LEARNING ALGORITHMS FOR CLASSIFICATION: A COMPARISON ON HANDWRITTEN DIGIT RECOGNITION

Yann LeCun, L. D. Jackel, Léon Bottou*, Corinna Cortes,
John S. Denker, Harris Drucker, Isabelle Guyon, Urs A. Müller,
Eduard Säckinger, Patrice Simard, and Vladimir Vapnik
AT&T Bell Laboratories,
Holmdel, NJ 07733, USA
E-mail: yann@research.att.com

ABSTRACT

This paper compares the performance of several classifier algorithms on a standard database of handwritten digits. We consider not only raw accuracy, but also training time, recognition time, and memory requirements. When available, we report measurements of the fraction of patterns that must be rejected so that the remaining patterns have misclassification rates less than a given threshold.

1. Introduction

Great strides have been achieved in pattern recognition in recent years. Particularly striking results have been attained in the area of handwritten digit recognition. This rapid progress has resulted from a combination of a number of developments including the proliferation of powerful, inexpensive computers, the invention of new algorithms that take advantage of these computers, and the availability of large databases of characters that can be used for training and testing. At AT&T Bell Laboratories we have developed a suite of classifier algorithms. In this paper we contrast the relative merits of each of the algorithms. In addition to accuracy, we look at measures that affect implementation, such as training time, run time, and memory requirements.

2. Databases

We begin by describing the databases we have used as benchmarks in the past and present. When we first began our research in character recognition we assembled our own database of 1200 digits, which is sometimes known as the "AT&T" database [7]. This database consisted of 10 examples of each digit from 12 different writers. The writers, who were cooperative friends of the researchers, were instructed to write each character in a box. This database was made available to other researchers in different institutions in Europe and the US. Usually, data from the first six writers comprised the training set, with the second six writers used for testing. We quickly discovered that this data was "too easy," since our recognizers and others' [5] soon achieved better than 99% accuracy on the test set. We therefore abandoned this database. Nevertheless, it is still occasionally used by other groups.

2.1. A Zipcode database

In order to obtain a database more typical of real-world applications, we contacted the US Postal Service and its consultants at Arthur D. Little, Inc. in Washington, DC. Through them we acquired a database of 7064 training and 2007 test digits that were clipped from images of handwritten Zipcodes. The digits were machine segmented from the Zipcode string by an automatic algorithm. As always, the segmented characters sometimes included extraneous ink and sometimes omitted critical fragments. These segmentation errors often resulted in characters that were unrecognizable or appeared mislabeled. (For example, a vertical fragment of a "7" would appear as a mislabeled "1".) These butchered characters comprised about 2% of test set, and limited the attainable accuracy. We could improve the accuracy of our recognizers by removing the worst offenders from the training set, but in order to maintain objectivity, we kept the butchered characters in our test set.

This database of cleaned training data, and uncleaned test data served for a time as a standard for our internal AT&T benchmarking. The US Postal Service requested that we not distribute this database ourselves and instead, the USPS, through Arthur D. Little, Inc., supplied other researchers with the unsegmented Zipcodes from which our database was derived. Segmenting was done by the users, often by hand. Thus no common database was available for meaningful comparisons.

Another shortcoming of this database was the relatively small size of the training and test sets. As our recognizers improved, we soon realized that we were starved for training data and that much better results could be had with a larger training set size. The size of the test set was also a problem. As our test error rates moved into the range of 3% (60 errors), we were uncomfortable with the large statistical uncertainty caused by the small sample size.

2.2. The NIST test

Responding to the community's need for better benchmarking, the US National Institute of Standards and Technology (NIST) provided a database of handwritten characters on 2 CD ROMs. NIST organized a competition based on this data in which the training data was known as NIST Special Database 3, and the test data was known as NIST Test Data 1.

After the competition was completed, many competitors were distressed to see that although they achieved error rates of less than 1% on validation sets drawn from the training data, their performance on the test data was much worse. NIST disclosed that the training set and the test set were representative of different distributions: the training set consisted of characters written by paid US census workers, while the test set was collected from characters written by uncooperative high school students. Examples from these training and test sets are shown in Figure 1. Notice that the test images contain some very ambiguous patterns. Although this disparity in distributions is certainly possible in a real world application, it is prudent (and usually possible) to guard against it. In general we can expect best test results when recognizers are tuned to the kind of data they are likely to encounter when deployed.

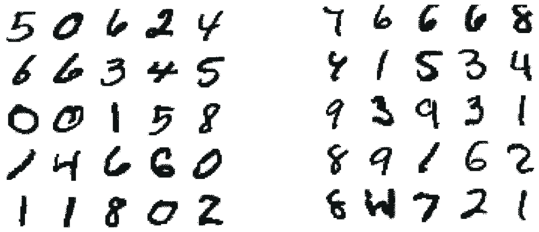

Fig. 1. a) Typical images from the NIST training set, and b) Typical images from the NIST test set.

A more subtle, but, for us, a more serious problem arises from having the training and test data belonging to different distributions. Most of our machine learning techniques now use the principles of Structural Risk Minimization [15] in which the capacity (roughly speaking, the number of free parameters) of a classifier is adjusted to match the quantity and the complexity of the training data. Because of the difference in distributions, we cannot use our full machine learning tool set on the NIST data when it is partitioned in this way.

2.3. Modified NIST (MNIST) training and test sets

For the reasons described above, we repartitioned the NIST data to provide large training and test sets that share the same distribution. We now describe how our new database was created. The original NIST test contains 58,527 digit images written by 500 different writers. In contrast to the training set, where blocks of data from each writer appeared in sequence, the data in the NIST test set is scrambled. Writer identities for the test set is available and we used this information to unscramble the writers. We then split this NIST test set in two: characters written by the first 250 writers went into our new training set. The remaining 250 writers were placed in our test set. Thus we had two sets with nearly 30,000 examples each. The new training set was completed with enough examples from the old NIST training set, starting at pattern # 0, to make a full set of 60,000 training patterns. Similarly, the new test set was completed with old training examples starting at pattern # 35,000 to make a full set with 60,000 test patterns. In the experiments described here, we only used the first 10,000 test images, but we used the full 60,000 training samples.

All the images were size normalized to fit in a 20x20 pixel box (while preserving

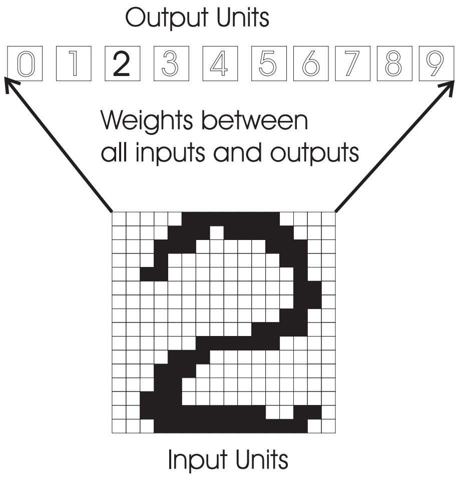

Fig. 2. Linear Classifier. Each input unit pixel value contributes to a weighted sum for each output unit. The output unit with the largest sum indicates the class of the input digit.

the aspect ratio). For some experiments, the 20x20 images were deslanted before being presented, i.e. slanted characters were straightened up using moment of inertia methods. For other experiments they were only centered in a larger input field using center of mass. Grayscale pixel values were used to reduce the effects of aliasing. Two methods (LeNet 1 and Tangent Distance) used subsampled versions of the images to 16 by 16 pixels. These are the training and test sets used in the benchmarks described in this paper. In this paper, we will call them the MNIST data.

3. The Classifiers

In this section we briefly describe the classifiers used in our study. For more complete descriptions readers may consult the references.

3.1. Baseline Linear Classifier

Possibly the simplest classifier that one might consider is a linear [4] classifier shown in Figure 2. Each input pixel value contributes to a weighted sum for each output unit. The output unit with the highest sum (including the contribution of a bias constant) indicates the class of the input character. In this kind of classifier there are $10N$ weights $+10$ biases, where N is the number of input pixels. For this experiment, we used deslanted 20x20 images of the MNIST characters. The network has 4010 free parameters. Because this is a linear problem, the weight values can be determined uniquely. The deficiencies of the linear classifier are well documented [11] and it is included here simply to form a basis of comparison for more sophisticated classifiers. On the MNIST data the linear classifier achieved 8.4% error on the test set.

3.2. Baseline Nearest Neighbor Classifier

Another simple classifier is a K-nearest neighbor classifier with a Euclidean distance measure between input images. This classifier has the advantage that no training time (nd no brain on the part of the designer) is required. However, the memory requirement and recognition time are large: the complete 60,000 twenty by twenty pixel training images (about 24 Megabytes at one byte per pixel, or 12 megabytes at 4 bits/pixel) must be available at run time. Much more compact representations could be devised with modest increase in recognition time and error rate. As in the previous case, deslanted 20x20 images were used. The MNIST test error for $k = 3$ is 2.4%. Naturally, a realistic Euclidean distance nearest-neighbor system would operate on feature vectors rather than directly on the pixels, but since all of the other systems presented in this paper operate directly on the pixels, this result is useful for a baseline comparison.

3.3. Large Fully Connected Multi-Layer Neural Network

Another classifier that we tested was a fully connected multi-layer neural network with two layers of weights (one hidden layer). The network was implemented on the MUSIC supercomputer [10] (for purposes of comparison, numbers quoted in Figures 8 and 9 are for equivalent times on a Sparc 10), and trained with various numbers of hidden units. Deslanted 20x20 images were used as input. The best results were 1.6% on the MNIST test set, obtained with a 400-300-10 network (approximately 123,300 weights). It remains somewhat of a mystery that networks with such a large number of free parameters manage to achieve reasonably low error rates on the test set, even though comparing their size to the number of training samples makes them appear grossly over-parameterized. Classical feed-forward multilayer networks seem to possess a built-in "self-regularization" mechanism. We conjecture that, due to the nature of the error surface, gradient descent training invariably goes through a phase where the weights are small. Recent theoretical analyses seem to confirm this (Sara Solla, personal communication). This is due to the fact that the origin of weight space (all the weights zero) is a saddle point that is attractive in almost every direction. Small weights cause the sigmoids to operate in the quasi-linear region, making the network essentially equivalent to a low-capacity, single-layer network. As the learning proceeds, the weights grow, which progressively increases the effective capacity of the network. A better theoretical understanding of these phenomena, and more empirical evidence, are definitely needed.

3.4. LeNet 1

To solve the dilemma between small networks that cannot learn the training set, and large networks that seem overparameterized, one can design *specialized* network architectures that are specifically designed to recognize two-dimensional shapes such as digits, while eliminating irrelevant distortions and variability. These considerations lead us to the idea of *convolutional network* [8]. There are well-known advantages to performing shape recognition by detecting and combining local features. We can require the network to do this by constraining the connections in the first few layers to be local: each unit takes its input from a local "receptive field" on the layer below. Furthermore, salient features of a distorted character might be displaced slightly from their position in a typical character, or the same feature can appear at different locations in different characters. Therefore a feature detector that is useful on one part of the image, is likely to be useful on other parts of the image as well. Specifying this knowledge we can be performed by forcing a set of units, located at different places on the image, to have identical weight vectors. The outputs of such a set of units constitute a *feature map*. A sequential implementation of this would be to scan the input image with a single unit that has a local receptive field, and store the states of this unit at corresponding locations in the feature map. This operation is equivalent to a convolution with a small size kernel, followed by a squashing function. The process can be performed in parallel by implementing the feature map as a plane of unitss that *share* a single weight vector. That is, units in a feature map are constrained to perform the same operation on different parts of the image.

An interesting side-effect of this *weight sharing* technique, is to reduce greatly the number of free parameters, since a large number of units share the same weights. In addition, this builds a certain level of shift invariance into the system. In practice, multiple feature maps, extracting different features types from the same image, are needed. It is important to stress that *all* the weights in the network are trained by gradient descent. Computing the gradient can be done with a slightly modified version of the classical backpropagation procedure.

The idea of local, convolutional feature maps can be applied to subsequent hidden layers as well, to extract features of increasing complexity and abstraction. Interestingly, higher level features require less precise coding of their location. Reduced precision on the position is actually advantageous, since a slight distortion or translation of the input will have reduced effect on the representation. Thus, each feature extraction layer in our network is followed by an additional layer which performs a local averaging and a subsampling, reducing the resolution of the feature map. Subsampling layers introduce a certain level of invariance to distortions and translations. The resulting architecture is a "bi-pyramid": The loss of spatial resolution in the

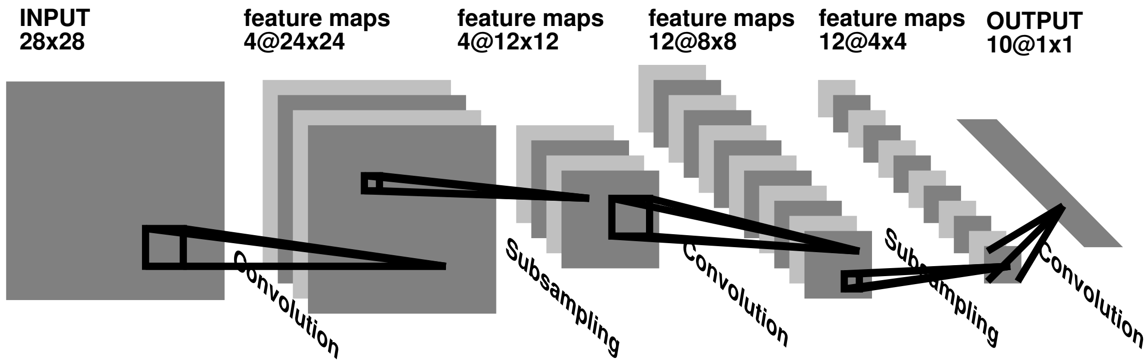

Fig. 3. Architecture of LeNet 1. Each plane represents a feaure map, i.e. a set of units whose weights are constrained to be identical. Input images are sized to fit in a 16 x 16 pixel field, but enough blank pixels are added around the border of this field to avoid edge effects in the convolution calculations.

feature maps (due to subsampling) is partially compensated by an increase in the number of feature types. It is important to stress that *all the weights in the network are adaptive*. The training process causes convolutional networks to automatically synthesize their own features.

One of our first convolutional network architecture, LeNet 1, shown in Figure 3, was trained on the MNIST database. Because of LeNet 1's small input field, the images were down-sampled to 16x16 pixels and centered in the 28x28 input layer. Although about 100,000 multiply/add steps are required to evaluate LeNet 1, its convolutional nature keeps the number of free parameters to only about 3000. The LeNet 1 architecture was developed using our own version of the USPS database and its size was tuned to match the available data. On the MNIST LeNet 1 achieved 1.7% error.

3.5. LeNet 4

Experiments with LeNet 1 made it clear that a larger convolutional network was needed to make optimal use of the large size of the MNIST set. LeNet 4 was designed to address this problem. It is an expanded version of LeNet 1 that has a 32x32 input layer in which the 20x20 MNIST images (not deslanted) were centered by center of mass. It includes more feature maps and an additional layer of hidden units that is fully connected to both the last layer of features maps and to the output units. LeNet 4 requires about 260,000 multiply/add steps and has about 17,000 free parameters. LeNet 4 achieves 1.1% error on the MNIST test.

3.6. LeNet 5

LeNet 5, has an architecture similar to LeNet 4, but has more feature maps, a larger fully-connected layer, and it uses a distributed representation to encode the categories at the output layer, rather than the more traditional "1 of N" code. LeNet 5 has a total of about 340,000 connections, and 60,000 free parameters, most of them in the last two layers. Again the non-deslanted 20x20 images centered by center of mass were used, but the training set was augmented with distorted versions of the original characters. The distorted characters were automatically generated using small, randomly chosen affine transformations (shift, scaling, rotation, and skewing). It achieves 0.9% error on the MNIST test.

3.7. Boosted LeNet 4

Several years ago, Schapire [13] proposed methods (called "boosting") for building a committee of learning machines that could provide increased accuracy compared to a single machine. Drucker et al. [6] expanded on this concept and developed practical algorithms for increasing the performance of a committee of three learning machines. The basic method works as follows: One machine is trained the usual way. A second machine is trained on patterns that are filtered by the first machine so that the second machine sees a mix of patterns, 50% of which the first machine got right, and 50% of which it got wrong. Finally, a third machine is trained on new patterns on which the first and the second machines disagree. During testing, in the Drucker method, all three machines are shown the unknown character and their output scores are added, with the highest total score indicating the most likely classification.

Notice that if the first machine is a version of LeNet 4, its 1% error rate means that an enormous amount of data must filtered to glean enough mis-classified patterns to train a second machine as complex as LeNet 4. Even more data is required to train the third machine. For this MNIST database there was insufficient data to train all three machines. As with LeNet 5, an unlimited number of training patterns was generated by distorting the training data with a set of affine transformations and line-thickness variations. This choice of distortions, in effect, builds some of our knowledge about character recognition into the training process. With this trick, a composite machine, consisting of three versions of LeNet 4, was trained. It attained a test error rate of 0.7%, the best of any of our classifiers. At first glance, boosting appears to require three times as much time to perform recognition as a single machine. In fact, with a simple trick, the additional computation cost is only about a factor of 1.75. This is because usually the first machine classifies patterns with high confidence and the outputs of the other two machines need not be evaluated.

3.8. Tangent Distance Classifier (TDC)

The TDC is a memory-based, k-nearest-neighbor classifier in which test patterns are compared to labeled, prototype patterns in the training set. The class of the training pattern "closest" to the test pattern indicates the class of the test pattern. The key to performance is to determine what "close" means for character images. In the naive approach, nearest-neighbor classifiers use the Euclidean distance: we simply take the squares of the difference in the values of corresponding pixels between the test image and the prototype pattern. The flaw in such an approach is apparent: a misalignment between otherwise identical images can lead to a large distance. The standard way of dealing with this problem is to use a hand-crafted feature extractor to enhance dissimilarity between patterns of different classes and decrease variability within each class.

Instead Simard and his coworkers [14] modified the *distance measure*, making it invariant against small distortions, including line thickness variations, translations, rotations, scale change, etc. If we consider an image as a point in a high dimensional pixel space, where the dimensionality equals the number of pixels, then an evolving distortion of a character traces out a curve in pixel space. Taken together, all these distortions define a low-dimensional manifold in pixel space. For small distortions, in the vicinity of the original image, this manifold can be approximated by a plane, known as the tangent plane. Simard et al. found that an excellent measure of "closeness" for character images is the distance between their tangent planes. Using this "tangent distance", a high accuracy classifier was crafted for use on the postal data. Tangent Distance was tested on the MNIST images downsampled to 16x16 pixels. An error rate of 1.1% was achieved. Prefiltering techniques using simple Euclidean distance at multiple resolutions allowed to reduce the number of necessary Tangent Distance calculations. The figure for storage requirement assumes that the patterns are represented at multiple resolutions at one byte per pixel.

3.9. LeNet 4 with K-Nearest Neighbors

As an alternative to a smart distance measures like the one used in the TDC, one can seek a change in representation so that Euclidean distance is a good measure of pattern similarity. We realized that the penultimate layer of LeNet 4, which has 50 units, can be used to create a feature vector that is appropriate for a Euclidean distance search. With these features, a 1.1% test error was attained, no improvement over a plain LeNet 4.

3.10. Local Learning with LeNet 4

Bottou and Vapnik [9] employed the concept of local learning in an attempt to get higher classifier accuracy. They had observed that the LeNet family of classifiers performs poorly on rare, atypical patterns, and interpreted this behavior as a capacity control problem. They surmised that the modeling capacity of the network is too large in areas of the input space where the patterns are rare and too small in areas where

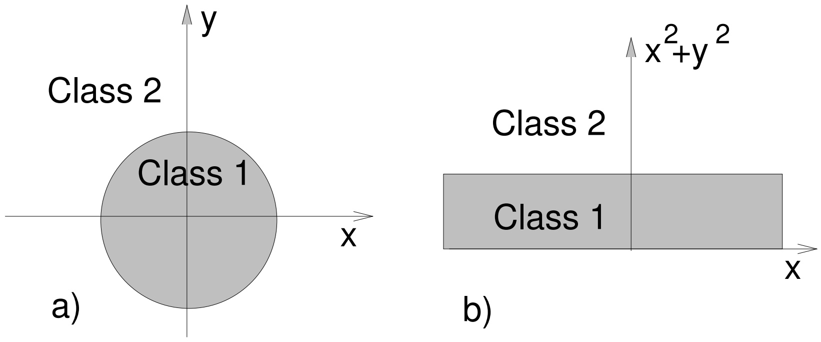

Fig. 4. The Optimal Margin Classifier can transform patterns from an input space in which they are not linearly separable to a new space in which they are linearly separable. a) shows the input space in which class 1 and class 2 are not linearly separable. b) shows the transformed space in which separation is possible.

patterns are plentiful. To alleviate this problem they proposed to retrain a simple linear classifier *every time a new test pattern is presented*. The linear classifier is only trained on the k patterns in the training set that are closest to the current test pattern, thereby producing an ephemeral *local* linear model of the decision surface. The linear classifier operates on feature vectors produced by the penultimate layer of LeNet 4. In order to control the capacity of these linear classifiers, they imposed a weight decay parameter g. The parameters k and g are determined by cross validation experiments. Unlike with previous experiment on the USPS database, the local learning method did not improve the test error rate over the original LeNet 4 on the MNIST test set: 1.1%.

3.11. Optimal Margin Classifier (OMC)

The Optimal Margin Classifier (OMC) is a method for constructing complex decision rules for two-group pattern classification problems. For digit recognition several such classifiers are constructed, each one checking for the presence of a particular digit. One way of constructing complex decision surfaces is to transform the input patterns into higher-dimensional vectors, and then to use a simple linear classifier in the transformed space. A simple example of such a transformation is shown in Figure 4. One classical transformation consists in computing products of k or less input variables. A linear classifier in that space corresponds to a polynomial decision surfaces of degree k in the input space. Unfortunately, this is impractical when the input dimension is large, even for small k. OMC is based on the idea that only certain linear decision surfaces in the transformed space are interesting, namely, the ones that are at the maximum distance from the convex hulls of the two classes. Interestingly, such planes can always be expressed as linear combinations of generalized dot products between the incoming pattern and a subset of training patterns, called the "support vectors". Support vectors in the transformed space are illustrated in

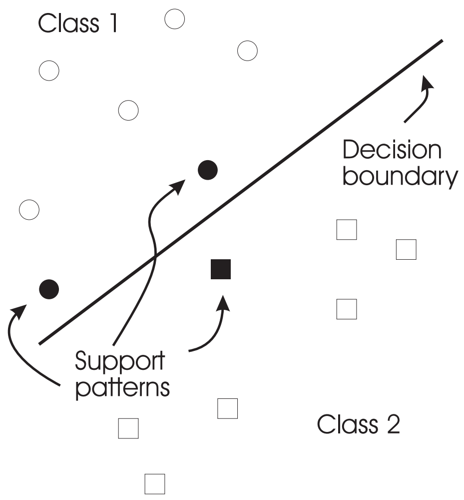

Fig. 5. The support patterns (filled squares and circles) defining the decision boundary are a subset of the training patterns (all squares and circles).

Figure 5. The resulting architecture can be viewed as a 2-layer "neural network" in which the weights of the first layer units are the support vectors. These units compute the dot-product between the input and their weight, and pass the result through a non-linear transformation (for a k-th degree polynomial surface, this transformation is simply an elevation to the kth power). The products are then linearly combined to produce the output. Finding the support vectors and the coefficients amounts to solving a high-dimensional quadratic minimization problem with linear constraints. OMC can be seen as a memory-based technique, since it involves comparing patterns to a set of prototypes, but it is less expensive than pure nearest-neighbor because only a subset of the training set must be stored (about 25,000 prototypes in our case).

The original OMC algorithm, developed by Boser, Guyon, and Vapnik [1] , only succeeds if the training set is linearly separable in the transformed space. The extended version of the technique, called Soft Margin Classifier, was proposed by Cortes and Vapnik to cover the non-separable case, and thus allows for labeling errors in the training set [2]. SMC yields a test error of 1.1% on the MNIST data.

4. Discussion

A summary of the performance of our classifiers is shown in Figures 6 -10. Figure 6 shows the raw error rate of the classifiers on the 10,000 example test set. Although all the classifiers, with the exception of the simple linear classifier, did well on the test set, Boosted LeNet 4 is clearly the best, achieving a score of 0.7%, closely followed by LeNet 5 at 0.9%. This can be compared to our estimate of human performance, 0.2%. Interestingly, substituting the last layer of LeNet 4 with more powerful classifiers did not change the raw accuracy.

Figure 7 illustrates another measure of accuracy, namely the number of patterns

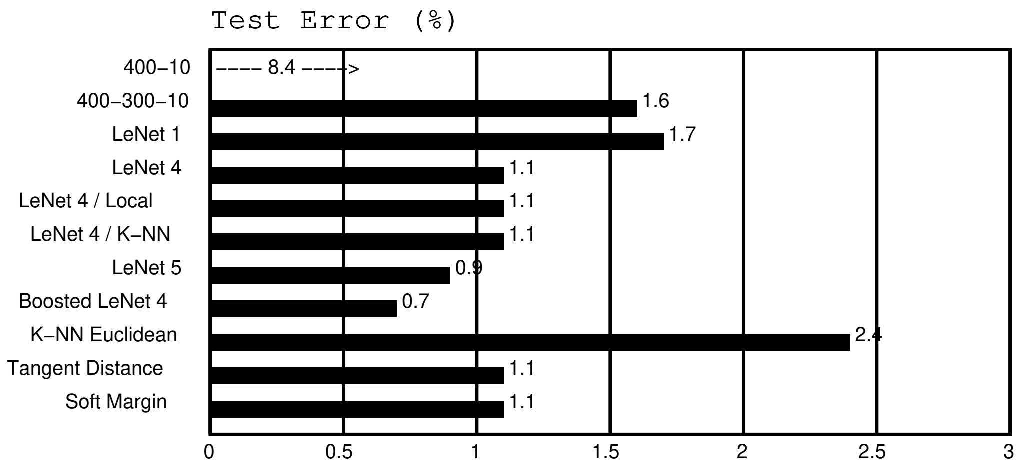

Fig. 6. Performance of classifiers on the MNIST test set. The uncertainty in the quoted error rates is about 0.1%.

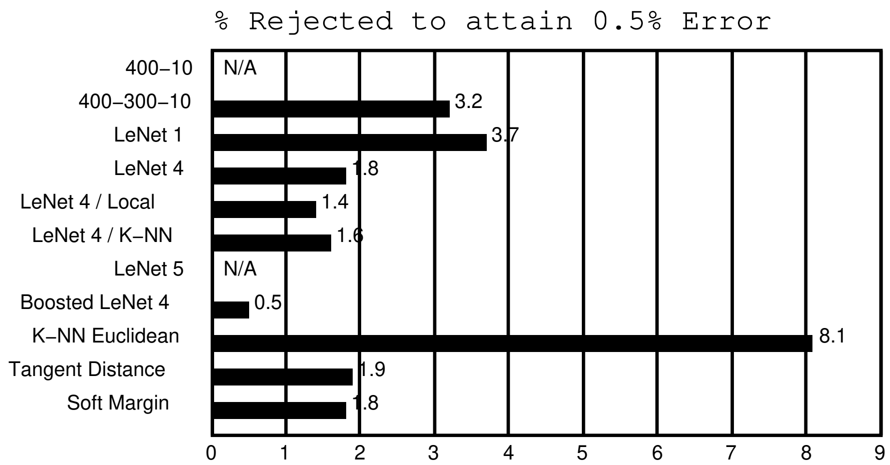

Fig. 7. Percent of test patterns rejected to achieve 0.5% error on the remaining test examples.

in the test set that must be rejected to attain a 0.5% error on the remaining test examples. In many applications, rejection performance is more significant than raw error rate. Again, Boosted LeNet 4 has the best score. The enhanced versions LeNet 4 did better than the original LeNet 4, even though the raw accuracy were identical.

Classification speed is also of prime importance. Figure 8 shows the time required on a Sparc 10 for each method to recognize a test pattern starting with a size-normalized pixel map image. Here we see that there is an enormous variation in speed. Expectedly, memory-based method are much slower than neural networks. The times shown in Figure 8 represent reasonably well-optimized code running on general purpose hardware. Using special purpose hardware, much higher speeds might be attained, provided that the hardware matches the algorithm. Single-board hardware designed with LeNet in mind performs recognition at 1000 characters/sec [12]. Cost-

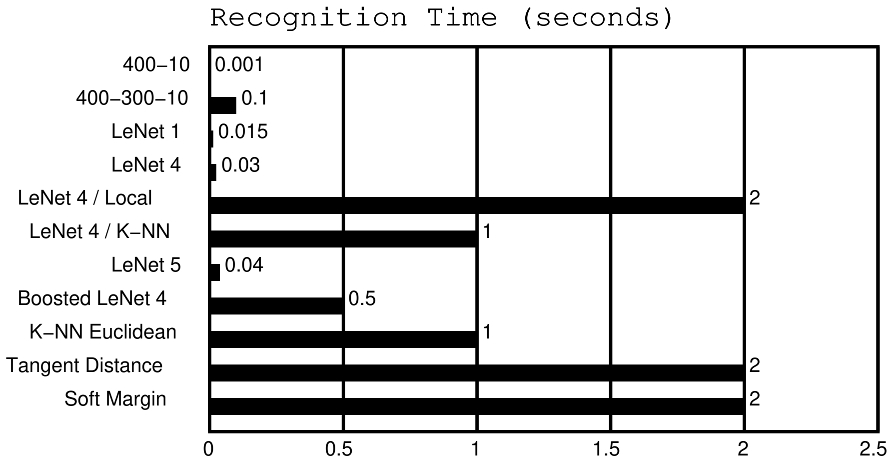

Fig. 8. Time required on a Sparc 10 for recognition of a single character starting with a size-normalized pixel map image.

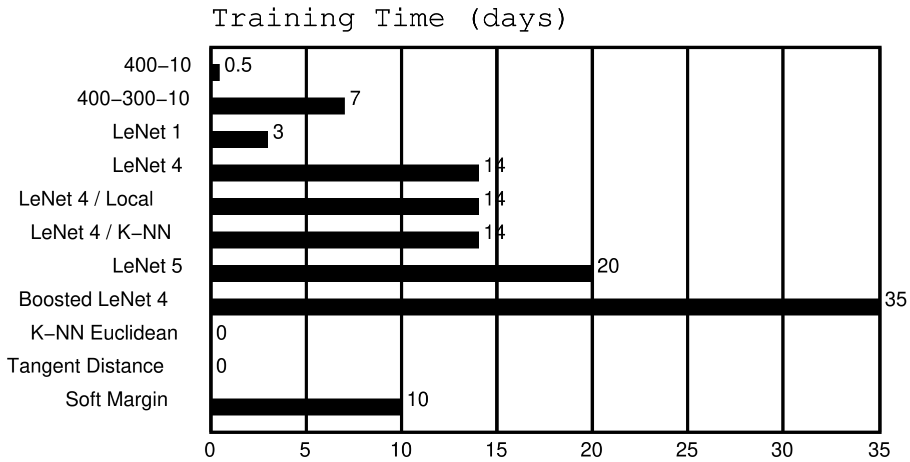

Fig. 9. Training time, in days, on a Sparc 10.

effective hardware implementations of memory-based techniques are more elusive, due to their enormous memory requirements.

Another measure with practical significance is the time required to train the classifiers. For the enhanced LeNet 4, training time is the time required to train the basic LeNet 4 which produces the feature vectors. For the other algorithms, again there is significant variation in the training time. Figure 9 shows the required training on a Sparc 10 measured in days. It is important to note that the training time, though an interesting measure for designers, is almost irrelevant to the customer.

Figure 10 shows a further measure of performance: the memory requirements of our various classifiers. Figures are based on 4 bit per pixel representations of the prototypes for K-Nearest Neighbors, 1 byte per pixel for Soft Margin, and Tangent Distance. They should be taken as upper bounds, as clever compression of the data

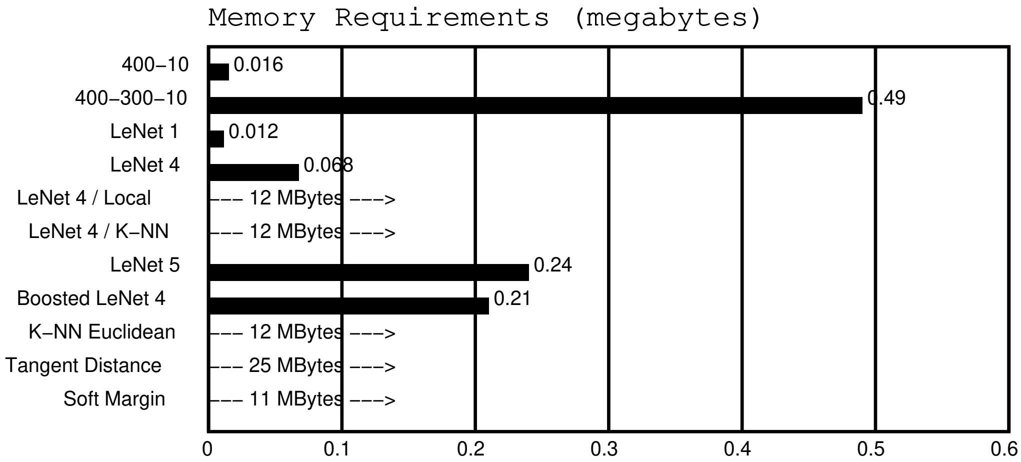

Fig. 10. Memory requirements for classification of test patterns. Numbers are based on 4 bit/pixel for K-NN, 1 byte per pixel for Soft Margin, and Tangent Distance, 4 byte per pixel for the rest.

and/or elimination of redundant training examples can reduce the memory requirements of some of the methods. Memory requirements for the neural networks assume 4 bytes per weight (and 4 bytes per prototype component for the LeNet 4 / memory-based hybrids), but experiments show that one byte weights can be used with no significant change in error rate. Of the high-accuracy classifiers, LeNet 4 requires the least memory.

Many real-world applications require a multi-character recognizer. This can be implemented by using a single-character recognizer to score character candidates provided by a heuristic segmenter. A graph search can be used to find the best consistent interpretation. The recognizers must be designed and trained to find not only the correct character (as discussed above), but also the correct segmentation [3]. To achieve this, the recognizer must be able to classify pieces of ink resulting from erroneous segmentations as non-characters. A big advantage of neural networks is that they can be trained "in the loop" to simultaneously recognize characters and reject non-characters. In addition, segmentation errors can cause small pieces of broken characters to be sent to the recognizer for scoring. If such pieces are size-normalized individually, they may be turned into shapes that the recognizer may have trouble telling apart from real characters. Because of this, characters are *not* individually normalized, rather, the normalization takes place at the string level. This causes large variations in the position and size of individual characters presented to the recognizer. Convolutional neural nets seem to be particularly good at handling such variations.

5. Conclusions

This paper is a snapshot of ongoing work. Although we expect continued changes in all aspects of recognition technology, there are some conclusions that are likely to remain valid for some time.

Performance depends on many factors including high accuracy, low run time, low memory requirements, and reasonable training time. As computer technology improves, larger-capacity recognizers become feasible. Larger recognizers in turn require larger training sets. LeNet 1 was appropriate to the available technology five years ago, just as LeNet 5 is appropriate now. Five years ago a recognizer as complex as LeNet 5 would have required several months' training, and more data than was available, and was therefore not even considered.

For quite a long time, LeNet 1 was considered the state of the art. The local learning classifier, the optimal margin classifier, and the tangent distance classifier were developed to improve upon LeNet 1 – and they succeeded at that. However, they in turn motivated a search for improved neural network architectures. This search was guided in part by estimates of the capacity of various learning machines, derived from measurements of the training and test error (on the large MNIST database) as a function of the number of training examples. We discovered that more capacity was needed. Through a series of experiments in architecture, combined with an analysis of the characteristics of recognition errors, LeNet 4 and LeNet 5 were crafted.

We find that boosting gives a substantial improvement in accuracy, with a relatively modest penalty in memory and computing expense. Also, distortion models can be used to increase the effective size of a data set without actually taking more data.

The optimal margin classifier has excellent accuracy, which is most remarkable, because unlike the other high performance classifiers, it does not include knowledge about the geometry of the problem. In fact, this classifier would do just as well if the image pixels were encrypted e.g. by a fixed, random permutation. It is still much slower and memory hungry than the convolutional nets. However, the technique is relatively new, therefore there is some room for improvement.

When plenty of data is available, many methods can attain respectable accuracy. Although the neural-net methods require considerable training time, trained networks run much faster and require much less space than memory-based techniques. The neural nets' advantage will become more striking as training databases continue to increase in size.

6. References

1. B. E. Boser, I. Guyon, and V. N. Vapnik, *A Training Algorithm for Optimal Margin Classifiers,* in Proceedings of the Fifth Annual Workshop on Computational Learning Theory **5** 144-152, Pittsburgh (1992).
2. Corinna Cortes and Vladimir Vapnik, *The Soft Margin Classifier,* Machine Learning, to appear (1995).

3. John S. Denker and Christopher C. J. Burges, *Image Segmentation and Recognition,* in The Mathematics of Induction, D. H. Wopert (ed.), Addison-Wesley (1994).
4. R. O. Duda and P. E. Hart, *Pattern Classification and Scene Analysis*, Chapter 4, John Wiley and Sons (1973).
5. S. Geman, E. Bienenstock, and R. Doursat, *Neural Networks and the Bias/Variance Dilemma,* Neural Computation **4** 1-58 (1992).
6. H. Drucker, R. Schapire, and P. Simard, *Boosting Performance in Neural Networks,* International Journal of Pattern Recognition and Artificial Intelligence **7** 705-720 (1993).
7. I. Guyon, I. Poujaud, L. Personnaz, G. Dreyfus, J. Denker, and Y. LeCun, *Comparing Different Neural Net Architectures for Classifying Handwritten Digits,* in Proc. 1989 IJCNN II 127-132, Washington DC. IEEE, (1989).
8. Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, R. Hubbard, and L. D. Jackel, *Handwritten digit recognition with a back-propagation network*, in D. Touretzky (ed), Advances in Neural Information Processing Systems **2**, Morgan Kaufman, (1990).
9. Léon Bottou and Vladimir Vapnik, *Local Learning Algorithms,* Neural Computation **4**, 888-900 (1992).
10. U. A. Müller, Bernhard Baumle, P. Kohler, A. Gunzinger, and W. Guggenbuhl, *Achieving Supercomputer Performance for Neural Net Simulation with an Array of Digital Signal Processors,* IEEE Micro Magazine, 55-65, October (1992).
11. M. L. Minsky and S. Papert, *Perceptrons*, MIT Press, Cambridge Mass. (1969).
12. Eduard Säckinger and Hans Peter Graf, *A System for High-Speed Pattern Recognition and Image Analysis*, Proc of the fourth International Conference on Microelectronics for Neural Networks and Fuzzy Systems, IEEE (1994).
13. R. Schapire, *The Strength of Weak Learnability*, Machine Learning **5** 197-227 (1990).
14. Patrice Y. Simard, Yann LeCun, and John Denker, *Efficient Pattern Recognition Using a New Transformation Distance,* Neural Information Processing Systems **5**, 50-58, Morgan Kaufmann (1993).
15. V. N. Vapnik, *Estimation of Dependencies Based on Empirical Data*, Springer-Verlag (1982).