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:

\[ \text{fitness} (p) = \text{fitness} (p) + 1 \]  \( (1) \)

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

\[ \text{fitness} (p) = \text{fitness} (p) - 1 \]  \( (2) \)

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 \( x \):

\[ w_0 = w_0 + \alpha (x - w_0) \]

\[ w_1 = w_1 - \alpha (x - w_1) \]  \( (3) \)

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 \( \alpha \) 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].

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:

\[ D(x) = (1 - \gamma)D(x) \]  

(4)

where \( \gamma \) is called the forgetting factor, and takes value in \([0, 1]\). For a large \( \gamma \), 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 Review. 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 \( m < n \) 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:

\[ D(x) = D(x) - \epsilon \]  

(5)

where \( \epsilon \) is a small positive number. In this paper, we define \( \epsilon \leq \frac{1}{\text{MAX}} \), where \( \text{MAX} \) 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.

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 \( \alpha = 0.1 \) 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 \( \alpha = 0.1 \) or 0.5.
7) (DP+ARR)-DBNN: DBNN trained by the DP-based \( R^4 \)-rule combined 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 “±”; 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 \( \gamma \) is, the smaller the training times are used in ALVQ-DBNN.
In particular, the training times for optdigits, pendigits, and isolet are significantly smaller compared to 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 higher 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 $\gamma$ and that between the training time and $\gamma$ for optdigits (one of the redundant databases) and vehicle (one...
of the non-redundant databases). The horizontal axis indicates the value of $\gamma$ 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 $\gamma$ has larger value. The recognition rate, on the other hand, is smaller for vehicle and is almost the same for optdigits when $\gamma$ has larger value. We think it means that we can set large $\gamma$ 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 $\gamma$ to keep the recognition rate and have to use large $\gamma$ to make the training time small. Therefore, we have to set an appropriate $\gamma$ to balance the recognition rate and 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 $\gamma$ has a suitable value. Also for non-redundant databases, a good $\gamma$ 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 $m = a \times n$ 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
We can find the difference between ALVQ and ARR by the comparison of Fig. 2 and 3. ARR makes the training faster for 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, ARR 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 opdigits 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