Handwriting Recognition with Large Multidimensional Long Short-Term Memory Recurrent Neural Networks

Paul Voigtlaender, Patrick Doetsch, Hermann Ney
Human Language Technology and Pattern Recognition, Computer Science Department
RWTH Aachen University
52056 Aachen, Germany
Email: {voigtlaender, doetsch, ney}@cs.rwth-aachen.de

***Abstract*—Multidimensional long short-term memory recurrent neural networks achieve impressive results for handwriting recognition. However, with current CPU-based implementations, their training is very expensive and thus their capacity has so far been limited. We release an efficient GPU-based implementation which greatly reduces training times by processing the input in a diagonal-wise fashion. We use this implementation to explore deeper and wider architectures than previously used for handwriting recognition and show that especially the depth plays an important role. We outperform state of the art results on two databases with a deep multidimensional network.**

***Keywords*-MDLSTM; LSTM; Long Short-Term Memory; Recurrent Neural Network; Handwriting Recognition;**

I. Introduction

Neural networks have become a key component in modern handwriting and speech recognition systems. While feedforward neural networks only use a limited and fixed amount of context of the input, recurrent neural networks (RNNs) can in principle make use of an arbitrary amount of context by storing information in their internal state. In particular, long short-term memory recurrent neural networks (LSTM-RNNs) have been very successful [1], [2], [3]. The LSTM architecture allows the network to store information for longer amounts of time and avoids vanishing and exploding gradients [4]. While normal LSTM-RNNs only use a recurrence over one dimension (the x-axis of an image or the time-axis for audio), multidimensional long short-term memory recurrent neural networks (MDLSTM-RNNs) [5] use a recurrence over both axes of an input image, allowing them to model the writing variations on both axes and to directly work on raw input images.

Recently, handwriting recognition competitions were won by MDLSTM-RNNs (e.g. [6], [7]) and very recently MDLSTM networks have also been shown to yield promising results for speech recognition [8]. However, the MDLSTM networks used for handwriting recognition in prior work, e.g. Pham et al. [9] who also use the same databases as we do, seem to be relatively small. One reason for this might be that usually CPU implementations are used for training which lead to high runtimes, e.g. Strauß et al. report, that the training of a single network usually lasts several weeks [6]. To the best of our knowledge, so far there is no publicly available GPU implementation of MDLSTM. In this work, we fill this gap and create an efficient GPU-based implementation which is described in Section IV and made publicly available.

We show that for deeper networks, a simple weight initialization scheme with fixed standard deviation results in convergence issues which can be solved by using the initialization scheme by Glorot et al. [10]. Furthermore, we use our implementation to train much larger and deeper networks as typically used for handwriting recognition and show that the results can thereby be substantially improved.

II. Multidimensional Long Short-Term Memory for Handwriting Recognition

A multidimensional recurrent neural network (MDRNN) is a generalization of a recurrent neural network, which can deal with higher-dimensional data such as videos (3D) or images (2D). Here we restrict ourselves to the two dimensional case which is commonly used for handwriting recognition tasks. A 2D-RNN scans the input image along both axes and produces a transformed output image of the same size. The hidden state $h(u, v)$ for position (u, v) of an MDRNN layer is computed based on the previous hidden states $h(u-1, v)$ and $h(u, v-1)$ of both axes and the current input $x(u, v)$ by

$$h(u,v) = \sigma(Wx(u,v) + Uh(u-1,v) + Vh(u,v-1) + b),$$

where W, U and V are weight matrices, b a bias vector and σ a nonlinear activation function. Like in the 1D case, MDLSTM introduces an internal cell state for each spatial position which is protected by several gates. The use of LSTM allows the network to exploit more context and leads to more stable training. It is common practice to use four parallel MDLSTM layers which each process the input in one of the four possible directions, e.g. from the top left to the bottom right. The four directions are later combined so that at every spatial position the full context from all directions is available.

The basic neural network architecture used in this work is depicted in Fig. 1. Similar to prior work [9], [6], [5], we stack multiple layers of alternating convolution and multidirectional MDLSTM and after the last MDLSTM layer, the two-dimensional sequence is collapsed to a one-dimensional sequence by summing over the height axis. After the collapsing, a softmax layer with a Connectionist Temporal Classification (CTC) [11] loss is used to handle the alignment between the input and the output sequence.

In contrast to Pham et al. [9], we use max pooling instead of strided convolutions, because we can afford the higher computational complexity due to the GPU implementation and max pooling is a standard component of many well-performing computer vision systems (e.g. [12], [13]). Additionally, we apply the first convolution directly to the input image and don't divide it into 2x2 blocks. Another difference is that we average the four directions and then feed them to a single convolutional layer instead of having one separate convolutional layer for each direction. Additionally, we found that the training can be very unstable with standard MDLSTM cells, as the internal state can quickly increase over time when two forget gates (one for each direction) are used [14]. Hence, we replaced them by stable LSTM cells, which were introduced by Leifert et al. [14] and solve this problem, for all experiments in this paper.

To keep the number of hyperparameters manageable, we fix all filter sizes to 3x3 and max pooling is always applied in non-overlapping blocks of 2x2. We use dropout of 0.25 for the forward connections of all convolutional layers, MDLSTM layers and the output layer, except for the first convolutional layer, where we don't use dropout as we don't want to drop the single color channel of the input image. Additionally, we only consider layer sizes which increase linearly with the layer index, i.e. layers get wider when they are closer to the output layer. For example, a network width of $15n$, where n is the layer index, for the network of Fig. 1 means that the lowest convolutional layer has 15 feature maps, the first MDLSTM layer has 30 hidden units per direction, the second convolutional layer has 45 feature maps and so on. We call this model with a width of $15n$ the basic model and use it for several experiments later on.

III. Databases

We used two databases for experiments. The IAM database [15] consists of 6,482, 976 and 2,915 lines of English handwriting for the training, development and evaluation sets. The line images of the training set have an average width of 1,781 pixels with 152 pixels standard deviation and an average height of 124 pixels with 34 pixels standard deviation. We used a smoothed word-based trigram language model with a perplexity of 420 on the training set and an out of vocabulary rate of 4% on the development set. The language model was combined with a 10-gram character language model to deal with out of vocabulary words [16].

The RIMES database [17] consists of 11,279 lines of French handwriting for training and 778 lines for evaluation. The line images of the training set have an average width of 1,665 pixels with 553 pixels standard deviation and a height of 160 pixels with 36 pixels standard deviation. We used a 4-gram word-based language model with a perplexity of 23.

IV. Implementation

As a direct implementation of MDLSTM in Theano using nested calls of scan, i.e. Theano's mechanism for loops, showed poor performance, we decided to implement it directly using CUDA and cuBLAS. Our GPU-based implementation is made freely available for academic research purposes as part of RETURNN, the RNN training software developed by our institute, and has already been briefly described in the corresponding publication [18]. We achieve good performance by pulling the non-recurrent parts of the computations out of the loops for the recurrences, using custom CUDA kernels for the MDLSTM gating mechanism and reusing memory wherever possible.

In previous (CPU-based) implementations [19], the image is processed row-wise in an outer loop and column-wise in an inner loop to make sure that for every pixel both predecessor pixels are processed before reaching it. However, we noticed that all pixels on a common diagonal of an image can be processed in parallel without violating this constraint (see Fig. 2). Hence, we process images diagonal-wise and also process multiple images and the four directions of multidirectional MDLSTMs simultaneously to exploit the massive parallelism offered by modern GPUs. Note that very recently and independently of this work, a similar parallelization scheme based on the idea to process images diagonal-wise has been proposed by van den Oord et al. [20]. In their implementation, the input map of an MDLSTM layer is explicitly skewed to transform the diagonals into columns which facilitates the computations. However, in our implementation this transformation is not needed, as we use batched cuBLAS operations which take an array of pointers, which point to the pixels on the diagonal, as arguments. Additionally, we make our implementation publicly available. In Stollenga et al. [21], parallelization is achieved by using a pyramidal connection topology which is easier to implement efficiently, but changes the available spatial context.

In Table I, we compare the runtime of the basic model for different mini-batch sizes with a GTX Titan X GPU on IAM. One can see, that with moderate batch sizes the runtime per epoch can be improved significantly, but at some point, the runtime almost saturates to roughly 15 minutes per epoch, possibly because the GPU is fully utilized. For comparison, we trained a similar network using RNNLIB [19] with an Intel Core i7-870 processor. RNNLIB supports only single-threaded CPU training and takes roughly three days for one epoch on the IAM dataset. Hence, without an efficient GPU

Figure 1: The basic network architecture used in this paper. The input image on the left is processed pixel-by-pixel using a cascade of convolutional, max-pooling and MDLSTM layers, and finally transcribed by a CTC layer on the right. Figure adapted from Pham et al. [9].

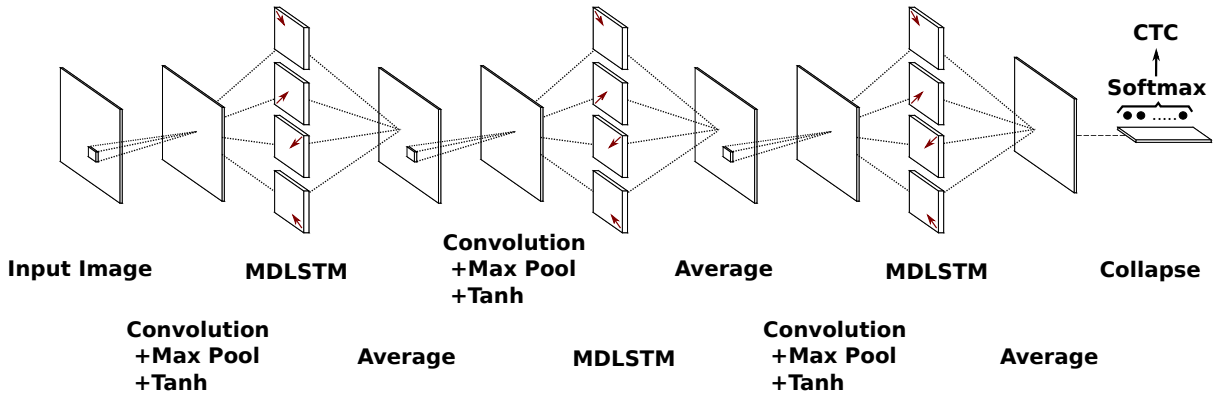
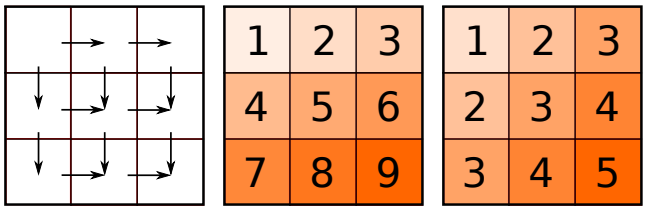


Figure 2: MDLSTM dependencies and order of computation. (a) The incoming arrows to a pixel have their origins at pixels which are needed for the computation of the current pixel. (b) Naive order of computation: the numbers indicate the order of computation which is from top to bottom, from left to right, one pixel at a time. (c) Diagonal order of computation: all pixels on a common diagonal are computed at the same time.



(a) (b) (c)

Table I: Speed and memory consumption for different batch sizes (i.e. an upper limit on the number of pixels in a mini-batch) using the basic model. The runtime column gives the duration for one epoch on the IAM corpus.

Batch size	Imgs/batch	Runtime[min]	Pixels/sec	Memory[GB]
1 image	1.00	54.3	0.38M	1.06
0.5M	1.53	41.5	0.49M	1.06
1.0M	3.26	24.8	0.82M	1.66
2.5M	7.81	16.5	1.24M	3.93
3.5M	10.64	15.0	1.36M	5.64
5.0M	14.88	14.5	1.41M	9.11

based implementation, the experiments in this paper would not have been possible in a reasonable amount of time.

V. Training and Weight Initialization

For optimization, we use Adam [22] with incorporated Nesterov momentum [23]. We start with a learning rate of 0.0005 and decrease it to 0.0003 in epoch 25 and to 0.0001 in epoch 35. Note that for a minibatch of multiple images, we sum over the CTC losses for every image and do not normalize by the batch size. We do not use any truncation or clipping of gradients. All following experiments are done on GTX 980 GPUs. In order to stay within the 4GB GPU memory limit for large networks, we restrict the batch size to 600k pixels for all networks, to keep the results comparable. During the training, we measure the CTC objective function value and the label error rate, i.e. the lowest character error rate of the network itself without lexicon or language model, on a holdout set of 10% of the training data. Training stops, when both measures do not improve for 20 epochs. For most networks, less than 80 epochs were necessary.

Unless stated otherwise, the images are used as input without any resizing, only a padding of 15 white pixels are added on all four sides of the images. The gray values have been linearly rescaled to values between 0 and 1. Images for which the output sequence have a shorter length than the number of characters in the reference are a problem during training, as they cannot be transcribed correctly by the CTC output layer. Hence, we decided to remove the 10 affected training images from RIMES, while on IAM this problem didn't occur.

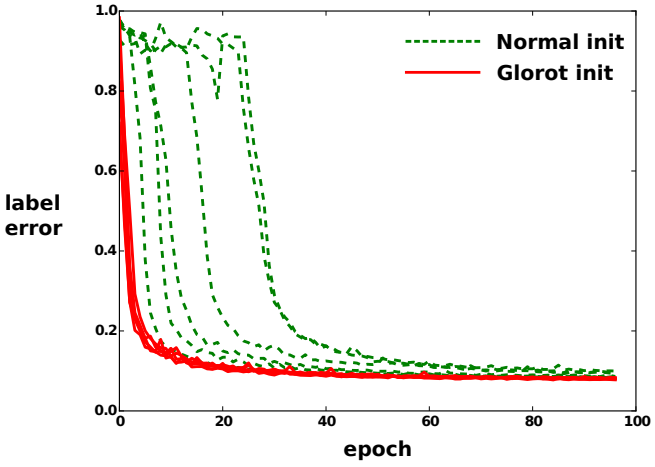
In Pham et al. [9], all network weights are initialized by a normal distribution with zero mean and standard deviation 0.01 (called normal initialization in the following). Especially when trying to train deeper networks, we often experienced slow convergence with this initialization scheme. Hence, we tried the popular initialization scheme proposed by Glorot et al. [10]. In this scheme, the weights are initialized with a uniform distribution given by

$$W \sim U\left(-\frac{\sqrt{6}}{\sqrt{n_{in}+n_{out}}}, \frac{\sqrt{6}}{\sqrt{n_{in}+n_{out}}}\right),$$

where n_{in} and n_{out} are the number of inputs of outputs of the layer. We performed several runs of training with a relatively large network with 9 hidden layers and both weight

initialization schemes to study the effect of the weight initialization for deep networks (see Fig. 3). The difference between multiple curves for the same initialization is only the random seed used for initialization. In the case of the normal initialization, the seed has a strong influence on the training progress, and convergence is often very slow. On the contrary, when using the initialization by Glorot et al. [10], the different runs show little variance and converge much faster. We also experimented with an orthogonal initialization based on Saxe et al. [24], which worked substantially better than the normal initialization, but slightly worse than the Glorot initialization. Consequently, we adopted the initialization by Glorot et al. for all further experiments.

Figure 3: Comparison of weight initializations on the IAM database. The green dotted curves show the training progress in terms of label error rate for networks initialized with a normal distribution with fixed standard deviation and different random seeds. The red lines show the training progress with Glorot initialization and different seeds. The Glorot initialization leads to much faster convergence and less dependence on the seed.



VI. Experiments

In order to study the effect of preprocessing and network topology, particularly the effect of the width and depth of the network, we conduct experiments on the IAM database introduced in Section III. After identifying a well-performing setup, we also evaluate it on the RIMES database to verify that the setup performs well across different databases.

For recognition, we used the RWTH Aachen University Speech Recognition System [25], [26]. We emulated the CTC topology in a hybrid hidden Markov model fashion by expressing the state emission probabilities as rescaled posteriors. We then used a regular HMM decoder with an appropriately adjusted number of states and transition probabilities. All recognitions were performed on paragraph level

Table II: Comparison of preprocessing methods. Deslanting alone without contrast normalization yields the best results.

Preprocessing method	WER[%]		CER[%]	
	dev	eval	dev	eval
Raw	8.5	11.0	3.0	4.3
Deslanted	8.0	10.2	2.5	3.8
Contrast norm.	8.2	10.7	2.8	4.1
Deslanted + contrast norm.	8.7	10.3	2.7	3.9

to include additional language model context. We performed recognitions with the models from different epochs including the epochs with the lowest CTC objective function value and the lowest label error rate. Additionally, the language model scale, prior scale and word insertion penalty were tuned to minimize the word error rate on the development set. Performance is measured in word error rate (WER) and character error rate (CER).

A. Preprocessing

In Bluche et al. [27], the authors found that preprocessing was not helpful for their MDLSTM optical model on an Arabic handwriting recognition task, while in Strauß et al. [6], several preprocessing steps were used. Hence, we decided to perform further experiments. In a first experiment for preprocessing, we considered deslanting the line images and contrast normalization using the algorithms from Kozielski et al. [28]. We used the basic network topology to compare all four possible combinations of these preprocessing methods in Table II. Contrast normalization in isolation provides a small gain, but using only deslanting yielded better results than combining both. Consequently, for all further experiments, we used the deslanted images without contrast normalization.

B. Topology

Compared to one-dimensional LSTM networks, which commonly have more than 10 million parameters (e.g. [2]), the MDLSTM networks used for handwriting recognition are usually relatively small, possibly also because the sizes of the networks are restricted by the high runtime of the used CPU implementations. For example, the network used in Pham et al. [9] has roughly 142k parameters, while our basic model has 766k parameters. Hence, we study the effect of width and depth on the WER.

In a first experiment, we vary the number of hidden units per layer from $5n$ to $30n$ while keeping the number of hidden layers fixed. The results in Table III show that a too small hidden layer size severely hurts the recognition performance, but when the layer sizes are large enough, further increasing them yields little differences and can even hurt.

Next, we vary the number of hidden layers and the position of the max pooling operations while keeping the hidden layer sizes fixed at $15n$ where n is the layer index.

Table III: Comparison of different network widths. $5n$ means that the number of hidden units in the nth layer is $5n$.

Network	Params	WER[%]		CER[%]	
		dev	eval	dev	eval
$5n$	88k	9.9	12.1	3.3	4.6
$10n$	342k	8.4	10.3	2.7	3.9
$15n$	766k	8.0	10.2	2.5	3.8
$20n$	1.35M	8.3	10.5	2.5	3.9
$30n$	3.04M	8.3	10.2	2.6	3.8

When changing the network topology, we always stack alternating convolution and MDLSTM layers as before. The combination of a convolutional layer and a MDLSTM layer can be seen as a building block, only the number of these blocks and the presence or absence of max pooling for each block is changed. We describe a network architecture by a string like LP-LP-LP-L-L, where LP is such a building block with max pooling and L is a building block without max pooling. Note that the number of parameters also strongly increases when increasing the depth, as the layer size is scaled proportional to the layer index. Since we are mainly interested in the effect of the depth here and we showed before that simply increasing the width of the network can sometimes even hurt performance, we decided to limit the number of hidden units per layer to a maximum of 120. The results in Table IV indicate that the positions of the max pooling layers only play a minor role, while increasing the depth from two to a total of five blocks of convolution and MDLSTM, i.e. ten hidden layers, greatly improves the results from a WER of 10.2% to 9.3% on the evaluation set. However, the 12 layer network again performs worse.

In addition to changing the width and the depth of the network, we also tried to replace the tanh nonlinearity for the convolutional layers with rectified linear units and the recently proposed exponential linear unit [29], but did not observe improvements.

From these experiments, it can be seen, that tuning the network topology, in particular the depth of the network, is important to achieve good performance and just strongly increasing the width does not lead to good results. Clearly, there is still a lot room for improvement of the architecture.

C. Final Results

Our best result on IAM is achieved by the ten layer network of the last subsection which yields a WER of 7.1% on the development set and 9.3% on the evaluation set. Table V compares our results to previously published results on IAM. Doetsch et al. [2] used a modified 1D LSTM network architecture. Voigtlaender et al. [30] used 1D LSTM networks and applied a sequence-discriminative training criterion which could also be applied to our model for further improvements. Pham et al. [9] used a smaller MDLSTM optical model and no open vocabulary approach

Table IV: Comparison of different network topologies. The depth of the network and the position of max pooling is varied. LP stands for a block of convolution and MDLSTM with max pooling and L stands for such a block without pooling. The deep networks with ten hidden layers achieve the best results.

Network	Hidden layers	Params	WER[%]		CER[%]	
			dev	eval	dev	eval
LP-LP-LP	6	766k	8.0	10.2	2.5	3.8
LP-L-LP-LP	8	1.68M	8.0	10.0	2.5	3.6
LP-LP-L-LP	8	1.68M	8.2	10.4	2.8	4.0
LP-LP-LP-L	8	1.68M	8.4	10.4	2.5	3.7
LP-LP-LP-L-L	10	2.63M	7.1	9.3	2.4	3.5
LP-L-LP-L-LP	10	2.63M	7.2	9.3	2.4	3.5
LP-L-LP-L-LP-L	12	3.81M	8.1	9.7	2.6	3.6

Table V: Comparison of the proposed system to results reported by other groups on the IAM database.

System	WER[%]		CER[%]	
	dev	eval	dev	eval
Our system	7.1	9.3	2.4	3.5
Doetsch et al. [2]	8.4	12.2	2.5	4.7
Voigtlaender et al. [30]	8.7	12.7	2.6	4.8
Pham et al. [9]	11.2	13.6	3.7	5.1

for recognition. For a fairer comparison, we also performed a closed vocabulary recognition which led to WERs of 10.1% on the development set and 11.7% on the evaluation set.

For RIMES, we trained the same network which achieved the best result on IAM and used the same preprocessing (i.e. only deslanting). This setup yielded a WER of 11.3% on the RIMES evaluation set. Table VI compares this result to previously published results on RIMES. The systems of the other publications in the table are the same as for IAM.

It can be seen, that on both corpora our large MDLSTM optical model achieves significant improvements both over one-dimensional LSTM models and over previously used smaller MDLSTM models.

VII. Conclusion

We presented our efficient GPU-based implementation of MDLSTM and showed that the network depth plays an important role for good performance. We trained deep networks with up to ten hidden layers and achieved significant performance improvements outperforming state of the art results on two databases. However, we think that these experiments are just the starting point for further progress in handwriting

Table VI: Comparison of the proposed system to results reported by other groups on the RIMES evaluation set.

System	WER[%]	CER[%]
Our system	9.6	2.8
Doetsch et al. [2]	12.9	4.3
Voigtlaender et al. [30]	12.1	4.4
Pham et al. [9]	12.3	3.3

recognition. With the help of our implementation, many more hyperparameters and novel architectural components like deep residual networks [13] can be quickly explored. Additionally, our software provides a general framework for training, from which also other applications like speech recognition or image segmentation can benefit in the future.

ACKNOWLEDGMENT

The authors would like to thank Mahdi Hamdani for help with the open vocabulary recognition setup.

REFERENCES

[1] A. Graves, "Supervised sequence labelling with recurrent neural networks," Ph.D. dissertation, Technical University Munich, 2008.

[2] P. Doetsch, M. Kozielski, and H. Ney, "Fast and robust training of recurrent neural networks for offline handwriting recognition," in *International Conference on Frontiers in Handwriting Recognition*, Sep. 2014, pp. 279–284.

[3] X. Li and X. Wu, "Constructing long short-term memory based deep recurrent neural networks for large vocabulary speech recognition," in *2015 IEEE International Conference on Acoustics, Speech and Signal Processing*, Apr. 2015, pp. 4520–4524.

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

[5] A. Graves and J. Schmidhuber, "Offline handwriting recognition with multidimensional recurrent neural networks," in *Advances in Neural Information Processing Systems 21*, 2008, pp. 545–552.

[6] T. Strauß, T. Grüning, G. Leifert, and R. Labahn, "Citlab ARGUS for historical handwritten documents," *arXiv preprint arXiv:1412.3949*, 2014.

[7] E. Grosicki and H. ElAbed, "ICDAR 2009 handwriting recognition competition," in *Proc. of the Int. Conf. on Document Analysis and Recognition*, 2009.

[8] J. Li, A. Mohamed, G. Zweig, and Y. Gong, "Exploring multidimensional lstms for large vocabulary asr," in *2016 IEEE International Conference on Acoustics, Speech and Signal Processing*, March 2016.

[9] V. Pham, T. Bluche, C. Kermorvant, and J. Louradour, "Dropout improves recurrent neural networks for handwriting recognition," in *International Conference on Frontiers in Handwriting Recognition*, 2014.

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

[11] A. Graves, S. Fernández, F. J. Gomez, and J. Schmidhuber, "Connectionist temporal classification: labelling unsegmented sequence data with recurrent neural networks," in *Proceedings of the International Conference on Machine Learning*, 2006, pp. 369–376.

[12] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems 25*, 2012, pp. 1097–1105.

[13] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *arXiv preprint arXiv:1512.03385*, 2015.

[14] G. Leifert, T. Strauß, T. Grüning, and R. Labahn, "Cells in multidimensional recurrent neural networks," *arXiv preprint arXiv:1412.2620*, 2014.

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

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

[17] E. Augustin, J.-m. Brodin, M. Carr, E. Geoffrois, E. Grosicki, and F. Prłteux, "RIMES evaluation campaign for handwritten mail processing," in *Proceedings of the Workshop on Frontiers in Handwriting Recognition*, no. 1, 2006.

[18] P. Doetsch, A. Zeyer, P. Voigtlaender, I. Kulikov, R. Schlüter, and H. Ney, "RETURNN: The RWTH extensible training framework for universal recurrent neural networks," *arXiv preprint arXiv:1608.00895*, 2016.

[19] A. Graves, "RNNLIB: A recurrent neural network library for sequence learning problems." [Online]. Available: http://sourceforge.net/projects/rnnl/

[20] A. V. D. Oord, N. Kalchbrenner, and K. Kavukcuoglu, "Pixel recurrent neural networks," *arXiv preprint arXiv:1601.06759*, 2016.

[21] M. F. Stollenga, W. Byeon, M. Liwicki, and J. Schmidhuber, "Parallel multi-dimensional lstm, with application to fast biomedical volumetric image segmentation," in *Advances in Neural Information Processing Systems 28*, 2015, pp. 2998–3006.

[22] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[23] T. Dozat, "Incorporating Nesterov momentum into Adam," Stanford University, Tech. Rep., 2015. [Online]. Available: http://cs229.stanford.edu/proj2015/054_report.pdf

[24] A. M. Saxe, J. L. McClelland, and S. Ganguli, "Exact solutions to the nonlinear dynamics of learning in deep linear neural networks," *arXiv preprint arXiv:1312.6120*, 2013.

[25] D. Rybach, S. Hahn, P. Lehnen, D. Nolden, M. Sundermeyer, Z. Tüske, S. Wiesler, R. Schlüter, and H. Ney, "Rasr - the rwth aachen university open source speech recognition toolkit," in *IEEE Automatic Speech Recognition and Understanding Workshop*, Dec. 2011.

[26] S. Wiesler, A. Richard, P. Golik, R. Schlüter, and H. Ney, "Rasr/nn: The rwth neural network toolkit for speech recognition," in *IEEE International Conference on Acoustics, Speech, and Signal Processing*, May 2014, pp. 3313–3317.

[27] T. Bluche, J. Louradour, M. Knibbe, B. Moysset, F. Benzeghiba, and C. Kermorvant, "The A2iA arabic handwritten text recognition system at the openhart2013 evaluation," in *International Workshop on Document Analysis Systems (DAS)*, 2014.

[28] M. Kozielski, P. Doetsch, and H. Ney, "Improvements in rwth's system for off-line handwriting recognition," in *International Conference on Document Analysis and Recognition*. IEEE, 2013, pp. 935–939.

[29] D. Clevert, T. Unterthiner, and S. Hochreiter, "Fast and accurate deep network learning by exponential linear units (elus)," *arXiv preprint arXiv:1511.07289*, 2015.

[30] P. Voigtlaender, P. Doetsch, S. Wiesler, R. Schlüter, and H. Ney, "Sequence-discriminative training of recurrent neural networks," in *IEEE International Conference on Acoustics, Speech, and Signal Processing*, Apr. 2015, pp. 2100–2104.