Contender’s network, a new competitive-learning scheme

Geok See Ng, S.S. Erdogan *, Pan Wei Ng

Nanyang Technological University, School of Applied Science, Division of Computer Engineering, Nanyang Avenue,
Singapore 2263, Singapore

Received 16 August 1993; revised 9 March 1995

Abstract

Artificial Neural Networks (ANNs) have been used to perform classification for Automatic Speech Recognition (ASR). In this paper, we propose a new neural network, the Contenders’ Network (CN) which requires little initial knowledge of the classification problem and lesser neurons than other ANNs.

Keywords: Contenders’ Network, Pattern classification

1. Introduction

ANNs have the ability to compute complex decision surfaces; the numerous processing elements present in the architecture have the ability to classify objects and make complex decisions. We use the ANNs in the area of ASR. Although numerous techniques have been developed for ASR, ASR is still not widely used due to the cost of its implementation. One of the objectives of our work is to make the processing of ASR cheap, fast and readily accessible. The hardware that we use is a simple 8-bit ADC/DAC from a SoundBlaster Pro card on a 80486 PC. Experiments are conducted in environments with some degree of background noise on spoken Mandarin digits and restricted to isolated word recognition.

Our approach is to develop a firm foundation of reusable software routines to perform speech signal processing and to use them with some ANNs to perform phoneme and word classifications. We implemented some signal processing routines to perform feature extraction. Other paradigms such as Restricted Coulomb Energy (RCE) (Reilly and Cooper, 1990), Learning Vector Quantization (LVQ) (Kohonen, 1987) and Dynamic Vector Quantization (DVQ) (Poirier, 1991) are investigated and compared with the proposed Contenders’ Network (CN).

This paper continues with the presentation of the CN, the motivation for its development, and its implementation. The performance of the CN is compared with RCE, LVQ and DVQ using some Artificially Generated Vectors (AGVs). Then, CN is applied on real speech feature vectors. Finally, for demonstration, a square puzzle game is built. This game uses speech as input and is run in real time with an accuracy of 93.6% for 12 commands. The response time is around 6 seconds per command.

* Corresponding author.
2. Some current neural network classifiers

ANNs are widely used in ASR (Furui, 1989; Huang et al., 1990; Morgan and Scofield, 1991; Rabinor and Schafer, 1985; Saito and Nakata, 1985; Waibel, 1989). However, many of them, especially those that use the backpropagation (Simpson, 1990) learning algorithm, require excessive training time. Hence, some ANNs that have a relatively shorter training time such as LVQ, DVQ and RCE are chosen for comparison. Another reason why these ANNs are chosen is that they have similar architectures. This allows us to train the ANN using the learning algorithm of one network, and perform classification using another.

The basic architecture of LVQ, DVQ and RCE is shown in Fig. 1. The model has 3 layers: one input, one internal, and one output. The number of input neurons corresponds to the number of elements in the input feature vector. The internal neurons are the reference patterns in the feature space from which comparisons according to some distance measures are made. The output neurons correspond to the various classes in the feature space. The input and the internal layer is fully connected. Each internal neuron has one and only one link to an output neuron. The network has only feedforward connections. Although the three ANNs share the same architecture, each has different learning and classification algorithms. However, all of them use supervised learning.

The RCE network creates a hyper-sphere with an initial radius when an input feature vector is not represented. It shrinks the radius of the hyper-sphere if the output class is not the desired class. The internal layer of the network is empty initially. As new input patterns are presented, the internal layer spawns new neurons to cover these patterns. If the input pattern falls within the hyper-sphere of a neuron of a different class, the radius of the hyper-sphere is shrunk. As training proceeds, various hyper-spheres are created in the feature space. A problem with the RCE is the choice of the initial radius. If it is too small, excessive internal neurons will be created. If it is too large, training will be slow. This is because time will be spent in shrinking the neurons to the right size. Also, an input pattern can be within the region and yet not be covered by any hypersphere. Furthermore, a pattern created cannot be shifted. This means that if a bad pattern is presented during training, the error will also be learnt.

In LVQ, class membership of an input vector is specified by the class of the nearest reference. The LVQ learning algorithm shifts the position of a neuron closer or away from the pattern presented depending on the classification result. The number of neurons in the internal layer is fixed. As training proceeds, the internal neurons (i.e. reference patterns) shift to their optimal position. A major problem with LVQ is the choice of the initial internal layer. Normally, some pre-processing of the feature space is done to derive the internal layer. An advantage of LVQ over RCE is that all points in the feature space belongs to a class and there is no undefined region. Also, it uses lesser internal neurons.

DVQ is another way of creating the initial internal layer (Poireir, 1991). It uses the following learning algorithm:

If the closest reference from the input vector $x$ does not belong to the same class as $x$ and the closest reference in the class of $x$ is too far from $x$ (i.e. distance greater than a threshold $\sigma$) then create a new reference equal to $x$

else use LVQ rule

The problem with this approach is the value of the threshold $\sigma$. If $\sigma$ is too small, excessive internal neurons will be created. If $\sigma$ is too big, fewer neurons will be created.
3. Contenders’ network

3.1. Motivation

In the feature space, there will be regions where class membership is ill-defined and the decision surface is complex. In these cases, more reference patterns are needed to determine the decision surface and the reference patterns will appear to be closer. There will also be cases where class membership is very well defined. In these cases, we need lesser reference patterns to determine membership, and the reference patterns will appear to be further apart.

The above three ANNs’ performance is limited by several factors. For RCE, the factors are the radius of the reference patterns (i.e. hyper-spheres) and the inability of shifting the reference patterns. For LVQ, the factor is the initial number of reference patterns. For DVQ, the factor is the value of threshold \( \sigma \). In other words, we want an ANN that will gather more information if the decision surface is complex, and use less information if the decision surface is simple. The ANN must also be insensitive to the scale of the feature space. The Contenders’ Network is proposed to tackle these problems.

3.2. Overcoming sensitivity to scale

Only some of the internal neurons are needed in the classification process. These reference patterns are those that are near the input vector. The key idea is to identify those neurons which are near to the input vector. We choose the top \( k \) references which are closest to the input \( x \). However, being in the top \( k \) references does not mean that a reference is close. We need a measure to limit them to a region close to \( x \). Instead of using a fixed distance, which is sensitive to scale, we use a percentage \( \beta \) which is stated as follows:

\[
D( x, n_i ) < \beta D( x, n_1 )
\]

where \( 1.0 < \beta < 2.0, D \) is the distance between vectors, \( n_i \) is the \( i \)th closest neuron for \( i = 1, 2, \ldots, k \) and \( n_1 \) is the closest neuron.

Hence, the sensitivity to scale is overcome by the use of two parameters, \( k \) and \( \beta \). Only \( t \) out of the \( k \) references satisfy Eq. (1). This is illustrated in Fig. 2.

The \( t \) references are called contenders, since they contend for the input pattern’s class membership during classification. The class membership of the input vector \( x \) is a function of these \( t \) references and itself, i.e.

\[
\text{class of}( x ) = f( x, n_1, n_2, \ldots, n_t ).
\]

3.3. Strong and ordinary contenders

Before we discuss the classification and learning algorithm, we present the concept of the strong and ordinary contenders. As was mentioned, references within the RCE could be erroneous. Hence, it will be advisable to reduce the weight of these erroneous patterns. In the CN, strong contenders are reference patterns with higher weights and ordinary contenders are those with lower weights. This is illustrated in Fig. 3.

During classification, the closest reference pattern is known as the winner. The remaining of the top \( t \) reference patterns are strong contenders and weak contenders. Membership of the input to a certain class depends on the rank (i.e. according to distance) of the top \( t \) references shown in Fig. 3. The weight is a monotonically decreasing function of the rank.

During learning, references are assigned as strong or
weak contenders. If no further assignment of strong or ordinary contenders can be made, a new reference is added.

3.4. Classification

CN uses an input vector \( \mathbf{v} \), which is fed into the network. The top \( k \) closest internal neurons (i.e. references) are considered. If none of these neurons are comparable with the closest neuron (which we shall call \textit{winner}, as in LVQ), then the output class is the class which the \textit{winner} belongs to. Otherwise, among the top \( k \) neurons, all those that are within range are chosen for consideration. There are \( t (t < k) \) of them. Among these \( t \) references, there may be some references which are of different class, with a tendency to compete with the winner (\textit{strong contenders}). If no \textit{strong contenders} exist within the top \( t \) references, then nobody likes to compete with the winner and the output of CN is the winner's class. Otherwise a \textit{contenders' meeting} is held to resolve the top \( t \) references. In the \textit{contenders' meeting}, each reference casts a \textit{weighted vote} to its own class. The weighted vote is defined by

\[
\text{weight of}(\mathbf{n}_i) = \begin{cases} 
1.1 - 0.1i & \text{if } n_i \text{ is not a strong contender of } n_1, \\
1 & \text{if } n_i \text{ a strong contender of } n_1 
\end{cases}
\]

where \( n_i \) refers to the top \( i \)th reference and \( s \) is a constant with value close to the weight of the winner. The weights follow a linear scale. The winner \((i = 1)\) has a weight of one, the next nearest has a weight of 0.9 and the next 0.8 and so on. The strong contender has a weight close to that of the winner. The class with the highest sum of votes will be the output class. The classification algorithm follows:

\textbf{Step 1.} Find top \( k \) closest neurons: \( n_1, n_2, \ldots, n_k \), where \( n_1 \) is the winner.

\textbf{Step 2.} Out of these \( k \) neurons, select neurons meeting the criterion.

\[ D(\mathbf{v}, n_i) < \beta \ D(\mathbf{v}, n_1). \]

where \( \beta > 1.0 \) and \( D \) is the distance between the vectors; let \( t \) be the number of neurons meeting this criterion.

\textbf{Step 3.} If \( t = 1 \) (i.e. only the winner is present) exit with the class of \( n_1 \) as output.

\textbf{Step 4.} If among these \( t \) neurons, none of them are strong contenders for \( n_1 \), exit with the class of \( n_1 \) as output.

\textbf{Step 5.} Each of the \( t \) neurons cast the vote to its own class with the defined weight (according to rank).

\textbf{Step 6.} The output class is the class with the highest sum of vote.

Intuitively, we will expect that most of the \( k \) nearest references (internal nodes) will belong to the same class as the input vector, especially the nearest reference (winner). If any of these \textit{near} references are far from the input vector compared with the winner, they are not considered as they do not belong to the decision region. That is why we use two criteria, controlled by two parameters, to perform this task. The first parameter is \( k \) which helps us consider the top \( k \) references in terms of distance ranking. The second being a percentage \( \beta \) which compares the distance in terms of order. These two comparisons are done in Steps 2 and 3 of the above algorithm to limit the number of references to be considered. There are now \( t \) references meeting the two criteria, including the winner. If the winner is the only one that meets these criteria (i.e. \( t = 1 \)), we conclude that the other references are far from the decision region and the best representation of the input vector class is the class of the winner. However, if there are several references a \textit{Contenders' Meeting} is held. The contenders cast a weighted vote to see which class best represents the class of the input vector. The weights associated with each contender depend on their ranking in terms of distance from the input vector. The winner has the highest vote. We use here a linear weighting function instead of some similarity measure because this is a fuzzy region in the decision space where distance measures do not play a major part anymore. Rather, it is the number of contenders and the order of distance which matters. Each contender will cast his own vote to his own class and the class with the highest sum of votes wins. We shall introduce the concept of \textit{strong contenders}. Strong contenders are the references which tend to compete with the winner. These strong contenders have stronger weights than other
ordinary ones. If within the top \( t \) contenders none of these strong contender for the winner is present, the contenders' meeting is not held and the winner outputs his own class.

3.5. Learning

Each neuron has a list of its own \( u \) strong contenders. During learning, either this list is updated or a new node is added. If the output class is correct, no learning is required. If the output class is wrong, then the appropriate corrections are made. There are three steps to be considered to suppress any addition of new references into the contenders' network. This is achieved by making some contenders strong or by making some contenders weak. This adds a certain bias to the distance measure function and to bring a reference closer to the input. But this movement is local to the winner and is not visible to the other references as each reference keeps his own list of strong contenders. The learning algorithm follows.

Step 1. When no meeting is held, there are two reasons. The first reason is that only the winner is close to the input vector \((t = 1)\). In this case, the only way to solve this is to call in another reference close to the input vector. Since there are none existing in the network, a new reference whose position is the same as the input vector is added. The second reason is that there are no strong contenders present in the meeting which allows the winner to assume leadership. The way to resolve this is to make one reference whose class is that of the input vector (supervised learning) a strong contender. If all references have the same class as the input vector, we cannot make any reference strong anymore. In this case, a new reference is added.

Step 2. A meeting is held and the winner is in the right class. The contenders’ meeting overrules the winner which is right in the first place. In other words, there are some strong contenders present of the wrong class and yet cause this wrong class to be the output. A natural solution to this problem is to find one of these strong contenders and make him an ordinary contender. If none of them is found, a new reference is added as in Step 1.

Step 3. The winner is in the wrong class and the contenders’ meeting is still unable to resolve the issue properly. In this case, a reference who has the same class as the input vector is found and made a strong contender. If all such contenders are already strong, the conflict is resolved by adding a new reference corresponding to the input vector.

CN involves a number of user defined parameters: \( u \) is the maximum number of strong contenders a neuron can have, \( k \) is the maximum number of neurons that are considered at a contenders’ meeting, \( \beta \) represents the percentage allowance for the top \( k \) neurons to be considered close to the input as the winner \((\beta > 1)\) and \( s \) is the weight with value close to the winner. These parameters provide flexibility in configuring the CN. When \( k = 1 \), the CN classification procedure becomes that of LVQ, and its learning procedure becomes almost similar to that of RCE. The number of neurons, \( t \), in a contenders’ meeting has its upper limit set as \( k \). \( t \) is dependent on \( \beta \) which is set to be 2.0 in our implementation. Using a percentage rather than a fixed value (e.g. gamma in DVO) eliminates the need for a priori knowledge about the problem. Hence the need for preprocessing of the data is not required. Parameters of the CN are not problem specific. This alleviates the need to perform preprocessing of the data within the applied domain. Such preprocessing is more often time consuming and requires some human intervention and expertise. The algorithm suppresses the need to add a new node by identifying nodes as strongly competitive (i.e. strong contenders) or weakly competitive. It tries its best to suppress the addition of a new reference pattern. If it is unable to resolve the conflict, it gives up and adds a new reference pattern, thus the number of reference patterns does not grow so fast. Furthermore, CN restricts learning locally to the winner (i.e. the nearest reference) by updating its list of strong contenders. This operation affects the decision region of the winner only. Decision regions of other references, even the strong contender itself, are not affected. The decision region will be affected only when new references are added. It also remembers which neurons have tendency to compete with one another. Such competition exists at the decision boundaries and provides vital information concerning the decision surface. Such information can be used for pruning the CN.
3.6. Analysis of algorithms

The user defined parameters of the CN are:

- \( k \): maximum number of neurons that are considered at a contenders' meeting.
- \( \beta \): percentage allowance for top \( k \) neurons to be considered close to the input as the winner.

These parameters provide flexibility in configuring the CN. When \( k = 1 \), the CN classification procedures becomes that of LVQ, and its learning procedure resembles the one of the RCE. The number of neurons, \( t \), in a contenders' meeting has its upper limit set as \( k \) and \( t \) is dependent on \( \beta \). \( \beta \) is normally set to be 2.0. Using a percentage rather than a fixed value (e.g. \( \sigma \) in DVQ) eliminates the need for \textit{a priori} knowledge about the problem. Hence there is no need for pre-processing the data. Parameters of the CN are not problem specific, hence there is no need for pre-processing. Such pre-processing is more often time consuming and requires some human intervention and expertise.

The algorithm suppresses the need to add new neurons by identifying neurons as strongly competitive (i.e. strong contenders) or weakly competitive. In any case where an erroneous reference pattern (i.e. labeled wrongly) is added, strong contenders will suppress it. In this way, the algorithm is tolerant to faulty inputs. The CN restricts learning to be local for each reference node. This is achieved by having each reference maintain its own list of strong contenders. Adding a strong contender to the list does not affect decision elsewhere.

4. Experiments and results

In this section, we evaluate the performance of the various ANNs in their ability to classify speech feature vectors into phonemes. Experiments are conducted to compare the performance of the LVQ (using LVQ PAK software package from Helsinki University of Technology, Finland), DVQ, RCE and CN. The number of nodes for LVQ PAK is set to an optimal value in order to minimize the error and the number of iterations of the networks. We also apply the CN in speech recognition.

The criteria used for comparison are:
1. Training speed (in terms of iterations).
2. Resulting number of internal cells after training.
3. Statistical hit rates when using test patterns different from training set.

The statistical results are evaluated using the following measures:

- **Hit rate** = \( \frac{\text{number of correct classifications}}{\text{total number of inputs}} \).
- **Miss rate** = \( \frac{\text{number of incorrect classifications}}{\text{total number of inputs}} \).

The data used for classification experiments come from two sources:
1. Artificially Generated Vectors (AGV).
2. Partial Correlation Feature Vectors of Speech (PCFV).

AGV are used to test the performance of the ANN classifiers to see how well they perform. AGV are
selected such that they represent difficult situations in classification, such as overlap. PCFV are used to evaluate the ability of ANN classifiers to classify speech data. In these experiments, only a single frame input vector is used as input to the ANN.

4.1. Synthetic data

The two classification problems shown in Fig. 4 are used for comparison. Problem 1 has two classes with some overlap. Problem 2 represents a difficult task in classification. The problem has two classes such that one of them lies exactly within the other. For each classification experiment two sets of data are generated. These two sets of data come from the same source (i.e. the same probability distribution) although the points in the two sets must not coincide. One set is used to train the ANN classifier and the other is used to test the classification ability of the ANN. The classification performance results are shown in Tables 1 and 2. In both tables, the row labeled 'Input' is the number of inputs of each class being fed into the network and the row labeled 'Output' gives the number of inputs classified as class 1 or 2 regardless of the input class. The rows for Hit and Miss are self-explanatory. The Error rate is the percentage of wrong classifications, i.e., the total number of misses over the total number of inputs for a given experiment.

4.2. Phonetic input

After testing on synthetic data, experiments are conducted with actual speech feature vectors. Phoneme classification is an important step in large-vocabulary word recognition systems. Phoneme classification reduces the dimensionality of the problem. That is, it converts a feature vector into a single symbolic unit, the phoneme as shown in Fig. 5. After phoneme classification, the observed feature vector sequence is converted to a phoneme sequence. The phoneme sequence is discrete and can be fed into the Hidden Markov Model (HMM) for recognition.

In the experiments PARCORR coefficients of order 8 are used. They are extracted from speech sampled at 12 kHz, 21 ms frame size with 15 ms frame interval. PARCORR is used throughout all our experiments as preliminary results suggest that it is the best feature among LPC, LPC cepstrum, and PARCORR. 12 phonemes corresponding to vowels and consonants in spoken mandarin pronunciation are used. The results are shown in Table 3.

---

**Table 1**

Results for Problem 1

<table>
<thead>
<tr>
<th>Classifier</th>
<th>CN</th>
<th>DVO</th>
<th>LVQ</th>
<th>PAK</th>
<th>RCE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Iterations</td>
<td>10</td>
<td>8</td>
<td>8</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>Nodes</td>
<td>72</td>
<td>175</td>
<td>32</td>
<td>34</td>
<td></td>
</tr>
<tr>
<td>Class</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>Input</td>
<td>150</td>
<td>150</td>
<td>150</td>
<td>150</td>
<td>150</td>
</tr>
<tr>
<td>Output</td>
<td>141</td>
<td>159</td>
<td>144</td>
<td>148</td>
<td>152</td>
</tr>
<tr>
<td>Hit</td>
<td>129</td>
<td>138</td>
<td>125</td>
<td>125</td>
<td>123</td>
</tr>
<tr>
<td>Miss</td>
<td>21</td>
<td>12</td>
<td>27</td>
<td>27</td>
<td>25</td>
</tr>
<tr>
<td>Error (%)</td>
<td>11.0</td>
<td>16.0</td>
<td>17.3</td>
<td>17.3</td>
<td></td>
</tr>
</tbody>
</table>

**Table 2**

Results for Problem 2

<table>
<thead>
<tr>
<th>Classifier</th>
<th>CN</th>
<th>DVO</th>
<th>LVQ</th>
<th>PAK</th>
<th>RCE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Iterations</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>Nodes</td>
<td>43</td>
<td>118</td>
<td>94</td>
<td>58</td>
<td></td>
</tr>
<tr>
<td>Class</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>Input</td>
<td>50</td>
<td>150</td>
<td>50</td>
<td>150</td>
<td>50</td>
</tr>
<tr>
<td>Output</td>
<td>55</td>
<td>145</td>
<td>44</td>
<td>156</td>
<td>47</td>
</tr>
<tr>
<td>Hit</td>
<td>37</td>
<td>132</td>
<td>31</td>
<td>137</td>
<td>32</td>
</tr>
<tr>
<td>Miss</td>
<td>13</td>
<td>18</td>
<td>19</td>
<td>13</td>
<td>18</td>
</tr>
<tr>
<td>Error (%)</td>
<td>15.5</td>
<td>16.0</td>
<td>16.5</td>
<td>17.5</td>
<td></td>
</tr>
</tbody>
</table>

---

**Fig. 5.** ANN phoneme classifier.
Several observations are drawn from the experiments. The LVQ PAK has rather good results as the initial references are generated by a single iteration of the RCE. However, after training, the LVQ PAK does not increase in classification capability. RCE’s poor performance is largely due to non-classification (i.e. input not within radius of any reference pattern).

The experiments show that CN selects a near-optimal number of reference patterns. Having too many reference patterns causes the ANN to behave as an associative memory, with no ability to make generalizations. Having too few reference patterns reduces the ANN’s ability to make decisions. DVQ’s ability to get the optimal number of reference patterns is very sensitive to $\sigma$. RCE suffers from the same problem but it is not as sensitive. LVQ PAK does not have this problem as it does not add reference patterns. The price to pay for CN’s performance is a slight increase in the number of iterations.

4.3. Speech recognition

Speech recognition requires specifying the feature of speech, e.g. Linear Predictive Coding (LPC), and some measurement criteria, e.g. distance. Based on these two specifications, classification can be made. For our case, we choose the sequence of speech frames for a word as input, and use a simplified form of Dynamic Time Warping (DTW) (Rabiner and Schafer, 1985) as the distance measure. The sampling rate used is 12 kHz and each frame is 256 samples of 8-bit each with 10 ms frame interval. Order-of-8 PARCORR is used and the CN is used as the classifier.

The application is a game which is to move squares on a 3 by 3 matrix using voice input as shown in Fig. 6. The voice commands are the 8 numbers and directions up, down, left and right. The game is run in real time with 93.6% success rate.

6. Conclusion

We have proposed a new ANN for classification and compared it with 3 other ANNs. We also applied the CN on speech. The proposed CN is insensitive to scale of feature space, tolerant to erroneous input and able to choose a near-optimal number of reference patterns. It has better classification results as compared with RCE, LVQ and DVQ.

References


