, allows arbitrary character strings between words, can optionally integrate a word-level LM and is faster than token passing. It comes with four different scoring-modes which govern the effect of the LM and can also be used to control the running time. The algorithm was evaluated on the IAM and Bentham HTR datasets. Experiments have shown that the algorithm is able to outperform best path decoding, VBS and token passing for both an ideal and a rudimentary LM. The running time of the *Words* and *N-grams* mode is in the order-of-magnitude of VBS. In case only a dictionary but no word-level LM is available, the *Words* mode is well suited which constrains the words of the beam-labelings.

ACKNOWLEDGMENTS

This work has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 674943 (project READ).

REFERENCES

[1] A. Graves, *Supervised sequence labelling with recurrent neural networks*. Springer, 2012.
[2] A. Graves, S. Fernández, F. Gomez, and J. Schmidhuber, "Connectionist Temporal Classification: labelling unsegmented sequence data with recurrent neural networks," in *Proceedings of the 23rd International Conference on Machine Learning*. ACM, 2006, pp. 369–376.
[3] K. Hwang and W. Sung, "Character-level incremental speech recognition with recurrent neural networks," in *IEEE International Conference on Acoustics, Speech and Signal Processing*. IEEE, 2016, pp. 5335–5339.
[4] A. Graves, M. Liwicki, S. Fernández, R. Bertolami, H. Bunke, and J. Schmidhuber, "A novel connectionist system for unconstrained handwriting recognition," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 31, no. 5, pp. 855–868, 2009.
[5] S. Ortmanns, A. Eiden, H. Ney, and N. Coenen, "Look-ahead techniques for fast beam search," *Computer Speech & Language*, vol. 14, no. 1, pp. 15–32, 2000.
[6] U. Marti and H. Bunke, "The IAM-database: an English sentence database for offline handwriting recognition," *International Journal on Document Analysis and Recognition*, vol. 5, no. 1, pp. 39–46, 2002.
[7] J. Sánchez, V. Romero, A. Toselli, and E. Vidal, "ICFHR2014 competition on handwritten text recognition on transcriptorium datasets," in *14th International Conference on Frontiers in Handwriting Recognition*. IEEE, 2014, pp. 785–790.
[8] A. Graves and N. Jaitly, "Towards end-to-end speech recognition with recurrent neural networks," in *Proceedings of the 31st International Conference on Machine Learning*, 2014, pp. 1764–1772.
[9] P. Brass, *Advanced data structures*. Cambridge University Press Cambridge, 2008.
[10] J. Aoe, K. Morimoto, and T. Sato, "An efficient implementation of trie structures," *Software: Practice and Experience*, vol. 22, no. 9, pp. 695–721, 1992.
[11] D. Jurafsky and J. Martin, *Speech and Language Processing*. Pearson London, 2014.
[12] T. Bluche, "Deep Neural Networks for Large Vocabulary Handwritten Text Recognition," Ph.D. dissertation, Université Paris Sud-Paris XI, 2015.
[13] B. Shi, X. Bai, and C. Yao, "An End-to-End Trainable Neural Network for Image-based Sequence Recognition and its Application to Scene Text Recognition," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 39, no. 4, pp. 2298–2304, 2016.