Complex Systems **6** (1992) 363–389

Strategic Application of Feedforward Neural Networks to Large-Scale Classification

Sung-Bae Cho[*]
Jin H. Kim
Center for Artificial Intelligence Research
and
Computer Science Department,
Korea Advanced Institute of Science and Technology,
373-1, Koosung-dong, Yoosung-ku, Taejeon 305-701, Republic of Korea

Abstract. Feedforward neural networks have been successfully applied to a variety of classification problems, but the number of classes used for experiments was too small to apply the results directly to large-scale problems. This paper presents several strategies for applying feedforward neural networks to large-scale, complex classification problems: a two-stage classification scheme, a rapid learning method, a training schedule called selective reinforcement learning, a training scheme including systematic noise, and a weight matrix reduction scheme. These strategies have been applied to the design of a printed Hangul (Korean script) recognition system. Experiments with the 990 most frequently used printed Hangul syllables confirm the usefulness of the presented strategies.

1. Introduction

A neural network is a parallel computational paradigm that solves problems by means of massive interconnections of simple processors. Several models of such networks have been proposed for use in a variety of difficult problems, especially classification problems [1]. Traditional classifiers test competing hypotheses sequentially, whereas neural network classifiers test them in parallel (thus providing high computational rates). The feedforward neural network classifiers in particular have been applied successfully to many

[*]Electronic mail address: `sbcho@gorai.kaist.ac.kr`.

problems, such as discriminating between underwater sonar returns [2] and forming text-to-phoneme rules [3].

Although there have been reports that feedforward neural network classifiers are adequate for tough pattern-recognition problems, the number of classes used for experiments was too small to apply the results directly to large-set classification problems. Larger networks require greater amounts of training time and patterns, and the computational complexity of the learning process can quickly reach unmanageable proportions [4]. Actually, a simple neural network approach often fails in many real world problems.

To cope with this difficulty, various temporary remedies have been proposed in the literature. In this paper we combine many of these techniques into a systematic framework of strategies for applying feedforward neural networks to large-scale, complex classification problems. Our strategies are devised for resolving problems primarily in three areas: architecture, learning, and generalization. For architecture we present a two-stage classification scheme; for learning, a rapid learning method and a training schedule called selective reinforcement learning; and for generalization, a training scheme that includes systematic noise and a weight matrix reduction scheme.

The two-stage classification scheme leads to a kind of modular architecture, thereby simplifying the decision-making process for classification. The rapid learning method accelerates learning speed by applying Aitken's Δ^2 process, which was originally developed for efficiently solving nonlinear optimization problems. Since a neural network tends to waste a great deal of time learning a few hard patterns, selective reinforcement learning focuses its attention on the hard patterns. The noise-included training scheme adds noises systematically to given training patterns. This has the same effect as expanding the number of training patterns, and therefore improves the generalization capability of the scheme. A large number of link weights do not contribute to decision making because of their small magnitude. The weight matrix reduction scheme cuts off insignificant links to improve recognition speed and generalization capability.

In order to investigate the behavior of a neural network with the above-described strategies, we designed and implemented neural networks for printed Hangul syllable recognition, using input data obtained from an optical scanner. Even for printed syllables, building a useful Hangul recognition system is not simple—such a system must be able to classify a large set of syllables that are very similar to each other.

The rest of this paper is organized as follows. In section 2, we introduce feedforward neural networks as classifiers. We describe several strategies for applying feedforward neural networks to large-scale classification problems in section 3. In section 4 we explain the printed Hangul syllable recognition system, based on the two-stage classification scheme. Experimental results with the 990 most frequently used printed Hangul syllables are presented in section 5. Finally, conclusions are discussed in section 6.

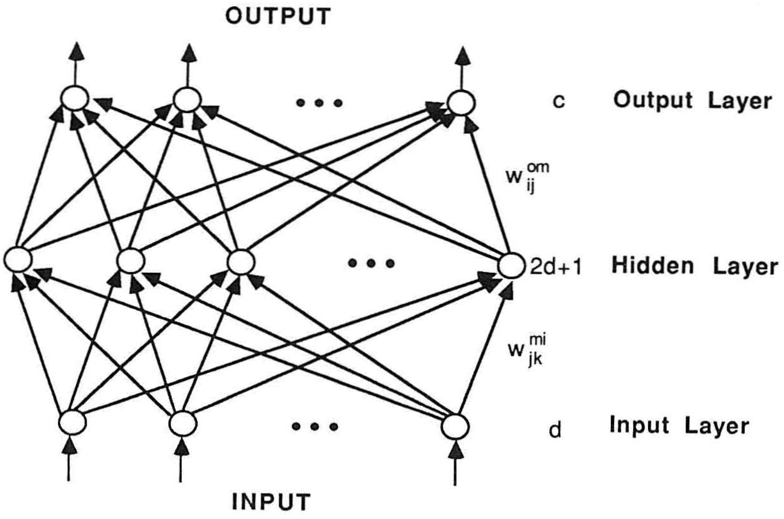


Figure 1: A two-layer neural network architecture. In this network, d is the number of features, c is the number of classes, and $2d+1$ is an appropriately selected number.

2. Feedforward neural network classifiers

A neural network can be considered a mapping device between input and output sets. Mathematically speaking, a neural network represents a function F that maps I into O; $F : I \to O$, or $\mathbf{y} = F(\mathbf{x})$ where $\mathbf{y} \in O$ and $\mathbf{x} \in I$. Since a classification problem is a mapping from a feature space to some set of output classes, we can formalize the neural network—especially the two-layer feedforward neural network trained with the backpropagation algorithm—as a classifier.

Figure 1 shows a two-layer feedforward neural network classifier with d neurons in the input layer, $2d+1$ neurons in the hidden layer, and c neurons in the output layer. Here, d is the number of features, c is the number of classes, and $2d+1$ is an appropriately selected number (this number is determined by Kolmogorov's theorem [5]). The network is fully connected between adjacent layers. The operation of the feedforward neural network classifier for this network is as follows.

Let $x = \{x_1, x_2, \ldots, x_d\}$ be the feature set. Then each neuron x'_i in the hidden layer calculates its output through

$$x'_i = f\left(\sum_{j=1}^{d} w_{jk}^{mi} x_j\right), \qquad 1 \le k \le 2d+1, \tag{1}$$

where w_{jk}^{mi} is a weight between the kth input neuron and the jth hidden neuron, and f is a sigmoid function such as $f(x) = 1/(1+e^{-x})$. After this process, each neuron y_j in the output layer calculates its output in a similar fashion:

$$y_j = f\left(\sum_{i=1}^{2d+1} w_{ij}^{om} x_i'\right), \qquad 1 \le j \le c, \tag{2}$$

where w_{ij}^{om} is a weight from the jth hidden neuron to the ith class output.

Let $\Omega = \{\omega_1, \omega_2, \ldots, \omega_c\}$ be the class set. Then the decision rule determines ω_{j^*} if

$$j^* = \arg\max_j y_j, \qquad 1 \le j \le c. \tag{3}$$

In other words, the neural network selects the largest output as the correct class.

This can be thought of as a nonlinear decision-making process. Given an unknown input, each output neuron estimates the possibility y_j of belonging to this class by

$$y_j = f\left\{\sum_{i=1}^{2d+1} w_{ij}^{om} f\left(\sum_{j=1}^{d} w_{jk}^{mi} x_j\right)\right\}, \tag{4}$$

and then the neuron having maximum value is selected as the corresponding class. The key consideration is how to determine the weight values w_{ij}^{om} and w_{jk}^{mi}.

The process of weight tuning for optimal classification is called learning. The backpropagation algorithm adjusts the weights between the layers according to the following equation [6]:

$$\Delta w_{ji} = \mu\, \delta_j\, y_i, \tag{5}$$

where μ is a learning rate, δ_j is the difference between the desired ouput and the actual output of neuron j, and y_i is the output of neuron i. δ_j can be rewritten as follows according to the layer in which the sigmoid function is used:

- when neuron j is at the output layer:

$$\delta_j = (d_j - y_j) y_j (1 - y_j) \tag{6}$$

- when neuron j is at the hidden layer:

$$\delta_j = y_j (1 - y_j) \sum_k \delta_k w_{kj} \tag{7}$$

In both cases d_j is a desired output of neuron j, and w_{kj} is the weight between neuron k and neuron j. The term $y_j(1 - y_j)$ is a result to differentiate the sigmoid function.

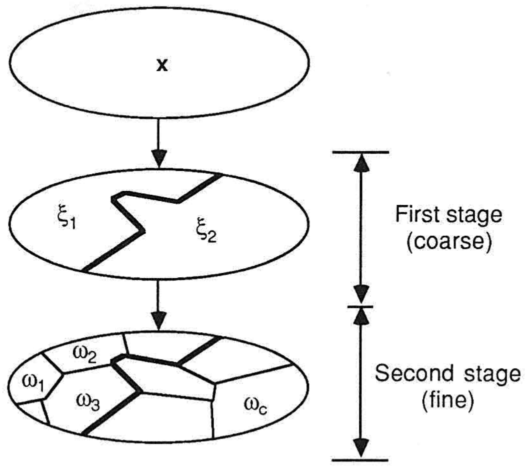


Figure 2: Two-stage classification scheme. The coarse partition of the total class is $\{\xi_1, \xi_2, \ldots, \xi_k\}$, and the fine partition is $\{\omega_1, \omega_2, \ldots, \omega_c\}$.

3. Application to large-scale classification

3.1 Two-stage classification

In the conventional neural network approach, 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 optimum 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, a conventional neural network classifier as just described tends not to converge to the solution state. If the network does converge, the time required for convergence may be prohibitive for practical purposes. The following section presents a method of neural network architecture design based on a "divide and conquer" approach.

According to the previous formalization of neural networks as classifiers, a neural network having d binary input neurons and c output neurons can be considered as a classifier to assign a given sample with d features to one of c predefined categories. 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 shown in figure 2). 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, $\{\omega_1, \omega_2, \ldots, \omega_c\}$.

Our two-stage classifier is shown in figure 3. The coarse network in the figure performs the mapping NN_c, which is a switch for selecting one of the

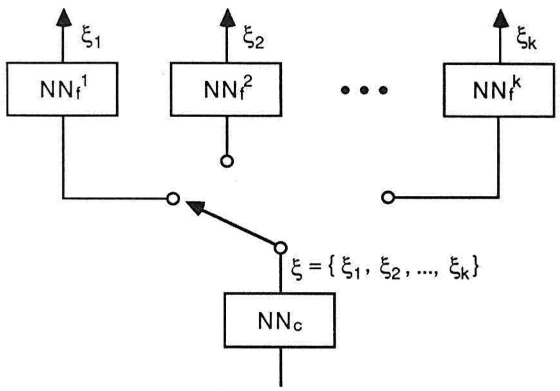


Figure 3: The two-stage classifier composed of several feedforward neural networks. The coarse network performs the mapping NN_c, which is a switch for selecting one of the k classifiers in the fine stage.

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.

The following demonstration will illustrate the degree to which complexity is reduced in a two-stage neural network classification scheme. In order to appreciate the difficulty of training a neural network with a large number of connections, let us first consider a single-layer network using linear discriminant functions.

Suppose there are n sample points in d dimensions, and each point is labeled either ω_1 or ω_2. Of the 2^n possible dichotomies of n points in d dimensions, a certain fraction $f(n,d)$ are said to be linear dichotomies. These are the labelings for which there exists a hyperplane separating the points labeled ω_1 from the points labeled ω_2. It can be shown [7] that this fraction is given by the following:

$$f(n,d) = \begin{cases} 1 & \text{if } n \le d+1 \\ \dfrac{2}{2^n}\displaystyle\sum_{i=0}^{d} \binom{n-1}{i} & \text{otherwise} \end{cases} \tag{8}$$

This means there may not exist a hyperplane to classify sample points if n is not less than $d+1$. However, if we think of a single-layer neural network as a linear pattern classifier, the number of connections becomes the number of degrees of freedom for the discriminant function, and the number of samples can be several times as large as the number of connections.

Multi-layer neural networks can act as nonlinear discriminants, therefore the above results cannot be used directly to analyze multi-layer neural network classifiers. Nevertheless, the basic idea of the neural network is to automatically generate the mapping rule between given data pairs of input and output, and the complexity of a multi-layer classifier can be measured by the number of possible mappings that should be considered.

Since we are given a neural network with d binary input neurons and c classes, the number of possible instances described by all these d features is 2^d, and the number of possible mappings considered by the neural network classifier is

$$ {}_{c}\Pi_{2^d} = c^{2^d}. \tag{9}$$

On the other hand, if we decompose the c classes into k subclasses $c_1, c_2, \ldots, c_k$ so that we can implement the classifier with a coarse network and k subnetworks, then the total number of possible mappings required by these networks becomes

$$ {}_{k}\Pi_{2^d} + \sum_{i=1}^{k} {}_{c_i}\Pi_{2^d} = k^{2^d} + \sum_{i=1}^{k} c_i^{2^d}. \tag{10}$$

However, the actual number of possible mappings required to classify a given sample point in subclass c_i is

$$ {}_{k}\Pi_{2^d} + {}_{c_i}\Pi_{2^d} = k^{2^d} + c_i^{2^d}. \tag{11}$$

Therefore, the load on the neural network classifier is greatly reduced. (For further explanation of the mathematical analysis, see [8].)

3.2 Rapid learning method

The learning of neural networks consists of the systematic adjustment of connection weights in order to approximate the desired output. In recent years, the backpropagation algorithm appeared to be one of the most successful learning procedures for multi-layer neural networks. This algorithm, however, is too slow to apply to the real world problems. (It solves problems by means of the gradient descent search method, which modifies iteratively the values of weights in the direction in which the error function E decreases most rapidly.)

Researchers have used several different approaches in attempting to speed up the convergence of backpropagation learning. Some have used more elaborate search methods. (Most of these are variations of Newton's method, and require the computation, or approximation, of second partial derivatives [9, 10, 11].) Others have attempted a systematic, empirical study of learning speed in the backpropagation algorithm, finding the heuristics for achieving faster rate of convergence [12, 13, 14]. Many algorithms that have been proposed for rapid learning use the approximated high-order derivative of the error function, which provides more information about the shape of the weight space, with the result that the rate of convergence is dramatically increased.

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

$$w_{n+1} = w_n + \mu \frac{\partial E}{\partial w_n} \tag{12}$$

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 [15, 16], Aitken's Δ^2 process is used to accelerate the learning speed. Equation (13) 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:

$$w_n^* = w_n - \frac{(w_{n+1} - w_n)^2}{w_{n+2} - 2w_{n+1} + w_n} \tag{13}$$

The value w_n^* is a better approximation of the solution than w_n or w_{n+1}. (A more detailed exposition of Aitken's Δ^2 process appears in Appendix A.)

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 learning process itself consists of two stages: acceleration and attention. In the earlier acceleration stage, a search process moves the network quickly across the solution space by means of equation (13). In the later attention stage, the search direction focuses slowly and accurately toward a minimum. The algorithm of this rapid learning method is as follows.

```
Algorithm 1: Rapid learning method with Aitken's $\Delta^2$ process
    Start learning with $w_0$;
    while Original sequence converges linearly do
        /* Step I : acceleration process */
        $w_{n+1} = w_n + \mu\frac{\partial E}{\partial w_n}$;
        $w_{n+2} = w_{n+1} + \mu\frac{\partial E}{\partial w_{n+1}}$;
        $w_n^* = w_n - \frac{(w_{n+1}-w_n)^2}{w_{n+2}-2w_{n+1}+w_n}$;
        $w_n = w_n^*$;
    end_while
    while Learning does not finish do
        /* Step II : attention process */
        $w_{n+1} = w_n + \mu\frac{\partial E}{\partial w_n}$;
    end_while
```

This rapid learning method not only accelerates the rate of convergence, but also induces convergence in some cases where the iteration diverges. Experimental results with the XOR problem confirm the superiority of the method presented here [17].

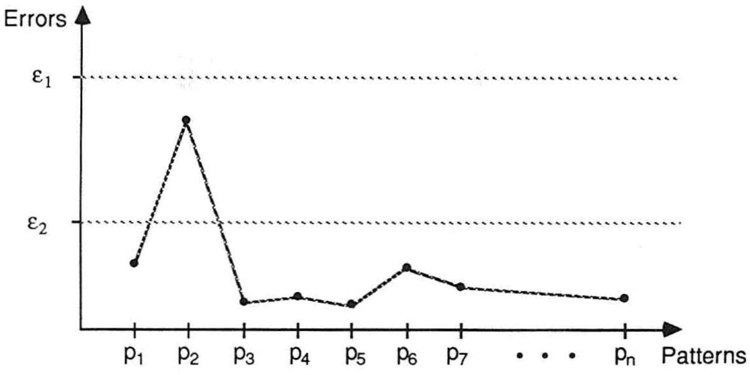


Figure 4: The intermediate state of the selective reinforcement learning process with respect to training patterns. After equally training all training patterns until the error of each is less than ϵ_1, the proposed learning scheme selectively trains unclassified patterns, for example p_2.

3.3 Selective reinforcement learning

A natural procedure for the design of classifiers is to construct a measure of the performance of the classifier on the training set, and subsequently to adjust the variables of the classifier such that this measure is optimized [7]. In supervised learning, each pattern that has to be learned is propagated through the network. Then the output is compared with the expected output, and the connection weights adjusted to minimize the observed error. (This process is repeated for all the patterns of the training set.) With a large problem, however, the learning process is likely to consume a great deal of time trying to learn a few unclassified patterns, while most of the other patterns already yield a correct result. Several learning schedules, such as *overlearning* [18] and *rapid incremental learning* [19], have been proposed to improve learning capability. Mori and Yokosawa [20] investigate the behavior of learning with various kinds of data presentation, and propose two learning schedules: the *review method* for reinforcing weak points of learning, and the *preparation method* for preventing networks from overtraining.

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, 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. Figure 4 shows an intermediate state of the learning process according to each pattern, and the overall algorithm is as follows.

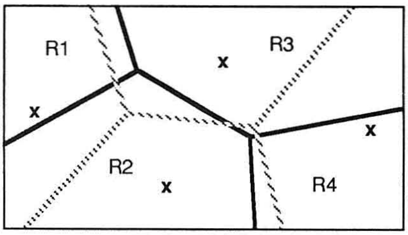


Figure 5: A four-class example of incorrect decision boundaries. Solid lines represent the correct decision boundaries; dashed lines represent the incorrect boundaries caused by inadequate training data (marked x).

```
Algorithm 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 trained once more. This automatic schedule of pattern presentation reduces training time and results in a more generalized network capable of achieving higher recognition performance.

3.4 Noise-included learning

Error tolerance is one of the most valuable properties of neural networks, yet simple neural networks are not likely to absorb the input noises 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; an example of incorrect decision boundaries resulting from inadequate training data is shown in figure 5. Since an inadequate set of training data is the cause of bad generalization, it is the major obstacle to achieve a higher recognition performance.

Researchers have tried several different approaches to overcome the geometrical variations of input data. Fukushima [21] 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 [22] 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 [24] manage the deformation of input patterns by using the *invariance net* at the front end, and Waibel et al. [25] 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 [26, 27].)

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 parent data. Though the answer depends primarily on the problem, the shifting method we have proposed performs well in many practical problems.

3.5 Weight matrix reduction

While the functionality of neural networks has been researched extensively, their structure has been studied relatively little. Most are fully connected layered networks, which operate well in simple problems having one or two hundred units, but which require much computation in large-scale problems. In order to remedy this shortcoming, Somani and Penla [28] designed two specially structured networks, the *Compact Neural Network* (CNN) and the *Reduced Interconnections Neural Network* (RINN), both of which use general network topologies such as hyper cube, rectangular grid, ring, and so on. Lehar and Weaver [29] proposed a developmental design scheme that constructs the network structure randomly, and modifies it by repeated mutation and selection in order to develop toward a desired functionality.

There are other methods of finding the smallest network that will perform a particular task that are based on pruning a solution network. A network which is too small may never solve a given problem, while a larger network may be inefficient, particularly on a conventional von Neumann computer [30]. Mozer and Smolensky [31] devised a technique for pruning some units from the solution network via relevance measure, but it requires a modification of the cost function and the examination of all units under the presentation of the entire training data. Karnin [32] proposed a method of cutting weights instead of units by estimating the sensitivity of the global error function with the inclusion/exclusion of each weight link in the neural network. Though Karnin suggests that this method would incur negligible computational overhead, it is easy to devise a simpler weight-cutting method (with little computational overhead) by carefully observing the behavior of the large-scale network. In this section, we present a simple method of reducing the number of links after training. The result is a small, efficient network that performs as well as, or better than, the original; that is, it improves the noise resistance of the network and reduces the processing time required to give a solution.

Let S_{ij} be the sensitivity with respect to connection w_{ij},

$$S_{ij} = E(w_{ij} = 0) - E(w_{ij} = w^*_{ij}), \tag{14}$$

where E is the global error of the network and w^*_{ij} is the solution value of the connection. S_{ij} represents the increase of the error E with the elimination of the connection between unit j and unit i. The sensitivity S_{ij} can be rewritten as follows:

$$S_{ij} = -\frac{E(w_{ij} = w^*_{ij}) - E(w_{ij} = 0)}{w^*_{ij} - 0} w^*_{ij} \tag{15}$$

$$\approx -\frac{E(w_{ij} = w^*_{ij}) - E(w_{ij} = w^i_{ij})}{w^*_{ij} - w^i_{ij}} w^*_{ij}, \tag{16}$$

where w^i_{ij} is the initial weight. Then, as derived in [32], the estimated sensitivity to the removal of connection w_{ij} can be evaluated as

$$S_{ij} = -\sum_{0}^{N-1} \frac{\partial E}{\partial w_{ij}}(n)\ \Delta w_{ij}(n)\ \frac{w^*_{ij}}{w^*_{ij} - w^i_{ij}} \tag{17}$$

where N is the number of training epochs. Karnin's approach is to calculate the sensitivity of each weight by equation (17) during backpropagation learning, then prune the low-sensitivity connections. (If w^*_{ij} is very close to zero, the sensitivity of equation (17) becomes zero, and the connection is very likely to be pruned.) A large-scale problem tends to result in a sparse network that has many links for which the solution values are close to zero. The weight matrix reduction scheme uses this fact in finding the proper size for a network, without incurring any computational overhead in the process of learning.

Since a unit used in neural networks sums N weighted inputs and passes the result through a nonlinearity [33], inputs of weight near zero may not be necessary for calculating output. The weight matrix reduction scheme eliminates these insignificant links, improving recognition speed and generalization capability. Figure 6 shows an example of unnecessary links. In this figure, the output remains the same even if only two links (a and d) are used in the calculation of the activation value. There is a trade-off between the acceleration of the recognition process and the degradation of its performance, and the determination of the optimal reduction is essential not only

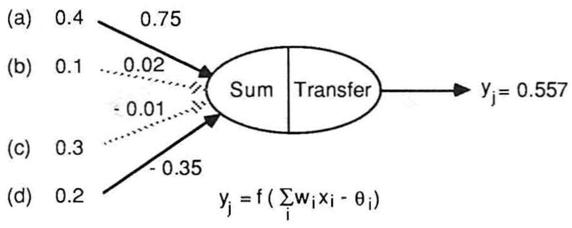


Figure 6: An example of link cuttings. The output remains the same even if only two links (*a* and *d*) are used for the calculation of the activation value.

to the acceleration of recognition but to the improvement of generalization performance as well. Le Cun [34] also confirms the fact that minimizing the number of free parameters in a network enhances generalization.

4. An application to Hangul recognition

The foregoing strategies have been applied to the design of a printed Hangul recognition system. The system is composed of a type classification network and six recognition networks. The former roughly classifies input syllable images into one of six types by their overall structure, and the latter further classifies them by character code. This two-stage architecture reduces network complexity, and faster training time and higher recognition performance naturally follow.

4.1 Structure of Hangul characters

The Korean script system Hangul consists of 24 characters each of which represents a phoneme. Ten of these are vowels and the rest are consonants. Characters are grouped together to form syllables. A syllable may consist of two to six characters. More than 11,000 syllables exist, but about 3,000 suffice for ordinary use. A word consists of a sequence of syllables.

The rules of character combination for making a syllable at first seem complicated but logical. The Korean vowels are shaped either vertically or horizontally elongated. The vertical vowels have their accompanying consonants on their left and the horizontal vowels have their accompanying consonants on their top. If a syllable has a consonant after a vowel, it is always written below the main vowel. Depending on its position and accompanying character, the shape of a character varies.

The general structure of Hangul syllables is presented in figure 7, where V1 indicates a vertically shaped vowel, V2 a horizontally shaped vowel, C1 a head consonant, and C2 a bottom consonant. According to the shape of the vowel included in the syllable and the presence or absence of a bottom consonant, Hangul syllables can be divided into six categories, as shown in figure 8.

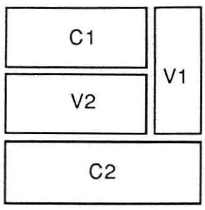


Figure 7: General structure of Hangul syllables. V1 signifies a vertically shaped vowel, V2 a horizontally shaped vowel, C1 a head consonant, and C2 a bottom consonant.

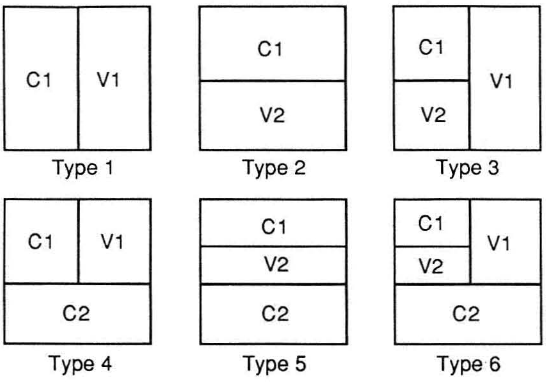


Figure 8: Six types of Hangul syllables. They are based on the shape of the vowel included in the syllable, and on the presence or absence of a bottom consonant.

4.2 Overall structure of recognition networks

At the beginning of our experiment, we implemented a simple neural network using backpropagation, with a single hidden layer that has 1,600 (40×40) input neurons, 40 hidden neurons and 10 output neurons. The output neurons distributively encode Hangul syllables. This network, however, did not converge when the number of classes of training samples exceeded approximately one hundred syllables.

We overcame this obstacle by utilizing the fact that a Hangul syllable is constructed by a combination of two to six characters. Syllables are grouped into six classes according to the types of character combination. As a result, the overall system is composed of a global type classification network and six recognition networks, one for each global type. Figures 9 and 10 show the algorithmic overview of the proposed classification scheme and the overall structure of the classifier, respectively. Two-stage neural classifiers have several advantages over single-layer backpropagation networks. Higher recognition performance is obtained with fewer training data, and the training time is shortened. These advantages result from the modular design.

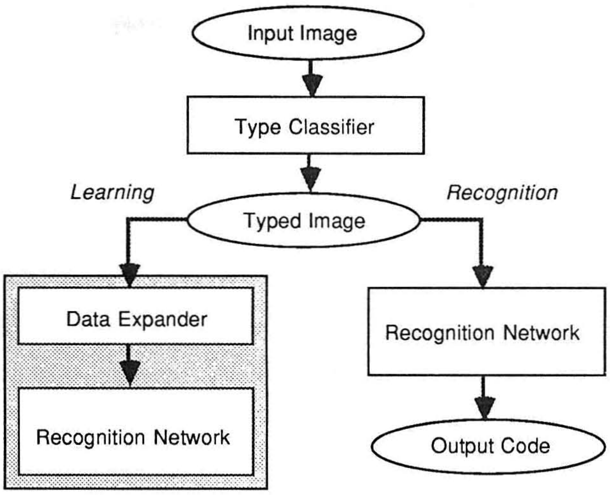


Figure 9: Overview of the Hangul character recognition system.

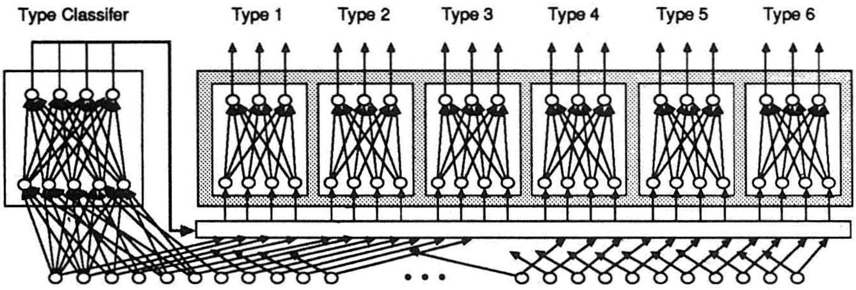


Figure 10: Overall neural network structure for Hangul character recognition. The system is composed of a type classification network and six recognition networks. The former roughly classifies input syllable images into one of six types by their overall structure, and the latter further classify them by character code.

4.3 The type classification network

Since Hangul syllables can be grouped into six global types, the recognition system is greatly simplified if the type of input image is known a priori. Therefore, the system determines the type of input syllable image and activates the corresponding network for syllable recognition. Figure 11 shows the structure of a type classification network.

The type classifier is a partially connected feedforward network that has 1,600 input neurons, 17 hidden neurons, and 6 output neurons. Each neuron of the hidden layer is connected to a specific input area, in which specific

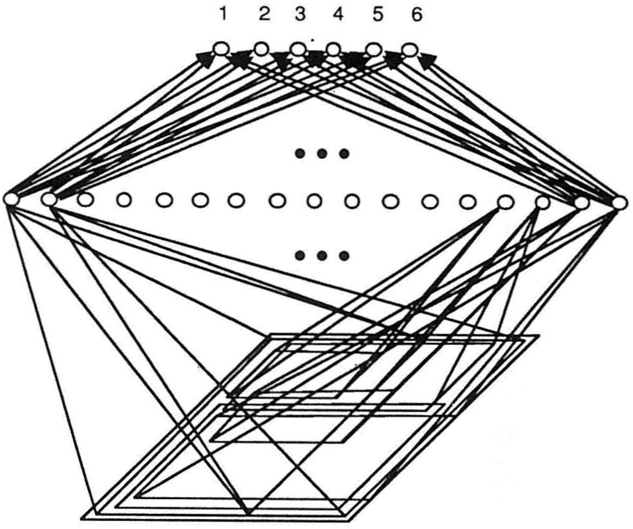


Figure 11: A type classification neural network. The type classifier is a partially connected feedforward network that has 1,600 input neurons, 17 hidden neurons, and 6 output neurons. Each neuron of the hidden layer is connected to a specific input area.

types of characters can appear. The first neuron, for example, is connected to the consonant area of type 1, the second neuron to the vowel area of type 1, and so on. Each hidden neuron arrives at a local decision, and output neurons summarize the local decisions and make the final decision of type determination. This structure can be justified by the observation that the position of the character in each syllable is an important consideration for classifying the type. According to this design, the number of interconnections is 5,915, while a fully connected network would require 27,302 $[(40 \times 40 \times 17) + (17 \times 6)]$.

4.4 The recognition networks

The recognition networks activated by the type classification network classify an input syllable image as a set of characters within a customized receptive field. The idea of a customized receptive field for each recognition network is based on the observation that every syllable of the same type has a structural similarity.

Each recognition network is structured for recognizing its character, and has a single hidden layer of 40 neurons. Figure 12 shows the structure of a recognition network for syllables of the first type, where the first network recognizes the head consonant and the second recognizes the vertically shaped vowel. As shown in this figure, the characters in a syllable image may touch each other. Recognition networks are capable of correctly recognizing characters despite the noise due to adjacent characters. It has been proven that

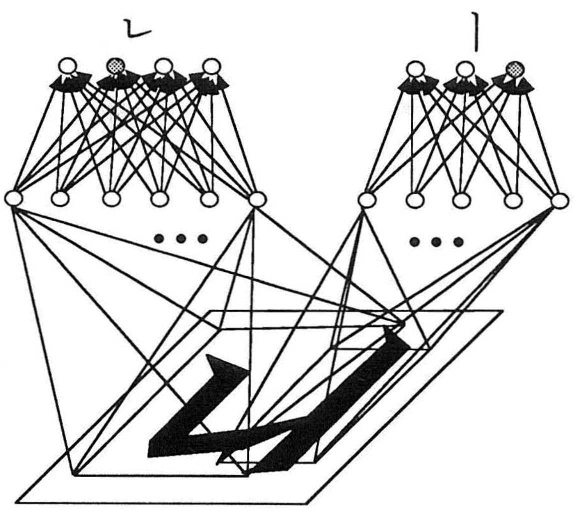


Figure 12: Neural network for typed character recognition (a case of type 1). In this figure, the characters in the syllable image may touch each other. Recognition networks are capable of correctly recognizing characters despite the noise due to adjacent characters.

a multi-layer perceptron with at most two hidden layers can form any arbitrarily complex decision regions in a feature space, which is the ultimate goal of any classifier [33].

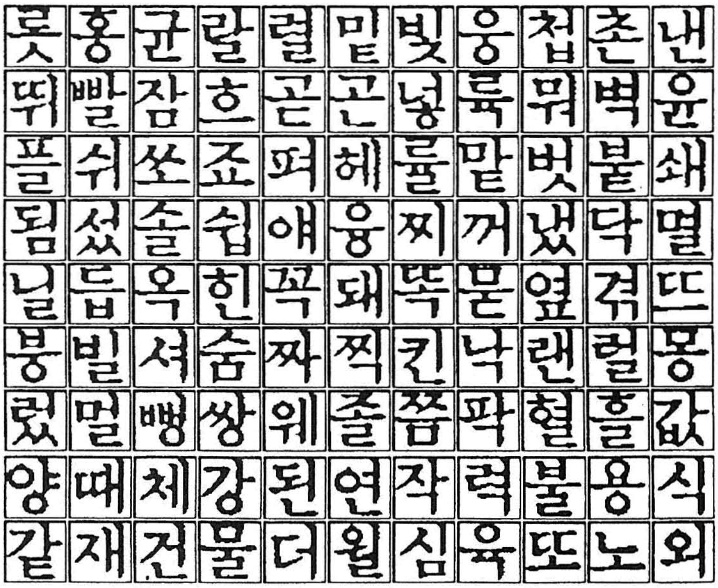
5. Experimental results

5.1 Environments for experiments

Experiments were conducted with the 990 most frequently used printed Hangul syllables on a Cray 2 supercomputer. We produced laser printer output using the Microtek MSF300C scanner at 300 DPI (dots per inch) resolution. After analyzing the horizontal and vertical projections, a maximum 40×40 binary image of each syllable can be extracted from the scanned document. Figure 13 shows a practical example of scanned syllables. Each recognition network with 40 hidden units was trained by the backpropagation algorithm, with a learning rate of 0.1, a momentum term of 0.9, and each weight randomly initialized between -0.1 and 0.1. The training was terminated when the total sum of squared errors became less than 0.04.

5.2 Analysis of results

The base system is a two-stage neural network trained with the rapid learning method. Figure 14 shows the decreasing rate of errors as the network learns, and confirms the improvement of learning speed resulting from the selective reinforcement learning scheme. This result does not, however, support a



롯 홍 균 랄 렬 밑 빛 웅 첩 촌 낸
뛰 빨 잠 흐 곧 곤 넣 륙 뭐 벽 윤
플 쉬 쏘 죠 뙤 헤 률 맡 벗 붙 쇄
됨 섰 솔 쉽 얘 융 찌 꺼 냈 닥 멸
닐 듭 옥 힌 꽉 돼 똑 묻 엷 겪 뜨
붕 빌 셔 숨 짜 찍 킨 낙 랜 럴 몽
렀 멀 뺑 쌍 웨 졸 쯤 팍 헐 흘 값
양 때 체 강 된 연 작 력 불 용 식
갈 재 전 물 더 월 심 육 또 노 외

Figure 13: A practical example of scanned syllables.

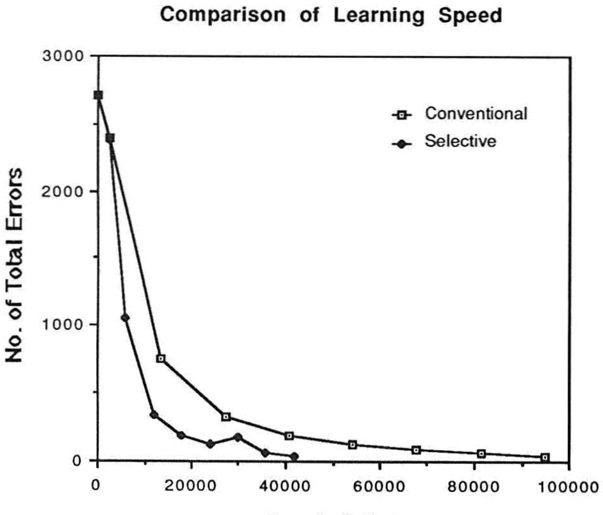


Figure 14: Fast learning by selective reinforcement. With selective reinforcement learning, the total number of training patterns is 79,349, while conventional learning requires 180,250.

group		*group identification rate*		*character recognition rate in each group*		*overall*	
id	#char	#error	%correct	#error	%correct	#error	%correct
1	114	0	100.00	0	100.00	0	100.00
2	68	3	95.59	2	97.06	5	92.65
3	42	0	100.00	6	85.71	6	85.71
4	480	1	99.79	16	96.67	16	96.67
5	251	0	100.00	15	94.02	15	94.02
6	35	3	91.43	0	100.00	3	91.43
overall	990	7	99.29	39	96.06	45	95.45

Table 1: Recognition rate of the conventional learning method.

group		*group identification rate*		*character recognition rate in each group*		*overall*	
id	#char	#error	%correct	#error	%correct	#error	%correct
1	114	0	100.00	0	100.00	0	100.00
2	68	3	95.59	0	100.00	3	95.59
3	42	0	100.00	2	95.24	2	95.24
4	480	1	99.79	7	98.54	7	98.54
5	251	0	100.00	2	99.20	2	99.20
6	35	3	91.43	0	100.00	3	91.43
overall	990	7	99.29	11	98.89	17	98.28

Table 2: Recognition rate of the noise-included learning method.

conjecture that the selective training of hard patterns makes the network learn better.

In order to compare recognition rates, the results with the conventional backpropagation learning are shown first in table 1. The data used for testing are different, of course, from the training data. Since the majority of patterns belong to types 4 or 5, the recognition rates of the two-stage networks show large variances according to type. The overall rate is 95.45%. The recognition rate of the noise-included learning method, shown in table 2, improves to 98.28%. Figure 15 compares both of these recognition rates according to type. The weakest point of the trained networks is type 6, due to the complexity of the shape and the small amount of training data.

The next experiment addresses weight-matrix reduction. The first issue is how pruning affects the performance of the network. The weight distribution of trained neural networks, shown in figure 16, seems to be normal with the exception of those less than -0.25 and those greater than 0.25. 63.56% of the links are between -0.15 and 0.15. Figure 17 shows the recognition rates when links in some intervals are pruned. The network continues producing

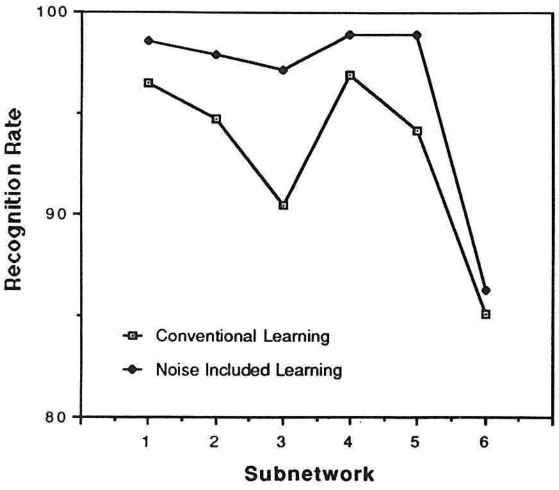


Figure 15: Comparison of conventional vs. noise-included learning. Conventional learning uses the original data. Noise-included learning uses 5 pairs of data, consisting of the original data and additionally modified patterns created by shifting them over a predetermined set of directions, such as up, down, left, and right.

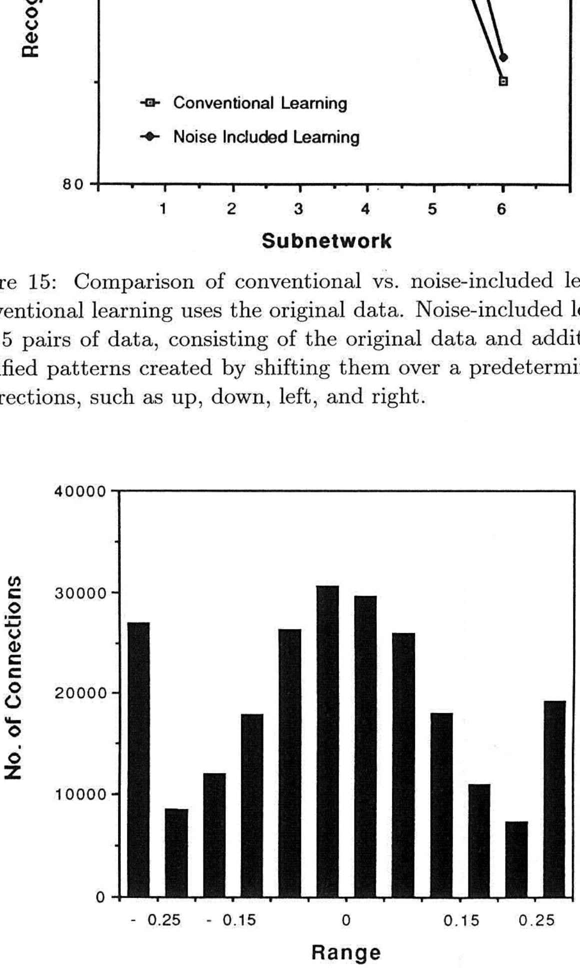


Figure 16: Weight distribution of a trained neural network. The weights seem to be normally distributed, with the exception of those less than −0.25 and greater than 0.25. Most of the weights—63.56%—are between −0.15 and 0.15.

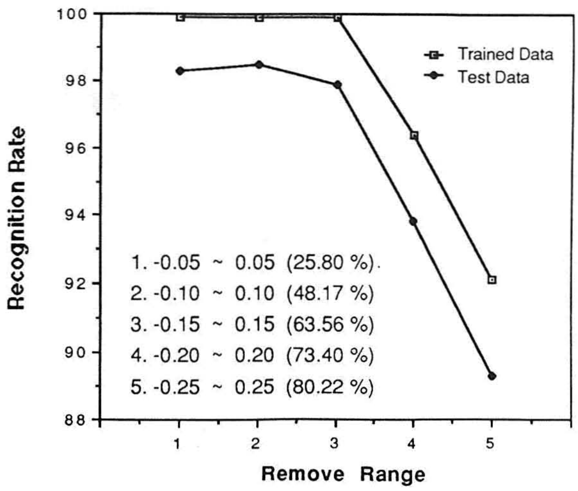


Figure 17: Recognition rates of the weight-matrix reduction. The network continues producing acceptable results until more than half of the links are pruned.

acceptable results until more than half of the links are pruned. Notice the improvement of the recognition rate with test data when pruning 48.17% of the links in the range $-0.10 \sim 0.10$. This means that redundant links actively detract from the network's ability to recognize distorted inputs. However, because further pruning degrades the network's performance, the proper limit of reduction is important.

To show the superiority of the neural network approach compared with conventional methods, some extra experiments were conducted. Figure 18 shows a comparison with a tree classifier, which is an efficient implementation of the nearest-neighbor rule [7]. In this figure, TC(i) represents the tree classifier constructed with one set of training data when i is 1, five sets when i is 2, and ten sets when i is 3; NN(i) represents the two-stage neural-network classifier trained with simple backpropagation when i is 1, and the noise-included training when i is 2. This comparison proves that multi-layer perceptron classifiers trained with backpropagation can perform as well as, or better than, conventional classifiers [35].

To generalize the capability of the proposed neural networks, we tried to recognize all 2,350 syllables using the system trained with 990 syllables. Table 3 shows the recognition results, which seem to be acceptable.

6. Concluding remarks

In this paper, we presented several strategies that are useful for applying feedforward neural networks to large-scale classification problems and applied them to the recognition of large character sets. Experiments with the

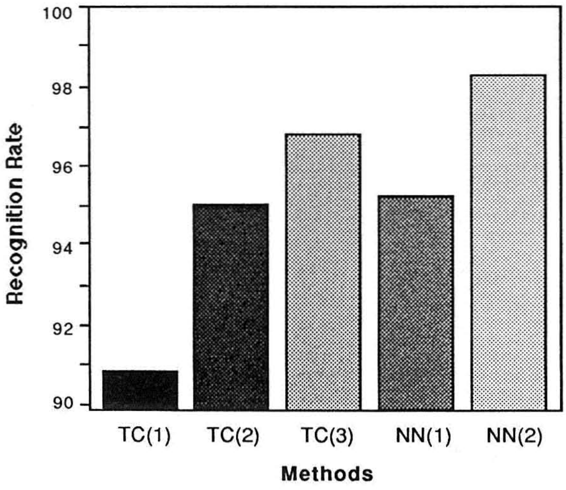


Figure 18: Comparison of the neural network with a statistical method. TC(i) represents the tree classifier constructed with one set of training data when i is 1, five sets when i is 2, and ten sets when i is 3; NN(i) represents the two-stage neural-network classifier trained with simple backpropagation when i is 1, and the noise-included training when i is 2.

group		*group identification rate*		*character recognition rate in each group*		*overall*	
id	#char	#error	%correct	#error	%correct	#error	%correct
1	149	4	97.32	4	97.32	8	94.63
2	90	2	97.78	5	94.44	7	92.22
3	109	14	87.16	48	55.96	52	52.29
4	1069	4	99.63	104	90.27	106	90.08
5	586	3	99.49	46	92.15	47	91.98
6	347	164	52.74	211	39.19	256	26.22
overall	2350	191	91.87	418	82.21	476	79.74

Table 3: Generalization capability. Recognition rate of 2,350 syllables using a neural network trained with 990 syllables.

990 most frequently used Hangul syllables were conducted to show the usefulness of these strategies. With selective reinforcement learning, the network learned twice as fast as conventional learning. For noise-included training, the recognition rate was 98.28%, which is superior to conventional methods. The network continued producing acceptable results until more than half of the links were pruned.

The fast learning method accelerated the learning speed using selective-reinforcement learning after the number of training patterns was increased by systematically adding the meaningful noises. In subsequent work, small computers may be used to solve large-scale problems by removing unnecessary weights from the trained ones.

Appendix A. Accelerated learning with Aitken's Δ^2 process

Let $\{w_n\}^\infty$ be a linearly convergent sequence of values converging to some point p; that is, for $e_n = w_n - p$,

$$\lim_{n\to\infty} \frac{|e_{n+1}|}{|e_n|} = \mu \leq 1. \tag{18}$$

To investigate the construction of a sequence $\{\underline{w}_n\}^\infty$, which converges more rapidly to p, suppose that the iteration $w_n = g(w_{n-1})$, $n = 1, 2, 3, \ldots$, converges linearly, so it satisfies

$$w_{n+1} - p = \mu(w_n - p) \tag{19}$$

and

$$w_{n+2} - p = \mu(w_{n+1} - p). \tag{20}$$

Solving equations (19) and (20) for p while eliminating μ leads to

$$\begin{aligned} p &= \frac{w_n w_{n+2} - w_{n+1}^2}{w_{n+2} - 2w_{n+1} + w_n} \\ &= w_n - \frac{(w_{n+1} - w_n)^2}{w_{n+2} - 2w_{n+1} + w_n} = w^* = \Delta^2(w_n) \end{aligned} \tag{21}$$

In general, the original assumption (19) will not be true; nevertheless, it is expected that the sequence $\{\underline{w}_n\}^\infty$, defined by

$$\underline{w}_n = w_n - \frac{(w_{n+1} - w_n)^2}{w_{n+2} - 2w_{n+1} + w_n}, \tag{22}$$

converges more rapidly to p than the original sequence $\{w_n\}^\infty$. The point w^* is a better approximation of p than is w_n or w_{n+1}. Graphically, the solution of $w = g(w)$ amounts to the problem of finding the point of intersection of the curves $y = w$ and $y = g(w)$, and w^* is the solution of the linear interpolant to $g(w)$ at w_n, w_{n+1} (see figure 19). If $g(w)$ is approximately a straight line between w_n and p, then the secant $s(w)$ is a very good approximation of

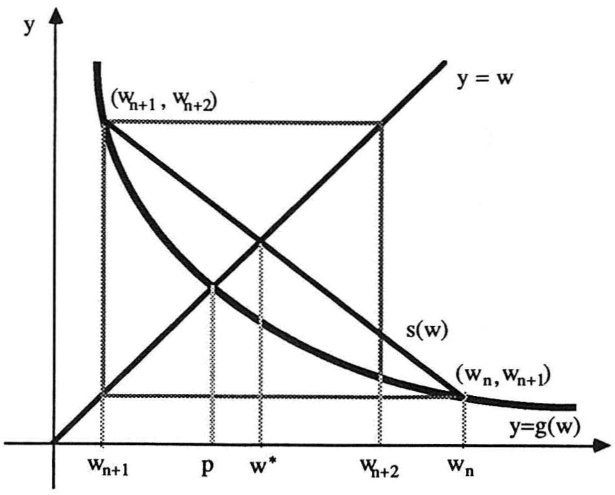


Figure 19: Graphical representation of repeated substitution. The solution of $w = g(w)$ amounts to the problem of finding the point of intersection of the curves $y = w$ and $y = g(w)$, and w^* is the solution of the linear interpolant to $g(w)$ at w_n, w_{n+1}.

$g(w)$ in that interval; hence the fixed point w^* of the secant is a very good approximation of the solution p. In this way, the better approximation is found in each iteration.

This process, called Aitken's Δ^2 process, accelerates the convergence of any sequence that is linearly convergent, and gives quadratic convergence without evaluating a derivative. Moreover, Aitken's Δ^2 process not only accelerates convergence but also converts divergence into convergence in some cases. It is easy to verify that if

$$w_n = 1 + p + \cdots + p^n, \tag{23}$$

then

$$\Delta^2(w_n) = \frac{1}{1-p}. \tag{24}$$

Therefore, Aitken's Δ^2 process immediately gives the limit of this particular sequence when it converges, and assigns a meaningful value even when it does not converge.

Acknowledgments

This work was supported in part by a grant from the Korea Science and Engineering Foundation (KOSEF), and by a grant from the Korea Advanced Institute of Science and Technology.

References

[1] R. Hecht-Nielsen, "Neurocomputer Applications," *NATO ASI Series F: Computer and Systems Sciences*, **41** (1988) 445–453.

[2] R. P. Gorman and T. J. Sejnowski, "Analysis of Hidden Units in a Layered Network Trained to Classify Sonar Targets," *Neural Networks*, **1** (1988) 75–89.

[3] T. J. Sejnowski and C. M. Rosenberg, "Parallel Networks that Learn to Pronounce English Text," *Complex Systems*, **1** (1987) 145–168.

[4] A. Waibel, "Connectionist Glue: Modular Design of Neural Speech Systems," pages 417–425 in *Proceedings of the 1988 Connectionist Models Summer School* (1988).

[5] A. N. Kolmogorov, "On the Representation of Continuous Functions of Many Variables by Superposition of Continuous Functions of One Variable and Addition," *Doklady Akademii Nauk SSSR*, **144** (1957) 679–681; *American Mathematical Society Translation*, **28** (1963) 55–59.

[6] D. E. Rumelhart and J. L. McClelland, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition* (Cambridge, MIT Press, 1986).

[7] R. O. Duda and P. E. Hart, *Pattern Classification and Scene Analysis* (New York, Wiley, 1973).

[8] S.-B. Cho and J. H. Kim, "A Two-stage Classification Scheme with Backpropagation Neural Network Classifiers," *Pattern Recognition Letters*, **13** (1992) 309–313.

[9] S. Becker and Y. le Cun, "Improving the Convergence of Back-Propagation Learning with Second Order Methods," pages 29–37 in *Proceedings of the 1988 Connectionist Models Summer School* (1988).

[10] D. B. Parker, "A Comparison of Algorithms for Neuron-like Cells," pages 327–332 in *Neural Networks for Computing*, edited by J. S. Denker (New York, American Institute of Physics, 1986).

[11] D. B. Parker, "Optimal Algorithms for Adaptive Networks: Second Order Backpropagation, Second Order Direct Propagation, and Second Order Hebbian Learning," pages 593–600 in *Proceedings of the IEEE International Conference on Neural Networks II* (1987).

[12] S. E. Fahlman, "An Empirical Study of Learning Speed in Back-Propagation Networks," Technical Report CMU-CS-88-162 (Carnegie-Mellon University, June 1988).

[13] P. Haffner, A. Waibel, H. Sawai, and K. Shikano, "Fast Back-Propagation Learning Methods for Large Phonemic Neural Networks," Technical Report TR-I-0058 (ATR Interpreting Telephony Research Laboratory, 1988).

[14] R. A. Jacobs, "Increased Rates of Convergence through Learning Rate Adaptation," *Neural Networks*, **1** (1988) 295–307.

[15] J. E. Angus, "On the Connection between Neural Network Learning and Multivariate Nonlinear Least Squares Estimation," *International Journal of Neural Networks: Research & Applications*, **1** (1989) 42–47.

[16] R. L. Watrous, "Learning Algorithms for Connectionist Networks: Applied Gradient Methods of Nonlinear Optimization," pages 619–627 in *Proceedings of the IEEE International Conference on Neural Networks II* (1987).

[17] S.-B. Cho and J. H. Kim, "An Accelerated Learning Method with Backpropagation," pages 605–608 in *Proceedings of the IEEE International Joint Conference on Neural Networks* (1990).

[18] M. Kam, R. Cheng, and A. Guez, "On the Design of a Content-Addressable Memory via Binary Neural Networks," pages 513–522 in *Proceedings of the IEEE International Conference on Neural Networks II* (1987).

[19] D. D. Nguyen and J. S. J. Lee, "A New LMS-Based Algorithm for Rapid Adaptive Classification in Dynamic Environments," *Neural Networks*, **2** (1989) 215–228.

[20] Y. Mori and K. Yokosawa, "Neural Networks that Learn to Discriminate Similar Kanji Characters," pages 332–347 in *Advances in Neural Information Processing Systems I*, edited by D. S. Touretzky (San Mateo, Morgan Kaufmann, 1989).

[21] K. Fukushima, "A Neural Network for Visual Pattern Recognition," *IEEE Computer*, **21**(3) (1988) 65–74.

[22] W. L. Reber, "An Artificial Neural System Design for Rotation and Scale Invariant Pattern Recognition," pages 277–283 in *Proceedings of the IEEE International Conference on Neural Networks IV* (1987).

[23] A. Khotanzad and J. H. Lu, "Distortion Invariant Character Recognition by a Multi-Layer Perceptron and Backpropagation Learning," pages 635–632 in *Proceedings of the IEEE International Conference on Neural Networks I* (1988).

[24] B. Widrow and R. G. Winter, "Neural Nets for Adaptive Filtering and Adaptive Pattern Recognition," *IEEE Computer*, **21**(3) (1988) 25–39.

[25] A. Waibel, T. Hanazawa, G. Hinton, K. Shikano, and K. J. Lang, "Phoneme Recognition Using Time-Delay Neural Networks," *IEEE Transactions on ASSP*, **37** (1989) 328–339.

[26] R. K. Bernhard and A. K. Wolfgang, "Design of Hierarchical Perceptron Structures and their Application to the Task of Isolated-Word Recognition," pages 243–249 in *Proceedings of the IEEE International Joint Conference on Neural Networks I* (1989).

[27] T. Matsuoka, H. Hamada, and R. Nakatsu, "Syllable Recognition Using Integrated Neural Networks," pages 251–258 in *Proceedings of the IEEE International Joint Conference on Neural Networks I* (1989).

[28] A. K. Somani and N. Penla, "Compact Neural Network," pages 191–198 in *Proceedings of the IEEE International Conference on Neural Networks III* (1987).

[29] S. Lehar and J. Weaver, "A Developmental Approach to Neural Network Design," pages 97–104 in *Proceedings of the IEEE International Conference on Neural Networks II* (1987).

[30] J. Sietsma and R. J. F. Dow, "Neural Net Pruning: Why and How," pages 325–332 in *Proceedings of the IEEE International Conference on Neural Networks I* (1988).

[31] M. C. Mozer and P. Smolensky, "Skeletonization: A Technique for Trimming the Fat from a Network via Relevance Assessment," pages 107–115 in *Advances in Neural Information Processing Systems I*, edited by D. S. Touretzky (San Mateo, Morgan Kaufmann, 1989).

[32] E. D. Karnin, "A Simple Procedure for Pruning Back-Propagation Trained Neural Networks," *IEEE Transactios on Neural Networks*, **1** (1990) 239–242.

[33] R. O. Lippmann, "An Introduction to Computing with Neural Nets," *IEEE ASSP Magazine*, (April 1987) 4–22.

[34] Y. le Cun, "Generalization and Network Design Strategies," pages 143–155 in *Connectionism in Perspective*, edited by R. Pfeifer et al. (Netherlands, North-Holland, 1989).

[35] W. Y. Huang and R. P. Lippmann, "Comparisons between Neural Net and Conventional Classifiers," pages 485–493 in *Proceedings of the IEEE International Conference on Neural Networks IV* (1987).