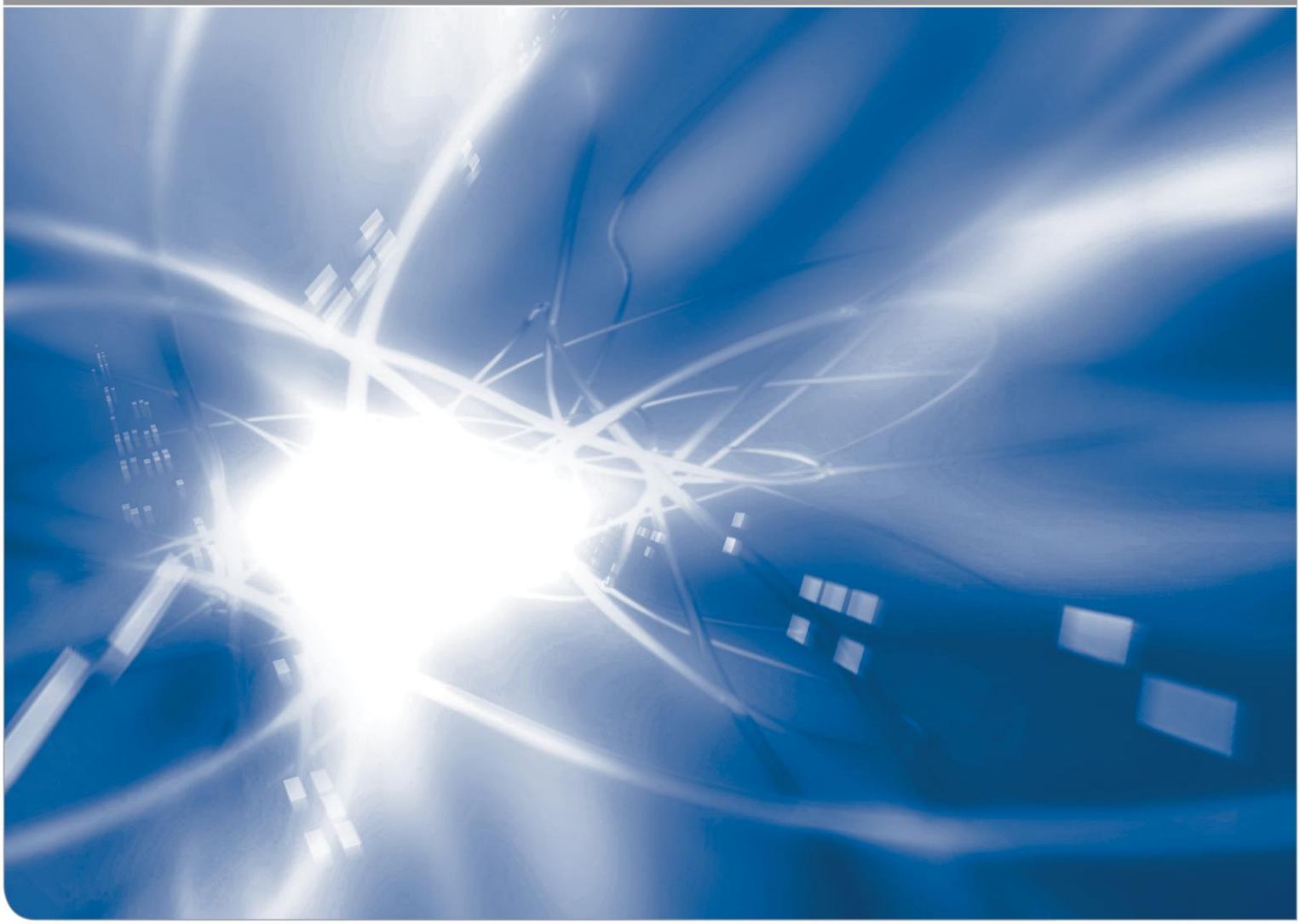
Karlsruhe Institute of Technology

On-line Recognition of Handwritten Mathematical Symbols

Open Access at KIT

by Martin Thoma[1], Kevin Kilgour[1], Sebastian Stüker[1], Alexander Waibel[1]

KIT SCIENTIFIC WORKING PAPERS 32



[1] Institute for Anthropomatics and Robotics

Institut für Anthropomatik
Humanoids and Intelligence Systems Lab
Adenauerring 2
76131 Karlsruhe
https://his.anthropomatik.kit.edu/

Impressum

Karlsruher Institut für Technologie (KIT)
www.kit.edu

2015

ISSN: 2194-1629

On-line Recognition of Handwritten Mathematical Symbols

Martin Thoma, Kevin Kilgour, Sebastian Stüker and Alexander Waibel

***Abstract*—The automatic recognition of single handwritten symbols has three main applications: Supporting users who know how a symbol looks like, but not what its name is, providing the necessary commands for professional publishing, or as a building block for formula recognition.**

This paper presents a system which uses the pen trajectory to classify handwritten symbols. Five preprocessing steps, one data multiplication algorithm, five features and five variants for multilayer Perceptron training were evaluated using $166{,}898$ recordings. Those recordings were made publicly available. The evaluation results of these 21 experiments were used to create an optimized recognizer which has a top-1 error of less than $17.5\,\%$ and a top-3 error of $4.0\,\%$. This is a relative improvement of $18.5\,\%$ for the top-1 error and $29.7\,\%$ for the top-3 error compared to the baseline system. This improvement was achieved by supervised layer-wise pretraining and adding new features. The improved classifier can be used via write-math.com.

I. Introduction

On-line recognition makes use of the pen trajectory. One possible representation of the data is given as groups of sequences of tuples $(x, y, t) \in \mathbb{R}^3$, where each group represents a stroke, (x, y) is the position of the pen on a canvas and t is the time.

[Kir10] describes a system called Detexify which uses time warping to classify on-line handwritten symbols and reports a top-3 error of less than $10\,\%$ for a set of 100 symbols. He did also recently publish his data on https://github.com/kirel/detexify-data, which was collected by a crowdsourcing approach via http://detexify.kirelabs.org. Those recordings as well as some recordings which were collected by a similar approach via http://write-math.com were merged in a single data set, the labels were semi-automatically checked for correctness and used to train and evaluated different classifiers. A more detailed description of all used software, data and experiments is given in [Tho14].

In this paper we present a baseline system for the classification of on-line handwriting into 369 classes of which some are very similar. An optimized classifier was developed which has a $29.7\,\%$ relative improvement of the top-3 error. This was achieved by using better features and supervised layer-wise pretraining (SLP). The absolute improvements compared to the baseline of those changes will also be shown.

In the following, we will give a general overview of the system design, give information about the used data and implementation, describe the algorithms we used to classify the data, report results of our experiments and present the optimized recognizer we created.

II. General System Design

The following steps are used for symbol classification:

1) **Preprocessing**: Recorded data is never perfect. Devices have errors and people make mistakes while using the devices. To tackle these problems there are preprocessing algorithms to clean the data. The preprocessing algorithms can also remove unnecessary variations of the data that do not help in the classification process, but hide what is important. Having slightly different sizes of the same symbol is an example of such a variation. Four preprocessing algorithms that clean or normalize recordings are explained in section IV-A.
2) **Data multiplication**: Learning systems need lots of data to learn internal parameters. If there is not enough data available, domain knowledge can be considered to create new artificial data from the original data. In the domain of on-line handwriting recognition, data can be multiplied by adding rotated variants.
3) **Feature extraction**: A feature is high-level information derived from the raw data after preprocessing. Some systems like Detexify take the result of the preprocessing step, but many compute new features. Those features can be designed by a human engineer or learned. Non-raw data features have the advantage that less training data is needed since the developer uses knowledge about handwriting to compute highly discriminative features. Various features are explained in section IV-B.

After these steps, it is a classification task for which the classifier has to learn internal parameters before it can classify new recordings.We classified recordings by computing constant-sized feature vectors and using multilayer perceptrons (MLPs). There are many ways to adjust MLPs (number of neurons and layers, activation functions) and their training (learning rate, momentum, error function). Some of them are described in section IV-C and the evaluation results are presented in section V.

III. Data and Implementation

We used 369 symbol classes with a total of $166{,}898$ labeled recordings. Each class has at least 50 labeled recordings, but over 200 symbols have more than 200 labeled recordings and over 100 symbols have more than 500 labeled recordings. The data was collected by two crowd-sourcing projects (Detexify and write-math.com) where users wrote symbols, were then given a list ordered by an early classification system and clicked on the symbol they wrote.

The data of Detexify and write-math.com was combined, filtered semi-automatically and can be downloaded via write-math.com/data as a compressed tar archive of CSV files.

All of the following preprocessing and feature computation algorithms were implemented and are publicly available as open-source software in the Python package `hwrt`.

IV. Algorithms

A. Preprocessing

Preprocessing in symbol recognition is done to improve the quality and expressive power of the data. It makes follow-up tasks like feature extraction and classification easier, more effective or faster. It does so by resolving errors in the input data, reducing duplicate information and removing irrelevant information.

Preprocessing algorithms fall into two groups: Normalization and noise reduction algorithms.

A very important normalization algorithm in single-symbol recognition is *scale-and-shift* [Tho14]. It scales the recording so that its bounding box fits into a unit square. As the aspect ratio of a recording is almost never 1:1, only one dimension will fit exactly in the unit square. For this paper, it was chosen to shift the recording in the direction of its bigger dimension into the $[0, 1] \times [0, 1]$ unit

square. After that, the recording is shifted in direction of its smaller dimension such that its bounding box is centered around zero.

Another normalization preprocessing algorithm is resampling [GAC+91], [JMRW01]. As the data points on the pen trajectory are generated asynchronously and with different time-resolutions depending on the used hardware and software, it is desirable to resample the recordings to have points spread equally in time for every recording. This was done by linear interpolation of the (x, t) and (y, t) sequences and getting a fixed number of equally spaced points per stroke.

Stroke connection is a noise reduction algorithm which is mentioned in [TSW90]. It happens sometimes that the hardware detects that the user lifted the pen where the user certainly didn't do so. This can be detected by measuring the Euclidean distance between the end of one stroke and the beginning of the next stroke. If this distance is below a threshold, then the strokes are connected.

Due to a limited resolution of the recording device and due to erratic handwriting, the pen trajectory might not be smooth. One way to smooth is calculating a weighted average and replacing points by the weighted average of their coordinate and their neighbors coordinates. Another way to do smoothing is to reduce the number of points with the Douglas-Peucker algorithm to the points that are more relevant for the overall shape of a stroke and then interpolate the stroke between those points. The Douglas-Peucker stroke simplification algorithm is usually used in cartography to simplify the shape of roads. It works recursively to find a subset of points of a stroke that is simpler and still similar to the original shape. The algorithm adds the first and the last point p_1 and p_n of a stroke to the simplified set of points S. Then it searches the point p_i in between that has maximum distance from the line p_1p_n. If this distance is above a threshold ε, the point p_i is added to S. Then the algorithm gets applied to p_1p_i and p_ip_n recursively. It is described as "Algorithm 1" in [VW90].

B. Features

Features can be *global*, that means calculated for the complete recording or complete strokes. Other features are calculated for single points on the pen trajectory and are called *local*.

Global features are the *number of strokes* in a recording, the *aspect ratio* of a recordings bounding box or the *ink* being used for a recording. The ink feature gets calculated by measuring the length of all strokes combined. The re-curvature, which was introduced in [HK06], is defined as

$$\text{re-curvature}(stroke) := \frac{\text{height}(stroke)}{\text{length}(stroke)}$$

and a stroke-global feature.

The simplest local feature is the coordinate of the point itself. Speed, curvature and a local small-resolution bitmap around the point, which was introduced by Manke, Finke and Waibel in [MFW95], are other local features.

C. Multilayer Perceptrons

MLPs are explained in detail in [Mit97]. They can have different numbers of hidden layers, the number of neurons per layer and the activation functions can be varied. The learning algorithm is parameterized by the learning rate $\eta \in (0, \infty)$, the momentum $\alpha \in [0, \infty)$ and the number of epochs.

The topology of MLPs will be denoted in the following by separating the number of neurons per layer with colons. For example, the notation $160{:}500{:}500{:}500{:}369$ means that the input layer gets 160 features, there are three hidden layers with 500 neurons per layer and one output layer with 369 neurons.

MLPs training can be executed in various different ways, for example with supervised layer-wise pretraining (SLP). In case of a MLP with the topology $160{:}500{:}500{:}500{:}369$, SLP works as follows: At first a MLP with one hidden layer ($160{:}500{:}369$) is trained. Then the output layer is discarded, a new hidden layer and a new output layer is added and it is trained again, resulting in a $160{:}500{:}500{:}369$ MLP. The output layer is discarded again, a new hidden layer is added and a new output layer is added and the training is executed again.

Denoising auto-encoders are another way of pretraining. An *auto-encoder* is a neural network that is trained to restore its input. This means the number of input neurons is equal to the number of output neurons. The weights define an *encoding* of the input that allows restoring the input. As the neural network finds the encoding by itself, it is called auto-encoder. If the hidden layer is smaller than the input layer, it can be used for dimensionality reduction [Hin89]. If only one hidden layer with linear activation functions is used, then the hidden layer contains the principal components after training [DHS01].

Denoising auto-encoders are a variant introduced in [VLBM08] that is more robust to partial corruption of the input features. It is trained to get robust by adding noise to the input features.

There are multiple ways how noise can be added. Gaussian noise and randomly masking elements with zero are two possibilities. [Deea] describes how such a denoising auto-encoder with masking noise can be implemented. The corruption $\varkappa \in [0, 1)$ is the probability of a feature being masked.

V. Optimization of System Design

In order to evaluate the effect of different preprocessing algorithms, features and adjustments in the MLP training and topology, the following baseline system was used:

Scale the recording to fit into a unit square while keeping the aspect ratio, shift it as described in section IV-A, resample it with linear interpolation to get 20 points per stroke, spaced evenly in time. Take the first 4 strokes with 20 points per stroke and 2 coordinates per point as features, resulting in 160 features which is equal to the number of input neurons. If a recording has less than 4 strokes, the remaining features were filled with zeroes.

All experiments were evaluated with four baseline systems $B_{hl=i}$, $i \in \{ 1, 2, 3, 4 \}$, where i is the number of hidden layers as different topologies could have a severe influence on the effect of new features or preprocessing steps. Each hidden layer in all evaluated systems has 500 neurons.

Each MLP was trained with a learning rate of $\eta = 0.1$ and a momentum of $\alpha = 0.1$. The activation function of every neuron in a hidden layer is the sigmoid function. The neurons in the output layer use the softmax function. For every experiment, exactly one part of the baseline systems was changed.

A. Random Weight Initialization

The neural networks in all experiments got initialized with a small random weight

$$w_{i,j} \sim U(-4 \cdot \sqrt{\frac{6}{n_l + n_{l+1}}}, 4 \cdot \sqrt{\frac{6}{n_l + n_{l+1}}})$$

where $w_{i,j}$ is the weight between the neurons i and j, l is the layer of neuron i, and n_i is the number of neurons in layer i. This random initialization was suggested on [deeb] and is done to break symmetry.

This can lead to different error rates for the same systems just because the initialization was different.

In order to get an impression of the magnitude of the influence on the different topologies and error rates the baseline models were trained 5 times with random initializations. Table I shows a summary of the results. The more hidden layers are used, the more do the results vary between different random weight initializations.

System	Classification error					
	Top-1			Top-3		
	Min	Max	Mean	Min	Max	Mean
$B_{hl=1}$	23.1 %	23.4 %	23.2 %	6.7 %	6.8 %	6.7 %
$B_{hl=2}$	21.4 %	21.8 %	21.6 %	5.7 %	5.8 %	5.7 %
$B_{hl=3}$	21.5 %	22.3 %	21.9 %	5.5 %	5.8 %	5.7 %
$B_{hl=4}$	23.2 %	24.8 %	23.9 %	6.0 %	6.4 %	6.2 %

TABLE I: The systems $B_{hl=1}$ – $B_{hl=4}$ were randomly initialized, trained and evaluated 5 times to estimate the influence of random weight initialization.

B. Stroke connection

In order to solve the problem of interrupted strokes, pairs of strokes can be connected with stroke connection algorithm. The idea is that for a pair of consecutively drawn strokes s_i, s_{i+1} the last point s_i is close to the first point of s_{i+1} if a stroke was accidentally split into two strokes.

59 % of all stroke pair distances in the collected data are between 30 px and 150 px. Hence the stroke connection algorithm was evaluated with 5 px, 10 px and 20 px. All models top-3 error improved with a threshold of $\theta = 10$ px by at least 0.2 percentage points, except $B_{hl=4}$ which did not notably improve.

C. Douglas-Peucker Smoothing

The Douglas-Peucker algorithm was applied with a threshold of $\varepsilon = 0.05$, $\varepsilon = 0.1$ and $\varepsilon = 0.2$ after scaling and shifting, but before resampling. The interpolation in the resampling step was done linearly and with cubic splines in two experiments. The recording was scaled and shifted again after the interpolation because the bounding box might have changed.

The result of the application of the Douglas-Peucker smoothing with $\varepsilon > 0.05$ was a high rise of the top-1 and top-3 error for all models $B_{hl=i}$. This means that the simplification process removes some relevant information and does not—as it was expected—remove only noise. For $\varepsilon = 0.05$ with linear interpolation some models top-1 error improved, but the changes were small. It could be an effect of random weight initialization. However, cubic spline interpolation made all systems perform more than 1.7 percentage points worse for top-1 and top-3 error.

The lower the value of ε, the less does the recording change after this preprocessing step. As it was applied after scaling the recording such that the biggest dimension of the recording (width or height) is 1, a value of $\varepsilon = 0.05$ means that a point has to move at least 5 % of the biggest dimension.

System	Classification error			
	Top-1	Change	Top-3	Change
$B_{hl=1}$	23.2 %	-	6.7 %	-
$B_{hl=2,SLP}$	19.9 %	−1.7 %	4.7 %	−1.0 %
$B_{hl=3,SLP}$	19.4 %	−2.5 %	4.6 %	−1.1 %
$B_{hl=4,SLP}$	19.6 %	−4.3 %	4.6 %	−1.6 %

TABLE II: Systems with 1–4 hidden layers which used supervised layer-wise pretraining (SLP) compared to the mean of systems $B_{hl=1}$–$B_{hl=4}$ displayed in Table I which used pure gradient descent. The SLP systems clearly performed worse.

D. Global Features

Single global features were added one at a time to the baseline systems. Those features were re-curvature re-curvature$(stroke) = \frac{\text{height}(stroke)}{\text{length}(stroke)}$ as described in [HK06], the ink feature which is the summed length of all strokes, the stroke count, the aspect ratio and the stroke center points for the first four strokes. The stroke center point feature improved the system $B_{hl=1}$ by 0.3 percentage points for the top-3 error and system $B_{hl=3}$ for the top-1 error by 0.7 percentage points, but all other systems and error measures either got worse or did not improve much.

The other global features did improve the systems $B_{hl=1}$ – $B_{hl=3}$, but not $B_{hl=4}$. The highest improvement was achieved with the re-curvature feature. It improved the systems $B_{hl=1}$ – $B_{hl=4}$ by more than 0.6 percentage points top-1 error.

E. Data Multiplication

Data multiplication can be used to make the model invariant to transformations. However, this idea seems not to work well in the domain of on-line handwritten mathematical symbols. We tripled the data by adding a version that is rotated 3 degrees to the left and another one that is rotated 3 degrees to the right around the center of mass. This data multiplication made all classifiers for most error measures perform worse by more than 2 percentage points for the top-1 error.

The same experiment was executed by rotating by 6 degrees and in another experiment by 9 degrees, but those performed even worse.

Also multiplying the data by a factor of 5 by adding two 3-degree rotated variants and two 6-degree rotated variant made the classifier perform worse by more than 2 percentage points.

F. Pretraining

Pretraining is a technique used to improve the training of MLPs with multiple hidden layers.

Table II shows that SLP improves the classification performance by 1.6 percentage points for the top-1 error and 1.0 percentage points for the top-3 error. As one can see in Figure 1, this is not only the case because of the longer training as the test error is relatively stable after 1000 epochs of training. This was confirmed by an experiment where the baseline systems where trained for 10,000 epochs and did not perform notably different.

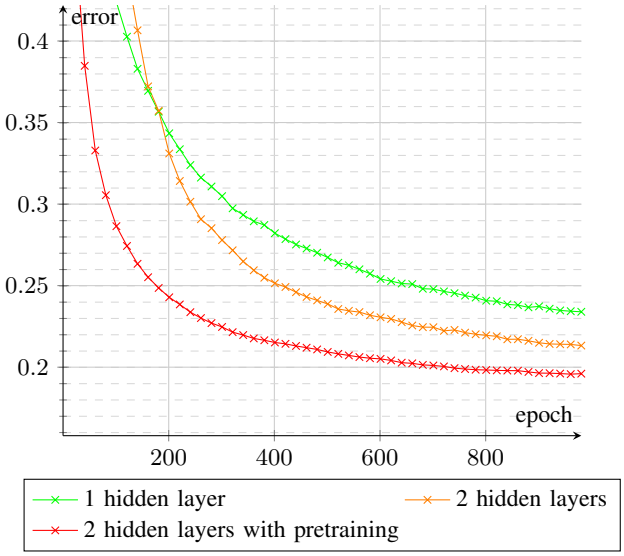


Fig. 1: Training- and test error by number of trained epochs for different topologies with supervised layer-wise pretraining (SLP). The plot shows that all pretrained systems performed much better than the systems without pretraining. All plotted systems did not improve with more epochs of training.

System	Classification error			
	Top-1	Change	Top-3	Change
$B_{hl=1,AEP}$	23.8 %	0.6 %	7.2 %	0.5 %
$B_{hl=2,AEP}$	22.8 %	1.2 %	6.4 %	0.7 %
$B_{hl=3,AEP}$	23.1 %	1.2 %	6.1 %	0.4 %
$B_{hl=4,AEP}$	25.6 %	1.7 %	7.0 %	0.8 %

TABLE III: Systems with denoising auto-encoder pretraining (AEP) compared to pure gradient descent. The auto-encoder pretraining (AEP) systems performed worse.

Pretraining with denoising auto-encoder lead to the much worse results listed in Table III. The first layer used a $\tanh$ activation function. Every layer was trained for 1000 epochs and the mean squared error (MSE) loss function. A learning-rate of $\eta = 0.001$, a corruption of $\varkappa = 0.3$ and a L_2 regularization of $\lambda = 10^{-4}$ were chosen. This pretraining setup made all systems with all error measures perform much worse.

VI. Summary

Four baseline recognition systems were adjusted in many experiments and their recognition capabilities were compared in order to build a recognition system that can recognize 396 mathematical symbols with low error rates as well as to evaluate which preprocessing steps and features help to improve the recognition rate.

All recognition systems were trained and evaluated with 166,898 recordings for 369 symbols. These recordings were collected by two crowdsourcing projects (Detexify and write-math.com) and created with various devices. While some recordings were created with standard touch devices such as tablets and smartphones, others were created with the mouse.

MLPs were used for the classification task. Four baseline systems with different numbers of hidden layers were used, as the number of hidden layer influences the capabilities and problems of MLPs.

All baseline systems used the same preprocessing queue. The recordings were scaled and shifted as described in IV-A, resampled with linear interpolation so that every stroke had exactly 20 points which are spread equidistant in time. The 80 (x, y) coordinates of the first 4 strokes were used to get exactly 160 input features for every recording. The baseline system $B_{hl=2}$ has a top-3 error of 5.7 %.

Adding two slightly rotated variants for each recording and hence tripling the training set made the systems $B_{hl=3}$ and $B_{hl=4}$ perform much worse, but improved the performance of the smaller systems.

The global features re-curvature, ink, stoke count and aspect ratio improved the systems $B_{hl=1}$–$B_{hl=3}$, whereas the stroke center point feature made $B_{hl=2}$ perform worse.

Denoising auto-encoders were evaluated as one way to use pretraining, but by this the error rate increased notably. However, supervised layer-wise pretraining improved the performance decidedly.

The stroke connection algorithm was added to the preprocessing steps of the baseline system as well as the re-curvature feature, the ink feature, the number of strokes and the aspect ratio. The training setup of the baseline system was changed to supervised layer-wise pretraining and the resulting model was trained with a lower learning rate again. This optimized recognizer $B'_{hl=2,c}$ had a top-3 error of 4.0 %. This means that the top-3 error dropped by over 1.7 percentage points in comparison to the baseline system $B_{hl=2}$.

A top-3 error of 4.0 % makes the system usable for symbol lookup. It could also be used as a starting point for the development of a multiple-symbol classifier.

The aim of this work was to develop a symbol recognition system which is easy to use, fast and has high recognition rates as well as evaluating ideas for single symbol classifiers. Some of those goals were reached. The recognition system $B'_{hl=2,c}$ evaluates new recordings in a fraction of a second and has acceptable recognition rates.

VII. Optimized Recognizer

All preprocessing steps and features that were useful were combined to create a recognizer that performs best.

All models were much better than everything that was tried before. The results of this experiment show that single-symbol recognition with 369 classes and usual touch devices and the mouse can be done with a top-1 error rate of 18.6 % and a top-3 error of 4.1 %. This was achieved by a MLP with a 167:500:500:369 topology.

It used the stroke connection algorithm to connect of which the ends were less than 10 px away, scaled each recording to a unit square and shifted as described in IV-A. After that, a linear resampling step was applied to the first 4 strokes to resample them to 20 points each. All other strokes were discarded.

The 167 features were

- the first 4 strokes with 20 points per stroke resulting in 160 features,
- the re-curvature for the first 4 strokes,
- the ink,
- the number of strokes and
- the aspect ratio of the bounding box

SLP was applied with 1000 epochs per layer, a learning rate of $\eta = 0.1$ and a momentum of $\alpha = 0.1$. After that, the complete model

was trained again for 1000 epochs with standard mini-batch gradient descent resulting in systems $B'_{hl=1,c}$ – $B'_{hl=4,c}$.

After the models $B_{hl=1,c}$ – $B_{hl=4,c}$ were trained the first 1000 epochs, they were trained again for 1000 epochs with a learning rate of $\eta = 0.05$. Table IV shows that this improved the classifiers again.

System	Classification error			
	Top-1	Change	Top-3	Change
$B_{hl=1,c}$	21.0 %	−2.2 %	5.2 %	−1.5 %
$B_{hl=2,c}$	18.3 %	−3.3 %	4.1 %	−1.6 %
$B_{hl=3,c}$	18.2 %	−3.7 %	4.1 %	−1.6 %
$B_{hl=4,c}$	18.6 %	−5.3 %	4.3 %	−1.9 %
$B'_{hl=1,c}$	19.3 %	−3.9 %	4.8 %	−1.9 %
$B'_{hl=2,c}$	17.5 %	−4.1 %	4.0 %	−1.7 %
$B'_{hl=3,c}$	17.7 %	−4.2 %	4.1 %	−1.6 %
$B'_{hl=4,c}$	17.8 %	−6.1 %	4.3 %	−1.9 %

TABLE IV: Error rates of the optimized recognizer systems. The systems $B'_{hl=i,c}$ were trained another 1000 epochs with a learning rate of $\eta = 0.05$.

VIII. Evaluation

The optimized classifier was evaluated on three publicly available data sets: `MfrDB_Symbols_v1.0` [SBPH12], CROHME 2011 [MVGK$^+$11], and CROHME 2012 [MVGK$^+$12].

`MfrDB_Symbols_v1.0` contains recordings for 105 symbols, but for 11 symbols less than 50 recordings were available. For this reason, the optimized classifier was evaluated on 94 of the 105 symbols.

The evaluation results are given in Table V.

Dataset	Symbols	Classification error	
		Top-1	Top-3
MfrDB	94	8.4 %	1.3 %
CROHME 2011	56	10.2 %	3.7 %
CROHME 2012	75	12.2 %	4.1 %

TABLE V: Error rates of the optimized recognizer systems. The systems output layer was adjusted to the number of symbols it should recognize and trained with the combined data from write-math and the training given by the datasets.

References

[Deea] "Denoising autoencoders (da)." [Online]. Available: http://deeplearning.net/tutorial/dA.html

[deeb] "Going from logistic regression to MLP." [Online]. Available: http://www.deeplearning.net/tutorial/mlp.html#going-from-logistic-regression-to-mlp

[DHS01] R. O. Duda, P. E. Hart, and D. G. Stork, *Pattern Classification*. John Wiley & Sons, 2001.

[GAC+91] I. Guyon, P. Albrecht, Y. L. Cun, J. Denker, and W. Hubbard, "Design of a neural network character recognizer for a touch terminal," *Pattern Recognition*, vol. 24, no. 2, pp. 105 – 119, 1991. [Online]. Available: http://www.sciencedirect.com/science/article/pii/003132039190081F

[Hin89] G. E. Hinton, "Connectionist learning procedures," *Artif. Intell.*, vol. 40, no. 1-3, pp. 185–234, Sep. 1989. [Online]. Available: http://dx.doi.org/10.1016/0004-3702(89)90049-0

[HK06] B. Huang and M.-T. Kechadi, "An HMM-SNN method for online handwriting symbol recognition," in *Image Analysis and Recognition*, ser. Lecture Notes in Computer Science, A. Campilho and M. Kamel, Eds. Springer Berlin Heidelberg, 2006, vol. 4142, pp. 897–905. [Online]. Available: http://dx.doi.org/10.1007/11867661_81

[JMRW01] S. Jaeger, S. Manke, J. Reichert, and A. Waibel, "Online handwriting recognition: the NPen++ recognizer," in *International Journal on Document Analysis and Recognition*, 2001, pp. 169–180.

[Kir10] D. Kirsch, "Detexify: Erkennung handgemalter LaTeX-symbole," Diploma thesis, Westflische Wilhelms-Universitt Mnster, 10 2010. [Online]. Available: http://danielkirs.ch/thesis.pdf

[MFW95] S. Manke, M. Finke, and A. Waibel, "NPen++: A writer independent, large vocabulary on-line cursive handwriting recognition system," in *Proceedings of the Third International Conference on Document Analysis and Recognition*, vol. 1, Aug 1995, pp. 403–408 vol.1. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=599023&tag=1

[Mit97] T. M. Mitchell, *Machine learning*, ser. McGraw Hill series in computer science. McGraw-Hill, 1997.

[MVGK+11] H. Mouchere, C. Viard-Gaudin, D. H. Kim, J. H. Kim, and U. Garain, "Crohme2011: Competition on recognition of online handwritten mathematical expressions," in *International Conference on Document Analysis and Recognition (ICDAR), 2011*, Sept 2011, pp. 1497–1500. [Online]. Available: http://hal.archives-ouvertes.fr/docs/00/61/52/16/PDF/CROHME_CRC511.pdf

[MVGK+12] H. Mouchere, C. Viard-Gaudin, D. Kim, J. Kim, and U. Garain, "Icfhr 2012 competition on recognition of on-line mathematical expressions (crohme 2012)," in *International Conference on Frontiers in Handwriting Recognition (ICFHR), 2012*, Sept 2012, pp. 811–816. [Online]. Available: http://hal.archives-ouvertes.fr/docs/00/71/78/50/PDF/Mouchere2012_CROHME.pdf

[SBPH12] J. Stria, M. Bresler, D. Prusa, and V. Hlavac, "Mfrdb: Database of annotated on-line mathematical formulae," in *Proceedings of the 2012 International Conference on Frontiers in Handwriting Recognition*, ser. ICFHR '12. Washington, DC, USA: IEEE Computer Society, 2012, pp. 542–547. [Online]. Available: http://dx.doi.org/10.1109/ICFHR.2012.231

[Tho14] M. Thoma, "On-line recognition of handwritten mathematical symbols," Karlsruhe, Germany, Nov. 2014. [Online]. Available: http://martin-thoma.com/write-math

[TSW90] C. C. Tappert, C. Y. Suen, and T. Wakahara, "The state of the art in online handwriting recognition," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 12, no. 8, pp. 787–808, 8 1990. [Online]. Available: http://dx.doi.org/10.1109/34.57669

[VLBM08] P. Vincent, H. Larochelle, Y. Bengio, and P.-A. Manzagol, "Extracting and composing robust features with denoising autoencoders," in *Proceedings of the 25th International Conference on Machine Learning*, ser. ICML '08. New York, NY, USA: ACM, 2008, pp. 1096–1103. [Online]. Available: http://doi.acm.org/10.1145/1390156.1390294

[VW90] M. Visvalingam and J. D. Whyatt, "The Douglas-Peucker algorithm for line simplification: Re-evaluation through visualization," in *Computer Graphics Forum*, vol. 9, no. 3. Wiley Online Library, 1990, pp. 213–225. [Online]. Available: http://www.bowdoin.edu/~ltoma/teaching/cs350/spring06/Lecture-Handouts/hershberger92speeding.pdf

KIT Scientific Working Papers
ISSN 2194-1629

www.kit.edu