Handwritten Digit Recognition: A Neural Network Demo

Berend-Jan van der Zwaag

EUREGIO Computational Intelligence Center
Dept. of Electrical Engineering
University of Twente
Enschede, the Netherlands
b.j.vanderzwaag@el.utwente.nl

Abstract. A handwritten digit recognition system was used in a demonstration project to visualize artificial neural networks, in particular Kohonen's self-organizing feature map. The purpose of this project was to introduce neural networks through a relatively easy-to-understand application to the general public. This paper describes several techniques used for preprocessing the handwritten digits, as well as a number of ways in which neural networks were used for the recognition task. Whereas the main goal was a purely educational one, a moderate recognition rate of 98% was reached on a test set.

1 Introduction

Handwritten digit recognition is already widely used in the automatic processing of bank cheques, postal addresses, etc. Some of the existing systems include computational intelligence techniques such as artificial neural networks or fuzzy logic, whereas others may just be large lookup tables that contain possible realizations of handwritten digits.

Artificial neural networks have been developed since the 1940s, but only the past fifteen years have they been widely applied in a large variety of disciplines. Originating from the artificial neuron, which is a simple mathematical model of a biological neuron, many varieties of neural networks exist nowadays. Although some are implemented in hardware, the majority are simulated in software.

Artificial neural nets have successfully been applied to handwritten digit recognition numerous times, with very small error margins, see e.g. [2] and [4].

The work described in this paper does not have the intention to compete with existing systems, but merely served to illustrate to the general public how an artificial neural network can be used to recognize handwritten digits. It was part of *NeuroFuzzyRoute in the* EUREGIO, an exposition in the framework of the world exposition EXPO2000 in Hannover.

2 Neural Networks

Artificial neural networks, usually called neural networks (NNs), are systems composed of many simple processing elements (neurons) operating in parallel

B. Reusch (Ed.): Fuzzy Days 2001, LNCS 2206, pp. 762–771, 2001.

whose function is determined by network structure, connection strengths, and the processing performed at computing elements or nodes [1] (other definitions can also be found). NNs exist in many varieties, though they can be categorized into two main groups, where the distinction lies in the learning method:

- supervised learning: the network is trained with examples of input and desired output;
- unsupervised learning: the network tries to organize the input data in a useful way without using external feedback.

In this work, a Kohonen self-organizing map [3] is used, which learns without supervision. The labeling stage is supervised, however, because digit class information is used to label the neurons. The Kohonen network is ideal for demonstrating how information can be stored in a neural network, as the information in the neurons has the same easy-to-interpret format as the input, namely 2-dimensional images of (parts of) numerals.

2.1 Learning

In the learning stage, the network is trained to recognize (parts of) handwritten numerals. This is done by first initializing the network and then repeatedly offering input patterns, determining the winning neuron and adjusting the neurons' feature vectors (see [3] for a more detailed description).

Initialization The self-organizing map consists of $N \times N$ neurons laid out in a rectangular grid. Each neuron contains (basically *is*) a randomly initialized feature vector with as many elements as the number of inputs, i.e., $M \times M$, being the size of the (binary) image that contains (part of) a numeral. The lateral distance between two neurons is the shortest distance between them in the map, usually defined along a straight connecting line or along the gridlines. Depending on the lateral distance between two neurons, they can be either inside or outside each other's neighborhood. The neighborhoods are important for the adjustment of the feature vectors during learning.

Finding the Winning Neuron One by one, training input patterns are offered to the network. For every neuron, the difference (usually Euclidian distance) between its feature vector and the input vector is calculated. The neuron with the smallest difference is then the winning neuron for that particular input example.

Vector Adjustment The winning neuron's feature vector is then adjusted in such a way that the difference with the input example decreases. This is also done for the neurons in the winning neuron's neighborhood, though the larger the lateral distance from the winning neuron, the smaller the adjustment. Often the neurons directly outside the winning neuron's neighborhood are adjusted in the opposite direction. The rate of adjustment as a function of the lateral

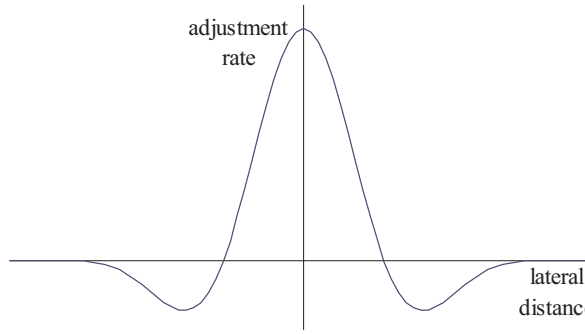


Fig. 1. Illustration of a Mexican hat function

distance to the winning neuron has the shape of a Mexican hat, as can be seen in Fig. 1. Over time, this hat is shrinking both horizontally (the neighborhood is getting smaller) and vertically (the adjustments are getting smaller), until the learning finishes.

2.2 Labeling

Before the network can classify unknown digits, the neurons need to be labeled. This happens after the learning stage has finished. Several ways of doing this are possible, some of which will be explained below. Once the neurons have been labeled, an unknown handwritten digit can be offered to the network. The label of the winning neuron then serves to classify the unknown digit.

Majority Voting Every neuron is assigned a number of counters, one counter for each digit class. Then, from every digit class an equal number of examples is offered to the network. Every time a neuron is the winning neuron, its counter for the corresponding digit class is incremented. After all examples of all classes have been offered, each neuron is labeled according to the class for which its counter was the highest. For example, if a neuron was winning neuron for seven instances of "5", two instances of "6", and never for the remaining digit classes, it will be labeled "5".

Neurons that never were the winning neuron, or for which the above method leads to ambiguities (e.g., as often winner for "5" as for "6"), can be labeled according to the labels of the neighboring neurons using majority voting. This may be done because of the topology preserving property of the self-organizing map: neighboring neurons contain similar information. Alternatively, neurons not labeled in the first round can be labeled "unknown."

Minimum Average Distance Instead of simply incrementing the class counters for winning neurons, more unambiguous labels can be obtained by minimum

average distance labeling. In this method, every time an example digit is offered to the network, the corresponding class counters for all neurons are increased by the lateral distance of the neuron to the winning neuron (analogous to the previous method, we use the term "counter" here, even though instead of counting occurrences we now accumulate distances). After all example digits of all classes have been presented, the average distances are calculated by dividing the class counters by the number of example digits for that class. For every neuron, the class with the smallest average lateral distance to the winning neuron forms the label. Neurons that never win but that are close to a winning neuron most of the time, will automatically be labeled accordingly.

Minimum Average Difference Yet another labeling method is labeling by minimum average difference. This method is very similar to the labeling by minimum average distance, but instead of accumulating the lateral distances to the winning neurons, we now accumulate the differences between the presented examples and the neuron's feature vector for each neuron. Again, after all example digits of all classes have been presented, the average differences are calculated by dividing the class counters by the number of example digits for that class. For every neuron, the class with the smallest average difference forms the label.

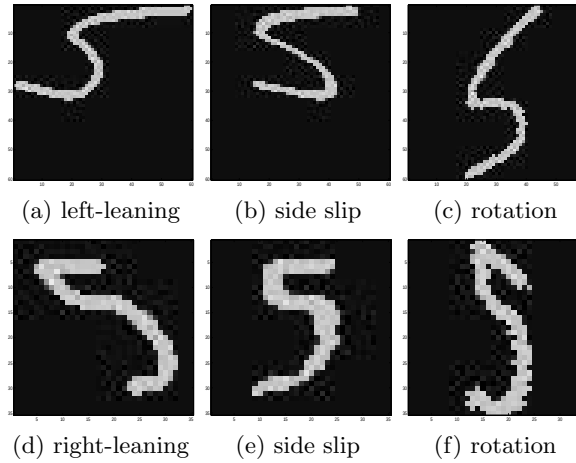
3 Handwritten Digit Recognition

3.1 Preprocessing

As digitized numerals may originate from a range of sources, it is important to make sure that all digits are preprocessed so as to eliminate extra problems due to non-class-specific differences in size, shear, line thickness, background and digit colors, resolution, etc. A few techniques to minimize these differences are discussed below.

Deskewing To help compensate for different handwriting styles (with different slant), digits that are slanting to the left or to the right are deskewed to put them all upright or to at least make them slant under the same angle. For some examples, see Fig. 2.

The deskewing process works as follows: first, an imaginary line is drawn through the digit so that the total of the horizontal distances from all digit pixels to this line is minimized in a least-squares sense. The imaginary line can be described by $x = ay + b$, where x and y are the horizontal and vertical pixel index, respectively, and a and b are line parameters. Then a side slip is performed: the pixels that form the digit are horizontally shifted to the left or to the right, while keeping their positions relative to the imaginary line constant. This is done in such a way that the imaginary line becomes vertical, with the digit effectively in an upright position. In formula: $x_{new} = x_{old} - ay$ for every pixel, resulting in the new line $x = b$. If the slant is removed by rotation, the recognition problem is likely to become more difficult, due to increased variation



(a) left-leaning (b) side slip (c) rotation

(d) right-leaning (e) side slip (f) rotation

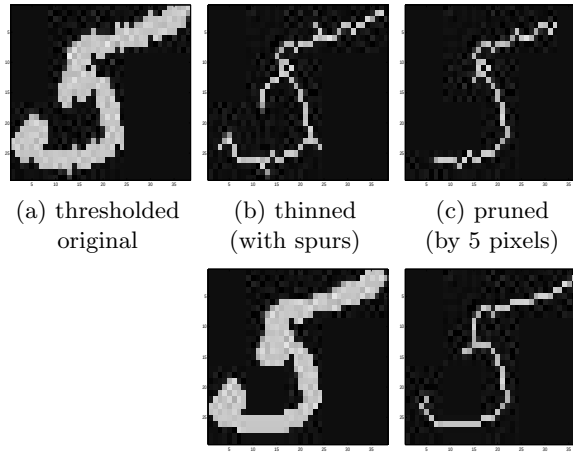
Fig. 2. Two examples of deskewing: (a) and (d): left and right leaning digits, resp.; (b) and (e): deskewed by side slip; (c) and (f): deskewed by rotation

among numerals from the same category. This may become clear by looking at Figs. 2c and 2f.

Thinning To overcome problems related to differences in the line thickness of the written characters, the digit images are thinned to a thickness of one pixel. This can lead to undesired side-effects, such as spurs, as can be seen from Fig. 3b. To correct for this, one can remove spur lines of up to a certain length from the resulting thinned image (Fig. 3c). A negative side-effect of this pruning process is that end-lines that are actually part of the digit are shortened. The length of these digit parts is usually still long enough for correct recognition of the digit, so that the benefits of the thinning process far outweigh the negative sides. Algorithms exist that try to correct for this ; however, the need for pruning can be avoided by smoothing the handwritten digit image before thinning, as seen in Figs. 3d and 3e. Moreover, this method results in an overall smoother thinned image.

Resizing Differences in digit size are due to different image resolutions and handwriting styles. To minimize the effect of these differences, all digits are resampled/rescaled to the same size in pixels, before being fed to the recognizer.

Smoothing Some acquisition methods may cause small artifacts on the digit's edges. These are removed by a smoothing operation. Some preprocessing techniques, including deskewing and resizing, may cause similar artifacts, so smoothing may be useful between several stages of the preprocessing. Effective smoothing can be accomplished by filling tiny gaps in both the foreground and the



(a) thresholded original (b) thinned (with spurs) (c) pruned (by 5 pixels)

(d) smoothed (e) thinned

Fig. 3. Sequence a-b-c shows the thinning and pruning process; sequence a-d-e shows an alternative: smoothing and thinning without pruning

(a) a gap in the black background (b) not a gap

Fig. 4. Illustration of how 5 can be significantly more than 3 but 6 not significantly more than 2 (see Sect. 3.1, Smoothing)

background. Filling a gap is nothing more than changing the foreground color of a gap pixel to the background color or vice versa. Tiny gaps in the foreground are background pixels with significantly more neighboring foreground pixels than neighboring background pixels. Tiny gaps in the background are defined in a similar way and can in fact also be seen as tiny foreground protrusions. The term "significantly more" cannot be defined as a simple number, but depends on the layout of the pixel neighborhood. For example, in the layout of Fig. 4a, 5 is significantly more than 3 and the central pixel can be safely smoothed away, whereas in the layout of Fig. 4b, 6 is not significantly more than 2 as the central foreground pixel cannot be smoothed away without breaking the foreground connectivity.

4 Network Configurations

4.1 Two NNs in Series

In this setup, one Kohonen map is trained with digit fragments, and another is trained with the ordering information in the trained first map. The idea is that

particular combinations of digit fragments are characteristic for a particular digit class; the first network identifies fragments, i.e., not digits, and the second network identifies combinations of fragments, i.e., digits. The second network will need to be labeled in order to be able to classify unknown digits. The first network remains unlabeled. Suppose the digit fragments are $M \times M$ pixels in size, then the first network (say $N \times N$ neurons), will have $M \times M$ inputs, being the pixel values of the digit fragments, meaning that the vectors stored in the neurons have a length of $M \times M$. The second network will then have $N \times N$ inputs, one for every neuron in the first network. Per digit, for every neuron in the first network is recorded for how many digit fragments that neuron was the winning neuron. These totals are input to the second network.

4.2 Template Matching Instead of NNs

Another method for the recognition is to create a template for each digit and to compare unknown digits to these templates to find out which template matches closest. One way of creating such templates is by calculating an average digit from an example set for each digit class.

4.3 Ten NNs in Parallel

In this setup, ten networks are trained for one digit class each. Thus, the networks are pre-labeled. They are trained with digit fragments (as before without location information). For the recognition stage, several methods are available, two of which are discussed below. In the special case that the networks consist of only one neuron each and the digit fragments actually contain the whole digit, this method is similar to the template matching method of Sect. 4.2 with the difference lying therein that creating the templates is now realized by training the networks, rather than calculating the average digits directly.

Minimum Distance Classification For classifying an unknown digit, (fragments of) that digit are offered to all ten networks, and per network the (Euclidian) distances between the winning neurons and the fragments are totaled over all fragments. Then, the label of the network with the smallest total distance gives the class of the unknown digit. In general, all ten networks will have equal numbers of neurons. If this is not the case, the total Euclidian distance is divided by the number of neurons to find the network with the smallest average distance.

Label Classification With label classification, (fragments of) the unknown digit are also offered to all ten networks, but now we are only interested in which network has the winning neuron with the smallest distance to a fragment, measured over all networks. This network is then called the winning network for that fragment. Finally, the label of the most frequently winning network, measured over all fragments, gives the class of the unknown digit.

4.4 Fragmented Digits vs. Whole Digits

Digit Fragments The idea to use fragments of digits rather than the whole digits, originates from the idea that for a specific digit class, digits as a whole show more variation than individual fragments that are characteristic for that class. When using fragments however, a decision needs to be made whether to include any position information or not. For the purpose of demonstrating the working of a Kohonen self-organizing map, the organization of the fragments is expected to be easier to understand if positional information is left out.

Whole Digits When using whole digits as inputs to the networks, what actually happens depends on the training method and the network architecture. With ten small networks, the networks basically store representative examples of the digits during the training stage. These are derived from the example set, although the stored examples are usually not present in the example set themselves, as the network tries to store generalizations of the example digits it receives at its inputs. The networks are pre-labeled and only trained with examples from their respective digit classes.

With one large network, all ten classes of digits are offered to the network during the training stage. The network will then store representative examples of all digits. Depending on the actual size of the network, more complicated numerals or numerals that can appear in several distinctive forms may cause the network to store more examples of that digit class than of another, simpler, class (e.g., consider the class "2", which can be written in very different ways, and the class "0", which most people write in much more similar ways, as can also be seen in Fig. 5).

5 Results

As the goal was a reasonably well working handwritten digit recognizer for demonstration purposes, and due to very limited time constraints, only a small number of experiments have been conducted to enable comparison between the various configurations. An existing set of handwritten digits with 25 digits per class was used. This set, shown in Fig. 5, was split into a training set of 15($\times$10) examples and a test set of 10($\times$10) examples. Previous research ([5] and others) had yielded a maximum 92% correct classification score (percentage correctly recognized digits) on the same test set. Using digit parts, this number now increased to 98% (see Table 1), which was considered sufficient for use in a demonstration. In these experiments, one network was used for each digit class, in total 10 networks. The size of the networks was the same for all 10 networks: 5$\times$5 and 7$\times$7 in two series of experiments. The size of the digits after preprocessing varied between 5$\times$5 and 50$\times$50 pixels and the size of the digit fragments varied between 3$\times$3 and 25$\times$25 pixels. Some results are shown in Table 1. In this table, the best classification results over a range of fragment sizes are shown for a selection of digit sizes, using minimum distance classification (as described in Sect. 4.3).

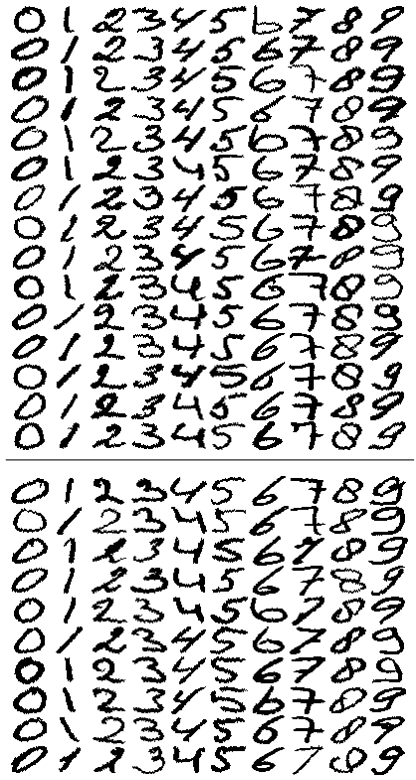


Fig. 5. Set of handwritten digits used for training (*top*) and testing (*bottom*)

A few experiments were conducted using whole digits rather than fragments. Results of these experiments are shown in Table 2. Per digit class one network was used, and classification was done by minimum distance as described in Sect. 4.3. Shown are the performances (on the test set) per digit size for two network sizes. The results are worse than in the experiments with fragmented digits, but strong conclusions cannot be drawn due to the limited number of experiments.

6 Digit Recognizing Demonstration

A continuous presentation has been installed at the Da Vinci Techno Activity Center in Enschede, the Netherlands, where visiting children and adults can learn more about technology. The presentation consists of a slide show explaining the whole Kohonen system of learning and recognizing, and a writing pad with display where visitors can write a digit and see the NN outcome. This NN is the same one that scored best in the experiments: 98% correct recognition rate, see Table 1. In the slide show, a set of unclassified digit fragments is shown

Table 1. Results of experiments with fragmented digits

network size	digit size [pixels]				
	5×5	10×10	15×15	20×20	25×25
5×5	73%	93%	98%	94%	93%
7×7	73%	90%	92%	86%	82%

Table 2. Results of experiments with whole digits

network size	digit size [pixels]				
	5×5	10×10	15×15	20×20	25×25
5×5	63%	82%	91%	92%	93%
7×7	70%	83%	92%	(not used)	

to give people also a chance to determine what digit the fragments come from. The whole presentation was part of "NeuroFuzzyRoute in the EUREGIO," an exposition in the framework of the world exposition EXPO2000 in Hannover.

Unfortunately, the digits written by visitors of the techno activity center have not been recorded, nor is anything known about the recognition rate. However, visitors have reacted enthusiastically to the hands-on demonstration.

7 Conclusions

This paper shows that a Kohonen self-organizing map can be used to recognize digits when given those digits in fragments without positional information. The Kohonen self-organizing map and the handwritten digit recognition have also proven to be a good neural network architecture and application for the purpose of introducing and demonstrating neural networks to the general public.

Acknowledgments

Thanks are due to Andre Beltman who wrote a fast Kohonen NN simulator, *NN Kohinoor*, and to Wubbe Jan Velthuis who designed the slide show.

References

1. DARPA: DARPA Neural Network Study. AFCEA International Press (1988) 60
2. Jain, L.C., Lazzerini, B. (eds.): Knowledge-based Intelligent Techniques in Character Recognition. CRC Press, Boca Raton FL (1999)
3. Kohonen, T.: Self-Organizing Maps. Springer-Verlag, Berlin (1995)
4. LeCun, Y., Jackel, L.D., Bottou, L., Brunot, A., Cortes, C., Denker, J.S., Drucker, H., Guyon, I., Muller, U.A., Sackinger, E., Simard, P., Vapnik, V.: Comparison of Learning Algorithms for Handwritten Digit Recognition. In: Fogelman, F., Gallinari, P. (eds.): Proc. 5th Int. Conf. on Artificial Neural Networks (1995) 53-60
5. Wentink, M.: Master-Slave Kohonen Network Applied to Digit Recognition. Internal Report BSC-021N95, University of Twente (1995)