Fast and robust training of recurrent neural networks for offline handwriting recognition

Patrick Doetsch, Michal Kozielski and Hermann Ney
Lehrstuhl für Informatik 6 - Computer Science Department
RWTH Aachen University
Aachen, Germany
{doetsch,kozielski,ney}@i6.informatik.rwth-aachen.de

***Abstract*—In this paper we demonstrate a modified topology for long short-term memory recurrent neural networks that controls the shape of the squashing functions in gating units. We further propose an efficient training framework based on a mini-batch training on sequence level combined with a sequence chunking approach. The framework is evaluated on publicly available data sets containing English and French handwriting by utilizing a GPU based implementation. Speedups of more than 3x are achieved in training recurrent neural network models which outperform state of the art recognition results.**

***Keywords*—handwriting recognition; recurrent neural networks; GPU; batch-training**

I. Introduction

Deep neural networks (DNNs) became a key element of modern speech and handwritten text recognition systems. Recurrent neural networks (RNNs) share many properties with DNNs and were applied to small handwriting recognition tasks with great success [1]. Recurrent network topologies have the advantage that contextual information can be modeled by the network dynamics while DNNs require the contextual information to be encoded in the input space [2] [3]. However, the training time of RNNs is much larger compared to the mini-batch training of neural network architectures with pure forward connections due to the interdependence of the observations in the sequence. Therefore a mini-batch must be defined over sequences which in turn leads to a higher variability of the data within each batch.

When recognizing handwritten text, especially the long short-term memory recurrent neural network (LSTM-RNN) is used in many modern handwriting recognition systems. Although there is a theoretical justification for the effectiveness of LSTM-RNNs compared to other RNN structures, the optimal design of the components within the LSTM-RNN is not known and modifications can lead to promising results [4].

In this paper we propose a trainable modification of the activation functions inside the gating units of a LSTM-RNN. We further adopt the idea of mini-batching in feed-forward neural networks and apply it to RNNs within an efficient training scheme that keeps the variability within a batch small. In the end we provide a reliable and fast training framework for LSTM-RNNs and evaluate it within a hybrid hidden Markov model (HMM) on publicly available databases containing English and French handwriting.

The paper is organized as follows: In Section II we describe the data, the underlying HMM recognition system and the basic concepts of the LSTM-RNN. Our proposed mini-batching strategy is described in Section III. Afterwards in Section IV we describe and analyze the proposed LSTM-RNN modification. Finally we evaluate our framework in Section V and compare its performance to results reported by other groups.

II. Experimental Setup

We evaluate the proposed methods on two databases containing English and French handwriting. For both databases we train a baseline HMM on the frames that are extracted as described in the following. Then the HMM is applied in a forced alignment to the training data in order to obtain a frame-wise labeling. These labeled frames serve as input to the LSTM-RNN which is trained to estimate a posterior distribution over the state labels by minimizing the cross-entropy error of a softmax output layer. Finally the posterior estimates are used to simulate the emission probability for each state s in a hybrid HMM:

$$p(x|s) = \frac{p(s|x) \cdot p(x)}{p(s)^{\alpha}} \tag{1}$$

where α denotes a priori scaling factor that is optimized empirically. Note that we can drop the $p(x)$ term on the right hand side of Equation 1 when searching for the most likely word hypotheses during decoding.

Performance is measured in terms of the frame error rate (FER), the word error rate (WER) and the character error rate (CER). The WER and CER correspond to the minimal edit distance between the hypothesis generated by the hybrid HMM and the transcription on word and character level respectively.

A. Databases

The IAM database [5] consists of handwritten English sentences and is divided in 747 paragraphs for training, 116 paragraphs for development and 336 for testing. A combined word and character based language model is used that has been trained on the the LOB [6], Brown [7], and Wellington [8] text corpora. The vocabulary contains the 50k most frequent of the training set. The word based 3-gram language model leads to a perplexity of 420 and an OOV rate of 4% on the development set of the IAM database. A 10-gram character based language model with a character inventory of 77 characters is used to

recognize words which are likely to be not in the vocabulary [9].

The RIMES database [10] contains handwritten French sentences. There are 1500 sentences for training and 100 sentences for testing. The training sentences were written by 672 different writers. We use a 4-gram word-based LM with a perplexity of 26 and a lexicon with a character inventory of 96 base characters.

B. Preprocessing

Before feature extraction the images are cleaned using a contrast normalization and deslanted by a sharing angle that was selected by taking the median of three projection profile based algorithms [11][12][13]. Afterwards an overlapping sliding window of width 8 is shifted over the unscaled images resulting in image slices of different height. Then a cosine window is applied to each slice to smooth the borders. The slices are translated to their center of gravity and further normalized by their standard deviation in order to generate fixed-size frames of 8×32 pixels. Finally the 256 dimensional vector is reduced by PCA to 20 components and augmented by its original moments in horizontal and vertical direction resulting in a 24 dimensional feature vector which are referred to as frames. With this method we extracted $3,742,335$ frames for the training set of the IAM database and $6,991,867$ frames for the training set of the RIMES database. In average a character is covered by 17 consecutive frames.

C. HMM Modeling

Initially an HMM was trained with six states per character by optimizing the maximum likelihood training criterion. Each emission probability distribution was modeled by a Gaussian mixture model with 128 densities per state. Afterwards he HMM is applied to the training set in a forced alignment mode in order to generate character length statistics. Based on these statistics a character dependent variable state HMM topology was estimated and used in further experiments. In total we obtain 563 state labels on the IAM database and 702 state labels on the RIMES database which also build the targets for LSTM-RNN training.

Training of the HMM is performed with the Viterbi algorithm using the Bakis topology. In order to assure that at least one observation is assigned to each Gaussian, two consecutive states within the HMM share the same emission probability distribution. During decoding we are looking for the word sequence $w_1^N := w_1, \ldots, w_N$ for which the trained model generates the highest posterior probability $p(w_1^N|x_1^T)$ when observing the input sequence $x_1^T := x_1, \ldots, x_T$:

$$x_1^T \rightarrow \hat{w}_1^N(x_1^T) = \arg\max_{w_1^N} \left\{ p^\kappa(w_1^N) p(x_1^T|w_1^N) \right\} \qquad (2)$$

where κ corresponds to a scaling exponent of the language model. We do not train the transition probabilities of the HMM. Instead, transition probabilities are modeled by fixed time distortion penalties which are optimized empirically.

In [14] we describe all preprocessing steps and components of the HMM recognition system in full detail.

D. RNN training

The parameters of the RNN are optimized in a gradient descend manner using the Backpropagation Through Time (BPTT) algorithm [15]. Training is performed by unfolding the recurrent connections of the RNN over time and to propagate the gradients back through the unfolded network. This procedure transforms the recurrent connections of the network into a forward structure which has the depth of the sequence length. Gradients propagated over so many steps vanish quickly, making learning of long-term dependencies almost impossible. It has been shown that only a linear recurrent unit with a fixed weight of 1 is able to avoid the problem of vanishing gradients [16]. On the other hand, only the non-linear activation functions allow the network to learn complex input-output relations. The long-short-term memory RNN (LSTM-RNN) [17] is designed to combine these insights by replacing the regular units of the neural network by so called memory cells. Each memory cell contains a central linear recurrent unit with an untrainable weight of 1 while a non-recurrent unit with a non-linear activation functions processes the incoming information of the central unit. In order to retain the gradient information, all incoming and outgoing connections to the central unit are scaled by gating units. These gating units receive their inputs from outside the cell in order to compute a squashing function that affects the information flow within the memory cell in a multiplicative way. In particular there are an input gate, a forget gate and an output gate protecting the incoming, outgoing and recurrent information flows respectively. The general structure of a memory cell is depicted in Figure 1. A comprehensive analysis of the gradient flow in RNNs can be found in [18].

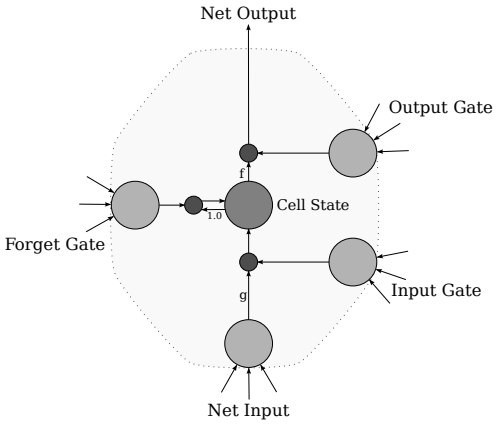

Fig. 1. LSTM-RNN memory cell with an input, output and forget gate. The gating units and the net input unit receive inputs from outside the cell. The output of these units is scaled by multiplicative units shown as small dark circles here [19].

In this work bidirectional LSTM-RNN training with BPTT is performed with GPU based implementation using the python library Theano [20]. The weights of the network between layer i and layer j of size n_i and n_j respectively are initialized according to the result of the analysis done by [21]:

$$W \sim U\left[-\frac{\sqrt{6}}{\sqrt{n_j + n_{j+1}}}, \frac{\sqrt{6}}{\sqrt{n_j + n_{j+1}}} \right]$$

The initialization forces the variance of the gradients to be

the same in all layers and is designed for hyperbolic tangent activation functions in both connecting layers.

During LSTM-RNN training we measure the performance of the network on a separate validation set consisting of 10% of the training data. On the IAM database 616 sequences (373,176 frames) are used for validation while on the RIMES database we use 1126 sequences (630,670 frames) for validation.

III. Accelerating BPTT training

For a fully connected RNN with n units BPTT has an efficient time complexity of $O(n^2)$ [15]. However, as long as the length of the observation sequence is unknown the required memory is also unbounded with a complexity of $O(Tn + n^2)$ for a sequence of length T. On today's GPU architectures system memory is very limited and therefore training of long sequences quickly becomes infeasible for large networks. Taking full advantage of the GPU further requires to parallelize the training procedure massively. However, propagating gradients back through time is an inductive task and therefore several frames within a sequence can not be trained in parallel. In order to train large and deep RNNs on a GPU we use two approaches tackling these issues:

1) Partitioning sequences in (overlapping) sub sequences
2) Forwarding multiple subsequences in parallel

Mini-batching is a common technique in neural networks with a nice theoretical justification, while on the other hand it increases the number of computations that can be performed in parallel. Similar to the mini-batching training on frame level of MLPs, a batching on sequence level can improve RNN training in a massively parallelized environment. In sequence based mini-batching all frames get forwarded in parallel which appear at the same time step. Therefore all sequences in a batch have to be of the same length and shorter sequences must be padded with zeros if required. In order to reduce the intra-batch variability of the sequence lengths and to control the memory requirement, we chunk the sequence in (possibly overlapping) subsequences of a constant length. Afterwards consecutive chunks are stacked into a 3D tensor and fed as input to the network which iterates over the third dimension of the tensor in the BPTT procedure. This approach is illustrated in Figure 2.

Through the partitioning and recombination of the partitions into batches, only a small set of sequences is mapped to each batch and therefore the variations regarding the writing style and background noise is small within a batch. In Table I we analyze the effect of the proposed batching method for different configurations on the IAM database. Experiments were performed with a fixed number of 40k frames per batch, i.e. the volume of the 3D tensor is kept constant. The LSTM-RNN consisted of one hidden layer with 500 memory cells.

The results suggest that a recombination of sequence chunks with less than 50 time steps leads to a degradation in system performance regarding the FER, CER and WER. With our feature extraction 50 time steps roughly correspond to three characters. Due to the powerful HMM decoding scheme the large difference in the FER does not transfer to the WER and CER of the final recognition system. Further it can be seen that the approach allows to recude the training time significantly

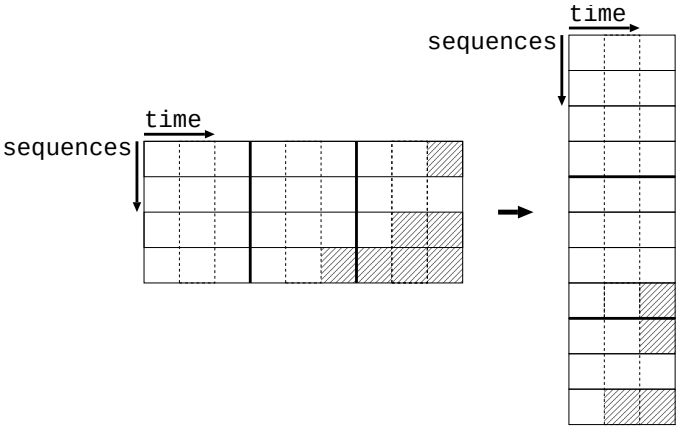

Fig. 2. Illustration of the chunking and batching approach. Each row in the graphics corresponds to one sequence and each column corresponds to a batch of the current time step. Each cell either corresponds to a feature vector or zero if it is textured. The sequences are partitioned into chunks of a particular length indicated by the bold borders and recombined into a large batch as shown on the right side of the graphics.

TABLE I. Recognition and runtime performance of different sequence chunk sizes on the IAM database using a batch size of 40k frames.

chunk size	chunks per batch	FER[%]	WER[%]	CER[%]	elapsed
10	4000	22.7	10.7	3.7	3.4 min
50	800	19.8	10.5	3.6	4.7 min
100	400	19.5	10.4	3.6	6.7 min
500	80	19.3	10.5	3.6	15 min
∞	≈ 65	19.2	10.4	3.5	16.2 min

while retaining good recognition results. We therefore chose a chunk size of 100 for all experiments described in the following sections.

IV. Scaling Gating Units

LSTM memory cells are usually configured to use logistic sigmoid activation functions for gating units and hyperbolic tangent (tanh) activation functions for the net input and net output. These activation functions are closely related since the logistic sigmoid is just a shifted version of the tanh activation function rescaled in output and input space:

$$sig(z) = \frac{1}{2}\left(1 + \tanh\frac{z}{2}\right) \tag{3}$$

Sigmoidal activation function share the characteristic to be composed of a symmetric non-linear part shortly before saturation to a limiting value, and an almost linear part in between of those saturation phases. Non-linearity is a key concept of activation functions and is required to learn complex patterns. On the other hand, the subsequent linear part produces large and stable gradients and therefore is important for convergence of the training procedure. In this work we propose a modification for gating units of an LSTM-RNN which enables the network to control the shape of the squashing function by introducing layer-specific shared weights $\lambda_\iota^{(l)}$, $\lambda_\phi^{(l)}$, and $\lambda_\omega^{(l)}$ for the input, forget and output gate of layer l respectively.

$$sig_{\chi}^{(l)}(z_{\chi_j}^{(l)}) = sig(\lambda_{\chi}^{(l)} z_{\chi_j}^{(l)}), \chi \in \{\iota, \phi, \omega\} \tag{4}$$

where $z_{\chi_j}^{(l)}$ denotes the net input of gate χ in memory cell j of layer l calculated over the output activations y_i that are provided through the forward and current connections:

$$z_{\chi_j}^{(l)}(t) = \sum_i \alpha_{\chi_j i}^{(l-1)} y_i^{(l-1)}(t) + \sum_i \alpha_{\chi_j i}^{(l)} y_i^{(l)}(t-1) \tag{5}$$

Taking the derivative of the local error E_n at training sequence n w.r.t. the gate weights $\lambda_{\chi}^{(l)}$ in layer l containing C memory cells results in the following update rules:

$$\frac{\partial E_n}{\partial \lambda_{\chi}^{(l)}} = \sum_{t=1}^{T} \sum_{j}^{C} \frac{\partial E_n}{\partial \lambda_{\chi}^{(l)} z_{\chi_j}^{(l)}(t)} \frac{\partial \lambda_{\chi}^{(l)} z_{\chi_j}^{(l)}(t)}{\partial \lambda_{\chi}^{(l)}} \tag{6}$$

$$= \sum_{t=1}^{T} \sum_{j}^{C} \delta_{\chi_j}^{(l)}(t) z_{\chi_j}^{(l)}(t) \tag{7}$$

The δ terms in (7) are calculated recursively starting at $t = T$ with $\delta_i(T+1) = 0 \ \forall i$. See e.g. [1] for a detailed description how to calculate these terms. All gate weights are initialized with a value of 1, so the network starts training with regular sigmoid activation functions.

From (6) it follows that the complete input information flowing into the layer is incorporated in the gate weight updates. All memory cells within a layer are therefore enforced to show a similar behavior regarding their gating activations. In Figure 3 we examine this specialization experimentally. For this purpose we trained an LSTM-RNN with three hidden layers and 500 memory cells in each layer on the IAM data set, and plot the magnitude of the gate weights over 50 epochs of training.

First, it is notable that the gate weights of the forget gates (green lines) converge to a similar value of 2 in all layers. Recalling the relation of the sigmoid and the tanh activation function mentioned in (3), the forget gate activation functions therefore converge to a shifted tanh that is rescaled in output space. The input and output gate weights (red and blue lines) show a coupled behavior that differs over the three layers. Considering the two hidden layers closest to the input layer, we can observe a high raise of the weight magnitudes until epoch 5 followed by a slope until epoch 20 where the gate weights in the layer closest to the input layer converge while the gate weights in the second layer start raising again. Interestingly, epoch 20 also produces the minimal frame error rate on the cross validation set used during LSTM-RNN training. The input and output gate weights in the layer closest to the output layer are raising almost monotonically, resulting in sharp activation functions that saturate very quickly. The shape of the activation functions learned in this layer is shown in Figure 4.

With the large gate weights learned from the network, the gating units tend to switch much more quickly between the saturation parts than in unscaled sigmoid activation functions.

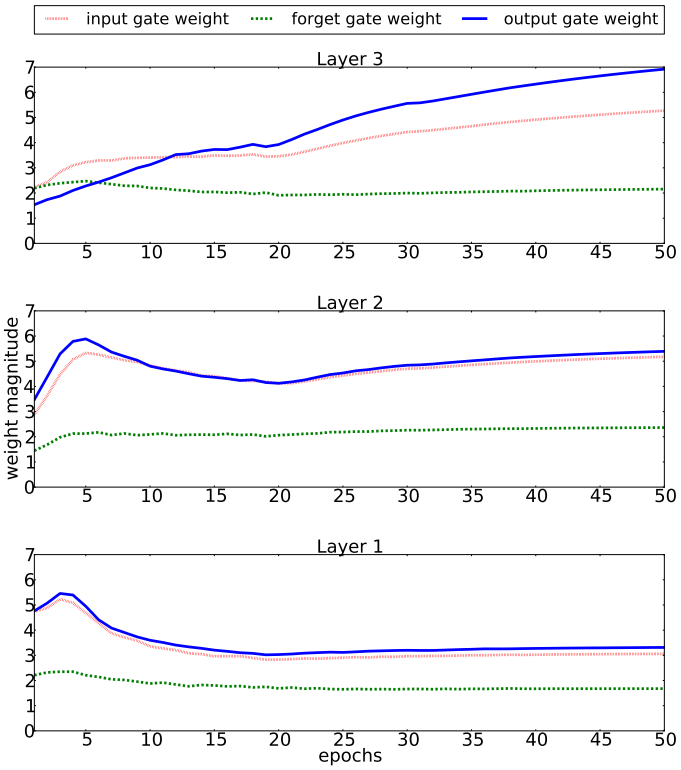

Fig. 3. Gate weights learned by an LSTM with three hidden layers. The solid lines corresponds to the last hidden layer (closest to the output layer), while the dashed and dotted lines correspond to the middle and first hidden layer respectively.

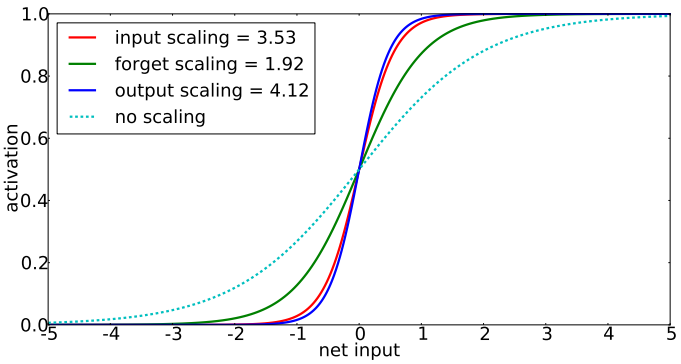

Fig. 4. Logistic sigmoid functions with different input scalings as trained by the RNN for the input, forget and output gate of the LSTM memory cells.

Figure 5 compares the output activations of the gates in a network with and without gate weights.

If we consider an activation to be saturated when its value deviates not more than 0.1 of activation function's bound, then we can see that more than 50% of the gate activations are in saturation when including gate weights in training. This is more than twice the number of saturated gate activations compared to an LSTM-RNN without gate weights. Especially small values of gate outputs are important because they allow the network to make a selective use of its input by introducing sparsity.

Comparing these results with Figure (3) we see that many forget gates are in saturation although the gate weights con-

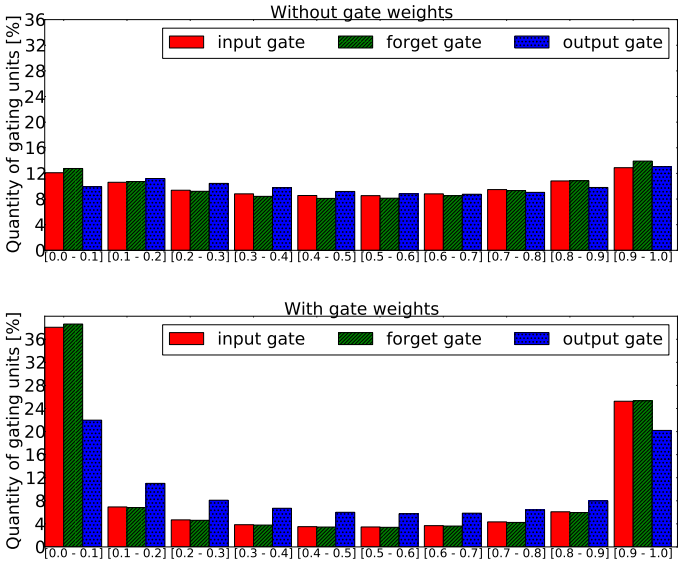

Fig. 5. Saturation analysis of sigmoid activation functions in gating units. The bars show the fraction of gate activation values that lie in the corresponding quantile. The lower and upper graphs show this distribution for an LSTM with and without gate weights respectively.

TABLE II. RECOGNITION PERFORMANCE ON THE VALIDATION SET OF THE IAM DATABASE.

Systems	FER[%]	WER[%]	CER[%]
Gaussian HMM	–	10.7	3.8
LSTM-RNN / HMM			
No gate weights	17.8	8.9	2.8
With gate weights	16.9	8.4	2.5

verge to a value that is much smaller than the gate weights of the input and output gate. This comes from the fact that the central unit of the memory cell is linear and therefore unbounded while the input and output gate receive their inputs from units with activation functions that are bounded in output space. Furthermore the amount of saturated output gates is smaller than the amount of saturated input and forget gates, although the gate weights of the output gates converge to the largest value. An explanation is that through the modified forget gate behavior the average state values of the memory cells are smaller which in turn leads to a larger sensitivity of the output gates on inputs that are close to zero.

V. EXPERIMENTS

Experiments were conducted on English and French handwriting using the databases introduced in Section II-A. Table II compares two LSTM-RNNs trained with and without gate weights and decoded in a hybrid HMM on the IAM database. Table III shows the results of the same experiment on the RIMES database. In both experiments we used an LSTM-RNN with three hidden layers having 500 memory cells in each layer resulting in approximately 10.7 million parameters. Note that the number of parameters only increases by 9 when using gate weights and therefore their influence on the runtime is negligible.

On both data sets we can observe an improvement in terms of FER, WER and CER when using layer specific shared

TABLE III. RECOGNITION PERFORMANCE ON THE VALIDATION SET OF THE RIMES DATABASE.

Systems	FER[%]	WER[%]	CER[%]
Gaussian HMM	–	15.7	5.5
LSTM-RNN / HMM			
No gate weights	18.0	13.3	4.6
With gate weights	17.4	12.9	4.3

TABLE IV. COMPARISON OF THE PROPOSED SYSTEM TO RESULTS REPORTED BY OTHER GROUPS ON THE IAM DATABASE.

Systems	Voc.	WER [%]		CER [%]	
		Dev.	Eval	Dev.	Eval
RWTH	50k	8.4	12.2	2.5	4.7
Kozielski et al. [14]	50k	9.5	13.3	2.7	5.1
Boquera et al. [22]	50k	19.0	22.4	-	9.8
Dreuw et al. [23]	50k	22.7	32.9	7.7	12.4
Bertolami et al. [24]	20k	26.8	32.8	–	–

gate weights. In particular we obtain a relative improvement of 5.6% on the IAM database and a relative improvement of 3.1% on the RIMES database when considering the final WER. It is worth mentioning that both baseline systems were carefully optimized in [14] and therefore even small absolute improvements are relevant. Furthermore, these improvements were achieved without increasing the system complexity significantly in terms of parameters.

In Table IV and V we compare the proposed system with results other groups reported on the IAM and the RIMES database respectively. Absolute improvements of about 1% in terms of WER compared to the best published systems prove the effectiveness of our method. Its is also notable that the systems in [14] are discriminatively trained LSTM-RNN tandem HMM systems that are more complex and require considerably more training effort compared to the hybrid HMM approach we used in this work.

VI. CONCLUSIONS

In this paper we described an efficient training framework for recurrent neural networks that operates on sequence chunks that were recombined into batches. We showed that we can achieve competitive performance on the IAM database when using the proposed training framework while at the same time we could reduce the runtime by a factor of more than 3.

Furthermore we described a modification for gating units in LSTM-RNNs that involve layer-specific gate weights that are applied to all gates of a particular type. We showed that the frame-wise recognition performance as well as the performance of the final HMM based recognition system can be improved using the proposed gate scaling method.

TABLE V. COMPARISON OF THE PROPOSED SYSTEM TO RESULTS REPORTED BY OTHER GROUPS ON THE RIMES DATABASE.

Systems	WER [%]	CER [%]
RWTH	12.9	4.3
Kozielski et al. [14]	13.7	4.6
A2IA [25]	15.2	7.2
Telecom ParisTech [10]	31.2	-

In future work the gate weights will be examined in greater detail. It has yet to be determined if a linear scaling is optimal and if a special treatment for the different gating units may be required.

REFERENCES

[1] A. Graves, "Supervised sequence labelling with recurrent neural networks," Dissertation, Technische Universität München, München, 2008.

[2] J. L. Elman, "Finding structure in time," *Cognitive science*, vol. 14, no. 2, pp. 179–211, 1990.

[3] H. Sak, A. Senior, and F. Beaufays, "Long short-term memory based recurrent neural network architectures for large vocabulary speech recognition," 2014.

[4] D. Monner and J. A. Reggia, "A generalized lstm-like training algorithm for second-order recurrent neural networks," *Neural Networks*, vol. 25, pp. 70–83, 2012.

[5] U.-V. Marti and H. Bunke, "The IAM-database: an english sentence database for offline handwriting recognition," vol. 5, no. 1, pp. 39–46, 2002.

[6] S. Johansson, E. Atwell, R. Garside, and G. Leech, *The Tagged LOB Corpus: Users Manual*, Norwegian Computing Centre for the Humanities, 1986.

[7] W. Francis and H. Kucera, "Brown corpus manual, manual of information to accompany a standard corpus of present-day edited american english," Tech. Rep., 1979.

[8] L. Bauer, "Manual of information to accompany the wellington corpus of written new zealand english," Tech. Rep., 1993.

[9] M. Kozielski, D. Rybach, S. Hahn, R. Schlüter, and H. Ney, "Open vocabulary handwriting recognition using combined word-level and character-level language models," *IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, pp. 8257–8261, May 2013.

[10] E. Grosicki and H. El Abed, "ICDAR 2009 handwriting recognition competition," in *Document Analysis and Recognition, 2009. ICDAR '09. 10th International Conference on*, july 2009, pp. 1398 –1402.

[11] M. Pastor, A. Toselli, and E. Vidal, "Projection profile based algorithm for slant removal," in *Image Analysis and Recognition*, ser. Lecture Notes in Computer Science, A. Campilho and M. Kamel, Eds. Springer Berlin / Heidelberg, 2004, vol. 3212, pp. 183–190.

[12] A. Vinciarelli and J. Luettin, "A new normalization technique for cursive handwritten words," *Pattern Recognition Letters*, vol. 22, no. 9, pp. 1043–1050, 2001.

[13] M. P. i Gadea, A. H. Toselli, V. Romero, and E. Vidal, "Improving handwritten off-line text slant correction," in *Procc. of The Sixth IASTED international Conference on Visualization, Imaging, and Image Processing (VIIP 06)*, 2006.

[14] M. Kozielski, P. Doetsch, and H. Ney, "Improvements in rwth's system for off-line handwriting recognition," *International Conference on Document Analysis and Recognition*, 2013.

[15] R. J. Williams and D. Zipser, "Gradient-based learning algorithms for recurrent networks and their computational complexity," 1995, ch. 13, pp. 433–486.

[16] S. Hochreiter, Y. Bengio, P. Frasconi, and J. Schmidhuber, "Gradient flow in recurrent nets: the difficulty of learning long-term dependencies," in *A Field Guide to Dynamical Recurrent Neural Networks*, S. C. Kremer and J. F. Kolen, Eds. IEEE Press, 2001.

[17] M. Liwicki, A. Graves, H. Bunke, and J. Schmidhuber, "A novel approach to on-line handwriting recognition based on bidirectional long short-term memory networks," in *Proceedings of the 9th International Conference on Document Analysis and Recognition (ICDAR)*, 2007.

[18] R. Pascanu, T. Mikolov, and Y. Bengio, "On the difficulty of training recurrent neural networks," in *ICML (3)*, 2013, pp. 1310–1318.

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

[20] J. Bergstra, O. Breuleux, F. Bastien, P. Lamblin, R. Pascanu, G. Desjardins, J. Turian, D. Warde-Farley, and Y. Bengio, "Theano: a CPU and GPU math expression compiler," in *Proceedings of the Python for Scientific Computing Conference (SciPy)*, 2010.

[21] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *International Conference on Artificial Intelligence and Statistics*, 2010, pp. 249–256.

[22] S. España Boquera, M. Castro-Bleda, J. Gorbe-Moya, and F. Zamora-Martinez, "Improving offline handwritten text recognition with hybrid HMM/ANN models," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 33, no. 4, pp. 767 –779, April 2011.

[23] P. Dreuw, P. Doetsch, C. Plahl, and H. Ney, "Hierarchical hybrid MLP/HMM or rather MLP features for a discriminatively trained gaussian HMM: a comparison for offline handwriting recognition," in *IEEE International Conference on Image Processing*, Brussels, Belgium, 2011.

[24] R. Bertolami and H. Bunke, "Hidden markov model-based ensemble methods for offline handwritten text line recognition."

[25] F. Menasri, J. Louradour, A.-L. Bianne-Bernard, and C. Kermorvant, "The A2iA French handwriting recognition system at the Rimes-ICDAR2011 competition." in *DRR*, ser. SPIE Proceedings, C. Viard-Gaudin and R. Zanibbi, Eds., vol. 8297. SPIE, 2012.