Stabilize Sequence Learning with Recurrent Neural Networks by Forced Alignment

Marc-Peter Schambach
Siemens AG
Bücklestraße 1-5, 78464 Konstanz, Germany
Email: marc-peter.schambach@siemens.com

Sheikh Faisal Rashid
Siemens AG
University of Kaiserslautern, Germany
Email: rashid@iupr.com

***Abstract*—Cursive handwriting recognition is still a hot topic of research, especially for non-Latin scripts. One of the techniques which yields best recognition results is based on recurrent neural networks: with neurons modeled by long short-term memory (LSTM) cells, and alignment of label sequence to output sequence performed by a connectionist temporal classification (CTC) layer. However, network training is time consuming, unstable, and tends to over-adaptation. One of the reasons is the bootstrap process, which aligns the label data more or less randomly in early training iterations. This also leads to the fact that the emission peak positions within a character are located unpredictably. But positions near the center of a character are more desirable: In theory, they better model the properties of a character. The solution presented here is to guide the back-propagation training in early iterations: Character alignment is enforced by replacing the forward-backward alignment by fixed character positions: either pre-segmented, or equally distributed. After a number of guided iterations, training may be continued by standard dynamic alignment. A series of experiments is performed to answer some of these questions: Can peak positions be controlled in the long run? Can training iterations be reduced, getting results faster? Is training more stable? And finally: Do defined character position lead to better recognition performance?**

I. Introduction

Cursive handwriting recognition has drawn continuous attention during the last years. Competitions measuring recognition performance for various scripts and languages [1], [2] are popular, and the number of public databases is constantly increasing. The focus on recognition performance has made recognition approaches comparable, and has led to professionalism in the field. State-of-the-art systems are either based on hidden Markov models (HMM), using sophisticated classification methods and topologies, or recurrent neural networks (RNN) with a final alignment layer, which will be used here.

The amount of training data available and the demand for high recognition performance require systems with millions of parameters. This makes parameter training time-consuming. Training data consists of images containing scanned lines of text, tagged with the textual content, but without character positions. The missing segmentation information makes the bootstrap process unstable: Training convergence time varies in a great range. And properties of the final system are undefined, especially the exact position of the characters during recognition.

The solution presented here is to guide the training during the first iterations with forced alignment of characters. The standard forward-backward algorithm for character alignment is replaced by either an estimated alignment, or by a fixed segmentation from other sources. After few iterations of the guided bootstrap training, standard forward-backward training is used again, which best adapts to the training data. The expectation is, that training startup time is reduced significantly, while the control over character positions is kept during the final training iterations.

The paper is organized as follows: Section II gives an overview of the recognition system, while section III presents the data used for visualization and experiments. Section IV explains the idea of forced alignment and describes the application in the given training context. Section V then describes the experiments and its results. Conclusions are drawn in the final section.

II. System

The system used here is based on recurrent neural networks and has been described by Alex Graves [3], [4].

It's a multi-layer neural network, which basically transforms a two-dimensional pixel plane into a sequence of class probabilities. It does so by sub-sampling the input pixel planes in each layer and finally collapsing the final plane in y-direction, getting away with a sequence in x-direction. Classes represent characters including whitespace, complemented by a non-character class which represents everything between characters. Recognition results are derived from the class probability sequence by dynamic programming.

Each layer is recurrent; it gets its input not only from the input pixel, but also from the neighboring cells within the layer. To get the whole context within two dimensions, four layers are implemented, each getting input from neighboring cells in *NW*, *SW*, *SE*, and *SW* direction, respectively. Cells are long short-term memory (LSTM) cells, which contain input, output and forget gates. These gates give the network a richer structure, and e.g. address the vanishing gradient problem. More details in [3].

The topology used in the experiments contains three hidden layers, with 4, 20 and 100 cells each. Sub-sampling layers have dimension 2×3 and sizes 6 and 30.

III. Data

Latin word images have been used for experiments and visualization, because Latin can be read by most people and

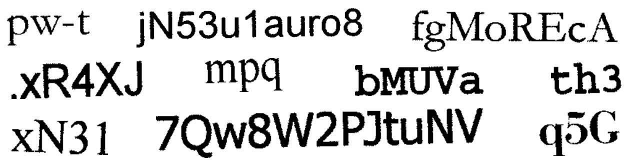

Fig. 1. Representative samples of synthetic words images: Random words of maximum length 12 have been rendered with various fonts and distortions.

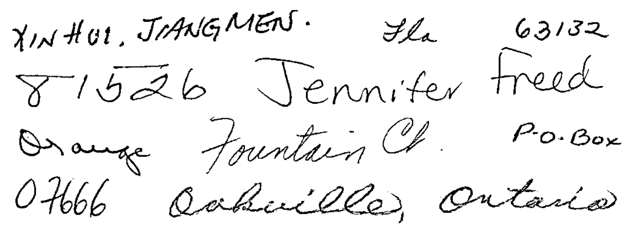

Fig. 2. Representative handwriting samples: Word images from US and Canadian postal applications, semi-automatically extracted and labeled.

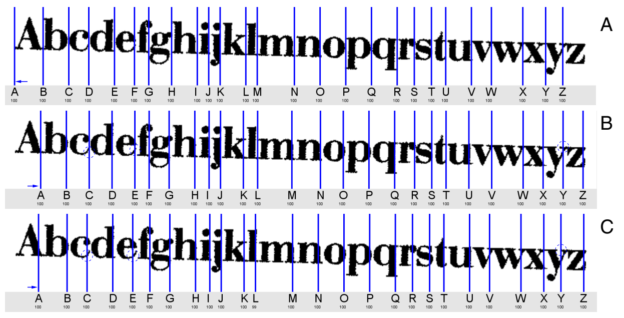

Fig. 3. Result of forward-backward training with different random weight initialization. While network A locates characters at the left side of the character image, networks B and C locate them at the right side. Network C tends to locate them a little bit to the center of the characters, as can be best seen with characters ‘c’,‘e’ and ‘y’.

systems are best understood. Both synthetic machine print and scanned handwriting has been investigated. Synthetic data needs less training time and allows for clear interpretation of segmentation results. Handwriting samples are used to rate recognition performance in realistic conditions.

A. Synthetic Data

Random words of size 1 to 12 have been used to synthesize word images. Samples are shown in Fig. 1. The alphabet includes 26 Latin characters, each in upper- and lower case, 10 digits, dash ‘–’ and space ‘ ’. Common fonts and font sizes have been used for rendering; rotation, additional noise and various binarization thresholds simulate typical artifacts from printing and scanning. 10k word images have been used for training, 1k each for validation and testing.

B. Handwriting

Handwritten word images have been taken from US and Canadian postal applications. Samples are shown in Fig. 2. Country names, city names, street names, personal names, and zip codes have been semi-automatically extracted and labeled. The alphabet has been restricted to 26 Latin characters (accents have been ignored), 10 digits and 12 special characters. Words and numbers are written in cursive script or hand block, in a wide range of sizes. They have been scanned at 212 dpi, with most image sizes in the range of 40 to 200 pixels height. 100k word images have been used for training, 10k for validation, and 4k testing.

IV. Forced Alignment

A. General concept

In back-propagation training, the difference between network output and input label gives the error, which is the basis for weight adaptation by gradient descend. For sequence learning, the correct label sequence has to be *estimated.*

The algorithmic concept is *expectation-maximization* (EM), a circular process, which estimates the assignment of data by a model (expectation), and then modifies the model using this assignment (maximization). It converges to a local optimum. It is the basis for Baum-Welsh training of HMM [5] which uses the forward-backward algorithm for “soft” alignment of characters. It has been proven that it is superior to Viterbi training, which performs “hard” character alignment, because it takes only the single best assignment path into account [6].

However, when training a randomly initialized network, in the first iterations the character probability distribution is smeared over the whole output sequence. In practice, this gives networks which locate the characters close to the boundaries of the training words. This tendency weakens when the network converges, but nearly always results in networks which finally locate the characters at the left or right edge of the character images. See Fig. 3 for examples.

It also slows down training, because the network doesn’t learn the correct character shapes in the first iterations. To avoid this, forward-backward soft alignment is replaced by an alignment with fixed positions. The “correct” character position gets probability one, while all others probability zero. The probability distribution corresponds to Viterbi training, but the positions are defined independent of the network output. This corresponds to the naïve approach in [6], but it will be only used for bootstrapping.

B. Estimation of “correct” position

There are several possibilities to define the position of a character within a word:

- Even distribution of characters: Every character is assumed to have the same width, so character positions have equal distance. While this method obviously unduly simplifies, it may still give an approximation good enough for bootstrapping. The method could be improved by fixed weights for each character, e.g. 0.7 for ‘i’ and 1.3 for ‘w’, which has not been implemented in this work.
- Existing recognition system: Use a system that has been already trained. From the behavior of the system it can be observed where it generally locates characters. This allows to specify the range of characters within training images. This can also be used to train a system with

different training data or extended character set, e.g. umlauts. When using the *same* training data, it can be used to control the peak position.
- Rendering: For synthetic data, the character position is known during rendering. It's the method of choice when using synthetic training data. It is not used in this work, because synthetic data has only be used for proof of concept.
- Manual segmentation: If manual or semi-automatical segmentation information is available, it can be used to define the character positions. Manual segmentation data has not available in this work.

All methods provide a range for the character position. The exact character position can be set at the left, center or right side of this range.

C. *Multi-step training*

Forced alignment is only useful in the first training iterations. Forward-backward training much better adapts to training data and should be executed after forced alignment training. When should training switch to forward-backward alignment? There are two possibilities:

- Perform forced alignment training until it finally converges.
- Perform forced alignment training until it is "good enough", which may be defined by a threshold for training error.

In this work, full convergence of forced alignment training has been used to show the effects of the method. If training times are a concern, the reduced number of bootstrap training iterations may be enough.

V. Experiments

A. *Character positions*

When training with plain forward-backward alignment, character positions are random, as has been shown in Fig. 3. Is it possible to control the positions where the characters are detected?

In the first experiment, networks have been trained with forced alignment to equidistant positions at the left, middle and right side of each image segment. Training images have been random words of up to length 12, as described in section III. Fig. 4 shows the effect on a long string containing the whole alphabet. Characters are in fact located at the left, middle and right position during recognition; small characters may even be detected outside of the proper image, showing the impact of the simplified alignment model.

Results are improved when forced-alignment training is performed based on an existing recognition system, which has been trained with standard forward-backward alignment. The base network indicates characters on a random, but stable relative position within the image; character segments are defined accordingly. Then, characters have been forced to the middle of these segments, where they in fact show up during recognition (Fig. 4, *supervised middle*). Even small characters are located correctly within their image.

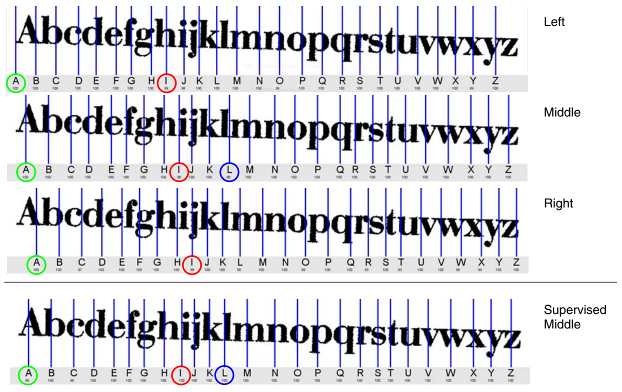
Abcdefghijklmnopqrstuvwxyz
Left
Middle
Right
Supervised Middle
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

Fig. 4. Result of forced alignment on synthetic machine print. Character position have been forced to the left, middle and right of the character by an equidistant model (top), and to the middle supervised by a previously trained system (bottom). See character *'A'* (green) for character positions. Especially small characters may be located outside their images due to the inadequate model (*'I'*, red). The model's tendency to equidistance does not exist in the bottom system (*'L'*, blue).

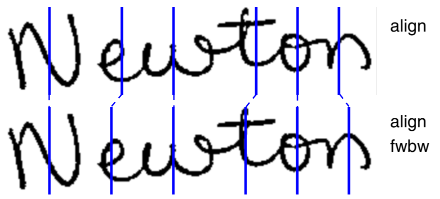

Fig. 5. Results of forced alignment on handwriting. Character positions have been forced to the center of the character by segmentation information from a previous training (top, *'align'*). Character position are stable even after re-training with forward-backward training (bottom, *'align–fwbw'*).

B. *Positional stability*

Forced alignment does not optimally adapt the network to the training data compared to dynamic alignment [6]. Network adaptation has to be continued with forward-backward training. How stable are character position, when alignment in not enforced any more?

The second experiment operates on handwritten words, which have been trained to middle position based on the segmentation of a previously trained network. Fig. 5 demonstrates that, as expected, characters are found at the middle position (*align*). Even after re-training with forward-backward alignment (*align–fwbw*), positions remain stable at the middle position.

In a third experiment, the forced-alignment training has been applied on an already trained system: Instead of random initialization, network parameters from a previous standard training have been used. While the base network (*fwbw*) locates characters at the left side (Fig. 6), the trained network (*fwbw–align*) locates them in the middle, as expected. However, when training is continued with forward-backward alignment again (*fwbw–align–fwbw*), characters move back nearly to the original positions!

The reason for this is important. Early features, contained in the first layers of the network, may be stable against re-training. When alignment is enforced, only character de-

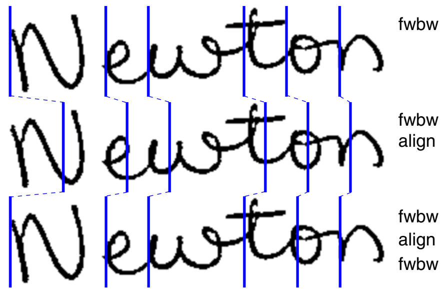

Fig. 6. Results of forced alignment on handwriting after retraining. A standard forward-backward training created character positions close to the left side of characters (top, *'fwbw'*). After forced alignment training, character position have moved to the center (middle, *'fwbw–align'*). Re-training with forward-backward training moved character positions nearly back to original positions (bottom, *'fwbw–align–fwbw'*). Only character *'o'* is still closer to the center

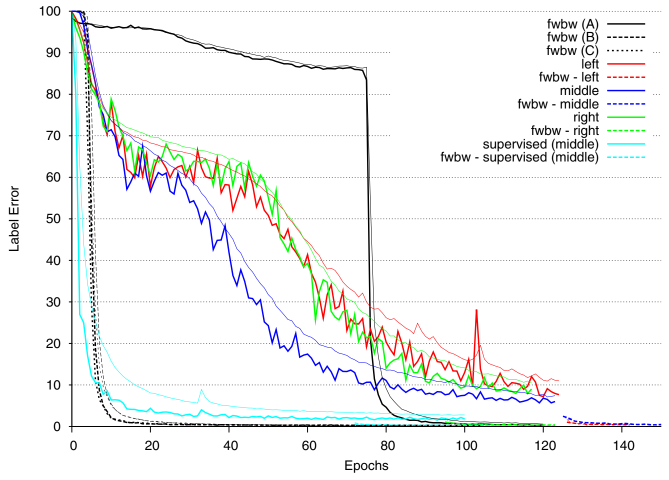

Fig. 7. Convergence of label error for different trainings using the same synthetic machine print data. Thick lines show error on validation data, thin lines on training data. Forward-backward trainings (*fwbw*) show a plateau in the first epochs, which extends up to epoch 75 for training *(A)*. Equidistant forced-alignment training (*left, middle, right*) converges best for *middle* alignment. Error is much reduced by final forward-backward training after epoch 125. Best results are obtained by *supervised* alignment with final forward-backward training.

scriptions from subsequent layers may be adapted, while the basic features are still best adapted to the original character descriptions. Accordingly, in order to really control character location, basic features have to be trained right from the beginning.

C. Convergence

The objective of this work has been to stabilize convergence of network training. Fig. 7 shows the instability of forward-backward training: Networks *A*, *B* and *C* have been initialized with different random parameters, but trained with the same data. While networks *B* and *C* start converging around epoch 5, network *A* needs more than 75 epochs until it starts converging. All three networks need some iterations until alignment is defined well enough to discriminitavely train parameters.

Forced alignment based on the assumption of equidistance of characters (*left, middle, right*) tends to converge faster in the

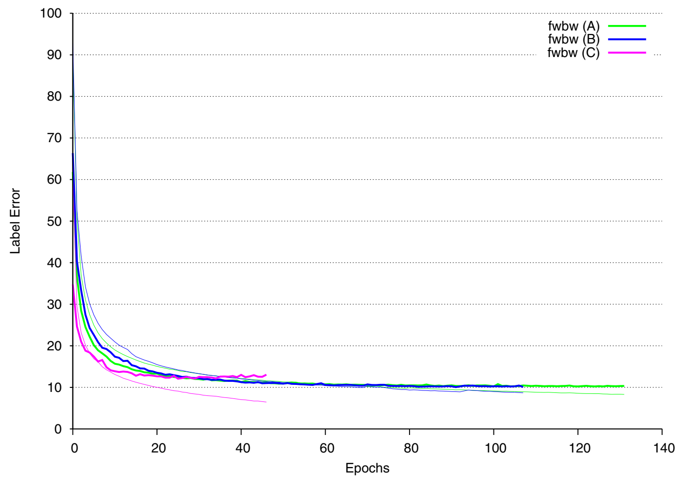

Fig. 8. Convergence of label error for three forward-backward trainings using the same handwriting data. Training *C* converges faster, but runs into over-adaptation. No plateau as for synthetic data in Fig. 7 has been observed.

beginning; subsequently it slows down due to the shortcomings of the alignment model. The middle position handles this shortcoming best. Training with *supervised* alignment converges fastest of all, but also slows down, because nailing characters to a fixed position is not optimal. In all cases, switching to forward-backward training gives fast convergence.

Fig. 8 shows the convergence of label error for handwriting data from section III-B. Compared to training of synthetic machine print, fast convergence right from the beginning can be observed for standard training. This is due to the larger amount of training data (100k images) which is processed in each epoch, which hides slow convergence in the first iterations. Network *C* is strongly over-adapting, and training stops early after a series of 20 epoch without improvements.

Forced alignment has been performed based on segmentation hypotheses calculated by an earlier trained system. Two experiments have been performed (Fig. 9): The first (*red*) starts with forced-alignment right from the beginning and continues with forward-backward training; the second (*green*) starts with standard training, then applies forced alignment for a few iterations, and continues with forward-backward training again. While the first experiment converges slower than forward-backward training alone, it yields better results in the long run. However, best results are provided by experiment two.

D. Recognition Performance

It has been shown in the last section, that forward-backward training sometimes goes into local minima, leading to sub-optimal recognition performance. The training seems to favor character location at the left and right edges of the character image. The expectation is to improve recognition performance by aligning characters to the middle, where most information is available.

Table I shows recognition rates on the independent test set of *synthetic* machine print words. Fixed character alignment in training strongly increases character error rate when applied

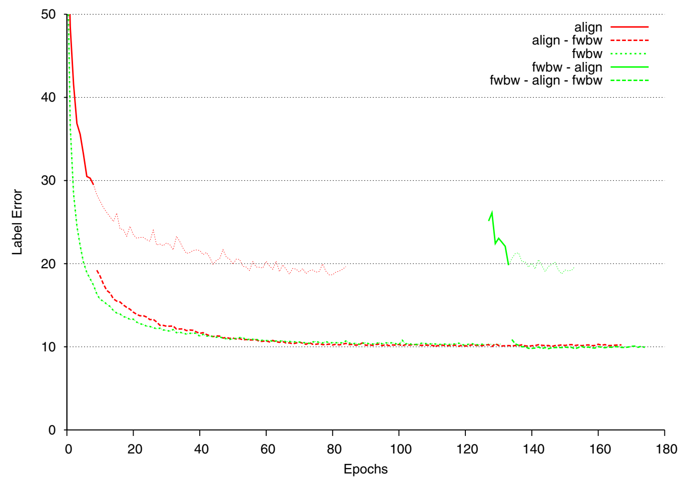

Fig. 9. Convergence of label error for forced-alignment training on handwriting data. The red lines show forced alignment applied from the beginning (*align*), continued by forward-backward alignment (*align–fwbw*) after nine epochs. The green lines show standard training (*fwbw*), continued by forced-alignment at epoch 127 (*fwbw–align*), and finally forward-backward at epoch 134 again (*fwbw–align–fwbw*). Small dotted lines show continuation of training after parameter transfer.

TABLE I
CHARACTER ERROR RATES ON 1K TEST SET OF SYNTHETIC WORDS.

Trained network		Character error rate	
		align	fwbw
standard	A		0.43%
	B		0.33%
	C		0.29%
equidistant	left	8.40%	0.31%
	middle	6.92%	0.33%
	right	7.34%	0.29%
supervised	middle	1.68%	0.42%

alone, as can be seen in the left result column. Even *supervised* alignment is far from the results of forward-backward training. However, when training is continued, recognition rates for *equidistant* alignment seem to be slightly better in average, but — surprisingly — *supervised* training is worst. However, most errors are random, mixing letter *‘O’* and digit *‘0’*, or letter *‘l’* and digit *‘1’*. These differences are not significant.

This changes with handwritten words, which are still a challenge for recognition systems. Table II shows recognition rates on the independent test set of the handwritten words. Standard forward-backward trainings *A*, *B* and *C* converge to networks with a big variance in character error rate. Supervised training from scratch *(align)* followed by forward-backward *(align–fwbw)* improved the best overall performance by an absolute 0.53%. Continuation of training *A* with forced alignment *(fwbw–align* and final forward-backward training *(fwbw–align–fwbw)* improves character error rate even more, by an absolute 0.68%. In applications, this makes a considerable difference.

TABLE II
CHARACTER ERROR RATES ON 4K TEST SET OF HANDWRITTEN WORDS.

Trained network	Character error rate
fwbw (A)	**9.13%**
fwbw (B)	23.19%
fwbw (C)	10.39%
align	18.98%
align–fwbw	8.60%
fwbw–align	19.22%
fwbw–align–fwbw	**8.45%**

VI. Conclusions

Forced alignment training has been proposed to stabilize the training of recurrent neural networks for unconstrained text recognition. The following observations have been made:

- Character positions can be effectively controlled by forced-alignment during training. Positions remain stable when alignment has been applied from the beginning. However, positions shift back to the original position when re-training an existing system
- Network convergence is more stable. Even if the average convergence speed is not improved, outliers with extremely bad convergence can be avoided by setting a good starting point with supervised alignment.
- By applying the technique, character error rate on handwriting images could be improved from 9.13% to 8.45%.

In future work, we want to find out, why character positions move back from the aligned positions to their original positions in final forward-backward training. Various small numbers of epochs with forced-alignment training will be executed in order to observe their influence on the final network. Besides, more statistical evidence on training convergence has to be collected.

References

[1] E. Grosicki and H. E. Abed, “ICDAR 2009 handwriting recognition competition,” in *10th ICDAR*, Barcelona, Spain, 2009.

[2] V. Märgner and H. E. Abed, “ICDAR 2009 — Arabic handwriting recognition competition,” in *10th ICDAR*, Barcelona, Spain, 2009.

[3] 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.

[4] A. Graves, “Supervised sequence labelling with recurrent neural networks,” Dissertation, Technische Universität München, 2008.

[5] L. R. Rabiner, “A tutorial on hidden Markov models and selected applications in speech recognition,” *Proceedings of the IEEE*, vol. 77, no. 2, pp. 257–285, Feb. 1989.

[6] A. Senior and T. Robinson, “Forward-backward retraining of recurrent neural networks,” in *in Advances in Neural Information Processing Systems 8*. MIT Press, 1996, pp. 743–749.