# Local Feature based Online Mode Detection with Recurrent Neural Networks

Sebastian Otte, Dirk Krechel
*University of Applied Sciences Wiesbaden*
*Wiesbaden, Germany*
*Email: {otte,krechel}@ecmlab.de*

Marcus Liwicki, Andreas Dengel
*German Research Center for AI (DFKI)*
*Kaiserslautern, Germany*
*Email: firstname.lastname@dfki.de*

***Keywords*-Neural Networks; RNN; Recurrent Neural Networks; LSTM; Long Short-Term Memory; Sequence Classification; Sequence Learning; Mode Detection; Gesture Recognition; Local Features**

***Abstract*—In this paper we propose a novel approach for online mode detection, where the task is to classify ink traces into several categories. In contrast to previous approaches working on global features, we introduce a system completely relying on local features. For classification, standard recurrent neural networks (RNNs) and the recently introduced long short-term memory (LSTM) networks are used. Experiments are performed on the publicly available IAMonDo-database which serves as a benchmark data set for several researches. In the experiments we investigate several RNN structures and classification sub-tasks of different complexities. The final recognition rate on the complete test set is 98.47 % in average, which is significantly higher than the 97 % achieved with an MCS in previous work. Further interesting results on different subsets are also reported in this paper.**

## I. Introduction

The task of drawing mode detection is to detect the mode of online handwritten strokes. For example, a mode detection system should be able to determine whether a user is producing deictic gestures (e.g. to mark an object on a map or to specify a route), handwritten text, or iconic object drawings (people, cars, etc.) [1]. If a reliable mode detection system is at hand the separation of text and non-text items can be used as a preprocessing step before either handwriting recognition or symbol recognition is applied.

With the growing amount of tablet devices available on the market, there is an increasing need for real-time applications where the user gets the recognition feedback immediately. Using real-time mode detection improves the user experience on such devices, because instead of having the user to switch manually between handwriting recognition, shape detection, and gesture recognition, the mode-detection system would be able to guess the user's intention based on the strokes themselves. The main motivation of this paper is pen-enabled multi-touch interfaces [2], especially the Touch & Write [3], which inherently distinguishes between touching and pen and touch interaction.

Mode detection can be seen as a special case of text/graphics segmentation for online documents [4], [5]. Instead of analyzing the online document as a whole, the handwritten strokes are analyzed shortly after putting them on the surface. This can be considered as a more difficult task because of several reasons. First, there is no information about the context of the other handwritten strokes, especially those entered in the future. Second, drawing modes typically increase the amount of classes to be distinguished from, i.e., besides the classes of text and graphics also gestures are possible. Finally, the result of drawing mode detection should be available in real-time, i.e., the processing time should be only a few milliseconds.

Recently [6], we introduced a multiple classifier system (MCS) for mode detection which improved state-of-the-art systems [1] by using better features, classifiers, and combinations of all. While the system outperformed previous work of [4] and [7], it has a critical drawback. Since a large amount of more or less complicated features have to be extracted and many classifiers are involved in the recognition phase, the processing time becomes too long for practical use.

In this paper we propose an orthogonal approach which extracts local features for each point and finally applies RNNs for recognition. This approach has several advantages over the global feature extraction and classification. First, quite simple features can be extracted which reduces the processing time. Second, the system can be applied on sequences with arbitrary length, i.e., no minimal or maximal length is required. Finally, RNNs have been successfully applied for several sequence learning tasks and therefore a good performance can be expected.

## II. Method

The first part of this section briefly discusses the local features used in this paper and how they are extracted from a stream of input vectors. The second part describes the two recurrent network architectures proposed in this work to be used for online mode detection.

### A. Feature Acquisition

A sample is given as a sequence of $n$ points $\mathbf{p}_i \in \mathbb{R}^2$ which is further partitioned into subsequences (strokes). A new stroke starts at each point where the pen-tip touches the surface and ends at the last point before the pen-tip is removed from the surface. Our approach is to acquire local

978-0-7695-4774-9/12 $26.00 © 2012 IEEE
DOI 10.1109/ICFHR.2012.229

Table I
Local online features

| No. | Description | Note |
|---|---|---|
| 1 | $sin(\alpha_i)$ | Sinus of the angle between the current and the last line segment |
| 2 | $cos(\alpha_i)$ | Cosinus of the angle between the current and the last line segment |
| 3 | $d_i$ | Summed lengths of the current two line segments $\|\mathbf{l}_i\|+\|\mathbf{l}_{i+1}\|$, normalized with Eq. (1) |
| 4 | $b_i$ | Pen-up/pen-down: 1.0 if one of the current two line segments bridges between two strokes, 0.0 otherwise |
| 5 | $s_i$ | Stroke index, normalized with Eq. (1) |

features only by streaming the points one by one in forward direction without any complex or global preprocessing. This makes the system applicable in real-time scenarios. Fig. 1

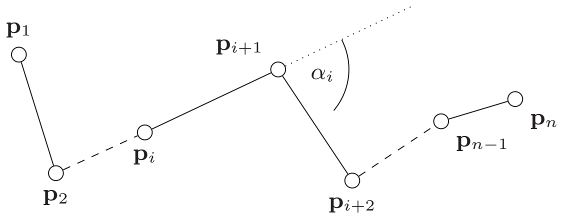


Figure 1. Local features of a point sequence

illustrates the acquisition of some features calculated in a local vicinity. The complete set of features is given and explained in Tab. I. In some cases, even if the range of input values is not standardized, it makes sense to normalize those inputs when dealing with neural networks. This paper follows [8] for normalization issues, i.e., the *mean* $m_k$ and the *standard deviation* $\sigma_k$ are used to compute the net input $\hat{v_k}$ for a given feature value $v_k$:

$$\hat{v_k} = \frac{v_k - m_k}{\sigma_k}. \tag{1}$$

For our experiments we compute means and standard deviations over all training samples. These values are stored and used for normalizing both training and test samples.

Note that feature 1 and 2, in literature generally known as *curvature*, give a relative and fully rotation invariant direction deviation and feature 3 represents the writing speed. In combination these features may provide the extraction of frequency information, which we assume are mainly relevant for the local features based sequence classification. Additionally feature 4 and 5 convey more structural information.

## B. Recurrent Neural Networks

While feedforward neural networks can only map between vectors in a static way, recurrent neural networks (RNNs) are able to map between sequences through dynamic information flow. They can generalize by observing data sequences even if the data is partially inaccurate or noisy. Due to this feature, RNNs seem to be a suitable choice for our task of sequence classification.

Our standard architecture (Fig. 2) is a two-layered RNN with a fully connected recurrent hidden layer as proposed in [8]. The input layer size depends on the specific number of

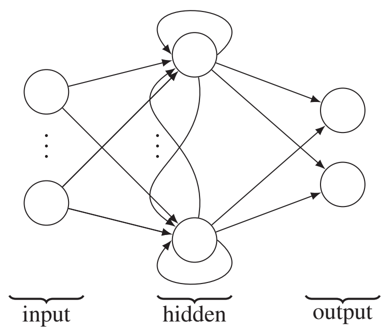


Figure 2. Recurrent architecture

used features. The hidden layer consists of $k$ units using the *hyperbolic tangent* ($\tanh$) for activation. The output layer consists of two $\tanh$-units. In the remainder of this paper, we refer such a network as RNN(k).

Training RNNs is done by *backpropagation through time* (BPTT) [9] with momentum term. Thereby, the gradients are computed over the complete input sequences. A frame-wise computation could be applicable as well but this is left for future work.

The second architecture used in this paper are *long short-term memory* networks (LSTMs) [10]. LSTMs overcome the main problem of traditional RNNs where the error vanishes dramatically over time while training. By simulating kind of differentiable memory cells, LSTMs are able to "trap" the error inside and allow to learn long time-lag problems with gradient descent [11]. We use LSTMs in their extended version with forget gates [12] and peephole connections [13]. The gates are activated with the standard *sigmoid function* $(1+e^{-x})^{-1}$ and cell inputs and outputs are squashed using $\tanh$.

The LSTM architecture is the same as for simple RNNs, however, the $k$ hidden units are replaced by $k$ LSTM blocks (see [8] for details). Analogously we denote the used architecture as LSTM(k). Similarly to RNNs, our LSTMs are also trained with BPTT using the gradients specified in [14].

The classification works as follows: First, a sequence is presented to the network. Second, the output $\mathbf{y} \in \mathbb{R}^m$ of the network after sequence processing is computed. Note that $m$ is also the number of classes. Finally, we select the class using the function $c : \mathbb{R}^m \rightarrow \{1, \dots, m\}$:

$$c(\mathbf{y}) = \min\left(\underset{1 \leq i \leq m}{\arg\max}(y_i)\right). \tag{2}$$

## III. Experiments and Results

All experiments in this section are performed on the IAMonDo database [7] containing $1,000$ online handwritten documents acquired from about 200 persons. Typical contents are text, drawings, diagrams, formulas, tables, lists, and markings. The traces contained by this database are widely unconstrained, which provides us to verify the robustness and performance of our approach in a real-world scenario.

In [7] a benchmark experiment for the task of distinguishing text and graphics has been performed. There the method of [4] achieved a performance of $91.3\,\%$ and the offline method proposed by [7] achieved $94.4\,\%$.[1] A previously introduced MCS performed with $97\,\%$ accuracy [6].

For training we used online BPTT with at most 200 epochs. However, mostly only 20-50 epochs were needed on average through early stopping. The learning rate ranges from $10^{-5}$ to $10^{-3}$ and the momentum rate is $0.9$. All results are achieved using an own Java implementation of RNNs and LSTMs.[2]

The aim of our experiments is to investigate the behavior of the different RNN structures on sub-tasks with different complexity. Therefore, we preform experiments on the following three setups:

1) We trained and evaluated on selected subsets (Exp. 1-6) of the database, which are *tables*, *rest* and *diagrams*. Note that tables usually just contain straight lines as drawings and are therefore *easy-to-learn*, while diagrams have much more variations making them *hard-to-learn* (see the examples in Fig. 5).
2) We trained on particular datasets and evaluated on the complete database (Exp. 7-8).
3) We trained and evaluated on the entire database (Exp. 9-10).

Note that the third setup is the setup which was used in all reference experiments in the previous work.

Each experiment has been repeated 10 times to ensure stable results independent from beneficial random-weight-initialization. The results for each architecture with optimal $k$ are listed in Tab. II.

Let us consider Exp. 1-4. The subsets tables and rest can be learned optimal with RNNs as well with LSTMs. Thus, only a short-term context is necessary to distinguish words and simple shapes as contained by these sets. The impact of increasing the network complexity is only marginal but verifiable existent as can be seen in Fig. 3, which depicts the results for different networks particular for the subset tables. An interesting result is that already a very simple RNN(2) performs quite well here.

[1] Note, however, that the offline results are not directly comparable to the online results, because another evaluation method is applied. [7]

[2] The toolkit (JANNLAB, see http://www.ecmlab.de/jannlab) has been verified on standard sequence labeling tasks proposed in the literature.

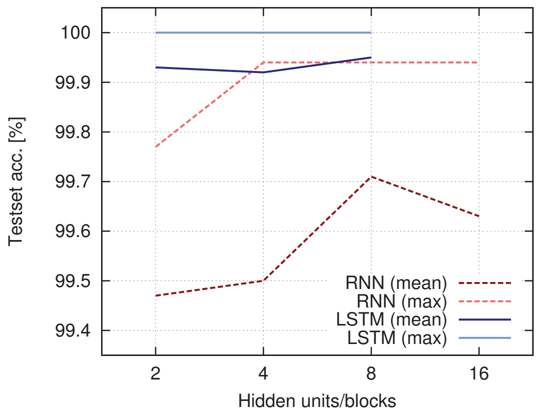


Figure 3. Tables experiments

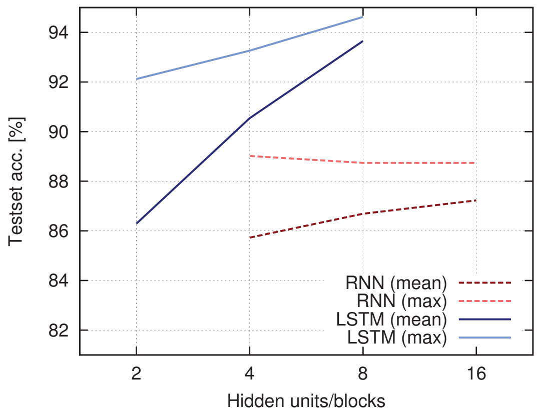


Figure 4. Diagrams experiments

The strength of LSTMs becomes important when the problem becomes harder. This follows from the results of Exp. 5-6. While the best RNN achieves a mean accuracy of $87.23\,\%$ the best LSTM achieves $93.65\,\%$. On comparing only the shapes-accuracy this issue appears still more clearly.

Another interesting observation was that for the hard diagrams set, the hidden layer size plays a decisive role for LSTMs but not for RNNs (see Fig. 4). Both architectures become more stable with a larger hidden layer. However, the maximum-performance of RNNs, which can only learn short-term dependencies, do not benefit from increasing the number of hidden units, whereas LSTMs perform dramatically better with more hidden blocks. It seems possible that the results can further be improved with $k > 8$ hidden blocks.[3]

Noteworthy are also Exp. 7-8: Only trained with the easy subset tables, both architectures yield good results on the entire database, especially the LSTM(4) with $97.13\,\%$ (mean) and $97.99\,\%$ (max.).

[3] This has not been tested yet, because of the exploding training time.

Table II
EXPERIMENTAL RESULTS

| Exp. | Network | Sample set (train/test) | Acc. [%] (train set) | Acc. [%] (test set) | | |
|---|---|---|---|---|---|---|
| | | | | All | Words | Shapes |
| 1 | RNN(8) | tables/tables | 99.50 | $99.71 \pm 0.27$ (99.94) | 99.91 (100.00) | 98.71 (99.67) |
| 2 | LSTM(8) | tables/tables | 99.94 | $99.95 \pm 0.05$ (100.00) | 99.98 (100.00) | 99.83 (100.00) |
| 3 | RNN(16) | rest/rest | 99.24 | $98.90 \pm 0.38$ (99.20) | 99.73 (100.00) | 90.76 (95.65) |
| 4 | LSTM(8) | rest/rest | 99.24 | $99.06 \pm 0.40$ (99.50) | 99.84 (100.00) | 91.41 (94.57) |
| 5 | RNN(16) | dia./dia. | 87.93 | $87.23 \pm 1.62$ (88.74) | 97.01 (100.00) | 61.88 (74.61) |
| 6 | LSTM(8)[ii] | dia./dia. | 96.01 | $93.65 \pm 0.73$ (94.62) | 97.28 (98.72) | 84.24 (88.28) |
| 7 | RNN(8) | tables/all | 99.50 | $95.83 \pm 0.64$ (96.68) | 98.96 (99.64) | 53.18 (59.28) |
| 8 | LSTM(4) | tables/all | 99.90 | $97.13 \pm 0.46$ (97.99) | 99.63 (99.77) | 63.02 (73.76) |
| 9 | RNN(16) | all/all | 96.49 | $96.87 \pm 0.55$ (97.64) | 98.76 (99.41) | 71.11 (78.04) |
| 10 | LSTM(8)[i] | all/all | 98.44 | $98.47 \pm 0.25$ (98.77) | 99.70 (99.84) | 81.67 (86.20) |

[i]learning rate $10^{-4}$, [ii]learning rate $10^{-3}$

Table III
TRAINING TIME

| Training set | Network | Mean time [s] |
|---|---|---|
| tables | RNN(16) | 159 |
| tables | LSTM(8) | 2412 |
| rest | RNN(16) | 319 |
| rest | LSTM(8) | 2329 |
| diagrams | RNN(16) | 558 |
| diagrams | LSTM(8) | 3760 |
| all | RNN(16) | 6208 |
| all | LSTM(8) | 24851 |

Finally, the results of Exp. 9-10 demonstrate the overall performance of our approach. Trained with the entire database an LSTM(8) achieves $98.47\,\%$ accuracy (mean) and even $98.77\,\%$ (max.). This is significantly better than the results of the MCS mentioned above.

Through all experiments LSTMs behave more stable than RNNs. This can be realized on comparing the standard deviations in Tab. II.

To comprehend the classification results, let us look at Fig. 5, which shows different classification examples (correct and wrong classified by an LSTM(8)) of the easy table subset and the hard diagram subset as well. Simple shapes (a) and typical words (c) and also more complex shapes mostly consisting of fast draw lines (b) are classified correct, while single fast drawn letters/digits and shapes with a lot of writing-like local variantions are classified wrong.

An RNN(16) with 368 weights processes[4] an input vector in only 0.05 ms. Respecting a mean of 63 vectors per sample results in on average 2.8 ms per sample. An LSTM(8) with 456 weights needs on average 16.8 ms per sample. This obviously fulfills any kind of realtime requirements.

Anyhow, during training the constant overhead from RNNs to LSTMs is clearly appreciable (see Tab. III). On training the tables set a RNN(16) is about 15 times faster than a LSTM(8). But we can observe, that this factor decreases on increasing the problem-complexity. On training the entire database this factor is only about 4.

[4]Java HotSpot(TM) 64-Bit Server VM (build 11.3-b02, mixed mode) on an Intel Core2Duo E6850 @ 3.0 GHz.

## IV. Conclusion and Future Work

We have presented a local feature based approach for online mode detection using RNNs and LSTMs. It was shown that on this problem the simple RNNs and the more complex LSTMs outperform other state-of-the-art classifiers including SVMs and MCS.

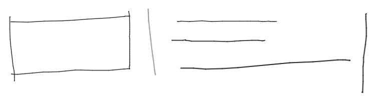
More specifically, on the IAMonDo-database RNNs outperform all previously reported single classifier systems and even achieve a slightly better performance than an MCS. The final maximal accuracy is about 97.6 %. Furthermore, LSTMs achieve an accuracy of close to 99 %. On particular subsets some LSTMs could even perform a perfect recognition. However, difficult subsets of the database still leave room for improvements.

One future task is to evaluate the behavior of bidirectional RNNs [15] or bidirectional LSTMs [14] for the presented problem. It seems possible that providing both past and future context can enhance the accuracy of mode detection. Similar investigations in handwriting recognition [16] are a promising support for this hypothesis.

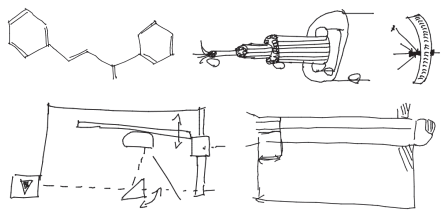
A further improvement could be the combination of a possibly bidirectional RNN (or LSTM) and a multilayer perceptron (MLP) where the RNN still performs on local feature sequences (maybe using more features) and the MLP uses global features. This can be extended to MCS as done in [6]. However, this would also result in longer time needed for classification. Thus, for real-time applications, an incremental recognition approach can be used, i.e., first the RNN outputs an initial guess and second, if more time is available, a better estimation can be made using a combination of several classifiers.

Nonetheless, extending the system also increases its complexity, which becomes apparent when comparing LSTMs and RNNs training time. In our framework an RNN can be trained 4-15 times faster then a comparably LSTM.

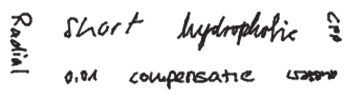
Another promising aspect of simple online computing RNNs, as we used here, is that they consume only minimal hardware resources, which allows a very efficient high



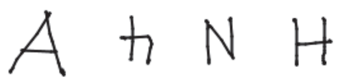
(a) Table (shapes) - (correct classified)



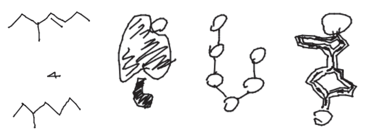
(b) Diagram (shapes) - (correct classified)



(c) Diagram (words) - (correct classified)



(d) Diagram (words) - (wrong classified)



(e) Diagram (shapes) - (wrong classified)

Figure 5. Different classification examples of the subsets table and diagram.

performance mode detection system directly implemented onchip in mobile devices.

Among other things, our results lead us to assume that RNNs and especially LSTMs work great also for general online gesture classification which we will evaluate in future work.

## REFERENCES

[1] D. Willems and L. Vuurpijl, “A bayesian network approach to mode detection for interactive maps,” in *Proceedings of the Ninth International Conference on Document Analysis and Recognition - Volume 02*. Washington, DC, USA: IEEE Computer Society, 2007, pp. 869–873.

[2] NUI Group Authors, *Multi-Touch Technologies*. NUI Group, 2009. [Online]. Available: http://nuicode.com/attachments/download/115/Multi-Touch_Technologies_v1.01.pdf

[3] M. Liwicki, O. Rostanin, S. M. El-Neklawy, and A. Dengel, “Touch & write: a multi-touch table with pen-input,” in *9th Int. Workshop on Document Analysis Systems*, 2010, pp. 479–484.

[4] A. Jain, A. Namboodiri, and J. Subrahmonia, “Structure in on-line documents,” in *Proceedings of the Sixth International Conference on Document Analysis and Recognition*. Washington, DC, USA: IEEE Computer Society, 2001, pp. 844–848.

[5] C. M. Bishop, M. Svensen, and G. E. Hinton, “Distinguishing text from graphics in on-line handwritten ink,” in *Proceedings of the Ninth International Workshop on Frontiers in Handwriting Recognition*, ser. IWFHR ’04. Washington, DC, USA: IEEE Computer Society, 2004, pp. 142–147.

[6] M. Weber, M. Liwicki, Y. T. Schelske, C. Schoelzel, F. Strauß, and A. Dengel, “MCS for online mode detection: Evaluation on Pen-Enabled multi-touch interfaces,” in *2011 International Conference on Document Analysis and Recognition (ICDAR)*. IEEE, Sep. 2011, pp. 957–961.

[7] E. Indermühle, M. Liwicki, and H. Bunke, “IAMonDo-database: an online handwritten document database with non-uniform contents,” in *9th Int. Workshop on Document Analysis Systems*, 2010, pp. 97–104.

[8] A. Graves, B. Bruegge, J. Schmidhuber, and S. Kramer, “Supervised sequence labelling with recurrent neural networks,” Ph.D. dissertation, Technische Universität München, 2008.

[9] R. J. Williams and D. Zipser, “Gradient-Based learning algorithms for recurrent networks and their computational complexity,” 1995.

[10] S. Hochreiter and J. Schmidhuber, “Long Short-Term memory,” *Neural Comput.*, vol. 9, no. 8, p. 1735–1780, Nov. 1997.

[11] ——, “LSTM can solve hard long time lag problems,” *Advances in Neural Information Processing Systems 9*, pp. 473—479, 1997.

[12] F. A. Gers, J. Schmidhuber, and F. Cummins, “Learning to forget: Continual prediction with LSTM,” *NEURAL COMPUTATION*, vol. 12, pp. 2451—2471, 1999.

[13] F. A. Gers, N. N. Schraudolph, and J. Schmidhuber, “Learning precise timing with LSTM recurrent networks,” *Journal of Machine Learning Research*, vol. 3, pp. 115—143, 2002.

[14] A. Graves and J. Schmidhuber, “Framewise phoneme classification with bidirectional LSTM and other neural network architectures,” *Neural Networks: The Official Journal of the International Neural Network Society*, vol. 18, no. 5-6, pp. 602–610, Jul. 2005, PMID: 16112549.

[15] M. Schuster and K. K. Paliwal, “Bidirectional recurrent neural networks,” *IEEE Transactions on Signal Processing*, vol. 45, no. 11, pp. 2673–2681, Nov. 1997.

[16] 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*, 2007.