Proceedings of the 2009 IEEE International Conference on Systems, Man, and Cybernetics
San Antonio, TX, USA - October 2009

Speed-up of the R^4-rule for Distance-Based Neural Network Learning

Naoki Tominaga and Qiangfu Zhao
System Intelligence Laboratory
The University of Aizu
Aizu-Wakamatsu, Japan
m5111124@gmail.com, qf-zhao@u-aizu.ac.jp

***Abstract*—The R^4-rule is a heuristic algorithm for distance-based neural network (DBNN) learning. Experimental results show that the R^4-rule can obtain the smallest or nearly smallest DBNNs. However, the computational cost of the R^4-rule is relatively high because the learning vector quantization (LVQ) algorithm is used iteratively during learning. To reduce the cost of the R^4-rule, we investigate three approaches in this paper. The first one is called the distance preservation (DP) approach, which tries to reduce the number of times for calculating the distance values, and the other two are based on the attentional learning concept, which try to reduce the number of data used for learning. The efficiency of these methods is verified through experiments on several public databases.**

***Index Terms*—Distance-based neural networks, nearest neighbor classifiers, neural networks, linear vector quantization, R^4-rule, attentional learning, pattern recognition.**

I. Introduction

A distance-Based neural network (DBNN) is a nearest neighbor classifier (NNC) realized in neural network (NN) form. In the literature, DBNN is widely known as self-organizing neural network, though DBNN is a model suitable both for un-supervised learning and for supervised learning [1] – [8]. An DBNN has a set P of neurons which are usually arranged in one layer. For any given input pattern x, the DBNN finds a neuron $p \in P$ whose weight vector is the nearest (or most similar) to x, and assigns the class label of p to x.

A direct way to design a DBNN is to put all training data into P, with the weight vector of each neuron being one of the training data. Unfortunately, the network so obtained is not efficient if the number of data is large. A more efficient way is to train a small DBNN using the training data, such that $|P|$, or the number of neurons, is much smaller than the number of data. The question is, how small $|P|$ should be for a given problem. To answer this question, we proposed an algorithm called the R^4-rule for finding the smallest or nearly smallest DBNN in [9]. One drawback in using the R^4-rule is that it is relatively time-consuming because the learning vector quantization (LVQ) algorithm is used iteratively during learning.

To reduce the cost of the R^4-rule, this paper investigates three approaches. The first one is called the distance preservation (DP) approach, which tries to reduce the number of times for calculating the distance values, and the other two are based on the attentional learning concept (AL), which try to reduce the number of data actually used for learning. In the DP approach, the distances between a neuron and all data are calculated and saved, and are used when needed. The distances are re-calculated only if the neuron is updated during learning.

The two AL based approaches are called the attentional LVQ (ALVQ) and the attentional R^4-rule (ARR), respectively. Both approaches try to reduce the number of data used for updating the neurons based on the *difficulty* or the *easiness* of each datum. The basic idea is that more difficult data should be used more frequently during learning. In the ALVQ, easy data are gradually forgotten, and are used with lower and lower frequencies during learning. In the ARR, difficult data are selected in each learning cycle, and only selected data are used for updating the neurons.

Experimental results with several public databases show that all three methods can accelerate the R^4-rule significantly while keeping the performance of the obtained DBNN relatively unchanged. The rest of paper is organized as follows. Section II provides a brief review of the R^4-rule. In Section III, we introduce the three methods for speedup of the R^4-rule. Section IV describes the experimental methods, the experimental results, and a discussion of the results. Section V provides the conclusion and some topics for future work.

II. A Brief Review of the R^4-Rule

The R^4-rule is a heuristic learning algorithm that can obtain the smallest or nearly smallest DBNN. It adds or removes neurons during learning so that to approximate the desired error rate with the least number of neurons. The R^4-rule consists of four basic operations with the initial 'R': *Recognition*, *Remembrance*, *Reduction*, and *Review*. One learning cycle contains one of the following two sets: *Recognition*, *Remembrance*, and *Review*; or *Recognition*, *Reduction*, and *Review*.

In *Recognition*, the recognition rate r of the DBNN and the fitness of each neuron are calculated. The fitness of a neuron shows how important the neuron is for recognition. The following explains how to calculate the fitness of a neuron:

1) All the fitness values are initialized to zero in each learning cycle.
2) When $x \in T$ (T is the training set) cannot be classified correctly after removing a neuron p from P, update

$fitness\ (p)$, the fitness of p, as follows:

$$fitness\ (p) = fitness\ (p) + 1 \tag{1}$$

3) When x can be classified correctly only after removing p from P, update $fitness\ (p)$ as follows:

$$fitness\ (p) = fitness\ (p) - 1 \tag{2}$$

The operation after *Recognition* is selected from *Remembrance* and *Reduction* based on r. When r is less than a desired value r_0, *Remembrance* is selected; otherwise, *Reduction* is selected.

In *Remembrance*, a new neuron p_0 is added to P. The weight of p_0 is one of the training examples which cannot be recognized correctly with the current P. In *Reduction*, the neuron with the lowest fitness is removed from P.

In *Review*, the weights of neurons in P are updated using all the examples in T with the DSM algorithm, which is one of the LVQ algorithms proposed in [7]. The weights are updated only if the current DBNN cannot recognize $x \in T$ correctly. The following shows how to update P using x:

$$\begin{aligned} w_0 &= w_0 + \alpha(x - w_0) \\ w_1 &= w_1 - \alpha(x - w_1) \end{aligned} \tag{3}$$

where w_0 is the weight of the neuron nearest to x and which is in the same class x belongs to; w_1 is the weight of the neuron nearest to x and which is in a different class than x; and α is the convergence ratio.

In using the R^4-rule, the desired recognition rate r_0 is an important parameter. We may determine r_0 using some other method. In this study, we just determine r_0 as follows.

1) Generate an appropriately large number of neurons at random for each class.
2) Train the DBNN with the same algorithm as used for *review*.

The DBNN so obtained is used as the initial network for R^4-rule based learning, and the recognition rate of this initial network is used as the desired recognition rate r_0.

Fig. 1 is the flowchart of the R^4-rule based learning. The parameter $count$ is the present number of learning cycles and MAX is the maximum number of learning cycles.

III. Methods for Speedup of the R^4-rule

The bottleneck of the R^4-rule is the computation of the distances between all training examples and all neurons for all epochs in *Review*. The basic idea of solving this problem is to reduce the number of distance calculations. In this paper, we consider three approaches for this purpose. The first one is the DP approach, the second one is ALVQ, and the third one is ARR. Actually, ALVQ has been proposed by us for fast induction of multivariate decision trees [10], [11]; and ARR is a new method proposed in [12].

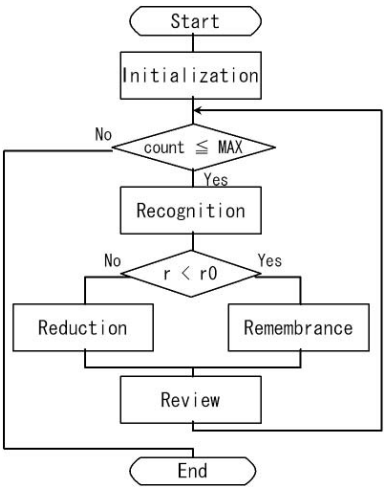

Fig. 1. Flowchart of the R^4-rule.

A. Distance Preservation

In the original R^4-rule, the distances between all neurons and all training examples are calculated in *Review*. The distances of the data to some of the neurons, however, do not have to be re-calculated if these neurons have not been updated since the last calculations. This is possible because we are using the DSM algorithm, in which the neurons are updated only if some datum is not classified correctly. Based on this fact, we proposed the R^4-rule with distance preservation (DP) [12].

In the DP approach, the distance $d(x, p)$ between an example x and a neuron p is saved after it is calculated. When $d(x, p)$ is needed, the saved value is used instead of calculating $d(x, p)$ again if p has not been updated since the last calculation of $d(x, p)$. The distance $d(x, p)$ is recalculated only if p has been updated. The DBNN trained with DP is exactly the same as that trained without DP and therefore there is no change in performance.

B. Attentional LVQ

The basic idea of attentional learning (AL) is to pay more attention to difficult data during learning. In the ALVQ approach, the concept of AL is applied to each epoch of *Review* in the R^4-rule. It is decided, based on a difficulty parameter, whether a training example is used or not for training in each epoch of *Review*. The larger the difficulty for an example, the more possible the example is used.

Specifically, in the ALVQ approach, each example x is assigned a parameter $D(x)$, which is a measure of the difficulty. The difficulty $D(x)$ takes its value in [0,1] and is initialized to 1 before *Review*. It is updated as follows in *Review*:

1) Generate a random number r in [0,1].
2) If $r > D(x)$, P and $D(x)$ is not updated using x; otherwise update $D(x)$ as explained below.
3) Find the nearest neuron p_0 in P.
4) If the label of p_0 is the same as that of x, that is, if the current DBNN can recognize x correctly, update $D(x)$

as follows:

$$D(x) = (1 - \gamma)D(x) \tag{4}$$

where γ is called the forgetting factor, and takes value in [0, 1). For a large γ, easy examples will be *forgotten* quickly during learning.

5) If the label of p_0 is different from that of x, that is, if the current DBNN cannot recognize x correctly, reset $D(x)$ to 1 and update neurons with x.

With the ALVQ, the computational cost can be reduced greatly. This is because, after several epochs, many training examples become recognizable and are used for training with low possibilities.

C. Attentional R^4-Rule

In the ARR, the concept of AL is applied to the whole process of the R^4-rule. It is decided, based on the difficulty parameter, whether a training example is used or not in *Reveiw*. The larger the difficulty of an example, the more probable the example is selected. In *Review* of the ARR, only a small number of selected examples (say, m examples, where $m \ll n$ and n is the number of training examples) are used, and thus the time for learning can be greatly reduced.

The experiments in [13] show that we can actually construct, for redundant training set, a smaller decision tree with better performance using part of the training examples. The partial training set is determined using the genetic algorithm in [13]. In the ARR, on the other hand, the partial training set is determined based on the AL concept, and is then used for training of DBNNs.

In the ARR, $D(x)$ is used to measure the difficulty of a example x as in the ALVQ. Each $D(x)$ is initialized to 1 before learning. In *Recognition*, $D(x)$ is updated for each example $x \in T$. If x is recognized correctly, update $D(x)$ as follows:

$$D(x) = D(x) - \epsilon \tag{5}$$

where ϵ is a small positive number. In this paper, we define $\epsilon \leq \frac{1}{MAX}$, where MAX is the maximum number of learning cycles in the R^4-rule. If x is not recognized correctly, $D(x)$ is reset to 1.

Before starting *Review*, $m(< n)$ examples are selected based on the difficulties. The following shows how m examples are selected.

1) Sort the examples in non-increasing order according to their difficulties $D(x)$.
2) Select the top m examples.

Only the selected examples are used in *Review*; all others are not used.

The computational cost of the R^4-rule can be much smaller with the ARR than that with the ALVQ. This is because only a small number of most difficult examples are used *Review*.

IV. Experiment

A. Experimental Method

To verify the efficiency of the proposed methods, we conducted experiments on the following databases taken from [14] : *cancer*, *diabetes*, *glass*, *iris*, *vehicle*, *optdigits*, *pendigits*, and *isolet*. Table I shows the parameters of the databases used in the experiments. To make the experimental results more reliable, five-fold cross-validation were conducted for 10 times for each database.

Table I
Parameters of the databases

Database	Number of Examples	Number of Features	Number of Classes
cancer	683	9	2
diabetes	768	8	2
glass	214	9	6
iris	150	4	3
vehicle	846	18	4
optdigits	5620	64	10
pendigits	10992	16	10
isolet	7797	617	26

Table II
Experiment parameters

Initial $\|P\|$	$5 \times n_c$
Maximum $\|P\|$	$8 \times n_c$
MAX (Maximum Number of Learning Cycles)	$16 \times n_c$
Initial Value for Learning Rate α	0.1
Number of Epochs in *Review*	20
Number of Epochs for Initialization	50

For comparison, we obtained the following results through the experiments for each database:

1) *DBNN-M*: DBNN with the neuron set containing all training examples.
2) *DBNN*: DBNN trained by the original R^4-rule.
3) *ALVQ-DBNN*: DBNN trained by the R^4-rule combined with the ALVQ, with $\gamma = 0.1$ or 0.5.
4) *ARR-DBNN*: DBNN trained by the R^4-rule combined with the ARR, with $m = 0.5n$ or $0.2n$.
5) *DP-DBNN*: DBNN trained by the DP-based R^4-rule.
6) *(DP+ALVQ)-DBNN*: DBNN trained by the DP-based R^4-rule combined with the ALVQ, with $\gamma = 0.1$ or 0.5.
7) *(DP+ARR)-DBNN*: DBNN trained by the DP-based R^4-rule combeind with the ARR, with $m = 0.5n$ or $0.2n$.

The main experiment parameters are shown in Table II. Here, n_c is the number of classes in the database and $|P|$ is the number of neurons.

B. Experimental Results

Table III and IV show the numerical results of the experiments. In the tables, *Size* is the number of neurons in the obtained DBNN; *Error* is the error rate for the test set; and *Time* is the training time in seconds. In each column of the tables, there are two values across the character "$\pm$"; the left is the average, and the right is the 95 percent confidence interval.

C. Discussion about Attentional LVQ

The experimental results show that the training times for *ALVQ-DBNN*, as expected, are smaller than those for *DBNN* for all the databases. In addition, the larger the forgetting factor γ is, the smaller the training times are used in *ALVQ-DBNN*.

Table III
COMPARISON OF DIFFERENT ALGORITHMS WITHOUT DP

Databases		DBNN-M	DBNN	ALVQ-DBNN $\gamma = 0.1$	ALVQ-DBNN $\gamma = 0.5$	ARR-DBNN $m = 0.5n$	ARR-DBNN $m = 0.2n$
cancer	Error	4.26±3.24	4.28±3.5	4.91±4.25	4.51±3.24	4.49±4.73	5.09±5.61
	Size	547±0	5.7±4.64	6.66±5.84	6.2±4.55	4.46±5.12	2.6±2.96
	Time	0±0	0.183±0.0848	0.118±0.0476	0.0607±0.0209	0.0868±0.0573	0.0285±0.0245
diabetes	Error	28.9±9.13	31.8±11.6	34.2±13.8	39±15.4	40.6±19	34.9±18
	Size	615±0	8.82±12.3	7.96±12.7	8.78±12.8	15.3±5.06	14.7±5.89
	Time	0±0	0.139±0.15	0.104±0.107	0.0693±0.069	0.0559±0.06	0.0323±0.0431
glass	Error	31±14.3	40±17.1	39.9±19.2	38.1±19.4	43±20.5	49±20.4
	Size	172±0	6±0	6.1±1.39	6.02±0.277	6.18±1.41	7.12±7.34
	Time	0±0	0.181±0.076	0.144±0.0881	0.112±0.0825	0.154±0.113	0.0941±0.0886
iris	Error	4.67±11.5	4.2±5.08	4.13±4.49	4.27±5.29	4.67±10.5	4.6±5.58
	Size	120±0	3.7±2.28	3.88±2.7	3.78±2.78	3.1±0.907	3±0
	Time	0±0	0.0399±0.0327	0.0255±0.0219	0.0117±0.0111	0.015±0.0103	0.00634±0.000463
vehicle	Error	30.7±4.41	22.6±6.76	22.7±6.52	24.7±6.86	25.7±6.91	39.9±10
	Size	677±0	10.8±14.7	10.4±13.4	10.2±10.4	8.34±13.7	30.4±11.4
	Time	0±0	0.97±0.562	0.723±0.379	0.432±0.237	0.453±0.32	0.15±0.184
optdigits	Error	1.37±0.639	3.62±0.892	3.79±1.02	3.81±1.27	3.87±1.13	3.98±2.89
	Size	4500±0	10±0	10±0	11.4±3.86	10±0	10±0
	Time	0±0	19±3.61	11±2.02	10.5±6.27	11.3±0.962	5.18±1.68
pendigits	Error	0.666±0.303	2.05±1.27	1.97±1.24	2.16±2.05	2.19±2.59	3.45±3.27
	Size	8790±0	17.1±4.73	17.3±3.77	19.3±6.2	17±4.67	13.8±4.54
	Time	0±0	56.7±4.24	35.1±5.73	17.6±5.46	31.9±5.68	14.8±3.27
isolet	Error	11.4±1.39	4.8±0.973	4.75±0.809	4.85±1.38	5.11±2.01	5.83±3.2
	Size	6240±0	26±0	26±0	26±0	26±0	26±0
	Time	0±0	1050±23.9	594±19.8	271±66	574±31.2	265±39.3

Table IV
COMPARISON OF DIFFERENT ALGORITHMS WITH DP

Databases		DP-DBNN	(DP+ALVQ)-DBNN $\gamma = 0.1$	(DP+ALVQ)-DBNN $\gamma = 0.5$	(DP+ARR)-DBNN $m = 0.5n$	(DP+ARR)-DBNN $m = 0.2n$
cancer	Error	4.28±3.5	4.91±4.25	4.51±3.24	4.49±4.73	5.09±5.61
	Size	5.7±4.64	6.66±5.84	6.2±4.55	4.46±5.12	2.6±2.96
	Time	0.0956±0.0451	0.0724±0.0285	0.0424±0.0136	0.0509±0.0325	0.0188±0.0154
diabetes	Error	31.8±11.6	34.2±13.8	39±15.4	40.6±19	34.9±18
	Size	8.82±12.3	7.96±12.7	8.78±12.8	15.3±5.06	14.7±5.89
	Time	0.132±0.159	0.106±0.119	0.0723±0.0796	0.0446±0.055	0.0184±0.0228
glass	Error	40±17.1	39.9±19.2	38.1±19.4	43±20.5	49±20.4
	Size	6±0	6.1±1.39	6.02±0.277	6.18±1.41	7.12±7.34
	Time	0.0907±0.0547	0.0795±0.0567	0.0691±0.0555	0.0849±0.0637	0.0531±0.0451
iris	Error	4.2±5.08	4.13±4.49	4.27±5.29	4.67±10.5	4.6±5.58
	Size	3.7±2.28	3.88±2.7	3.78±2.78	3.1±0.907	3±0
	Time	0.0151±0.015	0.0116±0.0118	0.00604±0.00648	0.00604±0.00695	0.0028±0.000734
vehicle	Error	22.6±6.76	22.7±6.52	24.7±6.86	25.7±6.91	39.9±10
	Size	10.8±14.7	10.4±13.4	10.2±10.4	8.34±13.7	30.4±11.4
	Time	0.666±0.395	0.552±0.286	0.369±0.199	0.357±0.269	0.0676±0.0881
optdigits	Error	3.62±0.892	3.79±1.02	3.81±1.27	3.87±1.13	3.98±2.89
	Size	10±0	10±0	11.4±3.86	10±0	10±0
	Time	8.02±3.31	5.57±2.36	10.9±8.9	4.47±2.2	3.07±1.94
pendigits	Error	2.05±1.27	1.97±1.24	2.16±2.05	2.19±2.59	3.45±3.27
	Size	17.1±4.73	17.3±3.77	19.3±6.2	17±4.67	13.8±4.54
	Time	70.7±10.4	55.8±10.8	33.2±9.91	50.4±12.2	59.8±25.3
isolet	Error	4.8±0.973	4.75±0.809	4.85±1.38	5.11±2.01	5.83±3.2
	Size	26±0	26±0	26±0	26±0	26±0
	Time	26.1±2.71	18.1±2.71	25.8±26.5	26±21.9	25.9±43

In particular, the training times for *optdigits*, *pendigits*, and *isolet* are significantly smaller compared with those in *DBNN*. We can, therefore, assume ALVQ is especially effective for databases with a large number of examples.

Also, the training times are shorter in *(DP+ALVQ)-DBNN* for all the databases except for *optdigits* than those in *DP-DBNN*. For *optdigits*, the times are longer when $\gamma = 0.5$ but shorter when $\gamma = 0.1$.

The training in *(DP+ALVQ)-DBNN* with $\gamma = 0.1$ is faster than that with $\gamma = 0.5$ whereas the training in *ALVQ-DBNN* is faster for *optdigits* and *isolet* when $\gamma = 0.5$ than when $\gamma = 0.1$. The possible reason is that ALVQ with $\gamma = 0.1$ has more distance calculations compared to that with $\gamma = 0.5$. DP can reduce more distance calculations when $\gamma = 0.1$ while it can reduce less distance calculations when $\gamma = 0.5$. It leads the faster training when $\gamma = 0.1$.

Compared with the error rates in *DBNN* and *DP-DBNN*, those of *ALVQ-DBNN* and *(DP+ALVQ)-DBNN* are higher for *diabetes*, *glass*, and *vehicle* while they are not significantly higher for *cancer*, *iris*, *optdigits*, *pendigits*, and *isolet*. We suppose it is because of the redundancy of the database. The former databases, we assume, have small redundancy; and the latter have large one.

Fig. 2 shows the relationship between the recognition rate of a test data set and γ and that between the training time and γ for *optdigits* (one of the redundant databases) and *vehicle* (one

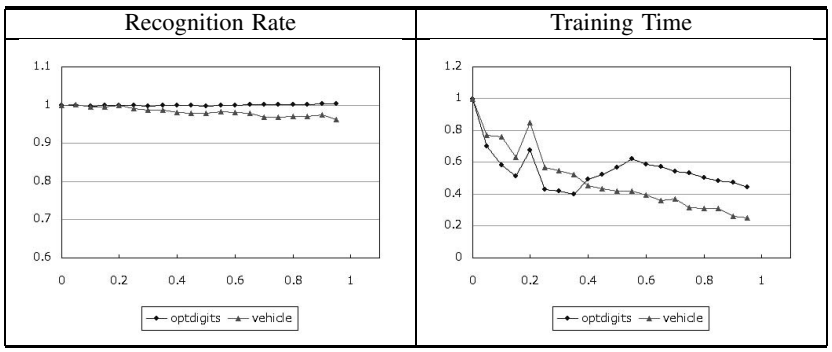

Fig. 2. Relation between ALVQ-based training of DBNNs and the forgetting factor γ.

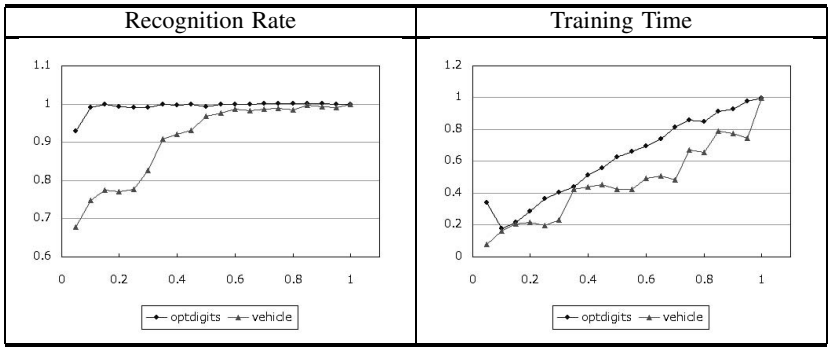

Fig. 3. Relation between ARR-based training of DBNNs and the training data set size m.

of the non-redundant databases). The horizontal axis indicates the value of γ and the vertical axis indicates the ratio how smaller (or larger) the recognition rate or the training time is with the ALVQ-based R^4-rule compared to those with the original R^4-rule.

We can see that the training time tends to be smaller for both databases when γ has larger value. The recognition rate, on the other hand, is smaller for *vehicle* and is almost the same for *optdigits* when γ has larger value. We think it means that we can set large γ to train fast *ALVQ-DBNN* with a high recognition rate for redundant databases. For non-redundant databases, however, we have trade-off between the recognition rate and the training time. We have to use small γ to keep the recognition rate and have to use large γ to make the training time small. Therefore, we have to set an appropriate γ to balance the recognition rate the training time.

To sum up, we can say ALVQ is effective for both the original R^4-rule and the DP-based R^4-rule especially when using databases with a large number of features and/or with large redundancy particularly when γ has a suitable value. Also for non-redundant databases, a good γ should be set for ALVQ to be effective.

D. Discussion about the Attentional R^4-Rule

The training times in *ARR-DBNN* are faster than those in *DBNN* for all the databases, as we expected. It is, of course, smaller with $m = 0.2n$ than with $m = 0.5n$. In particular, they are much shorter for *optdigits*, *pendigits*, and *isolet*. We can say ARR is effective specially for databases with the large number of examples. In addition, the training of *(DP+ARR)-DBNN* is faster than that of *DP-DBNN* for all the databases.

The numbers of neurons are smaller and the error rates are not significantly higher in *ARR-DBNN* and *(DP+ARR)-DBNN* for *cancer*, *iris*, *optdigits*, *pendigits*, and *isolet* than in *DBNN* and *DP-DBNN*. The numbers of neurons and the error rates are, however, much larger for *diabetes*, *glass*, and *vehicle*. We believe the reason here is again the redundancy of the database.

Fig. 3 shows the relationship between the recognition rate of a test data set and m and that between the training time and m for *optdigits* and *vehicle*. The horizontal axis indicates a where $m = a \times n$ and the vertical axis indicates the ratio how smaller (or larger) the recognition rate or the training time is with the ARR-based R^4-rule compared to those with the traditional R^4-rule.

The training time is smaller for both databases when m has small value. The recognition rate, on the other hand, is much smaller for *vehicle* and is almost the same for *optdigits* when m has small value. We think it means that we can set small m to train fast *ARR-DBNN* with a high recognition rate for redundant databases. It is, however, difficult to train fast *ARR-DBNN* with a high recognition rate for non-redundant

databases. We have to use large m to keep the performance and have to use large m to make the training time small. That is, we have to set an appropriate m to balance the recognition rate the training time.

Therefore, we can say that ARR is effective with redundant databases especially for databases with many examples. For non-redundant databases, m should be specified appropriately for good performance and fast training.

E. Comparison between Attentional LVQ and the Attentional R^4-Rule

We can find the difference between ALVQ and ARR by the comparison of Fig. 2 and 3. ARR makes the training faster for *optdigits* than ALVQ does while both methods can accelerate the R^4-rule while keeping the performance. For the database *vehicle*, both methods degrade the performance to a certain extent, but ARR degrades the performance more than ALVQ.

This implies, for redundant databases, ARR is more effective than ALVQ provided that a good m is selected. For non-redundant databases, on the other hand, ALVQ seems to be more effective when a good γ is specified. However, for a given database, we do not know it is redundant or not in advance, and therefore, it is difficult to provide good values for m and γ. This problem should be solved in the future to make the proposed methods more practically useful.

V. Conclusion and Future Works

In this paper, we have studied three methods for speedup of the R^4-rule. Experimental results show that the DP, although simple, is very effective. The two AL based approaches are also effective, although some parameters must be determined properly.

For further study, we should find γ and/or m which can make the R^4-rule based on ALVQ or ARR fast while keeping the performance of the trained DBNN. We can specify them by manual adjustment to be effective for most of the databases; or by proposal of an automatic method to identify the best value for a given example. The method to confirm the redundancy of a database before the learning of a DBNN is also necessary to make ALVQ and ARR more practically useful.

We have noticed that some interesting methods have been proposed in the literature for fast nearest neighbor search [15], [16], [17]. Specifically, [15] proposes avoidance of distance calculations based on a lower bound tree; [16] proposes the combination of projection search and a novel data structure; and [17] proposes partial distance search. No doubt, if we combine these methods with the ones proposed in this paper, we can speed up the R^4-rule further.

Acknowledgment

This research is supported in part by the Grants-in-Aid for Scientific Research of Japan Society for the Promotion of Science (JSPS), No.19500128.

References

[1] O. J. Murphy, "Nearest neighbor pattern classification perceptrons," in *Proc. IEEE*, vol. 78, no. 10, Oct. 1990, pp. 1595–1598.
[2] N. K. Bose and A. K. Garga, "Neural network design using voronoi diagrams," *IEEE Trans. on Neural Networks*, vol. 4, no. 5, pp. 778–787, Sept. 1993.
[3] T. Kohonen, "Self-organized formation of topologically correct feature maps," *Biological Cybernetics*, vol. 43, no. 1, pp. 59–69, 1982.
[4] T. Kohonen, "The self-organizing map," in *Proc. IEEE*, vol. 78, no. 9, Sept. 1990, pp. 1464–1480.
[5] G. A. Carpenter and S. Grossberg, "Art 2: self-organization of stable category recognition codes for analog input patterns," *Applied Optics*, vol. 26, no. 23, pp. 4919–4930, Dec. 1987.
[6] G. A. Carpenter and S. Grossberg, "The ART of adaptive pattern recognition by a self-organizing neural network," *IEEE Computer*, vol. 21, no. 3, pp. 77–88, Mar 1988.
[7] S. Geva and J. Sitte, "Adaptive nearest neighbor pattern classification," *Neural Networks, IEEE Transactions on*, vol. 2, no. 2, pp. 318–322, 1991.
[8] D. L. Reilly, L. N. Cooper, and C. Elbaum, "A neural model for category learning," *Biological Cybernetics*, vol. 45, no. 1, pp. 35–41, 1982.
[9] Q. F. Zhao and T. Higuchi, "Evolutionary learning of nearest-neighbor MLP," *IEEE Trans. on Neural Networks*, vol. 7, no. 3, pp. 762–767, 1996.
[10] Q. F. Zhao, "Inducing NNC-Trees with the R4-rule," *IEEE Trans. on Systems, Man, and Cybernetics/-Part B: Cybernetics*, vol. 36, no. 3, pp. 520–533, June 2006.
[11] Q. F. Zhao, "Inducing NNC-Trees quickly," in *Proc. IEEE International Conference on Systems, Man and Cybernetics (SMC06)*, Taipei, Oct. 2006, pp. 2784–2789.
[12] N. Tominaga, "Fast structural learning of distance-based neural networks," Master's thesis, The University of Aizu, Mar. 2009.
[13] T. Endou and Q. Zhao, "Generation of comprehensible decision trees through evolution of training data," in *Proceedings of the 2002 IEEE World Congress on Computational Intelligence*, 2002.
[14] D. J. Newman, S. Hettich, C. L. Blake, and C. J. Merz, "UCI repository of machine learning databases," 1998. [Online]. Available: http://www.ics.uci.edu/~mlearn/MLRepository.html
[15] Y.-S. Chen, Y.-P. Hung, and C.-S. Fuh, "Fast algorithm for nearest neighbor search based on a lower bound tree," in *Proc. of the 8th IEEE International Conference on Computer Vision*, vol. 1, Vancouver, Canada, Jul. 2001, pp. 446–453.
[16] S. A. Nene and S. K. Nayar, "A simple algorithm for nearest neighbor search in high dimensions," *IEEE Trans. on Pattern Analysis and Machine Intelligence*, vol. 19, no. 9, pp. 989–1003, Sep. 1997.
[17] D. Cheng, A. Gersho, B. Ramamurthi, and Y. Shoham, "Fast search algorithms for vector quantization and pattern matching," in *IEEE International Conference on Acoustics, Speech, and Signal Processing*, vol. 9, Mar. 1984, pp. 372–375.
[18] C. Bei and R. Gray, "An improvement of the minimum distortion encoding algorithm for vector quantization," *IEEE Trans. on Communications*, vol. 33, no. 10, pp. 1121–1133, 1985.