ELSEVIER

Neurocomputing 13 (1996) 151–170

NEUROCOMPUTING

A neurocomputing framework: From methodologies to application [†]

Sung-Bae Cho [*]

Department of Computer Science, Yonsei University, 134 Shinchon-dong, Sudaemoon-ku, Seoul 120-749, South Korea

Received 13 March 1995; accepted 11 September 1995

Abstract

In this paper, we present a practical framework of methodologies for increasing the efficiency of the training process and improving the generalization capability of neural networks. The methodologies are devised for resolving problems of neural networks primarily in three aspects: learning, architecture, and data representation. For learning we present a rapid learning method based on Aitken's Δ^2 process and a training schedule called selective reinforcement learning; for architecture, a two-stage classification scheme and a multiple network scheme; and for data representation, a data generation scheme with systematic noise and a preprocessing method by hidden Markov model. In order to investigate the behavior of neural network classifiers with the proposed methodologies, we designed and implemented neural networks for recognizing on-line handwriting characters obtained by an LCD tablet. Experimental results with a large set of on-line handwriting characters show the usefulness of the proposed methodologies.

Keywords: Methodologies for practical application of neural networks; Learning methods; Neural architecture; Training data preparation; Handwriting recognition

1. Introduction

A typical neural network classification problem is the learning of arbitrary decision boundaries for a classification task, based on a collection of labeled training samples.

[*] Corresponding author. Email: sbcho@csai.yonsei.ac.kr

[†] This work was supported in part from the Korea Science and Engineering Foundation (KOSEF) and Center for Artificial Intelligence Research (CAIR), the Engineering Research Center (ERC) of Excellence Program.

The boundaries are arbitrary in the sense that no particular structure, or class of boundaries, is assumed *a priori*. While there are many results which show the usefulness of feedforward neural networks, a number of researchers have also suggested that current-generation feed-forward neural networks are largely inadequate for difficult problems in pattern recognition and machine learning, regardless of parallel implementation issues [16].

The essence of the difficulties lies in the facts that the training algorithms for neural networks, especially backpropagation algorithm, are based on a simple steepest descent technique, and estimation error of neural networks can be decomposed into two components, known as bias and variance; whereas incorrect models lead to high bias, truly model-free learning suffers from high variance. Thus, model-free approaches to complex classification tasks are slow to converge, in the sense that large training samples are required to achieve acceptable performance. This is the effect of high variance, and is a consequence of the large number of parameters, indeed infinite number in truly model-free learning, that need to be estimated. Prohibitively large training sets are then required to reduce the variance contribution to estimation error. (For theoretical issues, such as VC dimension, to the necessary number of training examples, see [1,4].) Parallel architectures and fast hardware do not help here: this convergence problem has to do with training set size rather than implementation.

The only way to control the variance in complex classification problem is to use model-based estimation. However, and this is the other face of the dilemma, model-based classification is bias prone: proper models are hard to identify for these more complex (and interesting) inference problems, and any model-based scheme is likely to be incorrect for the task at hand, that is, highly biased.

The answer is that the bias/variance dilemma can be circumvented if one is willing to give up generality, that is, purposefully introduce bias. In this way, variance can be eliminated, or significantly reduced. The bias should contribute significantly to mean-squared error only if we should attempt to infer regressions that are not in the anticipated class. Therefore, in many cases of interest, one could go so far as to say that designing the right biases amounts to solving the problem. We will suggest that some of these important biases can be achieved through proper architecture and data representation.

The issues of neural network classifier design that we address are fundamentally pragmatic in nature. Although abstract issues such as theoretical questions of convergence are vitally important to the more general questions of whether neural network classifiers are provably workable, we do not address them. Instead, we make assumptions regarding convergence and learnability and focus on practical methodologies that lead to the applicational framework.

In nontrivial problems, a single network trained with conventional framework would be large and would require a long training time, owing to the requisite size of the training data set. Indeed, straight-forward neural network approaches to pattern recognition, in general, seem tractable only for relatively rudimentary tasks. The constraints of network size and training set size are joined by an additional constraint.

Given these constraints, a neural network approach to pattern recognition might employ some form of methodologies which are integrated into a global framework. In

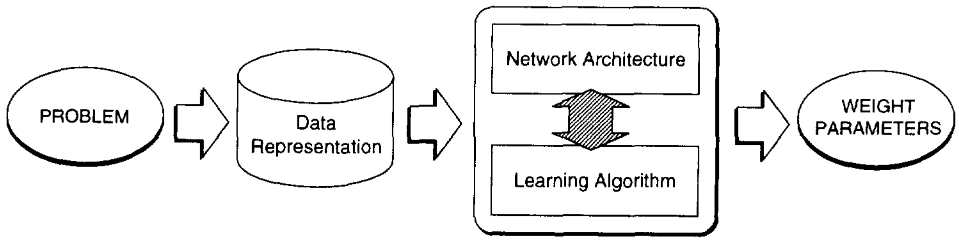

Fig. 1. A typical procedure for applying neural networks to practical problems.

order to implement such a framework, we can consider three aspects of the neural network approach as depicted by Fig. 1 [8].

- **The learning algorithm.** One of the most widely used learning algorithms is the backpropagation algorithm. The main reason for this success is that it produces a highly efficient network in spite of its simplicity. Since the backpropagation makes use of a simple gradient descent technique, the learning time explosively increases and the recall performance is greatly reduced for practical problems of large size and high complexity. In order to overcome this shortcoming, several investigators have examined methods for improving the rate of convergence of the backpropagation algorithm [9].
- **The architecture.** Once one fixes the structure of the network (i.e. chooses the number of hidden layers and the number of nodes in each hidden layer), the network adjusts its weights via the learning rule until the optimal weights are obtained. The corresponding weights along with the structure of the network create the decision boundaries in the feature space. In many practical pattern recognition problems, however, a conventional neural network classifier tends not to converge to the solution state. If the network does converge, the time required for convergence may be prohibitive for practical purposes. To overcome this difficulty, a variety of modular neural networks have been proposed [13].
- **The training data representation.** Learning in neural networks typically requires many training samples and relies more or less explicitly on some kind of syntactic preference bias such as 'minimal architecture,' but does not make use of explicit representations of domain-specific prior knowledge. If training data is deficient, learning a functional mapping inductively may no longer be feasible.

The rest of this paper is organized as follows. In Section 2, we show that neural network, as it is represented in some current neural networks, can be formulated as a Bayesian framework, thereby making the connection to the statistical pattern classification. And then we will focus on the limitations of these methods, at least as they apply to nontrivial problems in pattern recognition. In Section 3, we illustrate the methodologies for practical application of neural networks in the three categories: rapid training techniques, efficient design methods for architecture and methods for data representation. In order to investigate the behavior of a neural network with the methodologies, we designed and implemented neural networks for on-line handwriting character recognition, which is described in Section 4.

2. Neural network classifier

2.1 Bayesian formulation

The outputs of neural networks are not just likelihoods or binary logical values near zero or one. Instead, they are estimates of Bayesian *a posteriori* probabilities [30,24]. With a squared-error cost function, the network parameters are chosen to minimize the following:

$$E\left[\sum_{i=1}^{c}(y_i(X)-d_i)^2\right] \tag{1}$$

where $E[\cdot]$ is the expectation operator, $\{y_i(X): i=1,\dots,c\}$ the outputs of the network, and $\{d_i: i=1,\dots,c\}$ the desired outputs for all output nodes. Performing several treatments in this formula allows it to cast in a form commonly used in statistics that provides much insight as to the minimizing values for $y_i(X)$ [30]:

$$E\left[\sum_{i=1}^{c}(y_i(X)-E[d_i\mid X])^2\right]+E\left[\sum_{i=1}^{c} var[d_i\mid X]\right] \tag{2}$$

where $E[d_i \mid X]$ is the conditional expectations of d_i, and $var[d_i \mid X]$ is the conditional variance of d_i.

Since the second term in (2) is independent of the network outputs, minimization of the squared-error cost function is achieved by choosing network parameters to minimize the first expectation term. This term is simply the mean-squared error between the network outputs $y_i(X)$ and the conditional expectation of the desired outputs. For a 1 of M problem, d_i equals one if the input X belongs to class ω_i and zero otherwise. Thus, the conditional expectations are the following:

$$\begin{aligned} E[d_i\mid X] &= \sum_{j=1}^{c} d_i P(\omega_j\mid X) \\ &= P(\omega_i\mid X) \end{aligned} \tag{3}$$

which are the Bayesian probabilities. Therefore, for a 1 of M problem, when network parameters are chosen to minimize a squared-error cost function, the outputs estimate the Bayesian probabilities so as to minimize the mean-squared estimation error.

2.2 Bias / variance dilemma

The learning problem is to construct a function $f(x)$ based on a 'training set' $(x_1, y_1),\dots,(x_N, y_N)$, for the purpose of approximating y at future observations of x. this is sometimes called 'generalization'. To be explicit about the dependence of f on the data $D=\{(x_1, y_1),\dots,(x_N, y_N)\}$, we will write $f(x; D)$ instead of simply $f(x)$. Given D, and given a particular x, a natural measure of the effectiveness of f as a predictor of y is

$$E\left[(y-f(x;D))^2\mid x, D\right] \tag{4}$$

the mean-squared error (where $E[\cdot]$ means expectation with respect to the probability distribution P). In our new notation emphasizing the dependency of f on D, Eq. (4) reads

$$E\left[(y-f(x;D))^2 \mid x, D\right] = E\left[(y-E[y \mid x])^2 \mid x, D\right] + (f(x;D)-E[y \mid x])^2 \tag{5}$$

$E[(y-E[y \mid x])^2 \mid x, D]$ does not depend on the data, D, or on the estimator, f; it is simply the variance of y given x. Hence the squared distance to the learning function,

$$(f(x;D)-E[y \mid x])^2 \tag{6}$$

measures, in a natural way, the effectiveness of f as a predictor of y. The mean-squared error of f as an estimator of the regression $E[y \mid x]$ is

$$E_D\left[(f(x;D)-E[y \mid x])^2\right] \tag{7}$$

where E_D represents expectation with respect to the training set, D, that is, the average over the ensemble of possible D (for fixed sample size N).

It may be that for a particular training set, D, $f(x;D)$ is an excellent approximation of $E[y \mid x]$, hence a near-optimal predictor of y. At the same time, however, it may also be the case that $f(x;D)$ is quite different for other realizations of D, and in general varies substantially with D, or it may be that the average (over all possible D) of $f(x;D)$ is rather far from the regression $E[y \mid x]$. These circumstances will contribute large values in (7), making $f(x;D)$ an unreliable predictor of y. A useful way to assess sources of estimation error is via the bias/variance decomposition: for any x,

$$E_D\left[(f(x;D)-E[y \mid x])^2\right] = \underbrace{(E_D[f(x;D)]-E[y \mid x])^2}_{\text{bias}} + \underbrace{E_D\left[(f(x;D)-E_D[f(x;D)])^2\right]}_{\text{variance}} \tag{8}$$

If, on the average, $f(x;D)$ is different from $E[y \mid x]$, then $f(x;D)$ is said to be biased as an estimator of $E[y \mid x]$. In general, this depends on P; the same f may be biased in some cases and unbiased in others.

An unbiased estimator may still have a large mean-squared error if the variance is large: even with $E_D[f(x;D)] = E[y \mid x]$, $f(x;D)$ may be highly sensitive to the data, and, typically, far from the regression $E[y \mid x]$. Thus either bias or variance can contribute to poor performance. This paper attempts to solve the challenging problem on the design of appropriate bias.

3. Methodologies

In this paper we will develop a number of methodologies for increasing the efficiency of training and improving the generalization capability of neural network classifiers. The methodologies are devised for resolving problems primarily in three areas: learning,

Table 1
Methodologies for practical application of neural networks

	Category	Methodologies
1	Rapid learning algorithm	Aitken's Δ^2 process Selective reinforcement learning
2	Architecture design	Two-stage scheme Multiple network scheme
3	Data representation	Data generation with noise Preprocessing with HMM

architecture, and data representation. For learning we present several rapid training techniques for backpropagation according to the approach of three categories: numerical method based, heuristics based and learning strategy based.

In sequence, we illustrate two design methods for feedforward neural networks. They are described as mechanisms for incorporating *a priori* knowledge about a given problem. The first method, based on a two-stage scheme, decomposes the problem into more manageable ones, leading to a kind of modular architecture, thereby simplifying the decision-making process for classification. The second method on multiple network scheme combines incomplete decisions from several copies of networks for reliable decision-making.

Finally, we describe two kinds of methods for data representation: a noise-included training scheme and a hybrid method of hidden Markov models (HMMs) and a neural network classifier. The noise-included training scheme adds noise systematically to given training patterns. This has the same efffect as expanding the number of training patterns, and therefore improves the generalization capability. The proposed hybrid method exploits the discriminative capability of a neural network classifier while using HMM formalism to capture the dynamics of input patterns. Table 1 summarizes the methodologies.

3.1 Rapid learning algorithms

One of the major drawbacks of the backpropagation algorithm is its slow rate of convergence. Since the backpropagation makes use of a simple gradient descent technique, the learning time explosively increases and the recall performance is greatly reduced for practical problems of large size and high complexity. In order to overcome this shortcoming, several investigators have examined methods for improving the rate of convergence of the backpropagation algorithm: these can be classified into the following three categories.

First, some methods increases the learning speed by analytically incorporating more information about the error surface into the algorithm with techniques of numerical analysis [3,33]. Most of them are variants of the second-order method [27,28], which uses second derivatives in addition to the gradients when calculating the next weight values with the iterative formula.

Second, some methods accelerate the learning speed by using heuristic knowledge obtained by applying gradient descent techniques to learning. The basic idea here is concerned with the effective use of learning rate because it is believed that a fixed learning rate, regardless of the shape of the error curve, is the main cause of slow learning of the backpropagation algorithm. The typical examples of this category are the *dynamic adaptation method* of learning rate [22], and the *gradient reuse method* [21].

Finally, several methods do not attempt to revise the learning algorithm itself, but to mediate the shape and sequence of training patterns. These are related to learning schedule and preprocessing: to train hard patterns more frequently in the course of learning and to reduce the degree of correlation among training patterns before training.

In this section, we propose two rapid learning methods: one is based on the numerical method, called Aitken's Δ^2 process, and the other one is based on the learning strategy, called selective reinforcement learning.

3.1.1 Accelerated learning with Aitken's Δ^2 process

Let w_0 be the initial weight and $\{w_n\}_1^\infty$ the sequence generated by the learning algorithm of a neural network–backpropagation, for example. Then the weight-updating formula is usually as follows:

$$w_{n+1} = w_n + \mu \frac{\partial E}{\partial w_n} \tag{9}$$

where E is the error of the network and μ is the learning rate. Since the rapid learning method is simply considered as an iterative scheme performed by the network itself in order to solve nonlinear optimization, Aitken's Δ^2 process is used to accelerate the learning speed. Equation (10) illustrates the definition of this process. The purpose of Aitken's Δ^2 process is to obtain a sequence that converges more rapidly to a solution than the original sequence.

$$w_n^* = w_n - \frac{(w_{n+1} - w_n)^2}{w_{n+2} - 2w_{n+1} + w_n} \tag{10}$$

The value w_n^* is a better approximation of the solution than w_n or w_{n+1}.

We now present a rapid learning method that applies this technique to the weight updating formula in order to achieve a faster rate of convergence. The fast learning method presented applies this technique to the weight updating formula for achieving faster rate of convergence. The algorithm of the presented learning scheme is as follows.

```
Algorithm 3.1: Fast learning method with Aitken's $\Delta^2$ process
  Start learning with $w_0$;
  while Learning does not finish do
    $w_{n+1} = w_n + \mu \frac{\partial E}{\partial w_n}$;
    $w_{n+2} = w_{n+1} + \mu \frac{\partial E}{\partial w_{n+1}}$;
```

```
        $w_n^* = w_n - \frac{(w_{n+1} - w_n)^2}{w_{n+2} - 2w_{n+1} + w_n}$;
        $w_n = w_n^*$;
    end_while
```

This rapid learning method not only accelerates the rate of convergence, but also induces convergence in some cases where the iteration diverges [7].

3.1.2 Selective reinforcement learning

In selective reinforcement learning attention is focused on the hard patterns, since a great deal of time is required for a neural network to learn a few hard patterns. Selective reinforcement learning consists of two stages. In the first half, the weights are updated according to the sum of the errors of all training data in order to make the network grasp the outline of the training patterns. When the total sum of the errors (TSE) becomes smaller than a predefined tolerance ϵ_1, the training data for which the errors of output units exceed tolerance ϵ_2 are selectively presented.

```
Algorithm 3.2: Selective reinforcement learning
    Start learning with parameters $\epsilon_1$, $\epsilon_2$;
    while TSE > $\epsilon_1$ do
        /* It trains all the learning data */
        Training (ALL_DATA);
    end_while
    while TSE is not close to zero do
        /* It trains hard patterns more frequently */
        Train_data = Select_hard_patterns (ALL_DATA, $\epsilon_2$);
        Training (Train_data);
        Training (ALL – DATA);
    end_while
```

The second stage of selective reinforcement learning interlaces the total patterns with the hard patterns. After training the network using all patterns until approximately 50% of them are trained, the hard patterns are determined, and the network trains them once more. This automatic schedule of pattern presentation reduces the training time and results in a more generalized network capable of achieving higher recognition performance.

3.1.3 Simulation results

The representative methods presented above were compared empirically with the standard backpropagation algorithm. The problem chosen for comparison was the XOR problem because it is the most popular benchmark. The network structure used two incorporated hidden nodes and a bias node with a constant output of 1.0 connected to the hidden nodes and the output node.

The method of weight updating required a decision between update after presentation of each I/O pattern or after a complete cycle (epoch) of I/O patterns, both of which are

Table 2
A comparison of the number of epochs of the three methods with 25 trials

Method	No. of epochs				No. of successes
	Worst	Best	Average	S.D.	
ebp	18084	6084	13184.00	4056.94	7
ebp-m	19534	677	5721.40	5429.81	25
abp	4914	189	1120.81	1075.36	16
sbp	4626	3202	3881.44	314.85	25
sbp-m	11228	304	1176.40	2498.47	25

methods in current use. In this experiment, we adopted the method of changing the weights after each epoch. For all of the following comparisons, each consisted of 25 trials in which the initial weights were chosen randomly within the range -0.1 to $+0.1$. The average number of epochs taken for a successful trial was computed where success was deemed to be when the goal region was entered within 20,000 epochs. The goal region was defined as that region where the total sum of squared error of each pattern is less than 0.04.

With the conventional backpropagation, the average cycle over all the runs was 13184 epochs with standard deviation 4057, and only 7 cases out of 25 were successful. With the momentum augmentation, the average cycle became 5721 with standard deviation 5430. Though the difference between the worst and the best case becomes large, the average training time is shortened by twofold.

Table 2 shows the number of epochs of each method with 25 trials. In this table, ebp means the standard backpropagation, and the suffix '-m' represents the augmentation of momentum. Abp and sbp mean respectively Aitken's Δ^2 process and the selective presentation of samples.

From this experiment, the fastest method was Aitken's Δ^2 process. Despite that it does not require as much computation as the usual numerical method based techniques, it can achieve the performance of the second-order method. The only shortcoming is the relatively frequent cases of local minima. As compared with the speedup techniques based on the modification of algorithm, the learning strategy based method, sbp, turned out to be very stable because all the trials were successful.

3.2 Architecture design techniques

We present two design methods for feedforward neural networks: two-stage scheme and multiple network scheme. The relative superiority of the two-stage scheme to a single large neural network classifier is illustrated through the probabilistic interpretation of the two-stage scheme. And then, as an efficient fusion method of the multiple network scheme, we describe a combining method based on the fuzzy integral. One of the important advantages of this method over other conventional methods is that not only is the classification results combined but that the relative importance of the different networks is also considered. This approach may provide a possibility for incorporating any *a priori* knowledge regarding the underlying problem to improve the ability of the network to generalize.

3.2.1 Two-stage scheme

The two-stage neural network decomposes the classification problem into several more manageable ones. The design of a two-stage scheme for efficient classification entails some methodological considerations. The first step might be to find the partition $\{\xi_1, \xi_2, \ldots, \xi_k\}$ by using a clustering technique such as the *k-means algorithm*, or using some *a priori* knowledge about problem structure. As the result, we are given the coarse partition of the total class Ω, $\{\xi_1, \xi_2, \ldots, \xi_k\}$, and the fine partition of Ω, $\{\omega_1, \omega_2, \ldots, \omega_c\}$.

Let coarse categories r_j and s_k be chosen to have a unique correspondence with each class ω_i. For instance, r_j and s_k could be the row and column of a matrix of classes ω_i. Then,

$$P(\omega_i \mid X) = P(r_j, s_k \mid X). \tag{11}$$

If J is the number of r_j categories and K is the number of s_k categories, then $I = J \times K$. If we use the definition of conditional probability, without any simplifying assumptions the previous expression can be broken down as follows:

$$P(r_j, s_k \mid X) = P(s_k \mid X) \times P(r_j \mid s_k, X). \tag{12}$$

Thus, The desired probability is the product of one coarse category posterior probability and a second conditional probability. This scheme reduces the training of a single network with $J \times K$ outputs to the training of two smaller networks with J and K outputs, respectively.

Our implementation of the two-stage classifier is shown in Fig. 2. The coarse network in the figure performs the mapping NN_c, which is a switch for selecting one of the k classifiers in the fine stage. The networks in the second stage are realizations of the mappings NN_f^i. This approach, where the desired mappings are accomplished with several smaller neural networks, typically will require less training time than an approach that utilizes a single large network to carry out the mappings.

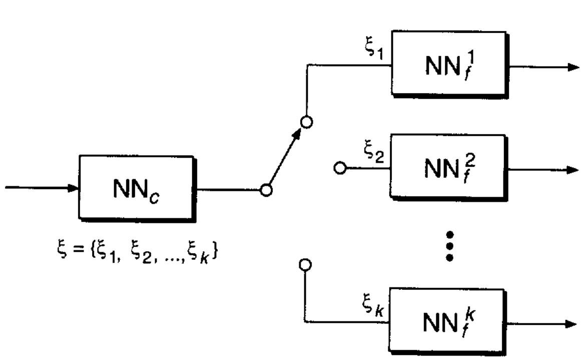

Fig. 2. The two-stage classifier composed of several feedforward neural networks. The coarse network performs the mapping NN_c, which is a switch for selecting one of the k classifiers in the fine stage.

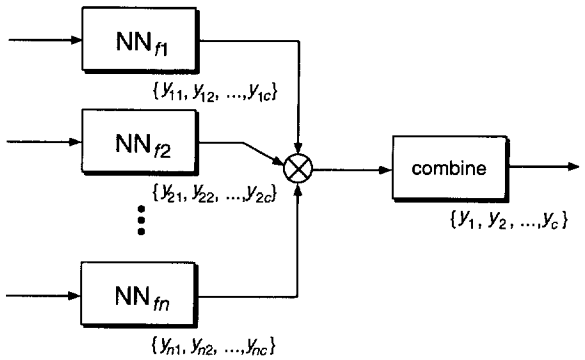

Fig. 3. The multiple neural network architecture with consensus scheme. n independently trained neural networks classify a given input pattern by using a consensus method to decide the collective classification.

3.2.2 Multiple network scheme

The networks train on a set of example patterns and discover relationships that distinguish the patterns. A network of a finite size does not often load a particular mapping completely or it generalizes poorly. Increasing the size and number of hidden layers most often does not lead to any improvements. Furthermore, in complex problems such as character recognition, both the number of available features and the number of classes are large. The features are neither statistically independent nor unimodally distributed. Therefore, if we can make the network consider the only specific part of the complete mapping, it will perform its job better.

The basic idea of the multiple network scheme is to develop n independently trained neural networks with particular features, and to classify a given input pattern by obtaining a classification from each copy of the network and then using a consensus scheme to decide the collective classification by utilizing combination methods [17] (see Fig. 3). Two general approaches, one based on fusion techniques and a second on voting techniques form the basis of the methods presented.

There have been proposed various neural network optimization methods based on combining estimates, such as boosting, competing experts, ensemble averaging, metropolis algorithms, stacked generalization and stacked regression. A general result from the previous works is that averaging separate networks improves generalization performance for the mean squared error. If we have networks of different accuracy, however, it is obviously not good to take their simple average or simple voting.

To give a solution to the problem, we developed a fusion method that considers the difference of performance of each network in combining the networks, which is based on the notion of fuzzy logic, especially the fuzzy integral [10,12]. This method combines the outputs of separate networks with importance of each network, which is subjectively assigned as the nature of fuzzy logic.

The fuzzy integral introduced by Sugeno and the associated fuzzy measures provide a useful way for aggregating information. Using the notion of fuzzy measures, Sugeno developed the concept of the fuzzy integral, which is a nonlinear functional that is defined with respect to a fuzzy measure, especially g_λ-fuzzy measure.

Definition 1. Let X be a finite set, and $h: X \to [0, 1]$ be a fuzzy subset of X. The fuzzy integral over X of the function h with respect to a fuzzy measure g is defined by

$$h(x) \bigcirc g(\cdot) = \max_{E \subseteq X} \left[\min\left(\min_{x \in E} h(x),\ g(E) \right) \right].$$

The calculation of the fuzzy integral with respect to a g_λ-fuzzy measure would only require the knowledge of the density function, where ith density, g^i, is interpreted as the degree of importance of the source y_i towards the final evaluation. These densities can be subjectively assigned by an expert, or can be generated from data. The value obtained from comparing the evidence and the importance in terms of the min operator is interpreted as the grade of agreement between real possibilities, $h(y)$, and the expectations, g. Hence fuzzy integration is interpreted as searching for the maximal grade of agreement between the objective evidence and the expectation. For further information, refer to [10].

3.3 Data representation

Many neural network models, such as a sufficiently large backpropagation network, can perform any arbitrary mapping from a continuous input space to a continuous output space [19,20]. In practice, the representation of the data is found to be important in determining the size of network required to perform the task in the time taken to train the network and the performance of the network when presented with noisy input. For these reasons, this section explores two efficient methods of preparing training data to improve the performance of neural network classifiers.

3.3.1 Noise-included data generation

Error tolerance is one of the most valuable properties of neural networks, yet simple neural networks are not likely to absorb the input noise of a large-scale problem. In large-scale classification problems the underlying training set contains only a small number of instances, and is not sufficiently representative for the underlying distributions of the classes. Classifiers trained with such sets will perform poorly.

Researchers have tried several different approaches to overcome the geometrical variations of input data. Fukushima [15] introduces complex cell planes to his model, *Neocognitron*. Though it may be appropriate for the biological point of view, the complex cell method requires too much storage and recognition time to be practical for large-scale problems. Reber [29] and Khotanzad and Lu [23] use preprocessing mechanisms (involving several techniques such as polar, log, and discrete Fourier transformation) for extracting the geometrically invariant features. Widrow and Winter [34] manage the deformation of input patterns by using the *invariance net* at the front end, and Waibel et al. [32] use *Time Delay Neural Network* (TDNN) for dealing with temporal deformation in speech recognition problems. (For speech recognition problems, it has been reported that recognition performance is improved by introducing random or temporal distortions into the training data [5,26].)

We present a distortion method to generate new training data. From a given training sample, this method automatically generates additional instances of its class by shifting the sample in a predetermined set of directions (such as up, down, left, and right). This method expands the number of training patterns in each class, and its artificial expansion can be performed optimally according to the problem. Networks trained via this method can respond in a more flexible way to unforeseen variations in future data. The critical question is how much noise can legitimately be added with the new data still belonging to the same class as the noiseless present data. Though the answer depends primarily on the problem, the shifting method we have proposed performs well in many practical problems.

Holmstrom et al. [18] showed that the additive noise is interpreted as generating a kernel estimate of the probability density that describes the training vector distribution. They did not prove that the introduction of additive noise to the training vectors always improves network generalization, but suggested mathematically justified rules for choosing the characteristics of noise if additive noise is used in training. Our method cannot be justified analytically, but the newly generated training samples to which noise has been added can be regarded as being drawn from a kernel estimate of the training vector density.

3.3.2 HMM preprocessing for sequence recognition

The key idea in the proposed HMM preprocessing architecture is (1) to convert a dynamic input sample to a static pattern sequence by using HMM-based recognizer and (2) to recognize the sequence by using an MLP-trained classifier. A block diagram of the hybrid architecture is shown in Fig. 4.

A usual HMM-based recognizer assigns one Markov model for each class. Recognition with HMMs involves accumulating scores for an unknown input across the nodes in

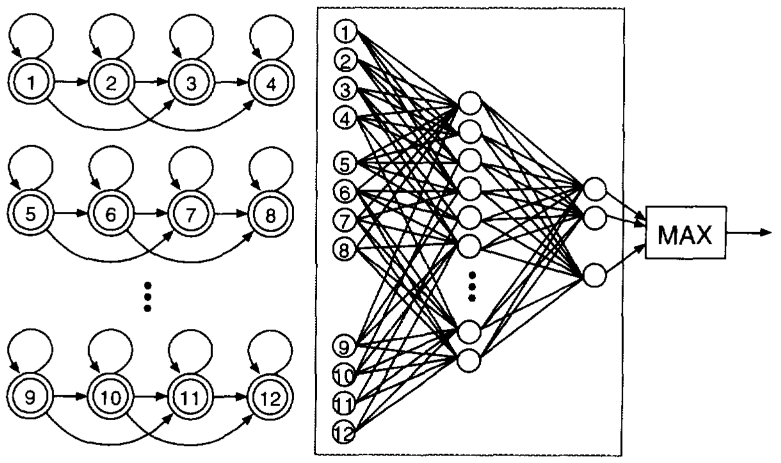

Fig. 4. The HMM preprocessing architecture.

each class model, and selecting that class model which provides the maximum accumulated score. On the contrary, the proposed architecture replaces the maximum-selection part with an MLP classifier.

The hybrid architecture takes the likelihood patterns inside the HMMs and presents them to an MLP to estimate posterior probabilities of class ω_i as follows:

$$P(\omega_i \mid X) \approx f\left\{ \sum_{k=1}^{H} w_{ik}^{om} f\left(\sum_{j=1}^{T} \sum_{l=1}^{N} w_{kjl}^{mi} \alpha_T(j, l) \right) \right\}, \tag{13}$$

where the w_{kjl}^{mi} is a weight from the jth input node at the lth state to the kth hidden node, w_{ik}^{om} is a weight from the kth hidden node to the ith class output, and f is a sigmoid function such as $f(x) = 1/(1 + e^{-x})$. Here, $\alpha_T(j, l)$ is the value of the forward variable $\alpha_T(l)$ at the jth HMM class model. Rather than simply selecting the model producing the maximum value of $P(X \mid \lambda_j)$, the proposed method have an MLP perform additional classification with all the likelihood values inside HMMs. In this method, the HMM yields a kind of static pattern of which the inherent temporal variations have been processed, and the MLP classifier discriminates them as belonging to one particular class.

The hybrid method automatically focuses on those parts of the model which are important for discriminating between sequentially similar patterns. In the conventional HMM based approach, only the patterns in the specified class are involved in the estimation of parameters; there is no role for any patterns in the other classes. The hybrid method uses more information than the conventional approach; it uses knowledge of the potential confusions in the particular training data to be recognized. Since it uses more information, there are certainly reasons to suppose that the hybrid method will prove superior to the conventional approach. In this method, the MLP will learn prior probabilities as well as to correct the assumptions made on the probability density functions used in the HMMs. For further account, see [11].

4. Application to on-line handwriting recognition

In order to give an idea of the practical application of the presented methodologies in pattern recognition, a data set of handwriting characters has been used as a source of training and test samples. An input character consists of a set of strokes, each of which begins with a pen-down movement and ends with pen-up movements. Several preprocessing algorithms were applied to successive data points in a stroke to reduce quantization noise and fluctuations of the writer's pen motion. The processes used are as follows: the wild point reduction, the dot reduction, the hook analysis, the three point smoothing, the peak preserving filtering, and the N point normalization. A sequence of preprocessed data points is approximated by a sequence of 8-directional straight-line segments which is the same as the chain code used by Freeman [14]. The procedure for collecting handwriting data is schematically presented in Fig. 5.

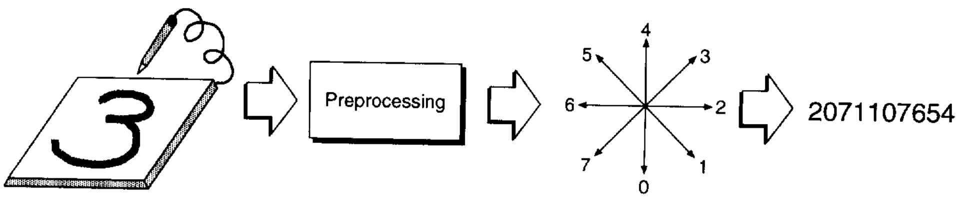

Fig. 5. Schematic diagram of the process for data collection.

In the experiment, handwriting characters were inputted to the computer (SUN workstation) by an LCD tablet of Photron FIOS-6440 which samples 80 dots per second. The tasks were to classify the Arabic numerals, the uppercase letters, and the lowercase letters which were collected from 13 writers. The writers were told to draw the numerals and letters into prepared square boxes in order to facilitate segmentation.

4.1 Multistage scheme

First of all, we tried to investigate the recognition rate of the neural network classifier. To recognize the on-line handwriting characters by neural network classifier, we implemented two-layer neural networks which have 10 input nodes, 20 hidden nodes and 10 or 26 output nodes. To train the neural network classifiers, forty examples for each class were used, while for recognition a further 500 examples were used as test inputs.

The recognition results with respect to the three tasks are reported in Table 3. The neural networks have been commonly presented as good static pattern classifiers, but this result implies that they still require further research for dynamic pattern recognition.

Moreover, this table reports the recognition results of the multistage neural networks. The groups at the coarse stage are as follows: For recognizing the numerals, (0, 6), (1, 5, 7), and (4, 8, 9); For the uppercase letters, (D, P), (E, F, I, O), (N, W, X, Y) an (U, V); For lowercase letters, (a, d, m), (b, p), (c, i, l), (e, z), (g, s), (h, n, y), (k, q, t), and (u, v). Though the proposed scheme did not increase the recognition rates remarkably, the training time was greatly shortened.

4.2 Multiple network scheme

To evaluate the performance of the multiple network scheme, we implemented three different networks, including the one used in the previous experiment, each of which is a

Table 3
Recognition rates of the neural network classifier and the multistage neural networks (%)

Subject	Simple NN	Multistage NN	
		Coarse	Fine
Numerals	77.4	83.2	77.8
Uppercases	73.2	81.8	76.0
Lowercases	59.0	73.8	68.8

Table 4
The result of recognition rates (%)

Networks	Numerals	Uppercases	Lowercases
NN_1	82.6	73.2	73.9
NN_2	81.2	66.8	71.8
NN_3	81.0	70.8	72.1
Voting	84.9	74.0	74.6
Average	86.9	75.2	78.2
Fuzzy	88.1	76.1	80.3

two-layer neural network having different number of input neurons and 20 hidden neurons. NN_1, NN_2 and NN_3 have 10 input neurons, 15 input neurons, and 20 input neurons, respectively. In this way each network makes the decision through its own resolution; NN_1 using sparsely sampled input produces the result by means of coarser view of input image, while NN_3 uses finer view. Thus, NN_1 has large possibility to overcome the variation or noise of input image though it utilizes rather blurred input.

Each of the three networks was trained with 40 samples per class, validated with another 500 samples, and tested on ten sets of samples collected from ten different writers: the recognition rate on the validation set was monitored in order to stop the training process. For all of the following experiments, each consisted of 10 trials in which the different data were made from different writers.

Table 4 shows the recognition rates of numerals, uppercase letters, and lowercase letters with respect to the three different networks and their combinations by utilizing consensus methods like majority voting, average, and the fuzzy integral. As can be seen, the overall classification rates for the fuzzy integral are higher than those for other consensus methods as well as individual networks.

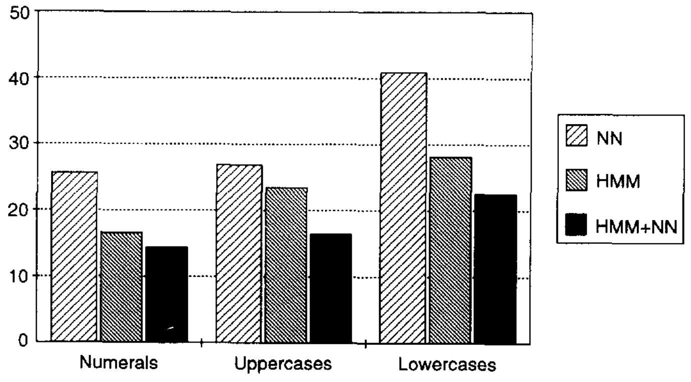

Fig. 6. A comparison of the error rates of neural network, HMM and the hybrid method.

4.3 HMM preprocessing

The next experiment is to recognize the same data set by HMMs. For the HMMs, we implemented left-right model in which no transitions are allowed to states whose indices are lower than that of the current state. It was composed of the ten nodes and the eight observation symbols in each node. The ten nodal matching scores of all models provided as inputs to the neural network classifier of the hybrid method. To train HMMs and a neural network classifier, forty examples for each class were used, while for recognition a further 500 examples were used as test inputs.

In order to apply the presented hybrid method for the numerals recognition, we implemented another two-layered neural network which has 100 input nodes, 20 hidden nodes and 10 output nodes. The input was provided by the ten HMM models consisting of ten nodes.

Fig. 6 compares the error rates of all the three methods with respect to the numerals, the uppercase letters, and the lowercase letters. The overall recognition rate for the ten classes with hybrid method is 85.40% on a total of 500 characters. This is a significant improvement over the performance obtained with HMMs trained with ML optimization (83.60% recognition rate), as well as with NN using the direction sequences of character as inputs (74.40% recognition rate). This improvement may be practically significant, but it is not impressive for a method which should give some net benefit by construction. However, the fact that similar (or bigger) improvements were obtained for upper and lower case letters provides evidence that this is a real effect.

5. Concluding remarks

In this paper we have developed a number of methodologies for increasing the efficiency of training and improving the generalization capability of neural network classifiers. However, there are a number of things to investigate further:

- In order to measure the efficiency of the proposed methodologies and compare them analytically, the connection to theoretical results should be explored. A competitive candidate may be Vapnik-Chervonenkis (VC) dimension, which is an established tool for analyzing learning from examples [1,6]. Simply stated, the VC dimension $VC(G)$ furnishes an upper bound for the number of examples needed by a learning process that starts with a set of hypotheses G about what f may be. It requires large G to find a good approximation of f, but the larger G is, the more examples of f we need to pinpoint the good hypothesis [2]. We will be able to show how the proposed methodologies shrink G without losing good hypotheses.
- Several works are also remaining for further research in the multiple network scheme. The relatively easy ones are to increase the recognition rate of each base classifier network for practical usage and to try the same experiments with larger number of classifiers. Furthermore, another fusion methods such as Dempster-Shafer method [31,25] may be applied to combine the decisions of the networks. Moreover, it is relevant to note here that this paper has dealt with the comparison of several methods

within the same framework of neural network architecture, so called the multiple neural networks. Therefore, it can be also possible to compare with any other neural network architectures, 'adaptive mixtures of local experts', for instance, as long as there is a fair way to compare the performance which are remained for further study.

- Finally, there are many possibilities of applying the proposed methodologies to more realistic, practical problems.

The time of the neural network classifiers is coming. It will probably be a golden age for those who are nimble enough — or powerful enough — to take advantage of it.

Acknowledgements

The author would like to thank Prof. Jin H. Kim of the Department of Computer Science at KAIST for continuous encouragement. This work was partly conducted while at: KAIST Center for Artificial Intelligence Research, 373-1 Koosung-dong, Yoosung-ku, Taejeon 305-701, Korea; and ATR Human Information Processing Research Laboratories, 2-2 Hikaridai, Seika-cho, Soraku-gun, Kyoto 619-02, Japan.

References

[1] Y.S. Abu-Mostafa, The Vapnik-Chervonenkis dimension: information versus complexity in learning, *Neural Computation* 1 (1989) 312–317.
[2] Y.S. Abu-Mostafa, Hints and the VC dimension, *Neural Computation* 5 (1993) 278–288.
[3] J.E. Angus, On the connection between neural network learning and multivariate nonlinear least squares estimation, *Int. J. Neural Networks-Res. & App.* 1 (1989) 42–47.
[4] E.B. Baum and D. Haussler, What size network gives valid generalization, *Neural Computation* 1 (1989) 151–160.
[5] R.K. Bernhard and A.K. Wolfgang, Design of hierarchical perception structures and their application to the task of isolated-word recognition, in: *Proc. IEEE Int. Joint Conf. Neural Networks* (San Diego, CA, 1989) (I) 243–249.
[6] A. Blumer, A. Ehrenfeucht, D. Haussler and M. Warmuth, Learnability and the Vapnik-Chervonenkis dimensions, *J. ACM* 36 (1989) 929–965.
[7] S.-B. Cho and J.H. Kim, An accelerated learning method with backpropagation, in: *Proc. IEEE Int. Joint Conf. Neural Networks* (Washington, DC, 1990) (I) 605–608.
[8] S.-B. Cho and J.H. Kim, Strategic application of feedforward neural networks to large-scale classification, *Complex Systems* 6 (4) (1992) 363–389.
[9] S.-B. Cho and J.H. Kim, Rapid backpropagation learning algorithms, *Circuits, Systems and Signal Processing* (Special Issue on Neural Processing) 12 (2) (1993) 155–175.
[10] S.-B. Cho and J.H. Kim, A multiple network architecture combined by fuzzy integral, in: *Proc. IEEE / INNS Int. Joint Conf. Neural Networks* (Nagoya, Japan, 1993) (II) 1373–1376.
[11] S.-B. Cho, A discriminative hidden Markov model recognizer with neural network postprocessor, in: *Proc. IEEE World Congress on Computational Intelligence* (Florida, FL, 1994) 2881–2884.
[12] S.-B. Cho and J.H. Kim, Combining multiple neural networks by fuzzy integral for robust classification, *IEEE Trans. on Systems, Man, and Cybernetics* 25 (2) (1995) 380–384.
[13] F. Fogelman Soulie, Neural network architectures and algorithms: a perspective, in: *Artificial Neural Networks* (Elsevier Science Publishers B.V., North-Holland, 1991) 605–615.

[14] H. Freeman, Computer processing in line drawing images, *Computing Survey* 6 (March 1974) 57–98.
[15] K. Fukushima, A neural network for visual pattern recognition, *IEEE Computer* 21 (3) (March 1988) 65–74.
[16] S. Geman, E. Bienenstock and R. Doursat, Neural networks and the bias/variance dilemma, *Neural Computation* 4 (1) (1992) 1–58.
[17] L.K. Hansen and P. Salamon, Neural network ensembles, *IEEE Trans. Pattern Analysis and Machine Intelligence* 12 (1990) 993–1001.
[18] L. Holmstrom and P. Koistinen, Using additive noise in backpropagation training, *IEEE Trans. Neural Networks* 3 (1) (1992) 24–38.
[19] K. Hornik, M. Stincombe and H. White, Multilayer feedforward networks are universal approximators, *Neural Networks* 2 (1989) 359–366.
[20] K. Hornik, M. Stinchcombe and H. White, Universal approximation of an unknown mapping and its derivatives using multilayer feedforward networks, *Neural Networks*, 3 (1990) 551–560.
[21] D.R. Hush and J.M. Salas, Improving the learning rate of back-propagation with the gradient reuse algorithm, in: *Proc. IEEE Int. Conf. Neural Networks* (San Diego, CA, 1988) (I) 441–447.
[22] R.A. Jacobs, Increased rates of convergence through learning rate adaptation, *Neural Networks* 1 (1988) 295–307.
[23] A. Khotanzad and J.H. Lu, Distortion invariant character recognition by a multilayer perceptron and backpropagation learning, in: *Proc. IEEE Int. Conf. Neural Networks* (San Diego, CA, 1988) (I) 635–632.
[24] D.J.C. MacKay, A practical Bayesian framework for backprop networks, *Neural Computation* 4 (3) (1992) 448–472.
[25] E. Mandler and J. Schuermann, Combining the classification results of independent classifiers based on the Dempster/Shafer theory of evidences, *Pattern Recognition and Artificial Intelligence* (1988) 381–393.
[26] T. Matsuoka, H. Hamada and R. Nakatsu, Syllable recognition using integrated neural networks, in: *Proc. IEEE Int. Joint Conf. Neural Networks* (San Diego, CA, 1989) (I) 251–258.
[27] D.B. Parker, A comparison of algorithms for neuron-like cell, in: *Neural Networks for Computing*, (J.S. Denker, eds., New York, American Institute of Physics, 1986) 327–332.
[28] D.B. Parker, Optimal algorithms for adaptive networks: second order backpropagation, second order direct propagation, and second order Hebbian learning, in: *Proc. IEEE Int. Cont. Neural Networks* (San Diego, CA, 1987) (II) 593–600.
[29] W.L. Reber, An artificial neural system design for rotation and scale invariant pattern recognition, in: *Proc. IEEE Int. Conf. Neural Networks* (San Diego, CA, 1987) (IV) 277–283.
[30] M.D. Richard and R.P. Lippmann, Neural network classifiers estimate Bayesian a *posteriori* probabilities, *Neural Computation* 3 (1991) 461–483.
[31] G. Shafer and R. Logan, Implementing Dempster's rule for hierarchical evidence, Artificial Intell. 33 (1987) 271–298.
[32] A. Weibel, T. Hanazawa, G. Hinton, K. Shikano and K.J. Lang, Phoneme recognition using time-delay neural networks, *IEEE Trans. on ASSP* 37 (1989) 328–339.
[33] R.L. Watrous, Learning algorithms for connectionist networks: applied gradient methods of nonlinear optimization, in: *Proc. IEEE Int. Conf. Neural Networks* (San Diego, CA, 1987) (II) 619–627.
[34] B. Widrow and R.G. Winter, Neural nets for adaptive filtering and adaptive pattern recognition, *IEEE Computer* (March 1988) 25–39.

Sung-Bae Cho was born in Seoul, Korea, in 1965. He received the B.S. degree in computer science from Yonsei University, Korea, in 1988 and the M.S. and Ph.D. degrees in computer science from KAIST (Korea Advanced Institute of Science and Technology), Korea, in 1990 and 1993, respectively.

He worked as a Research Staff at the Center for Artificial Intelligence Research at KAIST from 1991 to 1993. He was an Invited Researcher of Human Information Processing Research Laboratories at ATR (Advanced Telecommunications Research) Institute, Kyoto, Japan from 1993 to 1995. Since 1995, he has been an Assistant Professor in the Department of Computer Science, Yonsei University, Seoul, Korea. His research interests include neural networks, pattern recognition, intelligent man-machine interfaces, evolutionary computation and artificial life.

He was awarded outstanding paper prizes from the IEEE Korea Section in 1989 and 1992, and another one from the Korea Information Science Society in 1990. He was also the recipient of the Richard E. Merwin prize from the IEEE Computer Society in 1993. He was listed in Who's Who in Pattern Recognition from the International Association for Pattern Recognition in 1994, and nominated for biographical inclusion in the Fifth Edition of Five Thousand Personalities of the World from the American Biographical Institute in 1995.

Dr. Cho has served as a paper reviewer for a dozen international journals and two international conferences. He is a member of the Korea Information Science Society, INNS, IEEE Computer Society and IEEE Systems, Man and Cybernetics Society.