# Comparison of Kernel Estimators, Perceptrons, and Radial-Basis Functions for OCR and Speech Classification

Ethem Alpaydın, Fikret Gürgen
Department of Computer Engineering, Boğaziçi University, TR-80815 Istanbul Turkey
alpaydin@boun.edu.tr

*Neural Computing & Applications* (1995) 3: 38-49


*Abstract*—We compare kernel estimators, single and multi-layered perceptrons and radial-basis functions for the problems of classification of handwritten digits and speech phonemes. By taking two different applications and employing many techniques, we report here a two-dimensional study whereby a domain-independent assessment of these learning methods can be possible. We consider a feed-forward network with one hidden layer. As examples of the local methods, we use kernel estimators like $k$-nearest neighbor ($k$-nn), Parzen windows, generalized $k$-nn, and Grow and Learn (Condensed Nearest Neighbor). We have also considered fuzzy $k$-nn due to its similarity. As distributed networks, we use linear perceptron, pairwise separating linear perceptron, and multilayer perceptrons with sigmoidal hidden units. We also tested the radial-basis function network which is a combination of local and distributed networks. Four criteria are taken for comparison: Correct classification of the test set, network size, learning time, and the operational complexity. We found that perceptrons when the architecture is suitable, generalize better than local, memory-based kernel estimators but require longer training and more precise computation. Local networks are simple, learn very quickly and acceptably, but use more memory.

**Keywords**— Kernel Estimators; Perceptrons; Backpropagation; Radial-Basis Functions; Optical Character Recognition; Speech Recognition

## I. Introduction

Optical character recognition (OCR) and speech recognition are two tasks that have evident economic utility in improved person-machine communication. These two tasks have a number of common problems. One is the segmentation of character images/phonemes from a text image/speech. This is especially a problem when cursive handwriting/continuous phrases is the case. Another problem is writer/speaker independence. Recognition is also dependent on the domain. If only postal codes are to be optically recognized, then in most countries, only digits are valid classes. To minimize errors and rejects, more complicated tasks require also the integration of a vocabulary so as to be able to use lexical information to aid in recognition. This creates the problems of storing a large lexicon and accessing it efficiently.

Feedforward type neural networks are used for pattern classification. The input, a scanned bitmap image/preprocessed speech signal, is fed to the input layer. There may be a layer of feature extraction by the so-called hidden layer of the network and there are as many output units as there are classes to recognize, of which, only one should be active as classes are mutually exclusive.

### A. Literature Survey

Optical character recognition is one of the most popular applications of neural networks. This is both due its utility and its being a relatively easier before more complicated vision applications are attempted. An early review of neural character recognition systems is given in [2]. One early study is by Guyon et al. [17], where several neural recognizers are compared with classical methods like nearest neighbor and Parzen windows [11] in recognizing handwritten digits for automatic reading of postal codes. le Cun et al. [9] uses the back-propagation learning rule on multi-layer perceptrons [33] for recognizing handwritten digits. Matan et al. [27] explain a complete system with segmentation, noise and skew removal integrated with the neural recognizer given in [9]. An extensive study of backpropagation for optical recognition of both handwritten letters and digits is given by Martin and Pittman [26]. Neocognitron is another neural network used for recognition of digits [12]. A comparison of $k$-nn [11], back-propagation, and radial-basis functions [29] in recognizing handwritten digits is given by Lee [24]. A recent review of the task and several neural and conventional approaches are given by Senior [35].

Lippmann [25] and Waibel and Hampshire [38] give two reviews on using neural networks for speech recognition. Early work used recurrent neural networks for representation of temporal context but after the introduction of Time Delay Neural Networks by Waibel at al. [37], a feed-forward network can also be used for phoneme recognition. Lee and Lippmann [23], and Ng and Lippmann [30] for the same two artificial and two speech tasks compare a large number of conventional and neural pattern classifiers.

### B. Classification Methods

The classifiers considered are:

1. Kernel estimators [36] like $k$-nearest neighbor ($k$-nn) [11], Parzen windows, generalized $k$-nn, and Grow and Learn [2], which is a variant of the condensed nearest neighbor [18]. We have also tested fuzzy $k$-nn [21] which is quite similar to these methods.
2. Linear and pairwise separating perceptrons trained using delta rule [11] or multi-layer perceptrons with sigmoid hidden units trained using back-propagation [33].
3. Radial-basis functions [29] which is a two layer network with Gaussian hidden units.

### C. Performance Measures

The four criteria we used for comparing these methods are:

1. Generalization accuracy, measured in percent correct classification of patterns unseen during training,
2. Learning time, measured in epochs made over the training set,
3. Network size, measured in the number of modifiable connections times the number of bits required to store each connection weight with enough precision, and
4. Operational complexity, that takes into account the arithmetic precision required and ease of implementation, software or hardware.

Most work take generalization as the only quality measure but practical considerations like learning time, memory requirements, and implementation complexity are as much important when choosing one network model over another.

During classification, we have not added a "reject" option. Values reported are success percentages and the error percentages are obtained by complementing to 100%. Many of these methods depend on certain initial settings, e.g., initial weights, training set order, etc. To get rid of this dependence, we run each method twenty times and values reported are averages and standard deviations of these runs. We would like the average to be large if the measured value is success and small if it is the number of training epochs or the number of hidden units; in all cases, we desire the standard deviation to be as small as possible as it is the measure of uncertainty.

### D. Databases

The database we use for digit recognition has 1200 handwritten examples of 10 digits by 12 different writers. Each example is an isolated, normalized 16 by 16 binary image. One half is used for training and the second half for testing the network models. Other results on the same database are given in [17], [16].

For /b,d,g,m,n,N/ phoneme recognition experiments, the database contains 5240 Japanese isolated words and phrases. One hundred samples for each class (total of 600 samples) are taken from the even-numbered and odd-numbered words for training and testing. Phrases are used only for testing after having trained on isolated words. As it is known, the speaking style and speed of phrases differ from the isolated words. Phonemes are extracted from hand-labelled discrete word utterances and phrase utterances which have a sampling frequency of 12 KHz. 7 speech frames (each 10 ms) are used as input. For each 10 ms frame, 16 Mel-scaled FFT coefficients are computed as feature values. The final input fed to the neural network thus has 112 dimensions.

### E. Outline of the Paper

In the next section, we talk briefly about learning and generalization in neural networks. In section 3, we classify network architectures as local and distributed. Section 4 contains discussion of four different local architectures. Distributed architectures are explained in section 5. A combination of local and distributed approaches, named the radial-basis function network, is discussed in section 6. We discuss our results and conclude in section 7.

## II. Learning and Generalization

The setting for approximation is as follows: We are asked to map a multi-dimensional input vector $\mathbf{x}$ to one output $f(\mathbf{x})$. We do not know $f(\mathbf{x})$ but we are given a set of examples $\{\mathbf{x}, y\}$ on which to train our approximation function. Generally, there is error in reception in that $y = f(\mathbf{x}) + \eta$ where $\eta$ is taken as zero mean, input independent "noise." Classification is a special case where $f(\mathbf{x})$ is a binary function that denotes membership of class $\mathcal{C}$, i.e., $f(\mathbf{x})$ is 1 if $\mathbf{x} \in \mathcal{C}$, and 0 otherwise. This framework can be generalized to the multiple output case by defining separate $f^k(\mathbf{x})$ for different outputs. The function we use for approximation is of the form $\mathcal{F}(\mathcal{P}, \mathbf{x})$ where $\mathcal{F}()$ is the model and $\mathcal{P}$ is the set of parameters.

Learning in this context corresponds to the computation of $\mathcal{P}$ values to minimize the error between $y_i$ and $\mathcal{F}(\mathcal{P}, \mathbf{x}_i)$ over all training example pairs, $\{\mathbf{x}_i, y_i\}$ using a distance measure, e.g., sum of square errors:

$$E(\mathcal{P}) = \sum_{\{\mathbf{x}_i, y_i\}} [y_i - \mathcal{F}(\mathcal{P}, \mathbf{x}_i)]^2 \qquad (1)$$

In the case of neural networks, $\mathcal{F}()$ is defined by the network architecture, i.e., the number of hidden layers, the number of hidden units, connectivity between layers, etc. $\mathcal{P}$ corresponds to the modifiable "weights" of the connections of this network. Normally when the number of free parameters increases, the network's memorization ability increases, increasing the network's performance over the training set. But beyond a certain value, increasing further the complexity leads to a decrease in the "generalization" power of the neural network. Generalization is the ability of a system to give good outputs to inputs it has not been trained on. It is desired because in applications from the continuous domain, one can not train a system using all possible inputs. In such cases, we would like the system to be able to interpolate meaningfully from training examples to new cases.

Thus during learning, one wants to minimize both the error and the network complexity, thus Eq. (1) should be extended as:

$$E(\mathcal{P}) = \sum_{\{\mathbf{x}_i, y_i\}} [y_i - \mathcal{F}(\mathcal{P}, \mathbf{x}_i)]^2 + \gamma|\mathcal{F}| \tag{2}$$

where $|\mathcal{F}|$ denotes network complexity. This is in line with Occam's razor which states that simpler models are to be preferred over complex models unless the additional complexity is justifiable by improved performance on a separate cross-validation test set. This rule of thumb of minimizing cost is also good engineering. Simple models incorporate a large amount of bias because making something simple implies making a lot of assumptions and choices, and each choice made is one additional source of bias. Such simple models can perform well only if the assumptions and choices made are appropriate for the task. Models with large number of free parameters can approximate a larger distribution of functions sufficiently well; the cost paid is the large amount of training data required to effectively constrain the model so as to make it approximate the function inherent in the training samples. Our approach here is trial and error to determine the optimal number of free prameters. Another approach is the incremental modification of the network structure by piecewise adding/removing units/links (See review in [3]). Still another is to train many and take a vote where weights in vote are computed taking into account the success and complexities of the voting networks [5]. We have not tried these two latter approaches in the present article.

## III. Local vs. Distributed Network Architectures

The type of neural networks we are interested in for pattern classification are of feedforward type with one hidden layer. The output is a linear sum of a number of nonlinear basis functions:

$$\mathcal{F}(T, W, \mathbf{x}) = \sum_{h=1}^{H} T_h g_h(W_h, \mathbf{x}) + T_0 \tag{3}$$

where $g_h(W_h, \mathbf{x})$ are the basis functions, $T_h$ are their weights and $T_0$ is a threshold. This can be implemented as a two layer network where $g_h(W_h, \mathbf{x})$ denote the hidden unit activations, $T_h$ are the weights of the connections from the hidden units to the output unit and $T_0$ is the weight from the bias unit.

Depending on the form of the function $g(W_h, \mathbf{x})$ computed in its hidden units, a neural network is said to be *local* or *distributed*:

- In local neural networks, basis functions are local. Examples are the kernel estimators, and clustering methods. Each basis function has a certain reference point or center in the input space and its output is a function, $\phi()$, of the distance between the input and its center.

$$g_h(W_h, \mathbf{x}) = \phi(||W_h - \mathbf{x}||) \tag{4}$$

  where $||\cdot||$ is the Euclidean norm. This type of a network is also called *memory-based* [4] as $W$ make up a memory and the network basically functions like a table look-up. The $W$ and $T$ together make up the parameters of the model which we named $\mathcal{P}$ above.

- When the network is distributed, dot product is used as the distance measure instead of the Euclidean distance. The non-linearity $g(W_h, \mathbf{x})$ is the sigmoid function which is a smoothed version of thresholding:

$$g_h(W_h, \mathbf{x}) = \frac{1}{1 + \exp\left(-\sum_i W_{hi}\mathbf{x}_i - W_{h0}\right)} \tag{5}$$

  where $W_{hi}$ are the weights of the connections from the input units to the hidden units and $W_{h0}$ are the thresholds of the hidden units. Each basis function defines a hyperplane in the input space and the output is determined based on the distance of the input to the hyperplane. Because this hyperplane divides the whole input space into two, units respond globally and for a given input, more than one hidden unit may be active. In this case, the network is a distributed one as it forms a distributed representation of the input on its hidden units.

## IV. Local Networks

The simplest type of associative memory is a look-up table. Inputs and associated outputs can be stored in a memory and when an input is given, it is searched for in the table and the corresponding output is given. This is a universal learning method. Just as any boolean function can be learned as a disjunction of conjunctions, any continuous function can be approximated to a required precision by a juxtaposition of piecewise constant functions [6]. There is no reasonable upper bound on the size of the table. We can divide local or memory-based methods into three:

- *Kernel estimators.* The table size is as large as the training set and we interpolate from the training samples directly. This is basically kernel estimation of class densities [36]. K-nearest neighbors and Parzen windows discussed in the next two subsections are of this type.
- *Clustering methods.* When the training set is large, decreasing table size is possible by removing redundancy. One way is to define a criterion of importance for samples and to store only the important samples in the table. One example is the Grow and Learn method which we discuss in section 4.3. Another way to achieve data reduction is by clustering as in Learning Vector Quantization [22] or Adaptive Resonance Theory [7]. Yet another way is to extract the statistically important features, like the principal components, and make classification based on these; this technique is beyond the scope of this paper.
- *Sparse tables.* Here the table is much larger than the training set. The samples are not stored directly but are duplicated in several places where they "blend" with other samples and this causes interpolation to similar data. CMAC [28] and Sparse Distributed Memory [20] are examples. We have not tested these methods in this present article.

### *A. Kernel Estimation of Class Densities*

In parametric estimation, we assume the knowledge of a certain form of density family whose parameters we estimate from the data. In the nonparametric case, we directly estimate the entire density function. Then we need a large sample for our estimate not to be biased by the particular sample we use. Let us say we are given a sample of $n$ real $d$-dimensional observations $\mathbf{X}_1, \ldots, \mathbf{X}_n$. These samples are labelled as belonging to one of the possible classes $\omega_j$. The symbol $\hat{p}(\mathbf{x}, \omega_j)$ denotes our estimator of the joint sample, class density, $p(\mathbf{x}, \omega_j)$. The *kernel estimator* is given as

$$\hat{p}(\mathbf{x}, \omega_j) = \frac{1}{nh^d} \sum_{i=1}^{n} K\left(\frac{\mathbf{x} - \mathbf{X}_i}{h}\right) m_j(\mathbf{X}_i) \qquad (6)$$

where $m_j(\mathbf{x})$ is 1 if $\mathbf{x} \in \omega_j$ and 0 otherwise. $h$ is the *window width* or the *smoothing parameter.* Then using Bayes rule, we can compute the posterior probabilities:

$$P(\omega_j|\mathbf{x}) = \frac{p(\mathbf{x}, \omega_j)}{\sum_k p(\mathbf{x}, \omega_k)} \qquad (7)$$

For minimum error rate, we select the category having the highest posterior probability:

$$c = \mathrm{ArgMax}_k \left[P(\omega_k|\mathbf{x})\right] \qquad (8)$$

Depending on the shape of $K$, one can have various estimators [36]:

TABLE I

Percent recognition when Parzen windows is used as a function of $h$, the window width. Generalization is sensitive to $h$, with both small and large values of $h$ decreasing success.

| OCR | $h$ | | | | | |
|---|---|---|---|---|---|---|
| | 10 | 15 | 20 | 25 | 30 | 35 |
| Success | 93.5 | 93.8 | 94.2 | 92.7 | 90.3 | 87.7 |

| Speech | $h$ | | | | | | |
|---|---|---|---|---|---|---|---|
| | 0.6 | 0.7 | 0.8 | 0.9 | 1.0 | 1.1 | 1.2 |
| Test | 90.2 | 90.7 | 90.3 | 90.2 | 88.8 | 87.8 | 86.7 |
| Phrase | 67.2 | 68.7 | 68.2 | 67.9 | 67.1 | 65.3 | 63.1 |

- *Parzen windows.* For $\hat{p}$ to be a legitemate density function, $K$ should be nonnegative and integrate to one. For a smooth approximation, $K$ is generally taken to be the normal density [11]:

$$K(\mathbf{u}) = \left(\frac{1}{\sqrt{2\pi}}\right)^d \exp\left[-\frac{||\mathbf{u}||^2}{2}\right] \qquad (9)$$

  Here the kernel estimator is a sum of "bumps" placed at the observations. $K$ determines the shape of the bumps while $h$ determines their widths.

- *K-nearest neighbor (k-nn).* Let us denote the $k$th nearest sample to $\mathbf{x}$ as $\mathbf{x}^{[k]}$ and let $V^k(\mathbf{x})$ be the volume of the $d$-dimensional sphere of radius $r^k \equiv ||\mathbf{x} - \mathbf{x}^{[k]}||$; thus $V^k(\mathbf{x}) = r^k c_d$ where $c_d$ is the volume of the unit sphere in $d$ dimensions, e.g., $c_1 = 2$, $c_2 = \pi$, $c_3 = 4\pi/3$, etc. If out of the $k$ neighbors, $k^j$ of them are labelled $\omega_j$, then the $k$-nearest neighbor estimate is

$$\hat{p}(\mathbf{x}, \omega_j) = \frac{k^j}{nV^k(\mathbf{x})} = \frac{k^j}{nr^k c_d} \qquad (10)$$

  and thus $P(\omega_j|\mathbf{x}) = k^j/k$.

- *Generalized k-nn.* One can combine $k$-nn and Parzen windows by having smooth kernels with varying $h$. In the generalized $k$-nn, $h$ of Eq. (6) is taken as the distance to the $k$th nearest sample, $||\mathbf{x} - \mathbf{x}^{[k]}||$. Then $\hat{p}$ is a density function, if $K$ is. Thus the overall smoothing is governed by the choice of $k$, and the window width depends on the density in the neighborhood of $\mathbf{x}$.

Results with kernel estimators are given in Tables I and II for respectively Parzen windows and $k$-nn variants. In digit recognition, we used Hamming Distance as the metric which is the number of differing bits and in phoneme recognition we used the Euclidean distance. The correct choice of $k$ or $h$ is critical. If they are large, then even distant neighbors affect the density at $\mathbf{x}$ leading to a very smooth estimate. If $h$ or $k$ is small, $\hat{p}$ is the superposition of $n$ sharp pulses centered at the samples and is a "noisy" estimate.

With Parzen windows, when $h$ is small with a small sample, it is possible that no samples fall in the kernel leading to a zero estimate; $k$-nn guarantees that $k$ samples fall in the kernel. Small or large $h$ leads to a decrease in success. When $h$ is small, there are few samples and when it is large, there are too many. The good $h$ value is to be found using trial-and-error. In the case of generalized $k$-nn, instead of using one global $h$, we use different values depending on the probability density in the region.

For the $k$-nearest neighbor, it has been shown [8] that the performance of the 1-nearest neighbor in classification is never worse than twice the Bayesian risk where complete knowledge of the distributions are assumed. It can thus be said that at least half of this knowledge is provided by the nearest neighbor. The performance can be improved by increasing the number of neighbors considered whose error asymptotically converges to the Bayes risk.

When the samples are noisy, we expect $k$-nn with small $k$ not to perform well. Large $k$ takes into account the effect of very distant samples and thus is not good either. Generalized $k$-nn and fuzzy $k$-nn that we discuss in the next section, because they give less weight to distant neighbors perform better than the $k$-nn proper.

When $h$ (or $k$) is decreased to decrease bias, variance increases and vice versa. This can intuitively be explained as follows [16]. When $h$ (or $k$) is large, $\hat{p}$ is the weighted average of many samples and thus does not change much from one sample to another. The variance contribution is small, however response is biased toward the population response (In the extreme case, when $h \to \infty$, $\hat{p}$ is the sample average and is independent of the input $\mathbf{x}$). On the other extreme, when $h$ is small, there is small bias but the response is dependent on the particular sample used, therefore the variance contribution is high. The same argument holds with $k$-nn, with $k$ in place of $h$. Choosing $h$ or $k$ implies a trade-off between bias (systematic error) and variance (random error).

### B. Fuzzy k-nn

A related work is the "fuzzy" $k$-nn [21] where the computation of the fuzzy memberships is similar to the computation of class densities above. The fuzzy membership of input $\mathbf{x}$ to class $j$ is computed as:

$$u_j(\mathbf{x}) = \frac{\sum_{i=1}^{n} F(\mathbf{x} - \mathbf{X}_i) m(\mathbf{X}_i)}{\sum_{i=1}^{n} F(\mathbf{x} - \mathbf{X}_i)} \tag{11}$$

where the denominator is the normalizer to have $0 \leq u_j(\mathbf{x}) \leq 1$ and the function $F$ similarly decreases as distance between $\mathbf{x}$ and $\mathbf{X}_i$ increases, i.e., $F(\mathbf{u}) = 1/\|\mathbf{u}\|^{2/(m-1)}$. We use $m = 2$. When $m$ is larger, neighbors have equal effect regardless of their distance.

TABLE II

Percent recognition when $k$-nn is used as a function of $k$, the number of neighbors on the two test sets. g-$k$-nn is generalized $k$-nn and f-$k$-nn is fuzzy $k$-nn. Note that although $k$-nn falls off with increasing $k$, its variants that weigh neighbors' effects reciprocally with distance to the input seem more robust. 'n/a' indicates that the combination is not applicable.

| OCR | k | | | | | | |
|---|---|---|---|---|---|---|---|
| | 1 | 2 | 3 | 4 | 6 | 8 | 10 |
| $k$-nn | 93.3 | 90.8 | 94.2 | 92.3 | 92.5 | 91.0 | 90.8 |
| g-$k$-nn | na | 93.2 | 94.3 | 94.3 | 94.8 | 94.2 | 94.3 |
| f-$k$-nn | na | 92.8 | 94.3 | 93.3 | 93.8 | 93.0 | 93.5 |

| Speech | | k | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | | 1 | 2 | 3 | 4 | 6 | 8 | 10 |
| Test | $k$-nn | 90.2 | 88.7 | 89.0 | 89.2 | 88.5 | 88.2 | 87.0 |
| | g-$k$-nn | na | 90.2 | 89.8 | 90.0 | 90.8 | 91.3 | 91.3 |
| | f-$k$-nn | na | 90.2 | 90.0 | 89.0 | 89.7 | 89.2 | 89.0 |
| Phrase | $k$-nn | 67.4 | 63.0 | 62.5 | 64.6 | 62.8 | 63.8 | 62.2 |
| | g-$k$-nn | na | 67.4 | 65.7 | 66.9 | 67.1 | 67.5 | 67.2 |
| | f-$k$-nn | na | 67.4 | 64.1 | 65.9 | 65.0 | 66.0 | 64.6 |

Results achieved with this method are also given in Table II. This approach of weighting distant neighbors less prove also to be a good heuristic empirically.

### C. Grow and Learn (GAL)

The $k$-nn becomes infeasible when the training set is large. One possibility in minimizing the number of stored samples is by storing only a subset of the training set. This subset, named the consistent subset [18], contains those samples that can classify the remaining samples correctly using the nearest neighbor rule. There are many consistent subsets and one normally is interested in the minimal consistent subset, i.e., the subset with the minimum cardinality. Finding this corresponds to minimizing the following error measure [4] analogous to Eq. (2):

$$E(W) = \sum_{\mathbf{x}_i} L(\mathbf{x}_i) + \gamma |W| \tag{12}$$

$$L(\mathbf{x}_i) = \begin{cases} 1, & \text{if } \|W_c - \mathbf{x}_i\| = \min_j \|W_j - \mathbf{x}_i\| \text{ and} \\ & class(\mathbf{x}_i) \neq class(W_c); \\ 0, & \text{otherwise.} \end{cases} \tag{13}$$

$L(\mathbf{x}_i)$ is non-zero when sample $\mathbf{x}_i$ cannot be classified correctly by the stored subset, $W$. The first term in the cost measure is due to the error in classification and the second indicates the size of the subset stored. $\gamma$ indicates the relative importances of these two factors. Minimizing both leads to the minimal consistent subset. We believe however that finding the optimal subset is an NP-complete problem and thus algorithms with feasible complexity will yield sub-optimal solutions.

The Condensed nearest neighbor rule [18] or the Grow and Learn algorithm [2] or instance-based learning (e.g. IBL2 in [1]) passes one by one over the samples and adds

TABLE III

Performance results (average $\pm$ standard deviation) with the Grow and Learn method. Note that we make a few training epochs and store only a small part of the training set which is sufficient to get us to an acceptable level of performance.

| OCR | Units | Epochs | Train | Test |
|---|---|---|---|---|
| GAL | 127.0±6.3 | 3.9±1.2 | 100.0±0.0 | 89.0±1.1 |

| Speech | Units | Epochs | Train | Test | Phrase |
|---|---|---|---|---|---|
| GAL | 196.1±5.9 | 4.7±1.1 | 100.0±0.0 | 85.5±1.0 | 60.1±1.9 |

a sample to the subset only if it cannot be classified correctly with the already stored samples. The classification rule is 1-nn but other $k$-nn variants are also possible at the expense of more computation. This of course does not guarantee finding the minimal subset and furthermore, different subsets are found when the training set order is changed. It is shown [3] that the bias introduced due to particular training set orders can be decreased by training multiple GAL networks with the same training set but ordered differently and taking a vote.

The results achieved using this method is given in Table III. Note that by making a few sweeps over the training set and by storing only one fourth of the training set, one can achieve reasonably well performance. This method is promising when very rapid adaptation is necessary, e.g., in real-time systems like robotics, or when memory space is a premium.

## V. Distributed Networks

In the case of distributed networks, units do not respond locally but define hyperplanes that divide the input space into two. Linear perceptrons have no hidden layer. Pairwise separators have a hidden layer of hidden units with defined roles. In multi-layer perceptrons, hidden units learn.

The most popular method for training distributed networks is gradient-descent on a cost function like given in Eq. (1). This is named delta rule for the single layer case and back-propagation in multi layers. Given a differentiable error function $E$, we start with any set of weights and repeatedly change each weight by an amount proportional to $\partial E/\partial \mathcal{P}$:

$$\Delta \mathcal{P}_p = -\eta \frac{\partial E}{\partial \mathcal{P}_p} \tag{14}$$

where $\eta$ is the stepsize in descent. We use a momentum term and update learning rate during training for faster convergence. In the case of digit recognition, training stops when error per pattern falls below 0.0001 and for speech recognition, we stop when error fails to decrease any further in which case we take the "best so far."

### A. Linear Perceptrons

A linear perceptron computes just a weighted sum without any non-linearity. Its neural network implementation has only one layer of modifiable weights. When used for classification, we assume classes to be linearly separable. Perceptron for class $c$ is defined as follows:

$$\mathcal{F}^c(W, \mathbf{x}) = \sum_i W_i^c \mathbf{x}_i + W_0^c \tag{15}$$

### B. Pairwise Separation of Classes

The linear perceptron defines a hyperplane that separates all the positive examples of a class from all its negative examples. When classes are not linearly separable, one can use pairwise separation where for each pair of classes we have a linear perceptron [11]. When we have $n$ classes, we need $n(n-1)/2$ perceptrons. The class regions are convex and singly connected in such a case.

### C. Multi-Layer Perceptrons (MLP)

In the case of multi-layer perceptrons, we have a feedforward network with one hidden layer as given in Eq. (3) with the sigmoid basis function given in Eq. (5). It has been shown [13],[19] that this type of a neural network is a universal approximator, i.e., can approximate any continuous function with desired accuracy. It has also been shown [34] that multi-layer perceptrons estimate Bayesian posterior probabilities, building a link between multi-layer distributed networks and local kernel estimators. These theorems do not tell how many hidden units are necessary, so one should try several alternatives and choose the best.

Results achieved on the two applications using linear, pairwise and multi-layer perceptrons are given in Table IV. The back-propagation network converges to different minima starting from different initial weights, thus we run each network twenty times and report average and standard deviations. Both training sets are linearly separable but does not generalize to the test sets. Pairwise separating perceptron is better than the linear perceptron and is almost as good as the multi-layer perceptrons in terms of success. Note however that pairwise separating may not be a good idea when we have many classes as the number of hidden units is on the order of the square of the number of classes. Multi-layer perceptrons are good but because we train two layers at each iteration as opposed to one layer in single and pairwise cases, the computational complexity of training is more. Digit recognition seems to be easy but in the recognition of speech phonemes, training a multi-layer perceptron seems to require more epochs. This is because the hidden units have global response and many iterations with small weights updates are necessary as when we are updating a hyperplane to correct in one

TABLE IV

Performance results with the distributed methods: Linear, pairwise separating, and multi-layer perceptrons with one hidden layer of various number of hidden units. Values given are average and standard deviations of multiple runs starting from different initial weights. Pairwise separation seems to be promising as both success is high, computationally simple as we train only one layer, and memory requirement is low as we have a small number of classes.

| OCR | Units | Epochs | Train | Test |
|---|---|---|---|---|
| Linear | n/a | 14.1±3.1 | 100.0±0.0 | 89.2±1.2 |
| Pairwise | 45 | 22.2±3.1 | 100.0±0.0 | 95.3±0.5 |
| One hidden | 10 | 8.6±2.8 | 100.0±0.1 | 94.2±1.1 |
| | 20 | 6.4±2.0 | 100.0±0.0 | 95.5±0.8 |
| | 30 | 5.3±1.1 | 100.0±0.1 | 95.6±0.7 |
| | 40 | 5.5±1.1 | 100.0±0.1 | 95.3±0.9 |
| | 50 | 5.4±1.3 | 100.0±0.0 | 95.5±0.9 |

| Speech | Units | Epochs | Train | Test | Phrase |
|---|---|---|---|---|---|
| Linear | n/a | 64.3±7.8 | 98.8±0.3 | 93.6±0.3 | 57.3±0.8 |
| Pairwise | 15 | 26.4±2.7 | 97.7±0.2 | 93.6±0.3 | 67.0±0.5 |
| One hidden | 16 | 82.6±20.8 | 99.7±0.3 | 92.7±1.0 | 65.7±3.0 |
| | 32 | 53.5±11.2 | 99.7±0.2 | 93.1±0.6 | 65.1±3.3 |
| | 48 | 53.2±8.2 | 99.8±0.2 | 93.2±0.5 | 65.5±3.4 |
| | 64 | 53.9±13.3 | 99.8±0.2 | 93.5±0.6 | 65.3±1.5 |

part of the space, we are liable to forget what we have learned in another part.

## VI. Radial-Basis Functions

A radial-basis function network [29], [32] has one hidden layer with hidden units having Gaussian activation functions as opposed to a sigmoid:

$$g_h(W_h, \mathbf{x}) = \exp\left[-\frac{||W_h - \mathbf{x}||^2}{\sigma^2}\right] \qquad (16)$$

Unlike Parzen windows where we have one Gaussian for each sample, in radial-basis function networks we have less. One approach is to use $k$-means clustering [11] to compute the Gaussian centers. Another approach, named the *anchor* method, is to take a subset of the original samples randomly from the training set [24], [15]. Donoeux and Lengellé [10] propose to initialize an MLP with inputs and weights normalized to a constant length (as MLP uses the dot product as the distance measure instead of the Euclidean distance as in RBF). $\sigma$, the spread of Gaussians, is computed as the average of inter-center distances. The second is a single layer perceptron and is trained using the delta rule. Because units have local responses, only a small number of Gaussians are active for each input, thus one generally needs a lot more Gaussians than sigmoids, but learning is faster when only a few units need be updated for an input.

Simulation results with the same problem is given in Table V. We have tried three network sizes using 120, 240, and 360 Gaussians. Because we have a small training set (600 for both) we could not have more Gaussians otherwise there would be no advantage of having an RBF over Parzen windows and because the number of Gaussians is small, $k$-means clustering is necessary. We run each variant twenty times with differently initialized $k$-means and made twenty $k$-means epochs for computing the Gaussian centers. The success values achieved is not large; this is due to having a small number of Gaussians.

TABLE V

Simulation results with the radial-basis function network using three different number of Gaussians. Results given are averages of twenty runs with differently initialized $k$-means. 20 epochs of $k$-means are made to compute the Gaussian centers.

| OCR Units | Epochs | Train | Test |
|---|---|---|---|
| 120 | 20+35.4±8.6 | 95.1±1.2 | 90.9±1.4 |
| 240 | 20+35.6±9.5 | 97.4±0.9 | 93.5±1.2 |
| 360 | 20+25.8±6.1 | 98.2±0.6 | 93.7±0.9 |

| Speech Units | Epochs | Train | Test | Phrase |
|---|---|---|---|---|
| 120 | 20+14.7±6.4 | 83.8±3.3 | 80.4±3.2 | 58.6±3.8 |
| 240 | 20+13.7±3.9 | 91.0±1.2 | 85.6±1.3 | 62.9±2.9 |
| 360 | 20+13.1±4.4 | 93.8±1.2 | 87.1±1.3 | 64.5±2.7 |

TABLE VI

Summary of results with the best of each method.

| OCR | Units | Epochs | Train | Test |
|---|---|---|---|---|
| nn | 600.0 | 1.0 | 100.0 | 93.3 |
| $k$-nn ($k$=3) | 600.0 | 1.0 | 100.0 | 94.2 |
| Gen $k$-nn ($k$=6) | 600.0 | 1.0 | 100.0 | 94.8 |
| Fuzzy $k$-nn ($k$=3) | 600.0 | 1.0 | 100.0 | 94.3 |
| Parzen ($h$=20) | 600.0 | 1.0 | 100.0 | 94.2 |
| GAL | 118.2 | 3.0 | 100.0 | 90.2 |
| Linear | n/a | 14.1 | 100.0 | 89.2 |
| Pairwise | 45.0 | 22.2 | 100.0 | 95.3 |
| MLP ($H$=30) | 30.0 | 5.3 | 100.0 | 95.6 |
| RBF ($H$=360) | 360.0 | 45.8 | 98.2 | 93.7 |

| Speech | Units | Epochs | Train | Test | Phrase |
|---|---|---|---|---|---|
| nn | 600.0 | 1.0 | 100.0 | 90.2 | 67.4 |
| $k$-nn ($k$=1) | 600.0 | 1.0 | 100.0 | 90.2 | 67.4 |
| Gen $k$-nn ($k$=7) | 600.0 | 1.0 | 100.0 | 91.3 | 68.2 |
| Fuzzy $k$-nn ($k$=2) | 600.0 | 1.0 | 100.0 | 90.2 | 67.4 |
| Parzen ($h$=0.7) | 600.0 | 1.0 | 100.0 | 90.7 | 68.7 |
| GAL | 196.1 | 4.4 | 100.0 | 85.5 | 60.1 |
| Linear | n/a | 64.3 | 98.8 | 93.6 | 57.3 |
| Pairwise | 15.0 | 26.4 | 97.7 | 93.6 | 67.0 |
| MLP ($H$=64) | 64.0 | 53.9 | 99.8 | 93.5 | 65.3 |
| RBF ($H$=360) | 360.0 | 33.1 | 93.8 | 87.1 | 64.5 |

## VII. Concluding Discussions

### A. Overview of Results

We give a summary of results with all the methods considered for the two applications in Table VI. Out of the multiple runs with various parameter settings, we have chosen the one that gives the best compromise of success, learning time, and memory size.

### B. Analyses of Methods and Possible Improvements

Each learning method has a number of parameters that should be finetuned to get good performance. The main parameter however is the sample used, that is, how informative it is of the real field data. No learning method can have good generalization if the training data are significantly diferent from real data. We have compared methods using four criteria:

- *Generalization accuracy.*

  Local methods generalize sufficiently well. If there is noise in data or class labels, one needs a large sample and have $k$ or $h$ large to take into account the effect of many samples. In distance-based methods, one is advised to decrease as much as possible the dimensionality of the data by suitable preprocessing so as to alleviate the "curse of dimensionality." Just like fuzzy $k$-nn, GAL (and also clustering-based methods like LVQ and ART) can be "fuzzified" by taking into account more than one neighbor and weighting neighbors' effects inversely proportional to their distances to the input. This improves performance. Note that these methods are not limited to learning binary functions but can easily be generalized to approximate continuous functions as well.

  Distributed methods define boundaries supported by many samples and thus are more immune to noise and generalize better. They are slow for exactly the same reason.

  RBF method extends Parzen windows so that less storage is required and noise immunity is achieved. The idea is to have the best of two cases, namely fast learning and good generalization. One may also train both layers using back-propagation. This is not a good idea as unless an additional term is added, the spread of Gaussians need not stay small so locality may be lost and also this will not be any faster or better than usual back-propagation [29].

  With a small training sample, a distributed scheme is generally preferrable to a local scheme. A distributed unit is trained with all the samples; a local unit is trained by a small percentage of it. It is thus necessary to have a large training sample before using a local method like $k$-nn or RBF. In the case of RBF, when the sample is large, one can also use many Gaussians using the anchor method and can do without the time-consuming $k$-means computation.

- *Learning Time.*

  Local methods learn fast. Most of the time they just store the training sample as it is, which may be called *one-shot* incremental learning. GAL by being incremental needs a few epochs to select a subset of the training sample. Incremental allocation is good because it is a fast learning method. It is also bad as what one learns at one-shot may be noise.

  Distributed methods require a large number of iterations with small updates at each iteration as a sample point affects many hidden units and when updating to correct the discriminant in one part of the space, one disturbs the discriminant in another part.

  RBF method normally learns faster than MLP as we train only a one-layer network and only a small number of the Gaussians have non-zero output. In digit recognition in our case however this was not the case. When the training set is large and a large number of Gaussians can be allocated, the anchor method can be used. In the case when a small number of Gaussians are used, $k$-means clustering is necessary to optimize the Gaussian centers to reflect as well as possible the input probability density.

- *Memory Requirements.*

  The number of free parameters, $f$, is a linear function of the number of hidden units, $H$. When $d$ is the dimensionality of the input, from Equations (3) and (4) or (5), $f = d * H + H$. In the multi-layer perceptron, there are also $H + 1$ threshold parameters.

  Local methods like variants of $k$-nn and Parzen windows store the whole training sample. Thus these may be infeasible unless a method like GAL is used to store only the important subset or a clustering method like LVQ or ART. There is no study formalizing a reasonable upper bound on the number of samples GAL may store.

  Distributed methods do not have many free parameters thus the memory requirement is not large, however the weights require to be stored with much precision using many bits. In digit recognition for example all kernel estimator variants have their first layer weights binary, but distributed methods require more (we have not carried out an analysis to check how many precisely).

  RBF method requires much more Gaussians then an MLP with sigmoidal hidden units. The second layer (and first layer too if $k$-means clustering is used) requires precise weights as delta rule is used.

- *Implementation Complexity.*

  Local methods like $k$-nn, Parzen windows, and GAL, are quite simple. Finding the closest neighbor, requiring a "winner-take-all" mechanism implemented as a network of units, can be cheaply implemented in VLSI and there already are circuits for this. In software, finding the closest may be decreased to log time by using advanced data structures like k-d trees

[31] at the expense of increasing the complexity of the program.

Distributed methods use back-propagation which requires bi-directional flow of precise values and arithmetical circuitry manipulating those precise values. This is costly to do both in software simulations and hardware implementations.

RBF is also somewhat costly to implement. Both layers require precise weights. However because only a few of the Gaussians will be active for any input, in hardware implementation, one can use a systolic pipe-lining mechanism where the few small physical processing units may be used to compute the values of many virtual units.

### C. Final Comments

The readers of this journal should by now know that using a learning technique by itself is not sufficient. A good knowledge of the application and the learning methods is necessary to guide the network to learn the important and ignore the irrelevant. This can be done by task-dependent preprocessing of the signal or by constraining either the cost function or the network architecture. One also requires a large enough and representative sample for the approximation not to be biased by the idiosyncranisies of the particular sample used. The quality of the training sample is perhaps the most important factor that affects the quality of generalization. Of course here we are concerned with basically interpolatory tasks where the train and test sets, although distinct, are from the same domain. Extrapolation where the system is tested outside the train domain is a much harder task, especially for local methods.

It is surprising that methods quite simple like $k$-nn, or more generally memory-based approaches, are serious competitors to complex gradient-descending methods. We reach to the conclusion that much of the functionality of neural networks can be obtained from closely related but simpler techniques, for example, distance-based classifiers, without needing complex network structure, learning rules and precise weights at all.

More important is the fact that the results achieved on both tasks indicate that there is not one method that is significantly superior to others in all four respects of generalization accuracy, learning time, memory requirements and implementation complexity. The relative importances of these four factors may differ from one application to another one. Thus when choosing the best model, all of these constraints should be taken into account and not only the generalization accuracy as it has hitherto frequently been done.


### Acknowledgments

We would like to thank the anonymous reviewer whose comments improved the manuscript. The digit database is provided by Isabelle Guyon of AT&T Bell Labs. The Japanese speech database has been supplied by NTT Human Interface Laboratories, Speech and Acoustics Lab. (Furui Research Lab.) FG thanks Dr. Sadaoki Furui (executive manager) and the other members. EA thanks John Platt of Synaptics for informing about Hart's work on the condensed nearest neighbor rule and Francesco Masulli of University of Genoa for informing about Fuzzy $k$-nn. This work is supported by Grants EEEAG–41 and EEEAG–53 from TÜBİTAK, the Turkish Scientific and Technical Research Council and Grants 93HA0132 and 93HA0150 from Boğaziçi University Research Funds.


## References


[1] Aha DW, Kibler D, Albert MK. Instance-based Learning Methods, Machine Learning 1991; 6: 37–66.

[2] Alpaydın E. Neural Models of Incremental Supervised and Unsupervised Learning, PhD dissertation, No 869, Department d'Informatique, Ecole Polytechnique Fédérale de Lausanne, Switzerland, 1990.

[3] Alpaydın E. GAL: Networks That Grow When They Learn and Shrink When They Forget, International Computer Science Institute TR 91–032, Berkeley, USA, 1991.

[4] Alpaydın E. Single and Multiple Memory-Based Neural Networks for Efficient Function Approximation. In: O. Kaynak G. Honderd, E. Grant (Eds.) Intelligent Systems: Safety, Reliability, and Maintainability (pp. 194–204), NATO ASI series, Springer, 1993.

[5] Alpaydın E. Multiple Networks for Function Learning, International Neural Network Conference, Vol 1, San Francisco, USA, 1993: 9–14.

[6] Blum EK, Li LK. Approximation Theory and Feedforward Networks, Neural Networks 1991; 4: 511–515.

[7] Carpenter GA, Grossberg S. The ART of Adaptive Pattern Recognition by a Self-Organizing Neural Network, IEEE Computer 1988; 21: 77–88.

[8] Cover TM, Hart PE. Nearest Neighbor Pattern Classification, IEEE Transactions on Information Theory, 1967; 13: 21–27.

[9] le Cun Y, Boser B, Denker JS, Henderson D, Howard RE, Hubbard W, Jackel LD. Handwritten Digit Recognition with a Back-Propagation Network. In Touretzky, D. (Ed.) Advances in Neural Information Processing Systems 2 (pp. 396–404), Morgan Kaufmann, 1990.

[10] Denoeux T, Lengellé R. Initializing Back Propagation Networks with Prototypes, Neural Networks 1993; 6: 351–363.

[11] Duda RO, Hart PE. Pattern Classification and Scene Analysis, Wiley, 1973.

[12] Fukushima K. Neocognitron: A Hierarchical Neural Network Capable of Visual Pattern Recognition, Neural Networks 1988; 1: 119–130.

[13] Funahashi K. On the Approximate Realization of Continuous Mapping by Neural Networks, Neural Networks 1989; 2: 183–192.

[14] Furui S. Digital Speech Processing, Synthesis, and Recognition, Marcel Dekker, 1989.

[15] Gallant SI. Neural Network Learning and Expert Systems, MIT Press, 1993.

[16] Geman S, Bienenstock E, Doursat R. Neural Networks and the Bias/Variance Dilemma, Neural Computation 1992; 4: 1–58.

[17] Guyon I, Poujoud I, Personnaz L, Dreyfus G, Denker J, le Cun Y. Comparing Different Neural Architectures for Classi-

fying Handwritten Digits," International Joint Conference on Neural Networks 1989, vol 2, 127–132, Washington USA.
[18] Hart PE. The Condensed Nearest Neighbor Rule, IEEE Transactions on Information Theory 1968; 14: 515–516.
[19] Hornik K, Stinchcombe M, White H. Multilayer Feedforward Networks are Universal Approximators, Neural Networks 1989; 2: 359–366.
[20] Kanerva, P. Sparse Distributed Memory, MIT Press, 1988.
[21] Keller JM, Gray MR, Givens JA. A Fuzzy $K$-Nearest Neighbor Algorithm, IEEE Transactions on Systems, Man, and Cybernetics 1985; 15: 580–585.
[22] Kohonen, T. Self-Organization and Associative Memory, Springer, 1988.
[23] Lee Y, Lippmann R. Practical Characteristics of Neural Network and Conventional Pattern Classifiers on Artificial and Speech Problems. In D. Touretzky (Ed.) Advances in Neural Information Processing Systems 2 (pp. 168–177), Morgan Kaufmann, 1990.
[24] Lee Y. Handwritten Digit Recognition Using $K$-Nearest-Neighbor, Radial-Basis Function, and Backpropagation Neural Networks, Neural Computation 1991; 3: 440–449.
[25] Lippmann RP. Review of Neural Networks for Speech Recognition, Neural Computation 1989; 1: 1–38.
[26] Martin GL, Pittman JA. Recognizing Hand-Printed Letters and Digits Using Backpropagation Learning. In D. Touretzky (Ed.) Advances in Neural Information Processing Systems 2 (pp. 405–414), Morgan Kaufmann, 1990.
[27] Matan O, Baird HS, Bromley J, Burges CJC, Denker JS, Jackel LD, le Cun Y, Pednault EPD, Satterfield WD, Stenard CE, Thompson TJ. Reading Handwritten Digits: A Zip Code Recognition System, IEEE Computer 1992; 25(7): 59–62.
[28] Miller III WT, Glanz FH, Kraft III LG. CMAC: An Associative Neural Network Alternative to Backpropagation, Proceedings of the IEEE 1990; 78: 61–1567.
[29] Moody J, Darken CJ. Fast Learning in Networks of Locally-Tuned Processing Units, Neural Computation 1989; 1: 281–294.
[30] Ng K, Lippmann R. Practical Characteristics of Neural Network and Conventional Pattern Classifiers. In R. Lippmann, J. Moody, D. Touretzky (Eds.), Advances in Neural Information Processing Systems 3 (pp. 970–976), Morgan Kaufmann, 1991.
[31] Omohundro SM. Efficient Algorithms with Neural Network Behavior, Complex Systems, 1987; 1: 273–347.
[32] Poggio T, Girosi F. Networks for Approximation and Learning, Proceedings of the IEEE 1990; 78: 1481–1497.
[33] Rumelhart DE, Hinton GE, Williams RJ. Learning Internal Representations by Error Propagation. In D.E. Rumelhart, J.L. McClelland and the PDP Research Group (Eds). Parallel Distributed Processing: Explorations in the Microstructure of Cognition 1 (pp. 318–362), MIT Press, 1986.
[34] Ruck DW, Rogers SK, Kabrisky M, Oxley ME, Suter BW. The Multi-layer Perceptron as an Approximation to a Bayes Optimal Discriminant Function, IEEE Transactions on Neural Networks 1990; 1: 296–298.
[35] Senior AW. Off-line Handwriting Recognition: A Review and Experiments, Cambridge University Engineering Department, CUED/F-INFENG/TR 105, 1992.
[36] Silverman BW. Density Estimation for Statistics and Data Analysis, Chapman & Hall, 1986.
[37] Waibel A, Hanazawa T, Hinton G, Shikano K, Lang KJ. Phoneme Recognition Using Time-Delay Neural Networks, IEEE Transactions on Acoustics, Speech, and Signal Processing 1989; 37: 328–339.
[38] Waibel A, Hampshire II JB. Neural Network Applications to Speech. In P. Antognetti & V. Milutinović (Eds). Neural Networks: Concepts, Applications, and Implementations 1 (pp. 54–76), Prentice Hall, 1991.