Multi-Dimensional Connectionist Classification: Reading Text in One Step

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

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

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

***Abstract*—Offline handwriting recognition systems often use LSTM networks, trained with line- or word-images. Multi-line text makes it necessary to use segmentation to explicitly obtain these images. Skewed, curved, overlapping, incorrectly written text, or noise can lead to errors during segmentation of multi-line text and reduces the overall recognition capacity of the system. Last year has seen the introduction of deep learning methods capable of segmentation-free recognition of whole paragraphs. Our method uses Conditional Random Fields to represent text and align it with the network output to calculate a loss function for training. Experiments are promising and show that the technique is capable of training a LSTM multi-line text recognition system.**

I. Introduction

Current state-of-the-art systems for segmentation-free line-wise text recognition use Multi-Dimensional Long Short Term Memory *(MDLSTM)* networks [1] [2] trained with the Connectionist Temporal Classification *(CTC)* loss function [3]. Recognition systems that employ these techniques have shown high recognition rates for Latin handwritten text. CTC allows for the training of recurrent neural networks *(RNNs)* that recognize whole text lines without a prior segmentation into individual characters.

First systems [4] [5] capable of recognizing whole paragraphs without prior segmentation into text lines or characters were introduced recently. These techniques are based on three neural networks: an encoder network organized as a hierarchical subsampling network, followed by an attention network and a transcription network. The model uses the CTC loss function for recognition of multi-line texts while the attention mechanism segments the individual lines and brings them into correct sequential order. This requires repeated passes through the networks amount to a higher run-time for longer texts.

We propose to use a single hierarchical subsampling network with MDLSTM layers for end-to-end transcription of multi-line text from an image to a probabilistic classifier output, followed by an algorithmic decoding mechanism to obtain a nominal string of the transcription. Our approach uses Conditional Random Fields to train the neural network on this task. In contrast to the previously proposed methods [4] [5], our method transcribes multi-line text from an image in a single forward pass through a neural network only. This results in a low run-time of the text recognition system.

The proposed method for text alignment, network training and decoding avoids the typical problems during line segmentation caused by particularly skewed, curved or overlapping text lines. In our approach we train MDLSTM networks for a combined recognition of text lines and transcription of characters in one step. This is a step-up of MDLSTM networks that can handle input of multiple (variable sized) dimensions, but need to be forcefully collapsed to one dimension for CTC training.

We call our method of combining a MDLSTM network with a CRF-based loss function the "Multi-Dimensional Connectionist Classification" *(MDCC)*. MDCC is based on constructing an undirected and cyclic graphical model and approximate inference on it.

This paper is structured as follows: Section II describes the problem our proposed method tackles. Section III gives details of the proposed text alignment and loss function. Decoding of the probabilistic network output is described in Section IV. Sections V and VI describe experiments and results. We conclude the work by a discussion in Section VII.

II. Multi-line Text Recognition

Modern text recognition systems are often implemented by either processing each line without further segmentation, e.g. using a neural network trained with CTC, or following up with another segmentation step to retrieve fixed size character images for recognition. Systems employing line segmentation are prone to errors when processing text with skewed, curved or overlapping lines. Images of incorrectly segmented lines may contain cropped or distorted text lines and reduce the capacity of the recognition system to correctly transcribe text. The proposed method avoids this by leaving out the segmentation step. The text is transcribed from the whole image in one pass through the RNN.

Figure 1 shows a rough overview of the evolution of text recognition systems. Current systems employ one or two segmentation steps to break down the image into individual lines, words or characters. We propose to omit these segmentation steps and recognize the text from the whole image.

Our proposed method tackles the following problem: we assume a training set (X, Y_{text}) consisting of two-dimensional grayscale images X with multiple text lines, e.g. Figure 2, and

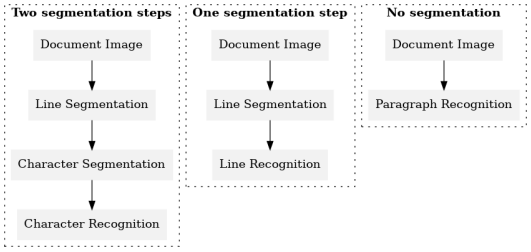

Fig. 1. Evolution of text recognition systems.

Fig. 2. Example image from the data set.

with transcribed text Y_{text} per image. The text recognition system is a MDLSTM hierarchical subsampling network trained in a supervised fashion. In contrast to previous work [6], our network does not require collapsing of the network prediction into a one-dimensional sequence. Since each $x \in X$ is a two-dimensional image and the network output is not forcibly collapsed, each network prediction $y_{\text{rnn}} = \text{RNN}(x, \mathbf{W})$ is a set of pixel-wise two-dimensional probability distributions. $\mathbf{W}$ is the set of weight and bias parameters of the RNN. y_{rnn} contains one probability distribution per symbol of the output alphabet. Since symbols are mutually exclusive at the same spatial position, a pixel-wise Softmax function is applied. y_{rnn} estimates the probability of a given pixel belonging to a specific symbol from the output alphabet.

Before aligning the text y_{text} over the RNN output y_{rnn}, we transform the text y_{text} to a label sequence l by the following rules:

- Symbols $\star$ map to their according label $g_\star$.
- Multiple adjacent occurrences of the same label $g_\star$ are separated by a glyph separator ϵ_g.
- Each line is started and ended with a single whitespace label $g_\sqcup$.
- Line separators ϵ_l are between each two lines, before the first and after the last line.

An example multi-line text "aa\nbc" maps to the sequence $l = (\epsilon_l, g_\sqcup, g_a, \epsilon_g, g_a, g_\sqcup, \epsilon_l, g_\sqcup, g_b, g_c, g_\sqcup, \epsilon_l)$ with length $|l| = 12$. $g_\star$ denote labels for visible symbols $\star$ and $\epsilon_\star$ denote invisible separators. We introduce two additional separators ϵ_l and ϵ_g into the neural network model, increasing the size of the output alphabet learned by the RNN by two. We will refer to the size of the output alphabet by $|g|$, which includes the two additional separators.

We use the following convention throughout the paper:

- y_{text} for the truth text of one sample.
- l of length $|l|$ for the label sequence representing y_{text}.
- y_{rnn} for the probabilistic RNN output.
- y_{align} for the alignment of truth text y_{text} over the pixel space of y_{rnn}.
- g of size $|g|$ for the output alphabet that is recognized by the RNN.
- s for pixels in the RNN output, each s being (x,y)-coordinates.
- x_s for indices into sequence l.

III. Conditional Random Field Alignment

Conditional Random Fields *(CRFs)* [7] are a generalization of Markov Random Fields *(MRFs)* [8] [9, p. 663] and as such undirected cyclic graphical models of multi-variate probability distributions. In contrast to MRFs, CRFs allow for conditioning of the probability distribution on an external set of features. CRF models are topologically constructed as a graph of nodes and undirected edges with the edges defining the neighborhood relations between the nodes. The proposed method applies discrete CRFs where each node has a probability vector of mutually exclusive labelings.

CRFs are defined by their *potential functions*, strictly positive functions that define the compatibility of a field configuration with the underlying model. Node potential functions model the a-priori compatibility $\psi_s(x_s)$ of a specific label x_s to a node s. Edge potentials accordingly define the compatibility $\psi_{s,t}(x_s, x_t)$ between two labelings x_s, x_t in their nodes s and t. Note that this work considers only CRFs with a maximum clique size of two.

We do inference on the CRF based on Loopy Belief Propagation *(LBP)* [10] [9, p. 769] [11], a standard method using in image processing. Belief propagation is a *message passing* algorithm for inference in graphical probabilistic models. Belief propagation was designed for acyclic models, but it can iteratively be applied to cyclic models and it shows convergence to the correct solution also in this case. This variant of belief propagation is named LBP. Beliefs are proportional to the probabilities of label x_s occurring in node s: $b_s(x_s) \propto p_s(x_s)$.

MDCC aligns the known truth text to the probabilistic RNN output by using LBP inference on a CRF. MDCC defines the node potential $\psi_s(x_s)$ and edge potential $\psi_{s,t}(x_s, x_t)$ functions for this task. The inferred probabilities will then be used to train the RNN in a supervised fashion. Defining the graph topology of the CRF is the first step towards this goal. We separate this into two parts, defining the CRF nodes and defining the CRF labels.

MDCC uses a grid-structured graph for representing the pixel space. Each node in the CRF represents one pixel of the RNN output and is connected to its four direct neighbors. We need to differentiate between geometric horizontal and vertical edges in pixel space to allow for a correct text alignment in later steps. We later use solid lines to represent horizontal edges and dotted lines for vertical edges.

Figure 3 displays the graphic representation of the truth text. Again, solid lines represent horizontal moves in the pixel space and dotted lines are vertical moves. Each node of the graph represents a specific position within the truth text. If the same symbol occurs multiple times in the text, it will translate to multiple nodes. Connecting two text positions in

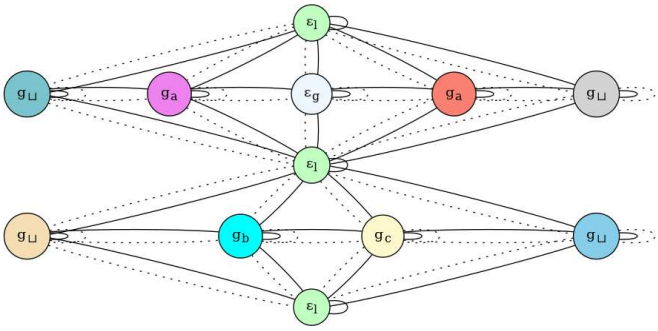

Fig. 3. Text representation for the CRF. Each node in this graph is one node labeling x_s in the CRF. Solid and dotted lines again represent horizontal and vertical transitions in pixel space. The encoded text is the example $l = (\epsilon_l, g_\sqcup, g_a, \epsilon_g, g_a, g_\sqcup, \epsilon_l, g_\sqcup, g_b, g_c, g_\sqcup, \epsilon_l)$ from Section II.

this graph means that those are allowed neighbors in pixel space. All symbols, with the exception of the line separator ϵ_l can be repeated in both horizontal and vertical direction. Line separators may only be one pixel in height. Adjacent symbols and lines may be connected in both horizontal and vertical directions to allow for slanted or curved text.

The two graphs of the pixel grid and truth text in Figure 3 simplify the construction of the CRF model by modeling pixel grid and text separately. The CRF topology is obtained by calculating the *graph tensor product* of the two graphs. Nodes from the pixel grid translate to CRF nodes s and nodes from the text graph to labels x_s of these CRF nodes s. We use $\sim_h$ for denoting neighboring nodes in horizontal direction and $\sim_v$ in vertical direction. CRFs have only one type $\sim$ of edge. An edge $(s, x_s) \sim (t, x_t)$ is added to the CRF iff $s \sim_h t \wedge x_s \sim_h x_t$ or $s \sim_v t \wedge x_s \sim_v x_t$.

MDCC restricts the graph tensor product by further application of the following rules to the relation $\sim$:

1) Edges may only be added to the CRF if their orientation (horizontal, vertical) is the same in the pixel and text graphs. This is expressed in the two edge types $\sim_h$ and $\sim_v$.
2) Only edges that still allow enough pixel space for the symbols and lines to the left, right, top and bottom of the current pixel may be added to the CRF. The truth text must not be truncated. For example the *h* of *hello* cannot occur in the four rightmost pixels because the remaining symbols would then not fit in.
3) An exception to rule 2 are leading and trailing whitespace symbols $g_\sqcup$ of each line, as well as the first and last line separator ϵ_l. These symbols are optional and the first visible symbol of the truth text may occur in the leftmost, rightmost, top or bottom pixel row or column.

We use a Potts model [9, p. 673] as the basis for our CRF and modify it. Edge potentials $\psi_{s,t}(x_s, x_t)$ are defined by a constant matrix of the size $|l| \times |l|$, in our example a 12×12 matrix for the 12 nodes in Figure 3. The diagonal of the matrix represents the same text position (CRF labels x_s and x_t) in two neighboring pixels (CRF nodes s and t) and favors these over movements in the text in the off-diagonals of the matrix if $w > 0$.

$$\psi_{s,t}(x_s, x_t) = \begin{pmatrix} e^w & e^0 & \cdots & e^0 \\ e^0 & e^w & \cdots & e^0 \\ \vdots & \vdots & \ddots & \vdots \\ e^0 & e^0 & \cdots & e^w \end{pmatrix} \tag{1}$$

Equation 1 is the template for the actual edge potential, which is different for each combination of nodes s and t because of *structural zeros*. Values of the matrix are set to exactly zero if the text positions x_s and x_t are not neighbors in nodes s and t according to the graph tensor product and above described ruleset. An edge potential $\psi_{s,t}(x_s, x_t) = 0$ effectively removes the edge from the CRF. Structural zeros allow to create a CRF that only calculates alignments for the given truth text and no variants or a truncation of it.

The weights are constant $w = 1.44$. The Potts model shows a *phase transition* behavior when modifying the weights w, as $w < 1.44$ results in many small segments, $w > 1.44$ in few large segments and $w = 1.44$ in a mixture of small and large segments. This is a general behavior [9, p. 673] of the Potts model and not specific to MDCC. Since symbols have different shapes and sizes, we chose to use the value of the phase transition and thus a mixture of small and large segments.

The node potentials $\psi_s(x_s | y_{\text{rnn}}, y_{\text{text}})$ define the compatibility between a text position x_s and the pixel s in the CRF model. It is dependent on the probabilities estimated by the RNN and favors text alignments for positions with a high RNN output. We take a two-dimensional Forced Alignment *FA* [12] $y_{\text{fa}}(s, l_{x_s})$ into account with a small factor of k_1, as well as a constant of k_2. Equation 2 details the node potential function that we used. FA defines probabilities $y_{\text{fa}}(s, l_{x_s})$ by placing the truth text in identical symbol widths and line heights in the pixel grid.

$$\psi_s(x_s | y_{\text{rnn}}, y_{\text{text}}) = e^{k_1 \times y_{\text{fa}}(s, l_{x_s}) + y_{\text{rnn}}(s, l_{x_s}) + k_2} \tag{2}$$

We chose values $k_1 = 0.25$ and $k_2 = 0.1$ for this work. This helps the initial alignment in the beginning of training but still allows a smooth transition to put more trust on the RNN output as its prediction gets more reliable. Text positions x_s in node s that violate the above ruleset, e.g. second text line in the top pixel row, are set to a node potential of zero.

Alignment of the truth text over the RNN pixels is approximated by running LBP on the described CRF model. We normalize the beliefs to $\sum_{x_s} b_s(x_s) = 1, \forall s$ and treat them like probabilities.

Figure 4 shows one alignment for our example text from Figure 3 to the RNN pixels. Please note that this is only one possible alignment, as e.g. the top g_a could also be a line separator ϵ_l. The number of alignments increases exponentially for larger pixel spaces. Each alignment yields different beliefs $b_s(x_s)$ for the text positions x_s in pixels s and LBP in *sum-product mode* calculates the mean over all alignments.

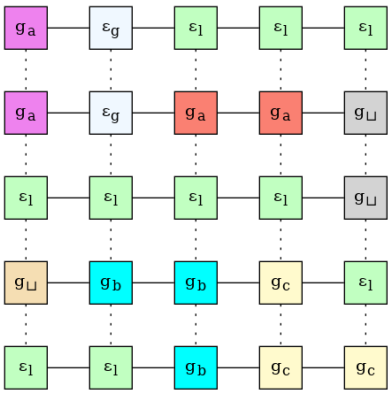

Fig. 4. One example alignment of the text of Figure 3 over the CRF nodes and thus RNN pixels.

LBP [9, p. 770] in sum-product mode can be seen as a generalization of the Forward-Backward algorithm [13]. The Forward-Backward algorithm infers probabilities in an acyclic directed graphical model by recursively passing messages along its edges. Messages are the product of the source node beliefs and the edge transition probability towards the target node. Incoming messages of a node are summed up. This is also called *Belief Propagation* because messages are the beliefs of the probability state of neighboring nodes. Message passing is repeated in forward and backward directions. Normalization of the beliefs per node will yield the exact probabilities. Inference in a tree-structure would start at the root node, pass messages towards the leaf nodes and then backwards again to the root node. LBP generalizes on this concept by repeatedly applying message passing to a cyclic graphical model until the beliefs have converged to a stable state.

Messages in LBP are the product of the source node belief $b_s(x_s)$, its node potential $\psi_s(x_s|y_{\text{rnn}}, y_{\text{text}})$ and the edge potential $\psi_{s,t}(x_s, x_t)$ towards the target node. Incoming messages are again summed up. Each iteration of LBP is the process of sending out every message in the CRF exactly once. Figure 5 shows message passing in LBP.

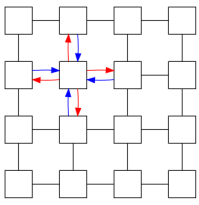

Fig. 5. Collecting (blue) and sending out (red) messages in LBP. This is repeated for all nodes and until convergence of beliefs.

LBP is an iterative algorithm and as such needs criteria for termination. Useful approaches are to check for the convergence of the passed messages or limiting the total number of iterations. We simply terminated LBP after exactly 75 iterations. This was enough to lead to a reliable result in our experiments, but other use cases may need a larger number of iterations. Convergence of the beliefs cannot be guaranteed for general graphs, but we applied several enhancements to increase the chances of convergence:

- Random shuffling of the order in which messages are processed. This helps both convergence and breaking symmetries in the alignment, such as always large segments in the top left corner.
- Asynchronous updates: using incoming message values from the current iteration instead of the previous one if they already have been updated.
- Normalizing incoming messages per node after each iteration.

For training the RNN, summing up of beliefs $b_s(x_s)$ is necessary since x_s are indices in l, which is a different set to g because of multiple occurrences of the same symbol within the sequence. We sum up all beliefs per symbol g, given that it is contained in the truth sequence l, to obtain the probabilities of a symbol occurring in a given pixel:

$$y_{\text{align}}(s, g) = \sum_{x_s, l(x_s)=g} b_s(x_s) \tag{3}$$

Both y_{rnn} and y_{align} are now of the same structure: two-dimensional probability distributions for each of the symbols of the output alphabet. y_{align} contains the target probabilities of a specific pixel belonging to a specific symbol and y_{rnn} the according network estimate. The actual loss function for supervised training of the RNN is a multinomial cross entropy loss. The general form of the cross entropy loss is $L = -\sum_i t_i \times \ln(o_i)$ with the target probabilities t_i and network output o_i. We substitute y_{align} and y_{rnn} for the proposed loss function in Equation 4, with its derivative in Equation 5.

$$L = -\sum_s \sum_g y_{\text{align}}(s, g) \times \ln(y_{\text{rnn}}(s, g)) \tag{4}$$

$$\frac{\partial L}{\partial y_{\text{rnn}}(s, g)} = \frac{-y_{\text{align}}(s, g)}{y_{\text{rnn}}(s, g)} \tag{5}$$

IV. Decoding

The previous Sections II and III describe the text alignment and loss function for training the RNN for multi-line text recognition. The network estimates a probability prediction y_{rnn} for a given multi-line document image. The predictions have two spatial dimensions and $|g|$ pixel-wise mutually exclusive probabilities for each of the symbols in the output alphabet. The text recognition system uses a decoding algorithm to obtain a symbol sequence from this probabilistic classifier.

We use a scan line for decoding the network output. The scan line extends over the whole width of the output and moves from top to bottom. Figure 6 shows our example with the current and next state of the scan line in solid lines. The scan line is not restricted to a straight line, but is allowed to jump

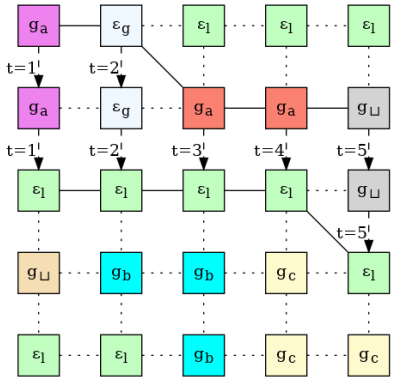

Fig. 6. Decoding procedure for the the example from Figure 4.

in the vertical dimension. This corresponds to diagonal lines from top-left to bottom-right or bottom-left to top-right.

The scan line moves vertically, alternating between line separators and visible text lines. Text lines are decoded while the scan line moves through them. Individual text text lines are obtained by moving the scan line through them, summing up glyph probabilities per column and then decoding these sum probabilities. Decoding of probabilities to sequences can be done by either *Best Path Decoding* or *Prefix Search Decoding* as described in [3]. This generates one sequence of symbols per text line. Moving the scan line through a visible text line is depicted by the time steps $t = 1$ through $t = 5$. In our example, it will read the sequence $(g_a, \epsilon_g, g_a, g_a, g_\sqcup)$ and move the state of the scan line to the line separators ϵ_l. Text lines are decoded to a human-readable string by removing adjacent duplicates from the sequence, followed by removing separators ϵ_g. The example text line is decoded to the string "aa ". After decoding a visible text line, the scan line moves through a stripe of line separators ϵ_l and adds a newline character to the decoded text. This process of moving the scan-line vertically is repeated until it moved through the complete output. The RNN prediction is then fully decoded to a readable string.

V. Neural Network Training

The neural networks in our experiments are hierarchical subsampling MDLSTM network with the topology and parameters detailed in Table I. We also conducted experiments with other layer sizes and subsampling windows, but the reported configuration has shown good and most reliable results.

MDLSTM layers are a stack of four two-dimensional LSTM layers, one per scan direction over the two spatial dimensions. MDLSTM layers contain peephole connections from the internal cell state to the gates. They have bias values for both the cell state and gates. The locally fully connected feedforward layers do not contain bias values, except for the last one. The described neural network contains a total of 978748 trainable parameters.

Weights were initialized by drawing from a uniform random distribution with a value range from -0.1 to $+0.1$. Bias values were initialized to zero. Optimization of the parameters has been done using mini-batches of size 32 and RMSProp [14] with learning rate $\mu = 0.001$, dampening factor $\epsilon = 1e^{-3}$ and decay rate $p = 0.95$.

TABLE I
Network topology for the conducted experiments.

Layer type	Parameters
Image input	8-bit grayscale, 300dpi
Subsampling	Window 3x3, stride 3x3
MDLSTM	8 cells per direction, tanh activation
Subsampling	Window 2x2, stride 2x2
Feedforward	64 neurons, no bias, tanh activation
MDLSTM	44 cells per direction, tanh activation
Subsampling	Window 3x3, stride 3x3
Feedforward	172 neurons, no bias, tanh activation
MDLSTM	80 cells per direction, tanh activation
Feedforward	$\|g\|$ neurons, with bias, linear activation
Softmax	Pixel-wise Softmax activation function

Training of the neural network was initiated by pre-training on single-word IAMDB data using CTC [3]. The collapsing layer and the last feedforward layer were removed after pre-training and a new randomly initialized feedforward layer added for multi-line training. Only this new layer was trained at first until no further reduction of error could be measured. We trained the whole network after this pre-training phase using the described CRF alignment for multi-line training.

The network error was measured in Character Error Rate *(CER)*. CER measures the ratio of the Edit distance [15] between the transcribed text and the truth text in relation to the length of the truth text. Lines are separated by one newline character for multi-line texts.

$$\mathrm{CER}(y_{\mathrm{rnn}}, y_{\mathrm{text}}) = \frac{100 \times \mathrm{Edit}(\mathrm{Decode}(y_{\mathrm{rnn}}), y_{\mathrm{text}})}{|y_{\mathrm{text}}|} \quad (6)$$

VI. Results

We conducted experiments on a data set extracted from the IAM offline handwriting database [16]. Each sample of the data set consists of 2 text lines with 3 words each. Words and lines were used in order in which they appear on the pages. Only words that were marked as *ok* were used. In total 11508 samples are in the data set, one is displayed in Figure 2. 5 percent were used for each of the validation and evaluation sets during training and evaluation.

In our experiment, CTC on the IAM word-images was used as pre-training until the error rate did not further improve on the validation set. CRF alignment and loss function together with the described multi-line data were used after this. Convergence rate on the training and validation sets over all training epochs is shown in Figure 7. Best CER were 7.15% on training, 11.5% on validation and 11.5% on evaluation data sets after 93 epochs of training and using *Best Path Decoding* [3]. CER on the evaluation set was reduced to 10.4% by using *Prefix Search Decoding* [3].

This is an improvement over the 10.9% CER reported by Bluche et al. [4, p. 6] on a similar data set. Our method takes $553\,\mathrm{ms}$ for transcribing one sample on an Intel i7-6560U at

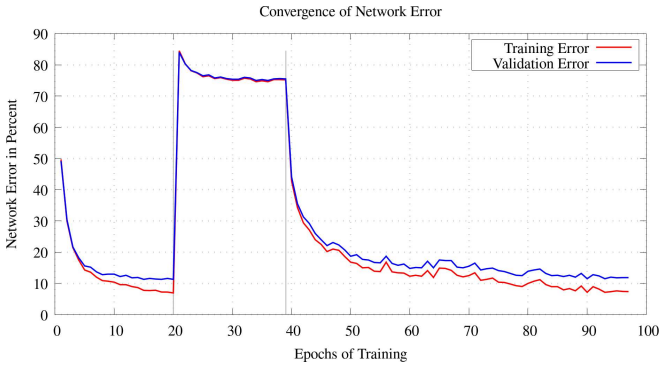

Fig. 7. Convergence rate of CER on the training (red) and validation (blue) sets. Ends of pre-training phases after 20 and 39 epochs are marked by the vertical lines.

2.2GHz or 506 ms on an Intel Xeon E5-2640 at 2.4GHz. No multi-threading was used for runtime measurements and the network implementation is CPU based with AVX2 as a vector extension.

Fig. 8. Left to right: RNN prediction, CRF alignment and error signal for the symbol *a* after 93 epochs.

Figure 8 show the RNN prediction, CRF alignment and error signal for the symbol *a* from Figure 2 after 93 epochs of training. The first and second columns show heat maps of the probabilities calculated by the RNN and CRF alignment. To show these probabilities in context, the heat maps are partially shown for high probabilities and then superimposed over the input image. The third column shows a complete heat map of the partial derivative of the loss function.

MDCC trains the RNN for producing the correct sequence of glyphs in order to decode the correct text. Localization of individual glyphs is not necessarily correct, as can be seen in Figure 8 were predictions of *a* and alignments are spatially higher than the actual characters in the image. The reported CER shows that localization information is not necessary for transcription using MDCC.

VII. Discussion

We introduced MDCC, a method for training a RNN for segmentation-free multi-line offline text recognition by using pre-training followed by a switch to a CRF text alignment and loss function. No explicit segmentation of lines or characters is utilized in this work. In contrast to other work [4] does our method only need a single forward pass through a MDLSTM network for transcription of a multi-line text. This makes MDCC suitable, depending on the size of the network and available computing resources, for real-time applications. We also propose a mechanism based on a horizontal scan-line for decoding the probabilistic network output to a multi-line character string. MDCC shows a lower CER than the previously published methods.

As other work with CRFs in the field of computer vision show, CRFs are capable of defining complex dependencies in pixel space while allowing approximate inference. This could allow for training RNNs for the recognition of complex text layouts in the future. MDCC can possibly be adapted to other weakly supervised problems, e.g. in three or four dimensional space, if ground truth information can be encoded in a CRF.

To the authors knowledge, MDCC is the first method for recognition of multi-line texts in a single pass through a neural network.

Acknowledgment

The authors would like to thank the Siemens Postal, Parcel & Airport Logistics GmbH for funding this work and providing computer hardware for carrying out the described experiments.

References

[1] S. Hochreiter and J. Schmidhuber, "Long short-term memory." *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.
[2] A. Graves, S. Fernandez, and J. Schmidhuber, "Multi-Dimensional Recurrent Neural Networks," IDSIA/USI-SUPSI, Tech. Rep., 2007.
[3] 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.
[4] T. Bluche and R. Messina, "Scan, Attend and Read: End-to-End Handwritten Paragraph Recognition with MDLSTM Attention," pp. 1–10, 2016. [Online]. Available: http://arxiv.org/abs/1604.03286
[5] T. Bluche, "Joint Line Segmentation and Transcription for End-to-End Handwritten Paragraph Recognition," in *NIPS 2016: Neural Information Processing Systems*, 2016.
[6] 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.
[7] J. Lafferty, A. McCallum, and F. C. N. Pereira, "Conditional random fields: Probabilistic models for segmenting and labeling sequence data," *ICML '01 Proceedings of the Eighteenth International Conference on Machine Learning*, vol. 8, no. June, pp. 282–289, 2001.
[8] J. Pearl, "Probabilistic Reasoning in Intelligent Systems," p. 552, 1988.
[9] K. P. Murphy, *Machine learning: a probabilistic perspective (adaptive computation and machine learning series)*. MIT Press, 2012.
[10] K. Murphy, Y. Weiss, and M. Jordan, "Loopy-belief Propagation for Approximate Inference: An Empirical Study," *15*, pp. 467–475, 1999.
[11] V. Martin, J. Lasgouttes, and C. Furtlehner, "The Role of Normalization in the Belief Propagation Algorithm," *arXiv preprint arXiv:1101.4170*, pp. 1–27, 2011. [Online]. Available: http://arxiv.org/abs/1101.4170
[12] M.-P. Schambach and S. F. Rashid, "Stabilize sequence learning with recurrent neural networks by forced alignment," in *Proceedings of the International Conference on Document Analysis and Recognition, ICDAR*, 2013, pp. 1270–1274.
[13] 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.
[14] Y. N. Dauphin, J. Chung, and Y. Bengio, "RMSProp and equilibrated adaptive learning rates for non-convex optimization," Tech. Rep., 2014.
[15] 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.
[16] U. V. Marti and H. Bunke, "The IAM-database: An English sentence database for offline handwriting recognition," *International Journal on Document Analysis and Recognition*, vol. 5, no. 1, pp. 39–46, 2003.