Label Transition and Selection Pruning and Automatic Decoding Parameter Optimization for Time-Synchronous Viterbi Decoding

Yasuhisa Fujii
Google Research
Mountain View, CA 94043
Email: yasuhisaf@google.com

Dmitriy Genzel
Google Research
Mountain View, CA 94043
Email: dmitriy@google.com

Ashok C. Popat
Google Research
Mountain View, CA 94043
Email: popat@google.com

Remco Teunen
Google Research
Mountain View, CA 94043
Email: remco@google.com

***Abstract*—Hidden Markov Model (HMM)-based classifiers have been successfully used for sequential labeling problems such as speech recognition and optical character recognition for decades. They have been especially successful in the domains where the segmentation is not known or difficult to obtain, since, in principle, all possible segmentation points can be taken into account. However, the benefit comes with a non-negligible computational cost. In this paper, we propose simple yet effective new pruning algorithms to speed up decoding with HMM-based classifiers of up to 95% relative over a baseline. As the number of tunable decoding parameters increases, it becomes more difficult to optimize the parameters for each configuration. We also propose a novel technique to estimate the parameters based on a loss value without relying on a grid search.**

I. Introduction

Markov-model-based classifiers have successfully been used in the context of Optical Character Recognition (OCR) [1], [2]. Our *Aksara* OCR system follows the same line in a generalized way with a log-linear model-based sequential classifier [3]. HMM-based classifiers have been effective especially for the cases where the segmentation is not known or difficult to obtain [2], since, in principle, all possible segmentation points can be taken into account. However, the benefit comes with a non-negligible computational cost. Time-synchronous Viterbi decoding with beam pruning is used for the decoding with an HMM-based classifier, where an input is represented as a sequence of frames which are processed one by one while expanding and pruning hypotheses. The bottleneck of the approach is that all hypotheses to be expanded at each potential segmentation point are extended to all output labels (characters). This can be quite expensive, especially if the system has a large number of labels. In this paper, we propose two simple yet effective pruning algorithms which directly deal with the problem and show that the new algorithms yield significant improvements on decoding speed.

The new pruning algorithms introduce three additional decoding parameters resulting in five pruning parameters in total. It is not trivial to optimize those parameters for each configuration (e.g., language) every time a component of the system has been updated. To solve this issue, we also propose a novel algorithm to optimize the parameters based on the quality of the decoding without relying on a grid search. It makes it possible for us to optimize the parameters every time a model is updated for a configuration using the same criterion. We present accuracy results comparing with two well-known OCR systems.

This paper is organized as follows: Section II describes our system based on the log-linear model-based sequential classifier. Section III elaborates on the new pruning algorithms. The algorithm to tune the decoding parameters will be described in Section IV. The experimental results will be shown in Section V followed by the conclusions and future work in Section VI.

II. System Description

A. Log-linear model-based sequential classifier

We define OCR as a task which takes an image as input, determines the text regions in it and produces sequences of Unicode points, or UTF-8 strings for them. In this work, we assume that each region corresponds to a horizontal or vertical line and such regions are provided a priori. We use a layout engine provided by Tesseract [4] to extract lines from images and focus only on the line recognition in this paper.

Aksara employs a log-linear model framework which can incorporate a variety of knowledge from different sources to compute the cost between an input line image X and a sequence of Unicode points Y through feature functions. We compute the cost between X and Y as follows:

$$C(X,Y) = \sum_i \lambda_i \Phi_i(X,Y) \tag{1}$$

where $\Phi_i(X,Y)$ is a cost between X and Y computed by a feature function i and λ_i is its weight. To make the decoding feasible, we assume that the input is represented as a sequence of frames and the cost is computed for each frame. Let $Z = (z_t)_1^T$ be a sequence of states of a feature function. We assume 1-to-n correspondence between Y and Z where a single Y can be uniquely inferred given a single Z. Let $\boldsymbol{z}_i(Y)$ be a function to return a set of Z from which Y is inferred for a feature function i. We compute $\Phi_i(X,Y)$ as follows:

$$\Phi_i(X,Y) = \min_{Z \in \boldsymbol{z}_i(Y)} \sum_t \phi_i(X, z_{t-1}, z_t, t) \tag{2}$$

We distinguish feature functions depending on whether they rely on z_{t-1} or not. The former are called *transition feature* *functions* while the latter are called *observation feature functions.*

The weights of the feature functions are optimized using *Minimum Error Rate Training* (MERT) [5]. We use character error rate (CER), defined as the character edit distance from the reference, divided by the reference length, as the error metric.

B. Decoding

The task of decoding is to find $\hat{Y}$ which yields the lowest cost for a given input X based on Eq. (1),

$$\hat{Y} = \operatorname*{argmin}_{Y} C(X, Y). \tag{3}$$

The cost computed by Eq. (1) can be divided into frames based on Eq. (2) for all feature functions. Therefore, we can use the standard time-synchronous Viterbi decoding to solve Eq. (3). The recognition states are created by combining the state of each feature function on-the-fly during decoding. Normally, beam pruning is used to reduce the number of hypotheses to be examined. The most basic pruning algorithm are based on two criteria: histogram pruning [6] and cost-width pruning [7]. Histogram pruning keeps only a specified number of hypotheses at each frame based on the costs of the hypotheses, while cost-width pruning keeps only those hypotheses whose costs are less than the cost of the best hypotheses plus a specified threshold. Let h be a hypothesis at a frame. Let $H(t, X)$ be a set of hypotheses to be pruned at frame t given an input X. Let $\boldsymbol{c}(h)$ be a function to return the cost of h. Let $\boldsymbol{r}(h, H)$ be a function to return the rank of h in H based on the cost. The set of hypotheses $H'_{\text{hist}}(t, X)$ and $H'_{\text{width}}(t, X)$ pruned by the histogram pruning and the cost-width pruning from $H(t, X)$, respectively, are then computed as follows:

$$H'_{\text{hist}}(t, X) = \{h \in H(t, X) |\ \boldsymbol{r}(h, H(t, X)) \le \theta_h\}, \tag{4}$$

$$H'_{\text{width}}(t, X) = \left\{h \in H(t, X) |\ \boldsymbol{c}(h) \le \boldsymbol{c}(\hat{h}) + \theta_w\right\}, \tag{5}$$

where

$$\hat{h} = \operatorname*{argmin}_{h \in H(t,X)} c(h), \tag{6}$$

and θ_h and θ_w are tunable parameters to control the maximum number of hypotheses and the maximum cost difference between a hypothesis and the best hypothesis at each frame, respectively. $H'(t, X)$, a set of hypotheses after pruning at frame t, is obtained as follows:

$$H'(t, X) = H'_{\text{hist}}(t, X) \cap H'_{\text{width}}(t, X). \tag{7}$$

C. Feature functions

We define feature functions from HMMs and a language model. An HMM is defined for each Unicode grapheme cluster[1] [8] unless there are hand-crafted rules to split the grapheme cluster into smaller chunks. If they exist, we use the split chunks as units of modeling. The units of such modeling are referred to as *atoms* in this paper. Specifically, the atoms are the labels output by HMMs. We assume a left-to-right topology for HMMs and allow skip transitions. As the emission model, we use either Gaussian mixture models (GMMs) or deep neural networks (DNNs) in the form of the hybrid approach as described in [9]. The transitions are defined based on the distance between the original and destination states and all transitions are tied across all HMMs. We use emission probabilities, transitions probabilities, transitions between labels (a.k.a. insertion penalty), transitions for a null label (for thin spaces between characters, i.e., single-frame spaces that are lingustically inert) as feature functions from the HMMs. State prior probabilities are also used as a feature function if DNNs are used as the emission model.

[1] *Grapheme cluster* and *character* are used interchangeably; the former to emphasize that a character may be composed of several parts.

Language models are defined over sequences of Unicode points. We use probabilities from the language models as a feature function. We do not use a dictionary or word-based language model because it is not well-defined in the written form for many languages, for example, Chinese. In addition, we prefer not to have the problem of out-of-vocabulary words which dictionary-based approaches entail. The method has not ruled out a hybrid approach that includes both character- and word-based language models although the present work uses only the former.

III. Label Transition and Selection Pruning

The system described in Section II allows us to support a large number of languages in a unified framework. However, the decoding speed can be slow with the standard beam pruning algorithm explained in Section II-B since any label (atom) can be connected to any label at all potential segmentation points, causing an explosion in the number of hypotheses created at each potential segmentation point. However, we can speed the decoding up if the number of label transition hypotheses (i.e. hypotheses to be expanded with new labels) and the number of labels to be connected at each frame are reduced appropriately.

A. Label transition pruning

The first approach is to reduce the number of label transition hypotheses. In speech recognition, it is empirically known that a tighter cost threshold can be used for word-ending states than for word-interior states [6]. In a similar vein, we propose to apply a different (tighter) cost width threshold only for the label transition hypotheses and change the criterion for the cost width pruning as follows:

$$H'_{\text{width}}(t) = \left\{h \in H(t, X)) |\ c(h) \le c(\hat{h}) + \theta_w(h)\right\}, \tag{8}$$

where

$$\theta_w(h) = \begin{cases} \theta_w & \text{if } h \text{ is not label transition hypothesis,} \\ \theta_{t_w} & \text{otherwise,} \end{cases} \tag{9}$$

where θ_{t_w} is a tunable parameter to determine the maximum difference of the costs of label transition hypotheses from the best hypothesis at each frame. This method is similar to language model pruning [6] which uses a tighter threshold for word ending states. The difference between our method and language model pruning is that we compute the difference of the costs against the best hypothesis of all the hypotheses, while language model pruning typically computes the difference within the word ending states.

B. *Label selection pruning*

The second approach is to reduce the number of labels to be connected to label transition hypotheses. In speech recognition, a technique known as phoneme look-ahead has been used to quickly obtain a set of labels to be connected by computing the costs of the connections in a fast but approximated way [10]. We propose a method to select labels to be connected based only on the costs incurred by observation feature functions without any approximations. Let $L(t)$ be the set of all possible labels to be connected to label transition hypotheses along with their costs computed by only observation feature functions at frame t. We propose to select $L'(t)$ based on the following criterion:

$$L'(t) = \left\{ l \in L(t) |\, \boldsymbol{r}(l, L(t)) \leq \theta_{l_s}, \boldsymbol{c}(l) \leq \boldsymbol{c}(\hat{l}) + \theta_{l_w} \right\} \quad (10)$$

where

$$\hat{l} = \operatorname*{argmin}_{l \in L(t)} \boldsymbol{c}(l), \quad (11)$$

and θ_{l_s} and θ_{l_w} are tunable parameters to determine the maximum number of labels allowed to be connected and the maximum difference of the costs between a label and the best label at each frame, respectively. As mentioned above, the cost of each label is computed only by using the observation feature functions. In our case, this corresponds to using the cost of the initial state of each label computed by the emission model of the HMM. The costs computed for the label selection can be cached and used when creating the hypotheses for the surviving labels. While it might seem that this does not reduce any computational costs, the label selection happens before new hypotheses are created, and therefore it reduces a significant amount of overhead as well as the computational costs for the transition feature functions (such as the language model).

IV. Automatic Decoding Parameter Optimization

As the number of tunable decoding parameters increases, it becomes more difficult to optimize them. Usually, a grid-search, which tries to find the best configuration by examining a set of configurations, is used to tune the parameters. This has two major problems, though. First, the range of values of a parameter can be different for each model (system) and therefore it is not known a priori. Second, it is time-consuming since it needs to decode all samples in a development set for each configuration.

To solve these problems, we propose a novel algorithm to optimize the parameters based on a loss value without relying on a grid search. We consider the following constrained optimization problem:

$$\hat{\Theta} = \operatorname*{argmin}_{\Theta} \mathcal{T}(\mathcal{D}; \Theta) \text{ s.t. } \mathcal{L}(\mathcal{D}; \Theta) \leq \alpha, \quad (12)$$

where $\mathcal{D} = \{(X^i, Y^i)\}_1^{|\mathcal{D}|}$ is the development set, $\mathcal{T}(\mathcal{D}; \Theta)$ is the decoding speed given $\mathcal{D}$ with the set of decoding parameters $\Theta = (\theta_h, \theta_w, \theta_{t_w}, \theta_{l_s}, \theta_{l_w})$, $\mathcal{L}(\mathcal{D}; \Theta)$ is a loss function for $\mathcal{D}$ with Θ, and α is an allowable loss value. We define the loss function as follows:

$$\mathcal{L}(\mathcal{D}; \Theta) = \sum_{(X,Y) \in \mathcal{D}} \min \left(1, \sum_i \delta \left(\hat{Z}^i, \hat{Z}^i_{\Theta} \right) \right), \quad (13)$$

where $\delta(\cdot, \cdot)$ is Kronecker's delta,

$$\hat{Z}^i = \operatorname*{argmin}_{Z \in \boldsymbol{z}_i(\hat{Y})} \sum_t \phi_i(X, z_{t-1}, z_t, t), \quad (14)$$

and $\hat{Z}^i_{\Theta}$ is the decoding result under Θ and the feature function i. We can obtain $\hat{\Theta}$ based on the following algorithm if we use Eq.(13) as the loss function, there is a monotonic relationship between the value of a decoding parameter and its computational cost and we optimize each parameter independently:

1) Decode $(X, Y) \in \mathcal{D}$ with $\dot{\Theta}$ and create a lattice for each sample. $\dot{\Theta}$ is empirically chosen so that the Viterbi paths for a lattice and the input are the same.
2) Compute the optimal value for each decoding parameter over all lattices. The actual algorithm to compute the value for each parameter is described below.
3) Sort the computed values in ascending order of the computational cost.
4) The k-th element in this sorted sequence is the optimal value for the parameter where $k = \lceil (1-\alpha) \cdot |\mathcal{D}| \rceil$ with $0 \leq \alpha < 1$.

Note that unlabeled data can be also used as $\mathcal{D}$ since it does not require transcriptions for the optimization. This algorithm requires samples in the development set to be decoded only once.

The algorithm to compute the optimal value for a lattice is different for each decoding parameter. The algorithms are derived based on the following assumption:

$$H(t, X) \subseteq \bar{H}(t, X) \rightarrow H(t+1, X) \subseteq \bar{H}(t+1, X). \quad (15)$$

We use h^v_t and $l^v_t$to indicate the Viterbi hypothesis in $H(t, X)$ and $L(t, X)$, respectively, and T_v to indicate a set of frames whose Viterbi hypotheses are label transition hypotheses. The following are the optimization algorithms for the decoding parameters we used in this paper.

Histogram pruning (θ_h):

$$\hat{\theta}_h = \max_t \boldsymbol{r}(v_t, H(t, X)) + 1. \quad (16)$$

Cost width pruning (θ_w):

$$\hat{\theta}_w = \max_t \boldsymbol{c}(v_t) - \min_{h \in H(t,X)} \boldsymbol{c}(h). \quad (17)$$

Label transition hypotheses pruning (θ_{t_w}):

$$\hat{\theta}_{t_w} = \max_{t \in T_v} \boldsymbol{c}(v_t) - \min_{h \in H(t,X)} \boldsymbol{c}(h). \quad (18)$$

Label selection pruning (θ_{l_s}, θ_{l_w}):

$$\hat{\theta}_{l_s} = \max_{t \in T_v} \boldsymbol{r}(l^v_{t+1}, L(t+1, X)) + 1, \quad (19)$$

$$\hat{\theta}_{l_w} = \max_{t \in T_v} \boldsymbol{c}(l^v_{t+1}) - \min_{l \in L(t+1,X)} \boldsymbol{c}(l). \quad (20)$$

V. Experiments

A. *Setup*

The effectiveness of the proposed algorithms were measured by performing OCR experiments for 5 languages: Arabic, English, Hindi, Japanese and Russian. These languages were selected to cover major scripts and various atom sizes.

We sampled around 2000 line images broadly from Google Books for each language (10,000 for English) and created manual transcriptions by 3 annotators for each line. Half of them were used as development data while the another half of them were used as evaluation data. In computing error rate, the transcription which is closest in edit distance to the one being scored is always taken as the reference. We report the normalized character error rate (N-CER) which does not distinguish characters that are visually the same, such as hyphen and dash, on the evaluation data. The development data was used to optimize feature weights and decoder options.

As described in Section II-C, the atom-based HMMs and the Unicode codepoint-based language modes were used as feature functions. The HMMs were trained based on the following procedure: 1) Render text obtained from Wikipedia (approximately 4M words) using Pango [11] with various fonts, sizes and resolutions and create synthesized text images. They are further artificially degraded with blur, rotation, binarization, contrast, etc. 2) Train atom-based left-to-right HMMs with GMMs on the degraded synthesized data. We allow one skip transition. The number of states for each atom is determined based on the statistics of the actual width of the atom. We use 45 two-dimensional DCT coefficients as features. 3) Perform the forced-alignment for the degraded synthesized data using the trained HMM and obtain the state-alignment for the data. 4) Train a feed-forward DNN using the data with the state-alignment information to form a hybrid system. We use DistBelief [12] to train the DNN. The input to the DNN is 360 two-dimensional DCT coefficients. The numbers of hidden nodes in the DNN are 1008, 752, 256 and 48, respectively, from bottom to top for all languages. 5) Optimize feature function weights using MERT on the development data. 6) Decode unsupervised data obtained from Google Books (up to 5M lines obtained from a different set of volumes than both the development and test sets) and create self-labeled data. The data is filtered based on its confidence probabilities which are computed using a regression tree. The threshold is set to 0.8. 7) Repeat steps 3-6 with the self-labeled data twice.

We trained 5-gram language models for English and Russian using text obtained from Wikipedia, 9-gram language models for Arabic and Hindi using text obtained from Google News, and trigram language model for Japanese using text obtained from Wikipedia. We used stupid-backoff [13] as a smoothing method.

Decoding time was measured on a Linux machine equipped with an Intel(R) Xeon(R) CPU E5-1650 and 32GB of memory. The measurements were conducted 5 times and the averaged value of them are reported.

B. Results: New pruning algorithms

Fig. 1 shows the trade-offs between N-CERs and the decoding speed per frame for each language with each pruning method. The graphs were created by decoding the evaluation data with decoding parameters optimized for various loss values and $\dot{\Theta} = (1000, \infty, \infty, \infty, \infty)$. The same loss value was used for all parameters to create one data point in the graphs. For example, we used the same loss value for θ_h and θ_w to create a data point of the graph for "Baseline". The graphs show that both pruning methods improved the decoding speed dramatically without loosing accuracy (40-60% by the label transition pruning and 70-95% by the label selection pruning). We obtained the best result when both methods were used. The relative improvements to the baseline were over 75% for English and Russian and 95% for Arabic, Hindi and Japanese in terms of speed without losing accuracy.

TABLE I. THE VALUES OF DECODING PARAMETERS FOR $\alpha = 0.04$.

Decoding parameter	Arabic	English	Hindi	Japanese	Russian
θ_h	424	126	268	788	123
θ_w	58.8	57.8	45.6	58.4	58.9
θ_{t_w}	35.0	32.2	18.5	18.6	23.8
θ_{l_s}	93	17	42	222	18
θ_{l_w}	9.5	7.2	9.4	8.7	6.1

TABLE II. N-CER [%] BY TESSERACT, ABBYY-V12 AND AKSARA.

Language	Tesseract	ABBYY-v12	Aksara
Arabic	20.4	15.3	5.20
English	1.30	1.02	0.89
Hindi	9.75	N/A	3.47
Japanese	13.40	8.42	6.51
Russian	2.62	1.66	0.88

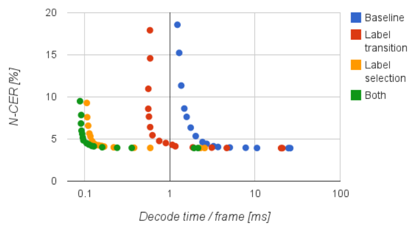
C. Results: Automatic decoding parameters optimization

We used the same set of loss values to create all curves in Fig. 1. This means that we did not need to know the range of each decoding parameter a priori. It solves one of the problems with the grid search. From the graphs, we found that the N-CERs increased after around $\alpha = 0.04$ for all configurations. Table I shows the actual values of the decoding parameters with $\alpha = 0.04$ for each language. It shows that the value of each decoding parameter was different for each language for the same loss value and the same trade-off point. Although the values of θ_w were close to each other in Table I, we found that the effective range of the value was significantly different when GMMs were used as the emission model instead of the hybrid approach. The proposed method can find appropriate values for loss values automatically regardless of the configuration. It verifies the effectiveness of the approach.

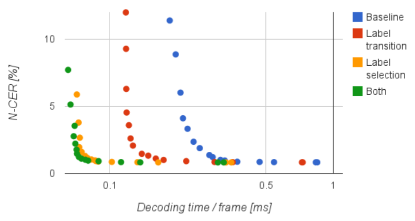
Table II compares Aksara optimized with $\alpha = 0.04$ and both pruning algorithms with Tesseract [14] and ABBYY-v12 [15] in terms of N-CER. The N-CERs were computed for the line images contained in the evaluation data but recognition was performed on the page images for a fair comparison with engines that use page-level information (e.g. adaption). We used a larger data set only for English (10871 lines from Google Books; from a different set of volumes than development set). The results showed that Aksara outperformed other systems on the data across the board.

VI. CONCLUSION

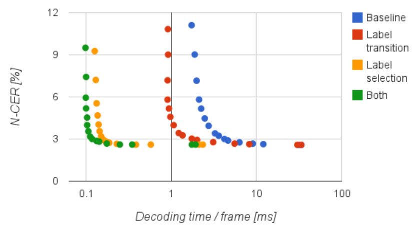
In this paper, we proposed two novel pruning algorithms which reduce the number of hypotheses created at label boundaries for time-synchronous Viterbi decoding. Experimental results showed that they were simple yet quite effective. We also proposed a novel algorithm to optimize decoding parameters based on a loss value without relying on a grid search. Experimental results showed the robustness and the effectiveness of the approach.



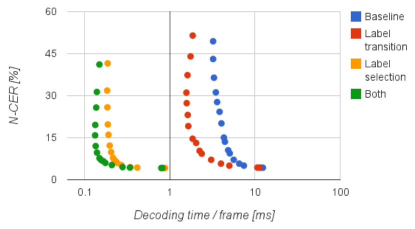
(a) Arabic (#Atom: 1208, #States: 4628)



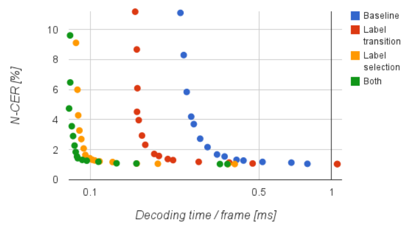
(b) English (#Atom: 346, #States: 2085)



(c) Hindi (#Atom: 2641, #States: 11527)



(d) Japanese (#Atom: 6675, #States: 45055)



(e) Russian (#Atom: 375, #States: 2159)

Fig. 1. N-CER vs. decoding time curves. X-axes use logarithmic scale. The baseline uses the histogram and cost width pruning. The label transition uses the label transition pruning in addition to the baseline. The label selection uses the label selection pruning in addition to the baseline. The both uses the label transition and selection pruning in addition to the baseline. The numbers in parentheses mean the number of atoms and states for each language. The width of the frame shift corresponds to one pixel of an image whose height is 30 pixels.

The effectiveness of the proposed pruning methods in other configurations such as a system with a word-based language model and domains such as speech recognition will need to be investigated in future work as well as more comparisons with other pruning methods. Extending the algorithm for Weighted Finite State Transducer (WFST)-based decoders will be another direction for future work, since input and output symbols are not necessarily synchronized in WFSTs and the proposed methods cannot be used directly. Currently, the loss function, which can be used in the parameter optimization algorithm, is limited to the one used in this paper. Extending the algorithm to use an arbitrary function such as CER will be future work. In addition, developing a method to optimize all decoding parameters at once will be another interesting problem.

Acknowledgment

The authors would like to thank Ray Smith for providing the layout engine.

References

[1] G. E. Kopec and P. A. Chou, “Document Image Decoding Using Markov Source Models,” *IEEE Trans. on Pattern Analysis and Machine Intelligence*, vol. 16, no. 6, pp. 602–617, Jun. 1994.

[2] I. Bazzi, R. Schwartz, and J. Makhoul, “An Omnifont Open-Vocabulary OCR System for English and Arabic,” *IEEE Trans. on Pattern Analysis and Machine Intelligence*, vol. 21, no. 6, pp. 495–504, Jun. 1999.

[3] D. Genzel, A. C. Popat, R. Teunen, and Y. Fujii, “HMM-based Script Identification for OCR,” in *Proceedings of the 4th International Workshop on Multilingual OCR*, ser. MOCR ’13. New York, NY, USA: ACM, 2013, pp. 2:1–2:5.

[4] R. Smith, “Hybrid page layout analysis via tab-stop detection.” in *Proceedings of the 10th International Conference on Document Analysis and Recognition*. IEEE, Jul. 2009, pp. 241–245.

[5] W. Macherey, F. J. Och, I. Thayer, and J. Uszkoreit, “Lattice-based Minimum Error Rate Training for Statistical Machine Translation,” in *Proceedings of the 2008 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2008, pp. 725–734.

[6] V. Steinbiss, B.-H. Tran, and H. Ney, “Improvements in beam search,” in *The 3rd International Conference on Spoken Language Processing, ICSLP 1994*, September 1994, pp. 18–22.

[7] H. Ney and S. Ortmanns, “Dynamic programming search for continuous speech recognition,” *Signal Processing Magazine*, vol. 16, no. 5, pp. 64–83.

[8] The Unicode Consortium. (2005) Unicode standard annex 29: Text boundaries. Technical report. [Online]. Available: http://unicode.org/reports/tr29/tr29-9.html

[9] H. A. Bourlard and N. Morgan, *Connectionist Speech Recognition: A Hybrid Approach*. Norwell, MA, USA: Kluwer Academic Publishers, 1993.

[10] S. Ortmanns, A. Eiden, H. Ney, and N. Coenen, “Look-ahead techniques for fast beam search,” in *Proceedings of the 32th International Conference on Acoustics, Speech, and Signal Processing*, April 1997, pp. 1783–1786.

[11] O. Taylor, “Pango, an open-source unicode text layout engine,” in *Proceedings of 25th Internationalization and Unicode Conference*, 2004.

[12] J. Dean, G. S. Corrado, R. Monga, K. Chen, M. Devin, Q. V. Le, M. Z. Mao, M. Ranzat, A. Senior, P. Tucker, K. Yang, and A. Y. Ng, “Large Scale Distributed Deep Networks,” in *Proceedings of the NIPS 2012: Neural Information Processing Systems*, Dec. 2012.

[13] T. Brants, A. C. Popat, P. Xu, F. J. Och, J. Dean, and G. Inc, “Large language models in machine translation,” in *In EMNLP*, 2007, pp. 858–867.

[14] R. Smith, “An overview of the Tesseract OCR Engine,” in *Proceedings of the 9th International Conference on Document Analysis and Recognition*. IEEE, Sep. 2007, pp. 629–633.

[15] ABBYY Production LLC. (2013) ABBYY® Finereader. [Online]. Available: http://www.abbyy.com/fr12guide_en.pdf