Comparison of Connectionist Temporal Classification Decoding Algorithms

Harald Scheidl
Student of Computer Science
Vienna University of Technology
e0725084@student.tuwien.ac.at

ABSTRACT

Recurrent Neural Networks (RNNs) are used for sequence recognition tasks such as Automatic Speech Recognition (ASR) and Handwritten Text Recognition (HTR). The output of such a RNN is a matrix containing label probabilities for each time-step. To decode these per-frame predictions into a final labeling multiple Connectionist Temporal Classification (CTC) decoding algorithms exist. Two algorithms are presented and compared in this paper. The first one is beam search decoding, which is an approximation of a breadth-first search through the tree of possible labelings. A character-level Language Model (LM) can be integrated. The second algorithm is token passing. The recognized text is constrained to a sequence of dictionary words. A word-level LM can be integrated to score consecutive words. This paper compares the concepts the algorithms are based on. Different types of LMs are presented and the integration into the CTC algorithms is discussed. A quantitative comparison is given for the Bentham handwriting dataset. A trained HTR system is coupled with both algorithms and the error rates are compared.

KEYWORDS

connectionist temporal classification, language model,
recurrent neural network, speech recognition, handwritten text recognition.

1 Introduction

Sequence recognition is the task of transcribing sequences of data with sequences of labels [2]. The focus of this paper lies on two well-known use-cases: HTR and ASR. RNNs are promising models to handle sequential data, however they suffer from the fact that training needs an error signal for each sequence element [3]. Graves [3] introduces the CTC operation which enables neural network training from pairs of data and target labelings. The RNN is trained to output the target sequence in a specific coding schema. Two selected decoding algorithms are analyzed and compared in this paper. Graves [4] proposes the *token passing* algorithm which outputs the most probable sequence of dictionary words. Hwang [5] introduces *beam search decoding* which iteratively searches the labeling with highest score. Both algorithms can be extended by a LM to incorporate information about language structure.

The objective of this paper is to compare beam search decoding and token passing and to give suggestions which algorithm to use for specific use-cases. The comparison is done regarding the concepts the algorithms are based on, time-complexity, issues arising in practice and the performance on a HTR task.

This paper is structured as follows: first the foundations of sequence recognition are presented. The coding schema used by the CTC operation is explained and both word-level and character-level LMs are introduced. Afterwards, the two mentioned CTC decoding algorithms are discussed. A theoretical comparison between both of them is given and the evaluation is done with the Bentham HTR dataset. Finally, the conclusion highlights the advantages and disadvantages of the algorithms and gives suggestions on when to use them.

2 Sequence Recognition

Sequential data occurs in the form of handwritten text, speech and gestures among others [3]. Hidden Markov Models (HMMs) show good performance on tasks such as HTR and ASR [4]. However, RNNs are also capable of modeling sequences and have advantages over HMMs from a theoretical and from a practical point of view [4]. When using RNNs without CTC, there are two disadvantages: the first one is that the training data must be annotated for each time-step and the second one is that the per-frame predictions must be postprocessed to yield the final labeling, i.e. the text in the case of HTR and ASR. When using the CTC, dataset creation does not require specifying the exact position of the characters in the input as shown in Figure 1. The annotation "The evidence" is assigned to the image as a whole. While training, the neural network figures out on its own where the text lies in the image, the same applies for the speech signal.

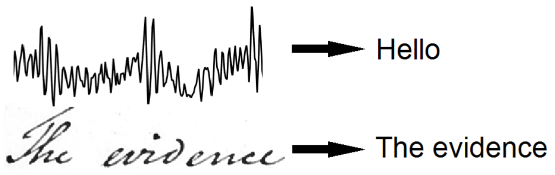

Figure 1: Two common sequence recognition problems. Top: part of a speech signal and its annotation. Bottom: an image of handwritten text and its annotation [7].

3 CTC Coding Schema

To understand the loss and decoding algorithms, it is essential to understand the coding schema used by the CTC operation. A 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 character is added to the RNN output which is called *blank* and is denoted by "-" in this paper. For each sequence element the character probabilities sum to 1 [4]. Picking one character for each time-step from the RNN output and concatenating these characters form a path π [4]. 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 blank labels [4]. A way to model this encoding is by using a Finite State Machine (FSM) [5]. The FSM is created as follows: the labeling is extended by adding a blank label in between all characters and at the beginning and at the end of the labeling. For each character (and blank) in the labeling, a state is inserted into the FSM. A self-loop is added to each state to allow repeated labels on a path. For consecutive characters, a direct transition and a transition through a blank is added. The only exception is the case of repeated characters (like in pi*zz*a), then only a transition through the blank is allowed to avoid ambiguousness. Figure 2 shows a FSM which produces valid paths for the labeling "ab": this is achieved by proceeding from a start state through arbitrary transitions and states to the final state and outputting the label of a state each time a state is reached. A valid path for "ab" is " - a a - - b - -", another one is "a b".

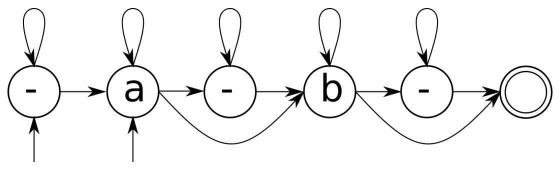

Figure 2: FSM which produces valid paths (encodings) for the labeling "ab". The two left-most states are the initial states while the right-most state is the final state.

To decode a path into a labeling, the encoding operation implemented by the FSM has to be inverted. This is done by the collapsing function B [5]. It is applied to a path π and yields a labeling l by first removing repeated labels 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 (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 or spoken text. A first approximation of decoding the RNN output is to take the most probable label 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 3. The given RNN output has a length of 2

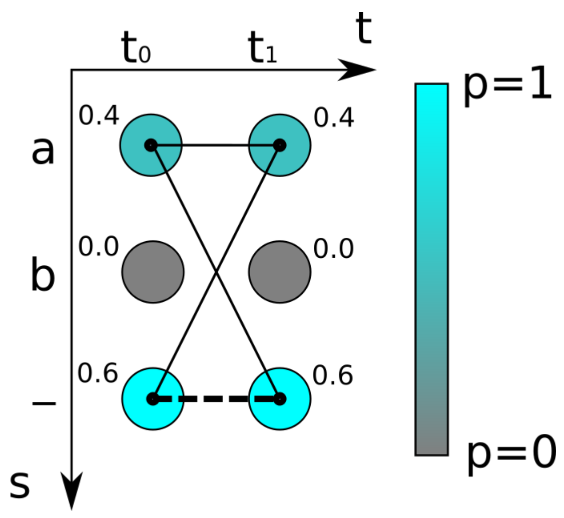

Figure 3: An example of a RNN output. The sequence has a length of 2 with time-steps t_0 and t_1. For each time-step a probability distribution over the possible characters (“a” and “b” and the blank character “-”) is outputted by the RNN. Lines indicate four different paths through the RNN output: the dashed line indicates the best path yielding the labeling “” while the thin lines indicate paths yielding the labeling “a”.

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 $2 \cdot 0.6 \cdot 0.4 + 0.4 \cdot 0.4 = 0.64$. In contrast to best path decoding, both beam search decoding and token passing are able to tackle such situations.

4 Language Model

A LM can be used to guide the CTC decoding algorithms by incorporating information about the language structure. The language is modeled on character-level or on word-level, but the basic ideas are the same for both. For simplicity only a word-level LM is discussed. A LM is able to predict upcoming words given previous words and it is also able to assign probabilities to given sequences of words [6]. For example, a well trained LM should assign a higher probability to the sentence “There are ...” than to “Their are ...”. A LM 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 $P(h) = P(w_1) \cdot P(w_2|w_1) \cdot P(w_3|w_1, w_2) \cdot ... \cdot P(w_n|w_1, w_2, ..., w_{n-1})$ [6].

It is not feasible to learn all possible word sequences, therefore an approximation called N-gram is used [6]. 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, that means they approximate $P(w_n|h)$ by $P(w_n|w_{n-1})$. The probability of a sequence is then given by $P(h) = \prod_{n=2}^{|h|} P(w_n|w_{n-1})$. 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. $P(h) = \prod_{n=1}^{|h|} P(w_n)$.

If a word is contained in the test text but not in the training text, it is called Out Of Vocabulary (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 [6].

5 Decoding Algorithms

This section presents beam search decoding and token passing. For both algorithms pseudo-code is shown and explained. Further, the time-complexity is derived based on the pseudo-code.

5.1 Beam Search Decoding

The following text is based on the paper from Hwang [5]. Hwang introduces beam search decoding in the context of ASR. A simplified version of Hwang’s algorithm is shown in Algorithm 1 which ignores depth-pruning. The labelings (or beams) y are created by iterating through time. The genesis beam is the empty labeling and is denoted by $\varnothing$. At each time-step, the beams in the set *Beams* are extended by all possible characters c from the alphabet C. The extensions are weighted by a character-level LM which scores seeing the new character c and the last character $y[-1]$ in the beam y next to each other. Besides the LM score, also the score according the the RNN output is incorporated. Extending all labelings by all possible characters for each time-step creates a tree of labelings as shown in Figure 4. Further, one instance of the original beam from the last time-step is kept: this accounts for paths extended by a blank or by the last character of the beam. As the number of vertices (representing labelings) of this tree grows exponentially in time, only the bw best-scoring labelings from the set *Beams* are kept (where the parameter bw is called beam width) to keep the algorithm feasible. A labelings score holds one LM weight per character, that means the longer the labeling, the more LM weights get multiplied. This yields low scores for long labelings, therefore a normalization step is needed at the end of the algorithm. Finally, the best labeling according to the scores at the final time-step T is returned by the algorithm.

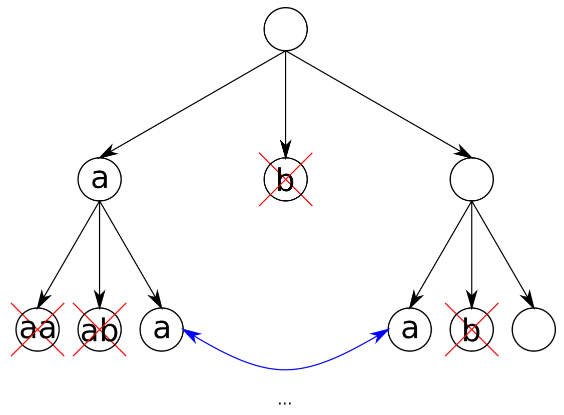

Figure 4: Iteratively extending labelings (from top to bottom) forms this tree of labelings. Only the two best-scoring beams are kept per time-step (i.e. $bw = 2$), all others are removed (red). Equal labelings get merged (blue).

The time-complexity is not specified in the paper from Hwang [5], however it can be derived from the presented pseudo-code. 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 $\mathcal{O}(T \cdot bw \cdot C \cdot log(bw \cdot C))$. The two inner loops are ignored because they only account for $\mathcal{O}(bw \cdot C)$.

5.2 Token Passing

This algorithm is proposed by Graves [4], however the following discussion is based on another publication from Graves [2]. A dictionary W containing words is given and the output of the algorithm is constrained to a sequence of these words. For each word a word-model is created, which essentially is a state machine connecting consecutive characters with regard to the CTC coding schema: a character of a word is represented by a path on which the character occurs one or multiply times and is optionally followed by one or multiply blanks (see CTC coding schema). 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

Algorithm 1: CTC Beam Search

```
Data: RNN output, beam width $bw$ and LM $P$
Result: most probable labeling
1  $Beams = \{\varnothing\}$;
2  $Score(\varnothing, 0) = 1$;
3  for $t = 1...T$ do
4      $\hat{B} = sort(Beams)[0 : bw]$;
5      $Beams = \{\}$;
6      for $y \in \hat{B}$ do
7          calculate $Score(y, t)$ by incorporating RNN output;
8          add $y$ to $Beams$;
9          for $c \in C$ do
10             $y' = y + c$;
11             calculate $Score(y', t)$ by incorporating RNN output and $P(c|y[-1])$;
12             add $y'$ to $Beams$;
13         end
14     end
15 end
16 normalize $Score(y, T)$ according to $|y|$;
17 return $y$ with highest $Score(y, T)$;
```

of already visited words. Figure 5 shows three word-models which are put in parallel.

Pseudo-code is shown in Algorithm 2. The algorithm iterates through all time steps t. At each time-step t, a loop goes through all dictionary words w. For each word w starting at the current time-step, the most probable word w^* ending at the previous time-step is searched. This accounts for leaving a word-model and entering another word-model. The new score is calculated by using the score of the ending word w^* multiplied by the LM score of seeing both words w and w^* next to each other. A new token holding the calculated score and the word history extended by the word w is added. Afterwards, the most probable alignment of the extended word w' (corresponding to the mentioned word-model) is calculated and a token is added which accounts for the end-state of the new word. The extended word w' has a blank label at the beginning, the end and in between all characters and accounts for possible alignments of the word. Finally, the algorithms looks for the best scoring token at the final time-step T. The word history stored in this token is returned as a result.

The time-complexity of this algorithm is $\mathcal{O}(T \cdot W^2)$, where T denotes the sequence length and W the dictionary size [2]. This can easily be checked by looking at the pseudo-code: for T time-steps and W words, the most likely preceding word is searched which needs another loop

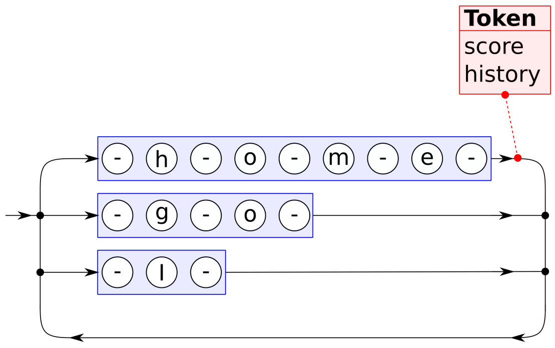

Figure 5: Three word models (blue) are put in parallel. Information flow is implemented by tokens (red) which are passed through the states and between the words.

over the W dictionary words.

Algorithm 2: CTC Token Passing

```
Data: output of RNN, dictionary $W$ and LM $P$
Result: most probable word sequence
1  Initialize all tokens for all words at time-step $t = 1$;
2  for $t = 2...T$ do
3      for $w \in W$ do
4          get $w^* \in W$ for which end-token $tok$ at time $t-1$ exists and for which $tok.score \cdot P(w|w^*)$ is maximal;
5          calculate score by incorporating LM and add $w$ to history of new token;
6          add token;
7          $w' = extend(w)$;
8          for $s = 1...length(w')$ do
9              find best alignment of $w'$ by incorporating RNN output and calculate score;
10             add token;
11         end
12         add end-token representing end of this word;
13     end
14 end
15 get end-token $tok$ at time-step $T$ with maximum $tok.score$;
16 return $tok.history$;
```

6 Results

First, the concepts of the algorithms are compared from a theoretical point of view. Afterwards, a HTR dataset and a trained neural network are used to evaluate the performance of both algorithms.

6.1 Conceptual Comparison

Token passing constraints its output to dictionary words. The larger the dictionary, the more words can be recognized, however increasing the dictionary size quadratically increases the running time. OOV words are not handled by token passing, therefore the behavior in this case is undefined (a similar word may be detected at the position of the OOV word). The problem of OOV words also includes numbers (e.g. detecting numbers from 1 to 10000 requires adding these 10000 numbers to the dictionary) and hyphenation. Beam search decoding on the other hand allows arbitrary output texts.

The word-level (character-level) bigram LM is trained by sampling the frequency of two words (characters) next to each other in a training text. At least W^2 words (characters) are needed in the training text such that each possible pair of W unique words (characters) is encountered at least once. Character-level LMs require less training text than word-level LMs if the number of unique characters is lower than the number of unique words as it is the case with the Bentham dataset: there are 93 unique characters and 8274 unique words.

Regarding the running times, both algorithms depend linearly on the sequence length of the RNN output. As already stated, the possible dictionary size of token passing is limited due to its quadratic influence on the runtime. Beam search decoding depends quasi-linearly on the alphabet size (fixed for a given dataset) and on the beam width (tunable parameter). Decreasing the beam width may yield lower accuracy, however it also decreases the running time.

Both algorithms can be implemented using log-likelihood. This avoids numerical problems when using the limited precision floating point arithmetic of a computer.

6.2 Evaluation

To evaluate the performance of the algorithms, Character Error Rate (CER) and Word Error Rate (WER) are used as error measures. The CER is the Levenshtein distance between the ground truth text and the recognized text, normalized by the length of the ground truth text [1]. The same applies for WER, but on word-level. Equation 1 shows how the CER is calculated, calculating the WER goes accordingly: the minimal number of edit operations (insert, delete, substitute) is divided by the length of the ground truth text GT. Values greater than 100% occur when the length of the ground truth text is less than the edit distance.

Type	Description
Input	gray-value line-image
Conv+Pool	kernel 5×5, pool 2×2
Conv	kernel 5×5
Conv+Pool+BN	kernel 3×3, pool 2×2
Conv	kernel 3×3
Conv	kernel 3×3
Conv+BN	kernel 3×3
Conv+Pool	kernel 3×3, pool 2×2
MDLSTM	bidir., 512 hidden cells
Mean	avg. along vert. dim.
Collapse	remove dimension
Project	project onto C classes
CTC	decode or loss

Table 1: Architecture of the neural network. Abbreviations: average (avg), bidirectional (bidir), vertical (vert), dimension (dim), batch normalization (BN), convolutional layer (Conv).

$$CER = \frac{\#ins + \#del + \#sub}{length(GT)} \tag{1}$$

The algorithms are compared using the Bentham HTR dataset, a sample is shown in Figure 6. It is written by Jeremy Bentham (1748-1832) and his secretarial staff [7]. It contains 433 pages and is mostly written in English, some parts are Greek or French [7]. Data is annotated on line-level and partly on word-level. For this paper a neural network using the TensorFlow framework is implemented and trained. It consists of three main parts: first, the convolutional layers, then the recurrent layers and finally the decoding layer. The recurrent layers use a RNN type called Multidimensional Long Short Term Memory (MDLSTM). MDLSTM essentially propagates information through both dimensions (vertical and horizontal) of the image. An overview of the architecture is given in Table 1. The input image has a size of 800×64 pixels and the output text has a length of at most 100 characters.

Evaluation is done by combining the algorithms presented with this neural network. A baseline result is obtained by best path decoding. The results are shown in Table 2, where each result is given in the format CER/WER. As can be seen, beam search decoding outperforms best path decoding for both CER and WER. This is due to correctly incorporating multiple paths into the result and due to the help of the LM. Token passing outperforms beam search decoding regarding the WER. This makes sense as token passing constraints its output to dictionary words. However, if token passing makes a wrong word prediction, the word usually differs by many characters, which explains the high CER.

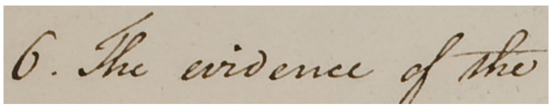

Figure 6: A sample from the Bentham HTR dataset [7].

Dataset	Path	Beam	Token
Bentham	5.72/16.74	5.55/16.18	8.24/9.34

Table 2: Result for the Bentham HTR dataset. The results are given in the format CER/WER. Columns: best path decoding (Path), beam search decoding (Beam) and token passing (Token).

7 Conclusion

This paper first gave an introduction to sequence recognition and afterwards presented two decoding algorithms for RNNs trained with the CTC loss function. Best path decoding is a simple approximation and produces the correct labeling if the per-frame predictions are sharply peaked. Beam search decoding uses a character-level LM and achieves the best results regarding CER on the Bentham dataset. In contrast to word-level LMs, character-level LMs have the advantage that they do not suffer from OOV words. Token passing constrains its output to a sequence of dictionary words. The following list helps to decide which algorithm to use:

- If (nearly) all words to be recognized are known in advance, token passing achieves better results than the other algorithms regarding WER.
- If speed matters, best path decoding is the way to go.
- In all other cases, beam search decoding is recommended.

References

[1] T. Bluche. *Deep Neural Networks for Large Vocabulary Handwritten Text Recognition.* PhD thesis, Université Paris Sud-Paris XI, 2015.

[2] A. Graves. *Supervised sequence labelling with recurrent neural networks.* Springer, 2012.

[3] 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*, pages 369–376. ACM, 2006.

[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*, 31(5):855–868, 2009.

[5] K. Hwang and W. Sung. Character-level incremental speech recognition with recurrent neural networks. In *IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 5335–5339. IEEE, 2016.

[6] D. Jurafsky and J. Martin. *Speech and Language Processing*. Pearson London, 2014.

[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*, pages 785–790. IEEE, 2014.