Increasing robustness of handwriting recognition using character n-gram decoding on large lexica

Martin Schall
Institute for Optical Systems
University of Applied Sciences
Constance, Germany
Email: martin.schall@htwg-konstanz.de

Marc-Peter Schambach
Siemens Postal, Parcel &
Airport Logistics GmbH
Constance, Germany
Email: marc-peter.schambach@siemens.com

Matthias O. Franz
Institute for Optical Systems
University of Applied Sciences
Constance, Germany
Email: mfranz@htwg-konstanz.de

Abstract*—Offline handwriting recognition systems often include a decoding step, that is retrieving the most likely character sequence from the underlying machine learning algorithm. Decoding is sensitive to ranges of weakly predicted characters, caused e.g. by obstructions in the scanned document. We present a new algorithm for robust decoding of handwriting recognizer outputs using character n-grams. Multidimensional hierarchical subsampling artificial neural networks with Long-Short-Term-Memory cells have been successfully applied to offline handwriting recognition. Output activations from such networks, trained with Connectionist Temporal Classification, can be decoded with several different algorithms in order to retrieve the most likely literal string that it represents. We present a new algorithm for decoding the network output while restricting the possible strings to a large lexicon. The index used for this work is an n-gram index with tri-grams used for experimental comparisons. N-grams are extracted from the network output using a backtracking algorithm and each n-gram assigned a mean probability. The decoding result is obtained by intersecting the n-gram hit lists while calculating the total probability for each matched lexicon entry. We conclude with an experimental comparison of different decoding algorithms on a large lexicon.

I. Introduction

The problem of recognizing unconstrained and unsegmented handwriting text using artificial neural networks has received considerable attention in recent years. Current publications show that artificial neural networks can be trained successfully to recognize single lines of unsegmented handwritten text [1]. Practical applications of offline handwriting recognizers include scanned texts with obstructed, damaged or dirty parts. Decoding of the neural network output at such positions is difficult.

State of the art solutions use recurrent neural networks containing Long-Short-Term-Memory *LSTM* cells [2] [3] [4] in multi-dimensional hierarchical subsampling [5] [6] networks. Connectionist Temporal Classification *CTC* [7] [1] is applied as a supervised training for neural networks, training both character classification and localization on unsegmented handwritten text. CTC trains the network using a variant of the forward-backward-algorithm [8] to infer the posterior distribution of the characters using the networks prediction and the known correct label. Handwriting recognition based on this technology has been awarded top positions in international competitions. The output of such a network is a one-dimensional sequence of label probabilities. In order to interpret the output activations as label probabilities, the Softmax function is applied on each position of the sequence individually. The data structure of the final network is a two-dimensional matrix with one dimension representing the spatial distribution of the characters and the other dimension specifying the learned labels for character recognition. Probabilities at each position in the output sum up to one with each individual probability in the range between zero and one.

A network trained with CTC classifies handwritten text from an input image and produces a sequence of label probabilities. CTC is a supervised training for artificial neural networks which trains the network for both the classification and localization of characters. As such it does not require the input to be pre-segmented or the training data labeled with spatial information. CTC does train the network to learn the correct label classifications and order of the sequence but not necessarily the exact character positions [9, chap. 7.2].

Decoding of the network output generates a readable character sequence with a high combined probability given the label probabilities estimated by the network. Decoding can be done both with and without constraining the output strings to a given lexicon or grammar [9, chap. 7.5].

The algorithm proposed in this paper generates an n-gram index for a given lexicon of allowed character strings and uses this index to decode the output of an recurrent neural network trained with CTC. Decoding of the output is done by finding n-grams with above-average combined probability in it and, in a second step, combining these n-grams to match entries of the given lexicon. Extraction of the n-grams is done in a robust way. The result of one decoder run is a small subset of lexicon entries with combined probabilities assigned to them. Combined probabilities are constituted of weighted combinations of the contained n-grams.

The paper begins in section II by detailing the proposed decoding algorithm starting with the n-gram index generation in II-A, followed by information on the network output in II-B, extraction of n-grams from it in II-C, intersection of n-grams in II-E and confidence estimation in II-F. A modification of the decoding algorithm to allow single word detection or whole line decoding is given in II-G. Details on the algorithm are followed by the results of an experimental comparison with other decoding algorithms in section III. The paper concludes with a discussion of the results in section IV.

II. Methodology

A. Generating the index

The n-gram index consists of a map from n-grams to one hit list per n-gram referencing the lexicon entries containing the mapped n-gram. This map contains each key (the n-gram) only once and uses exact look-up algorithms to find n-grams. Several different data structures, such as binary trees, tries or hash maps, are suitable for this task. Since building the n-gram index is done once and after that accessed read-only, emphasis should be placed on a low complexity for the look-up of an n-gram. The hit lists contain references to lexicon entries and the position of the n-gram within each specific entry.

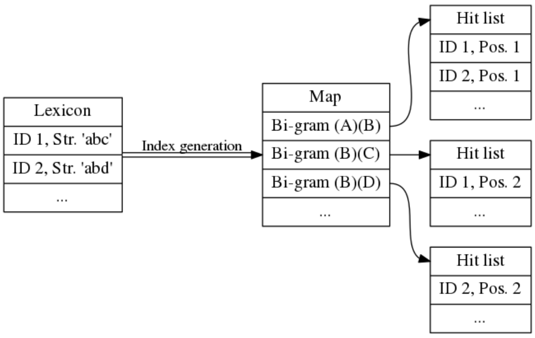

Fig. 1. Example index generation

Figure 1 shows an example index using bi-grams and with two example strings. The digits of the bi-grams are enclosed in brackets to illustrate that they are class labels and not the actual characters. The double lined arrow in the figure represents the index generation to compile the provided word lexicon into the n-gram index for later use. The single lined arrows illustrate references within the data structure, pointing from n-grams within the map to their according hit lists.

N-grams within the index are generated by traversing each string contained in the lexicon once and extracting the n-gram starting at the current position. Since the network output is restricted to a predefined set of labels and only these can be matched, the lexicon entry must be converted to its label sequence before extracting the n-grams. Characters need to be mapped to labels, learned by the neural network, before n-gram generation. Characters that are not used as a label are ignored. Characters are mapped to labels, the classes learned by the neural network, in order to reduce the number of trained classes and to consolidate similar looking characters. The length of the individual n-grams is kept constant for the whole index, common choices are lengths of two or three (bi-grams or tri-grams). Using shorter n-grams allows for more error tolerance, but more n-grams with less information gain will be generated, resulting in a negative impact on the run-time. The choice of the n-gram size is thus a trade-off between error tolerance and run-time. Each generated n-gram is inserted into the map and the ID of the lexicon entry appended to the assigned hit list.

Generating the n-gram index for a given lexicon is dependent on the configuration of the used artificial neural network but not on the actual output of the network for a specific input. The set of all label classes and the assignment from character to label class must be known from the configuration of the network before generating the index. This allows to build the index once for a given lexicon and network configuration in advance in order to reduce run-time usage by storing and reusing the index.

Later decoding of the network output requires an intersection of the hit lists assigned to the found n-grams. Intersection of multiple hit lists is the task of collecting entries that occur in multiple such hit lists and has been in current literature [10] [11] [12]. To reduce the run-time of this algorithm, the hit lists are sorted by the lexicon entry's ID in ascending order, hit list items with equal entry ID are further ordered by the ascending position of the n-gram within the entry. This order is easily achieved by generating the n-grams starting with the first lexicon entry and appending the entry ID to the end of the related n-gram hit lists.

N-grams occurring in a large number of lexicon entries result in a large hit list within the index. Long hit lists require a high amount of disk space for storage and have a negative impact on run-time for hit list intersection. On the other hand, frequently occurring n-grams contain little amount of information that can be used for distinguishing between multiple lexicon entries. Consequently, very long hit lists should be removed from the index. In our paper, n-grams that occur in more than ten percent of the lexicon entries were removed from the index.

B. Network output activations

Connectionist Temporal Classification [7] *CTC* is an output layer for recurrent neural networks. It uses a variant of the forward-backward-algorithm [8] to infer posterior probabilities for the label classes based on the networks estimation and the known correct label sequence. The network itself is then trained using backpropagation [13] with the negative logarithmic likelihood target function. The output of a network trained with CTC is a one-dimensional sequence of label probabilities. As such the output has a non-fixed length growing linearly with the width of the input image. In fact the output length is determined by the combined width of the sub-sampling windows applied by the chosen network topology. Each position in the output sequence has a fixed number of values, one for each label class that can be recognized by the network. The application of the Softmax function individually at each output sequence position allows the interpretation of the output as label probabilities over the sequence.

Networks trained with CTC recognize one more label class than necessary for the printable characters used in the written language. This additional label class is called the 'blank-label' or 'non-label' and is an invisible separator label. A network trained with CTC does predict labels in the output as a series of 'spikes', one for each label in the sequence.

Since after subsampling individual characters are still likely to have a width greater than one in the network output, the number of spikes (localized label predictions) is lower than the length of the network output. This and the unknown image segmentation makes the *blank-label* necessary [7]. The *blank-label* separates individual characters within the output sequence and allows distinguishing between one character in the input triggering multiple adjacent output activations and true repetitions of the same character within the input.

Note that the prediction of the *blank-label* does not necessarily indicate the absence of a visible character, but possibly the continuation of the previous or following character.

TABLE I
EXAMPLE OUTPUT SEQUENCE FOR THE INPUT STRING 'ABBC'

Position	1	2	3	4	5	6	7	8	9	10
(A)	0.5	0.9	0.1	0.0	0.3	0.0	0.1	0.0	0.3	0.1
(B)	0.0	0.0	0.1	0.7	0.3	0.9	0.7	0.0	0.2	0.0
(C)	0.3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.4	0.7
'blank'	0.2	0.1	0.8	0.3	0.4	0.1	0.2	1.0	0.1	0.2

Table I shows the possible network output sequence for the example input string 'abbc'. Note that the same label can be repeatedly strongly activated both with and without repetition of the same character within the input. A change of the strongest activation from one label to another or the strong activation of the *blank-label* signals the transition from one label class to another. The total activations at each output position always sum up to one with a minimal activation of zero.

One basic decoding algorithm for this type of network output is detailed in [9, chap. 7.5.1], called 'Best Path Decoding'. It is based on finding the label with the maximum activation at each position of the network output and concatenating these to a intermediate label string. Removing repetitions of the same label and the invisible blank-label from this label string yields the decoded string. Since this algorithm yields non-optimal results for network outputs with multiple weakly predicted labels, 'Prefix Search Decoding' [9, chap. 7.5.2] should be preferred for practical applications.

C. Extracting n-grams from the network output

A network trained with CTC is primed to produce output activations in spikes, that is in short and confined subsequences within the output sequence. For good recognizable, distinguishable handwritten input these spikes of the output activations alternate between the *blank-label* and labels related to printable characters or symbols.

For the proposed decoding algorithm, each n-gram must start and end with a printable character or symbol because the *blank-label* is used solely as a invisible separator label. Also the *blank-label* is not used during index generation, making the matching of n-grams containing *blank-labels* impossible. In total, a n-gram of n labels consists of n printable labels and up to $n-1$ *blank-labels*. A total of at last n and up to $(2 \times n) - 1$ spikes within the output activations are used per n-gram. Contained *blank-labels* between the visible character labels is seen as the normal case because the network output is most likely longer than the correct string with the labels predicted as spikes, thus containing predicted *blank-labels* in between. *Blank-labels* in this case mark either regions without a drawn character or continuations of neighboring characters.

N-grams are extracted from the network output by using a backtracking [14] algorithm. Backtracking starts at a given position within the sequence and produces n-grams starting at this position. To extract n-grams covering the whole output sequence, this backtracking algorithm must be started for each position in the output sequence.

The backtracking algorithm proposed in this work does collect the n-grams and calculate their probability at the same time. It uses a recursive depth-first strategy for building n-grams and calculating the mean probability. Repetitions of the same label are resolved by using only the maximum activation of the label within the repeating sub-sequence. This way only the peak of the activation spike for each label of the n-gram is used for probability calculation. On the other hand, the spikes for different labels are allowed to be distributed over wide sub-sequences, which makes the n-gram extraction more resilient against false-positive activations or multiple mediocre activations at the same position.

The probability of each extracted n-gram is calculated as the mean probability

$$p(G|y) = \frac{1}{(n \times 2) - 1} \sum_{g \in G} y_{l_g}(i_g) \tag{1}$$

with G being the set of $(n \times 2) - 1$ activation spikes that contributed to the n-gram, each a tuple of label l_g and position i_g. $y_l(i)$ defines the network output for label l at position i. This allows for further interpretation of the n-gram probability in terms of stochastic probabilities.

The backtracking algorithm is restricted by thresholds for the activations within the network output. Only paths with an activation higher than the threshold can be followed, resulting in a lower number of extracted n-grams but also reduced runtime requirements. The activation threshold is allowed to be different for each label. In this work, a threshold of 0.25 was used for printable labels and 0.001 for the *blank-label.*

D. Index access with incomplete information

The following sections detail the extraction of n-grams from the networks output activations and intersection of their hit lists. Equations for estimating the probabilities of both individual n-grams and lexicon entries are based on calculating the weighted mean probability over their elements. General decoding algorithms calculate probabilities with respect to principles of the Viterbi-algorithm [15] [16], a product of the element probabilities.

Accessing the proposed n-gram index should be done, in order to reduce run-time, with as few n-grams as possible while still using n-grams contributing to the distinction between relevant and non-relevant lexicon entries. This leads to incomplete information during the intersection of the extracted n-gram hit lists. As a result, lexicon entries having not all their included n-grams extracted from the network output will be processed during hit list intersection. Probabilities for these n-grams are thus unknown to the algorithm.

Correct probabilities for the missing n-grams can be calculated by an online evaluation of the current lexicon entry based on the network output. Since the indexed lexicon and the hit lists are potentially large, this approach would degrade the run-time of the proposed decoding algorithm. Because of this, a simple approximation of the missing n-gram probabilities is used by the proposed decoding algorithm.

A Viterbi-like-decoder, calculating the product over all best path element probabilities, needs an estimation of these unmatched n-gram probabilities. Estimations of zero or one probabilities lead to the lexicon entry being discarded entirely or unmatched parts having no effect at all. This would mean either biasing partially matched entries positively or discarding them entirely. Probability estimations in between, without further evaluation of the network output, act as a constant coefficient in the final probability that can be arbitrarily chosen.

All three potential behaviors in Viterbi-like-decoders with incomplete information from extracting the n-grams within the network output are unwanted in this context. It is to be expected that not all n-grams related to any given lexicon entry were extracted from the network output.

The proposed algorithm calculates the lexicon entries probability as a weighted mean and allows unmatched n-grams to be included in the equation with zero probability as estimate. This penalizes partly matched entries without discarding them entirely. Pruning of the n-gram extraction in this case reduces the run-time without directly affecting the results in a negative way. Pruning during the n-gram extraction then needs to be chosen carefully in order to allow extraction of n-grams that are strongly predicted by the network. Weakly predicted n-grams can be discarded during extraction in order to reduce the run-time of the algorithm.

E. Intersection of n-gram hit lists

The final result of the proposed algorithm is a set of lexicon entries related to the network output activations in terms of a high mean probability of the lexicon entries labels given the network output activation. Mean probabilities of the lexicon entries are provided by the algorithm as a measure of confidence for further pruning or processing of the result.

Retrieval of the matched lexicon entries is related to the *multiple search* and *t-intersection* problems. These problems describe the task of intersecting two or more ordered sequences with the additional constraint that each element of the intersection must be contained in at least t of the sets. Algorithms for solving these problems are published [10] [11] [12].

Input for this step of the proposed algorithm is a set of n-grams extracted from the network output and the generated n-gram index itself. Hit lists for the extracted n-grams are retrieved used the map contained in the n-gram index. N-grams that occur in no lexicon entry, and thus do not map to lexicon entries, are discarded. Frequent n-grams with their hit lists removed are also discarded.

Matching of the lexicon entries is done by intersecting the hit lists of the extracted n-grams. Additional constrains are put in order that require the positions of the extracted n-grams within the network output sequence and the lexicon entry to be in the same order. This prevents usage of the n-grams in arbitrary order and thus generation of arbitrary strings that may not even be contained in the lexicon.

During intersection, each matched lexicon entry is rated with its mean probability over the used n-grams. Mean probabilities of the matched entries can be used both for ordering and pruning the result of the intersection. Calculation of the mean probabilities per lexicon entry is detailed in section II-F.

F. Confidence value for matched entries

Part of the matching of the network output sequence against the lexicon is not only finding relevant entries but also to measure the relevancy in terms of a probability. This allows ordering and pruning of the result but also possibly influences following algorithms in the whole system.

The total probability of a matched lexicon entry is calculated out of the probabilities of the matched n-grams. As detailed before, the probabilities of extracted n-grams are the mean of their $(n \times 2) - 1$ activation spikes in the network output. To continue this idea, the total probabilities of lexicon entries are calculated as the weighted mean over their matched n-grams. N-grams that are included in the lexicon entry, but not extracted from the network output are assumed to have a probability of zero.

N-grams are allowed to overlap while matching a lexicon entry with a maximum of n n-grams per position of the entry. The exception are the front and rear $n - 1$ positions of the entry that can not be matched by as many n-grams. This makes weighting the n-gram probabilities for mean calculation necessary in order to assign equal significance to each position of the entry. N-grams at the front and rear of the sequence must be weighted stronger than the n-grams in the middle.

Each entry position can potentially be matched by 1 to n n-grams. For weight calculation, a total weight of n is assigned to each entry position and this weight has to be shared between the n-grams overlapping this position. This gives a total weight of $n \times l$ for n-grams of size n and an entry of length l to the entry. N-grams have a weight of at least n with higher weights in the front and rear position because they share their weights with fewer or no other n-grams in these positions.

The total weight of an n-gram of size n at position i of an entry of length l is defined as

$$w(i, n, l) = \sum_{x=i}^{i+n} \frac{n}{s(x, n, l)} \tag{2}$$

with

$$s(x, n, l) = \max(n - 1, \min(x, l - (x + n))) + 1 \tag{3}$$

yielding the number of n-grams that share the entry position x. Positions in these equations are counted starting with zero.

TABLE II
EXAMPLE DISTRIBUTION OF TRI-GRAMS OVER THE STRING 'ABCDEFG'

Weight							
5.5	(A)	(B)	(C)				
3.5		(B)	(C)	(D)			
3.0			(C)	(D)	(E)		
3.5				(D)	(E)	(F)	
5.5					(E)	(F)	(G)

Table II gives an example for tri-grams ($n = 3$) and the lexicon entry 'abcdefg' ($l = 7$). The total weight of the lexicon entry is 21 that has to be shared by the included n-grams. The middle tri-gram shares all its positions with two other tri-grams, it thus has a weight of 3. The first tri-gram is the only one including the first entry position and shared the second position with only one other tri-gram, receiving a weight of $3.0 + 1.5 + 1.0 = 5.5$. Weight calculation is similar for the other n-grams.

Final probability of a matched lexicon entry e based on the extracted n-grams is

$$p(e|M, n, y) = \frac{1}{n \times |e|} \sum_{m \in M} w(i_m, n, |e|) \times p(G_m|y) \tag{4}$$

with the set of n-grams matched by the lexicon entry and network output M, the length of a single n-gram n, the length $|e|$ of the lexicon entry and network output y. $p(G_m|y) = 0$ for n-grams not extracted from the network output.

G. Single word detection vs. whole line decoding

Equation 4 defines the probability of a matched lexicon entry as the mean probability with respect to the length of the lexicon entry. This allows lexicon entries to reach the maximum probability of 1 even when fully matching the lexicon entry, but only a small part of the network output. This behavior of the algorithm is not unwanted for some use-cases because it allows one network output to produce multiple best matches, each covering a sub-sequence of the network output. This can be used as a word detection algorithm for localization of single words within longer texts.

However, the algorithm can be easily modified for decoding whole lines at once with the best match being the lexicon entry that covers the full network output. For this the normalization term in equation 4 has to be changed from $\frac{1}{n \times |e|}$ to $\frac{1}{n \times |y|}$ with $|y|$ being the length of the network output. With this modification, a lexicon entry has probability 1 if it does fully match the network output.

The probability expression for lexicon entries can be further modified to suite specific use-cases. Word detection with emphasis on longer lexicon entries can be achieved by calculating both the unmodified and modified probability 4 for the lexicon entries and calculating a weighted mean of the two. The proposed decoding algorithm can then be configured for any wanted emphasis on single word detection or whole line decoding.

III. RESULTS

This section contains results of the comparison of different artificial neural network decoding algorithms and their configurations in the context of offline handwriting recognition. The used network was for all tests identical, the only difference being the decoding algorithms and configurations.

The network topology was a three-layered multidimensional hierarchical subsampling network [6] [5] using LSTM cells [2] [3] [4]. Training was done using CTC [7]. Data for training and evaluation of the network were handwritten postal addresses from both the USA and Canada. A pre-processing step binarized the images and separated them into single lines of text. 135000 such images were used for training, 3000 each for validation and test. There were no overlaps between these three sets.

The trained neural network showed an 6.86% Character Error Rate *CER* on the test set, 7.01% CER on the validation set and 5.50% CER on the training set. CER was measured using nominal results from the Prefix Search Decoding described by [9, chap. 7.5.2]. CER is the percentage of the Edit-distance [17] to the length of the correct string.

Tests were done using the trained neural network and the test data set with a lexicon containing the labels for the images within all three data sets. The lexicon contained 423170 different strings, including the correct labels and variants of them. Since the lexicon contained the correct labels, CER was lower than with a nominal-only decoding.

Comparison was done between the following decoding algorithms:

- *Constrained Decoding on full set* as described by [9, chap. 7.5.3].
- *Best Path Decoding & Levenshtein* using Best Path Decoding [9, chap. 7.5.1] to generate a nominal string and followed by a search for similar lexicon entries using a full-table-scan with the Levenshtein distance [17] [18] as the measure. This provided a baseline for the error rate.
- *n-Gram index decoding & Constrained Decoding* using the proposed decoding algorithm to filter the full lexicon, generating a small subset of it. Constrained Decoding was applied on this reduced lexicon to produce the final result.
- *n-Gram index decoding only* with the proposed decoding algorithm, but without the afterward evaluation of the reduced lexicon by Constrained Decoding. Instead the top ranked lexicon entry was used for comparison.

Application of a ten percent threshold for frequent tri-grams resulted in zero tri-grams being removed from the index.

The results are shown in table III with the best results in Character Error Rate and wall clock run-time highlighted in bold text. CER and run-time were calculates as the mean over 3000 test set images. The result with a reasonable trade-off between CER and run-time is marked in italic font.

TABLE III
RESULTS OBTAINED USING DIFFERENT DECODING ALGORITHMS ON THE DESCRIBED TEST SET

Decoder	Configuration	CER	Run-time
Constrained Dec. on full set	beam-width 10000	1.05%	81.9ms
Constrained Dec. on full set	beam-width 100000	0.76%	2297.7ms
Constrained Dec. on full set	beam-width unlimited	**0.58%**	7444.5ms
Best Path Dec. & Levenshtein		1.52%	756.6ms
n-Gram index dec. & Constrained Dec.	tri-grams, filter-size 100, beam-width 10000	0.87%	15.0ms
n-Gram index dec. & Constrained Dec.	tri-grams, filter-size 500, beam-width 10000	*0.65%*	*18.9ms*
n-Gram index dec. only	tri-grams	3.05%	**13.3ms**

IV. Discussion

Using the proposed decoding algorithm as a filter of the lexicon in combination with Constrained Decoding showed CER values near of the unrestricted Constrained Decoding algorithm in our experiments, while using only a portion of the run-time. This suggests that the network output produced by a hierarchical subsampling network using LSTM cells trained with CTC is expressive enough to allow extraction of n-grams without using context information. This in turn allows the application of common information retrieval algorithms for the problem of decoding an artificial neural networks output.

On the other hand, experiments show also that using the proposed decoding algorithm alone results in sub-optimal CER values while using equal configurations. This leads to the conclusion, that the proposed algorithm acts as a efficient filter of the lexicon, but the ranking of lexicon entries is not optimal.

Further research is thus necessary to improve the proposed algorithm or adapt other information retrieval algorithms to this problem in order to leave out the Constrained Decoding in the process without loosing a large portion of the CER.

The original character strings of the lexicon should be stored separately for use cases in which they are required as part of the decoding result since the generated n-gram index does not allow for a loss-less reconstruction of the character strings. The index does not allow loss-less reconstruction since characters not trained by the artificial neural network and very frequent n-grams were removed from it. Removal of very frequent n-grams is less of a practical concern since in our experiments, no n-gram occurred in more than ten percent of the lexicon entries.

Acknowledgment

The authors would like to thank Jörg Rottland for his valuable suggestions and proof-reading of this paper.

References

[1] 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.

[2] S. Hochreiter and J. Schmidhuber, "Long short-term memory." *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[3] F. A. Gers, J. Schmidhuber, and F. Cummins, "Learning to forget: continual prediction with LSTM." *Neural computation*, vol. 12, no. 10, pp. 2451–2471, 2000.

[4] K. Greff, R. K. Srivastava, J. Koutník, B. Steunebrink, and J. Schmidhuber, "LSTM: A Search Space Odyssey," IDSIA/USI-SUPSI, Tech. Rep., 2015.

[5] A. Graves, S. Fernandez, and J. Schmidhuber, "Multi-Dimensional Recurrent Neural Networks," IDSIA/USI-SUPSI, Tech. Rep., 2007.

[6] A. Graves and J. Schmidhuber, "Offline Handwriting Recognition with Multidimensional Recurrent Neural Networks," in *Advances in Neural Information Processing Systems 21, NIPS'21*, 2008, pp. 545–552.

[7] A. Graves, S. Fernandez, 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 Press, 2006, pp. 369–376.

[8] L. R. Rabiner, "A tutorial on hidden Markov models and selected applications in speech recognition," *Proceedings of the IEEE*, vol. 77, no. 2, pp. 257–286, 1989.

[9] A. Graves, *Supervised Sequence Labelling with Recurrent Neural Networks*. Springer, 2012.

[10] J. Barbay and C. Kenyon, "Adaptive intersection and t-threshold problems," in *Proceedings of the thirteenth annual ACM-SIAM symposium on Discrete algorithms*. Society for Industrial and Applied Mathematics, 2002, pp. 390–399.

[11] R. Krauthgamer, A. Mehta, V. Raman, and A. Rudra, "Greedy list intersection," Weizmann Institute; IBM Almaden Research Center; Google Inc.; Department of Computer Science and Engineering, University of Buffalo, State University of New York, Tech. Rep., 2007.

[12] D. Tsirogiannis, S. Guha, and N. Koudas, "Improving the Performance of List Intersection," *Proceedings of the VLDB Endowment*, vol. 2, no. 1, pp. 838–849, 2009.

[13] R. Rojas, "The Backpropagation Algorithm," in *Neural Networks*. Springer, 1996, pp. 151–184.

[14] C. E. Leiserson, R. L. Rivest, C. Stein, and T. H. Cormen, *Introduction to Algorithms, Third Edition*. MIT Press, 2009.

[15] J. Forney, G.D., "The Viterbi Algorithm," *Proceedings of the IEEE*, vol. 61, no. 3, 1973.

[16] G. D. Forney Jr, "The Viterbi Algorithm: A Personal History," *arXiv:cs/0504020*, 2005.

[17] V. I. Levenshtein, "Binary codes capable of correcting deletions, insertions, and reversals," *Soviet Physics Doklady*, vol. 10, no. 8, pp. 707–710, 1966.

[18] R. A. Wagner and M. J. Fischer, "The String-to-String Correction Problem," *Journal of the ACM*, vol. 21, no. 1, pp. 168–173, 1974.