On-line Recognition of Handwritten Mathematical Symbols

Martin Thoma

***Abstract*—Writing mathematical formulas with LaTeX is easy as soon as one is used to commands like `\alpha` and `\propto`. However, for people who have never used LaTeX or who don't know the English name of the command, it can be difficult to find the right command. Hence the automatic recognition of handwritten mathematical symbols is desirable. 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 which were collected with two crowdsourcing projects. 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 an improvement of $18.5\,\%$ for the TOP-1 error and $29.7\,\%$ for the TOP-3 error compared to the baseline system.**

I. Introduction

On-line recognition makes use of the pen trajectory. This means 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. One handwritten symbol in the described format is called a *recording*. One approach to classify recordings into symbol classes assigns a probability to each class given the data. The classifier can be evaluated by using recordings which were classified by humans and were not used to train the classifier. The set of those recordings is called *test set*. The TOP-n error is defined as the fraction of the symbols where the correct class was not within the top n classes of the highest probability.

Various systems for mathematical symbol recognition with on-line data have been described so far [KR98], [MVGZ$^+$13], but most of them have neither published their source code nor their data which makes it impossible to re-run experiments to compare different systems. This is unfortunate as the choice of symbols is crucial for the TOP-n error and all systems used different symbol sets. For example, the symbols o, O, $\circ$ and 0 are very similar and systems which know all those classes will certainly have a higher TOP-n error than systems which only accept one of them.

Daniel Kirsch describes in [Kir10] a system called Detexify which uses time warping to classify on-line handwritten symbols and claims to achieve a TOP-3 error of less than $10\,\%$ for a set of 100 symbols. He also published his data on https://github.com/kirel/detexify-data, which was collected by a crowd-sourcing 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 used to train and evaluated different classifiers. A complete description of all involved software, data and experiments is given in [Tho14].

II. Steps in Handwriting Recognition

The following steps are used in all classifiers which are described in the following:

1) **Preprocessing**: Recorded data is never perfect. Devices have errors and people make mistakes while using 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 III-A.
2) **Data multiplication**: Learning algorithms 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) **Segmentation**: The task of formula recognition can eventually be reduced to the task of symbol recognition combined with symbol placement. Before symbol recognition can be done, the formula has to be segmented. As this paper is only about single-symbol recognition, this step will not be further discussed.
4) **Feature computation**: A feature is high-level information derived from the raw data after preprocessing. Some systems like Detexify simply take the result of the preprocessing step, but many compute new features. This might have the advantage that less training data is needed since the developer can use knowledge about handwriting to compute highly discriminative features. Various features are explained in section III-B.
5) **Feature enhancement**: Applying PCA, LDA, or feature standardization might change the features in ways that could improve the performance of learning algorithms.

After these steps, we are faced with a classification learning task which consists of two parts:

1) **Learning** parameters for a given classifier. This process is also called *training*.
2) **Classifying** new recordings, sometimes called *evaluation*. This should not be confused with the evaluation of the classification performance which is done for multiple topologies, preprocessing queues, and features in Section IV.

The classification learning task can be solved with multilayer perceptrons (MLPs) if the number of input features is the same for every recording. There are many ways how to adjust MLPs and how to adjust their training. Some of them are described in section III-C.

III. Algorithms

A. Preprocessing

Preprocessing in symbol recognition is done to improve the quality and expressive power of the data. It should make follow-up tasks like segmentation and feature extraction 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 in 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. There are multiple ways how to shift the recording. For this paper, it was chosen to shift the bigger dimension to fit into the $[0, 1] \times [0, 1]$ unit square whereas the smaller dimension is centered in the $[-1, 1] \times [-1, 1]$ square.

Another normalization preprocessing algorithm is resampling. 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 with linear interpolation of the (x, t) and (y, t) sequences and getting a fixed number of equally spaced points per stroke.

Connect strokes is a noise reduction algorithm. 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 would be to reduce the number of points with the Douglas-Peucker algorithm to the most relevant ones 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 $p_1 p_n$. If this distance is above a threshold ε, the point p_i is added to S. Then the algorithm gets applied to $p_1 p_i$ and $p_i p_n$ 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 [MFW94], 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` is the probability of a feature being masked.

IV. Evaluation

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 into $[-1, 1] \times [-1, 1]$ as described in section III-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_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 $\text{sig}(x) := \frac{1}{1+e^{-x}}$. 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 might 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	range	min	max	range
B_1	23.08 %	23.44 %	0.36 %	6.67 %	6.80 %	0.13 %
B_2	<u>21.45 %</u>	<u>21.83 %</u>	0.38 %	5.68 %	<u>5.75 %</u>	0.07 %
B_3	21.54 %	22.28 %	0.74 %	<u>5.50 %</u>	5.82 %	0.32 %
B_4	23.19 %	24.84 %	1.65 %	5.98 %	6.44 %	0.46 %

TABLE I

THE SYSTEMS B_1 – B_4 WERE RANDOMLY INITIALIZED, TRAINED AND EVALUATED 5 TIMES TO ESTIMATE THE INFLUENCE OF RANDOM WEIGHT INITIALIZATION.

B. Connect strokes

In order to solve the problem of interrupted strokes, pairs of strokes can be connected with stroke connect 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 connect algorithm was tried with 5 px, 10 px and 20 px. All models TOP-3 error improved with a threshold of $\theta = 10\,\text{px}$ by at least 0.17 %, except B_4 which improved only by 0.01 % which could be a result of random weight initialization.

C. Douglas-Peucker Smoothing

The Douglas-Peucker algorithm can be used to find points that are more relevant for the overall shape of a recording. After that, an interpolation can be done. If the interpolation is a cubic spline interpolation, this makes the recording smooth.

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_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 % worse for TOP-1 and TOP-3 error.

System	Classification error			
	TOP-1	change	TOP-3	change
$B_{1,p}$	23.75 %	0.41 %	7.19 %	0.39 %
$B_{2,p}$	<u>22.76 %</u>	1.25 %	6.38 %	0.63 %
$B_{3,p}$	23.10 %	1.17 %	<u>6.14 %</u>	0.40 %
$B_{4,p}$	25.59 %	1.71 %	6.99 %	0.87 %

TABLE II

SYSTEMS WITH DENOISING AUTO-ENCODER PRETRAINING COMPARED TO PURE GRADIENT DESCENT. THE PRETRAINED SYSTEMS CLEARLY PERFORMED WORSE.

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.

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_1 by 0.27 % for the TOP-3 error and system B_3 for the TOP-1 error by 0.74 %, but all other systems and error measures either got worse or did not improve much.

The other global features did improve the systems $B_1 - -B_3$, but not B_4. The highest improvement was achieved with the re-curvature feature. It improved the systems $B_1 - -B_4$ by more than 0.6 % 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. It was tried to triple the data by adding a rotated 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 % for the TOP-1 error.

F. Pretraining

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

Figure 1 shows the evolution of the TOP-1 error over 1000 epochs with supervised layer-wise pretraining and without pretraining. It clearly shows that this kind of pretraining improves the classification performance by 1.6 % for the TOP-1 error and 1.0 % for the TOP-3 error.

Pretraining with denoising auto-encoder lead to the much worse results listed in table II. 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 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.

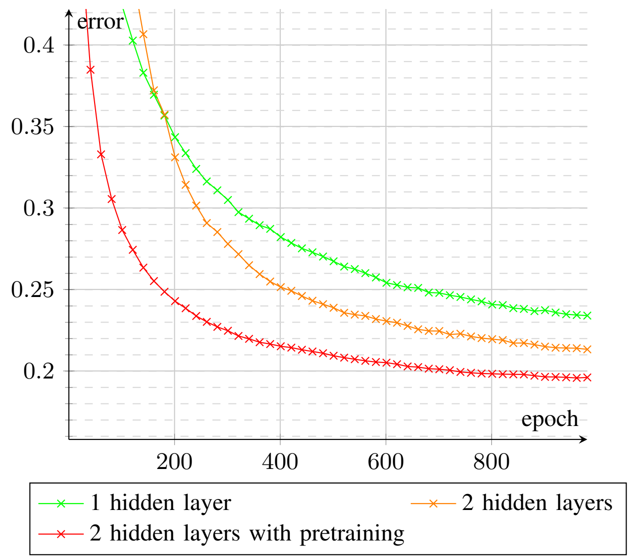

Fig. 1. Training- and test error by number of trained epochs for different topologies with SLP. The plot shows that all pretrained systems performed much better than the systems without pretraining. All plotted systems did not improve with more epochs of training.

System	Classification error			
	TOP1	change	TOP3	change
$B_{1,c}$	20.96 %	−2.38 %	5.24 %	−1.56 %
$B_{2,c}$	18.26 %	−3.25 %	4.07 %	−1.68 %
$B_{3,c}$	18.19 %	−3.74 %	4.06 %	−1.68 %
$B_{4,c}$	18.57 %	−5.31 %	4.25 %	−1.87 %
$B'_{1,c}$	19.33 %	−1.63 %	4.78 %	−0.46 %
$B'_{2,c}$	17.52 %	−0.74 %	4.04 %	−0.03 %
$B'_{3,c}$	17.65 %	−0.54 %	4.07 %	0.01 %
$B'_{4,c}$	17.82 %	−0.75 %	4.26 %	0.01 %

TABLE III
ERROR RATES OF THE OPTIMIZED RECOGNIZER SYSTEMS. THE SYSTEMS $B'_{i,c}$ WERE TRAINED ANOTHER 1000 EPOCHS WITH A LEARNING RATE OF $\eta = 0.05$. THE VALUE OF THE COLUMN "CHANGE" OF THE SYSTEMS $B'_{i,c}$ IS RELATIVE TO $B_{i,c}$.

G. Optimized Recognizer

All preprocessing steps and features that were useful were combined to create a recognizer that should perform 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 TOP1 error rate of 18.56 % and a TOP3 error of 4.11 %. This was achieved by a MLP with a 167:500:500:369 topology.

It used an algorithm to connect strokes of which the ends were less than 10 px away, scaled each recording to a unit square and shifted this unit square to $(0, 0)$. 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

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.

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

V. Conclusion

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 to fit into a unit square, shifted to $(0, 0)$, 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_2 has a TOP-3 error of 5.75 %.

Adding two slightly rotated variants for each recording and hence tripling the training set made the systems B_3 and B_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_1–B_3, whereas the stroke center point feature made B_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 connect 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 $B'_{2,c}$ had a TOP-3 error of 4.04 %. This means that the TOP-3 error dropped by over 1.7 % in comparison to the baseline system B_2.

A TOP-3 error of 4.04 % 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 $B'_{2,c}$ evaluates new recordings in a fraction of a second and has acceptable recognition rates. Many algorithms were evaluated. However, there are still many other algorithms which could be evaluated and, at the time of this work, the best classifier $B'_{2,c}$ is only available through the Python package `hwrt`. It is planned to add an web version of that classifier online.

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.

[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

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

[KR98] A. Kosmala and G. Rigoll, "Recognition of on-line handwritten formulas," in *In Proceedings of the Sixth International Workshop on Frontiers in Handwriting Recognition*, 1998, pp. 219–228. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.41.9056

[MFW94] S. Manke, M. Finke, and A. Waibel, "Combining bitmaps with dynamic writing information for on-line handwriting recognition," in *Proceedings of the ICPR-94*, 1994, pp. 596–598.

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

[MVGZ$^+$13] H. Mouchere, C. Viard-Gaudin, R. Zanibbi, U. Garain, D. H. Kim, and J. H. Kim, "ICDAR 2013 CROHME: Third international competition on recognition of online handwritten mathematical expressions," in *12th International Conference on Document Analysis and Recognition (ICDAR), 2013*, Aug 2013, pp. 1428–1432. [Online]. Available: http://www.isical.ac.in/~crohme/CROHME2013.pdf

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

[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