The goal of Inductive Learning is to produce general rules from a set of seen examples, which can then be applied to other unseen examples. ID3 is an inductive learning algorithm that can be used for the classification task. The input to the algorithm is a set of tuples of description and class. The ID3 algorithm learns a decision tree from these input examples, which can then be used for classifying unseen examples given their descriptions. ID3 faces a problem called the replication problem.

An algorithm called the Expert-Gate algorithm is presented in this thesis. The aim of the algorithm is to tackle the replication problem. We discuss the various issues involved with each step of the algorithm and present results corroborating our choices. The algorithm was tested on various artificially created problems as well as on a real life problem. The performance of the algorithm was compared with that of Fringe.

The algorithm was found to give excellent results on the artificially created problems. The Expert-Gate algorithm gave satisfactory results on the NETtalk problem. Overall, we believe the algorithm is a good candidate for testing on other real life domains.
The Expert-Gate Algorithm

by

Varad Vidyadhar Joshi

A Thesis submitted to
Oregon State University

in partial fulfillment of the
requirements for the degree of

Master of Science

Completed November 23, 1992
Commencement June 1993
APPROVED:

Redacted for privacy

Professor of Computer Science in charge of major

Redacted for privacy

Head of Department of Computer Science

Redacted for privacy

Dean of Graduate School

Date thesis presented November 23, 1992
Typed by Varad Vidyadhar Joshi for Varad Vidyadhar Joshi
ACKNOWLEDGEMENTS

Dedicated to My Parents.

I would like to take this opportunity to thank my advisor, Thomas Dietterich, who was a constant source of ideas and encouragement. I would also like to thank Thomas Dietterich and Hussein Almuallim and for their insightful comments on the rough draft.
# Table of Contents

1 Introduction

1.1 Inductive Learning ........................................... 1

1.2 Related Algorithms ........................................... 3
    1.2.1 ID3 ..................................................... 3
    1.2.2 Fringe ............................................... 6

1.3 Thesis Outline .............................................. 8

2 The Expert-Gate Algorithm ..................................... 11

2.1 Motivation .................................................. 11

2.2 The Algorithm ............................................... 12

2.3 Decisions Regarding the Algorithm ........................... 14
    2.3.1 Choice of Constants .................................. 14
    2.3.2 Choosing The Best Expert ............................. 15
    2.3.3 Number of Experts .................................... 17
    2.3.4 Gate/Expert Depth .................................... 18
    2.3.5 Gate/Expert Learning Algorithms ....................... 19

2.4 Summary .................................................. 21

3 Performance Evaluation ...................................... 24

3.1 Parameter Selection ......................................... 24
# List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Algorithm ID3</td>
<td>5</td>
</tr>
<tr>
<td>2. A decision tree for $x_1 \cdot x_2 + x_3 \cdot x_4$</td>
<td>6</td>
</tr>
<tr>
<td>3. Algorithm Fringe</td>
<td>7</td>
</tr>
<tr>
<td>4. Procedure Find-features</td>
<td>9</td>
</tr>
<tr>
<td>5. An example of new features created by procedure find-features</td>
<td>10</td>
</tr>
<tr>
<td>6. The Expert-Gate Algorithm</td>
<td>13</td>
</tr>
<tr>
<td>7. The Expert-Gate Algorithm (modified)</td>
<td>23</td>
</tr>
<tr>
<td>8. Comparison of run times on a DNF target concept</td>
<td>31</td>
</tr>
<tr>
<td>9. Comparison of run times on a CNF target concept</td>
<td>33</td>
</tr>
<tr>
<td>10. Results for a data set from the NETtalk domain problem</td>
<td>38</td>
</tr>
<tr>
<td>11. An example of Gate-Experts for a DNF target concept</td>
<td>44</td>
</tr>
</tbody>
</table>
# List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Iterations vs. Accuracy</td>
<td>25</td>
</tr>
<tr>
<td>2. Number of Experts vs. Accuracy</td>
<td>27</td>
</tr>
<tr>
<td>3. Gate depth vs. Accuracy</td>
<td>28</td>
</tr>
<tr>
<td>4. Accuracy for DNF target concepts</td>
<td>30</td>
</tr>
<tr>
<td>5. Accuracy for CNF target concepts</td>
<td>32</td>
</tr>
<tr>
<td>6. Accuracy for configurations of Experts-Gate as target concepts</td>
<td>34</td>
</tr>
<tr>
<td>7. An example of NETtalk data</td>
<td>34</td>
</tr>
<tr>
<td>8. Accuracy for phoneme bits</td>
<td>36</td>
</tr>
<tr>
<td>9. Accuracy for stress bits</td>
<td>36</td>
</tr>
<tr>
<td>10. Phoneme accuracy on NETtalk data</td>
<td>37</td>
</tr>
<tr>
<td>11. Stress accuracy on NETtalk data</td>
<td>37</td>
</tr>
<tr>
<td>12. Comparison of the EG algorithm with other algorithms</td>
<td>39</td>
</tr>
</tbody>
</table>
A new inductive learning algorithm called Expert-Gate algorithm is presented in
this thesis. This algorithm uses the principles of feature discrimination and com-
petitive learning. There have been a number of well-known inductive learning
algorithms like ID3, Fringe, DCfringe etc. Fringe has been found especially suc-
cessful. A comparative study with Fringe is done in this work along with discussion
of the Expert-Gate algorithm itself.

The next sections briefly review inductive learning and the algorithms ID3
and Fringe, as it is necessary to understand these algorithms to understand the
Expert-Gate algorithm. The last section gives an outline of this thesis.

1.1 Inductive Learning

Inductive learning or empirical learning is typically accomplished by reasoning
from externally supplied examples to produce general rules which can be applied
to other unseen examples. More formally, the learner learns a function describing
the given target concept from a finite number of examples of the target concept.
For example, given the description of a number of human beings in terms of their height, weight, hair color etc., with a classification of each of them as "Tall" or "not Tall", the learning system can learn the concept of "Tall." In this example, the target concept is "Tall", the instance space is all human beings and the preclassified examples are pairs of the form (description, tall/not tall).

The inductive leap in a learning algorithm from a finite number of values to a total function is not justifiable unless we make some assumptions about the nature of the function we are trying to learn. These assumptions or additional constraints are called the "Bias" of the algorithm. The concept of bias and the need for it is discussed in detail by Mitchell [Mit90]. At first glance, it may appear that it is desirable to remove all bias from the learning procedure, but as Mitchell has proved, removing all the bias makes the learning procedure completely useless. An unbiased learning system's ability to classify examples is equivalent to lookup from a table of examples previously seen, so it can not deal with previously unseen examples.

Biases are typically of two types:

1. restricted hypothesis space bias, and
2. preference bias.

A restricted hypothesis space bias assumes that the function $f$ to be learned belongs to a restricted class of hypotheses. The bias is usually described in terms of the representation of the hypotheses. For example, it might be assumed that the hypothesis can always be expressed as a logical conjunction. Hypothesis spaces such as logical conjunctions, linear threshold functions, k-DNF functions, and k-CNF functions have been extensively studied. An example of an algorithm that uses restricted hypothesis space bias is Mitchell's Version Space algorithm [Mit82].

A preference bias is incorporated in one of the most popular inductive algorithms called ID3 [Qui86]. This algorithm is discussed in the next section as it
is very relevant to this thesis. The preference bias, instead of restricting the hypothesis space, places a preference ordering over the hypothesis space. In short, if the learning algorithm comes up with two hypotheses which are equally good, then the preference bias tells which hypothesis to prefer. Most preference biases attempt to minimize the syntactic complexity of the hypothesis representation by, for example, preferring the shortest decision list, or the shortest logical conjunction etc. These are all variations of Occam's Razor, which states that the simpler of two competing hypotheses should always be preferred [BEHW90].

Attempts have been made to analyze various inductive learning algorithms. The Valiant framework [Val84, BEHW90, Hau88] provides theoretical bounds on the number of training examples required in order to have sufficient confidence in the correctness of the learning algorithm. This style of analysis is called PAC (probably approximately correct) learning. Blumer et al. [BEHW90] have shown how to extend these results for preference biases. Haussler [Hau88] used the idea of the Vapnik-Chervonenkis dimension (VC-dimension) for handling infinite hypothesis spaces.

1.2 Related Algorithms

1.2.1 ID3

Quinlan [Qui86] described an inductive learning algorithm called ID3, which is still considered one of the best algorithms in inductive learning. ID3 makes a number of assumptions about its universe. ID3 can be used only for classification tasks, although, as Quinlan points out, many problems are classification problems or can be recast as classification problems. For ID3, the examples of the instance space are described in terms of a collection of attributes. The attributes should provide adequate information for classification of the examples in that no two examples with same values for each attribute should belong to different classes. It is further assumed that all the relevant attributes are provided to the algorithm.
Each object of the instance space belongs to one of a set of mutually exclusive classes. In general, it is assumed that there are only two classes, although the algorithm can be extended to handle more than two classes. The two classes are denoted P and N, (also sometimes denoted as T and F) and the objects belonging to these classes are termed positive and negative instances of the target concept. The class P can be seen as standing for the target concept.

The induction task in ID3 is to develop a decision tree which acts as a classification rule. Each non-leaf node in the decision tree is labeled with an attribute. Each such node represents a test on that attribute with a branch for each outcome. (In general, the test is a simple branching based on the value of the attribute.) The leaves of the decision tree are the class names (in this case, P or N). In order to classify an example, we start from the root, and depending on the result of the test at each node, we choose a branch and follow it until we reach a leaf node. The label of the leaf node is the predicted class of the example.

ID3 implements a preference bias. Simpler trees are preferred over more complex ones while learning. Although the simplest or the shortest tree which correctly classifies all the training examples would be most favorable, there is no polynomial time algorithm for finding such a tree. ID3 builds a reasonably simple tree, though the algorithm can not guarantee finding the simplest tree.

ID3 works as follows. The input to the algorithm is a set of examples with their classification. The algorithm is also given a set of possible tests that can be performed on the feature values of the examples. For example, if the features are all boolean, each test simply determines the truth value of a feature. The algorithm then chooses the test that best separates the examples. The test is run on all examples, and the examples are divided into subsets depending on the result of the test. For each subset, a check is made to see whether all the examples in the subset belong to the same class. If that is the case, the node is labeled as a terminal node with the class name as the label; otherwise ID3 is called on the subset. The full algorithm is given in Figure 1.
Algorithm ID3
1. Choose the best test, that is the test that best separates the examples.
2. Let $S_1, S_2, \ldots, S_v$ be the resulting subsets such that all the examples in $S_i$
have outcome $O_i$ on the test.
3. For $i = 1$ to $v$
   
   If all examples belong to one class (either P or N) then
   classify $S_i$ as that class
   else
   call ID3 with $S_i$ as the input examples

Figure 1. Algorithm ID3

Assuming that all the features have boolean values, each test is a check of the
truth value of a feature. The best test is found using the information theoretic
technique of finding the information gain. The test with the maximum information
gain is chosen as the best test. Consider that $S$ is the given set of examples. Let
$p$ be the number of positive examples and $n$ be the number of negative examples
in $S$.

The information gain of a feature $A$ is defined as,

$$gain(A) = I(p,n) - E(A).$$

$I(p,n)$ is the expected information for the correct classification of the examples.

$$I(p,n) = - \frac{p}{p+n} \log_2 \frac{p}{p+n} - \frac{n}{p+n} \log_2 \frac{n}{p+n}.$$

$E(A)$ is the expected information required for the tree with $A$ as a root. Let
$S_1, S_2, \ldots, S_v$ be the subsets of $S$ where $S_i$ has outcome $O_i$ on feature $A$. (For
boolean features there are only two outcomes.) Let $p_i$ and $n_i$ be the number of
positive and negative examples in \( S_i \). Then \( E(A) \) is given by,

\[
E(A) = \sum_{i=1}^{v} \frac{p_i + n_i}{p + n} I(p_i, n_i).
\]

1.2.2 Fringe

ID3 faces a problem, called the replication problem, when learning boolean functions with small DNF descriptions. Consider the boolean function \( x_1 \cdot x_2 + x_3 \cdot x_4 \). A decision tree for this boolean function is given in Figure 2. The subtree for the term \( x_3 \cdot x_4 \) is replicated in the decision tree. While learning this decision tree, the examples belonging to the term \( x_3 \cdot x_4 \) are fragmented into two subsets belonging to the two leaves denoted A and B in Figure 2. This causes the algorithm to require a large number of examples in order to ensure that the information gain computations are accurate. This problem is called the replication problem. Fringe [PH90] tries to solve the replication problem by building conjunctive features from the primitive features.

\[\text{Leaf A}\]
\[\text{Leaf B}\]

\textbf{Figure 2.} A decision tree for \( x_1 \cdot x_2 + x_3 \cdot x_4 \)

In each iteration of the Fringe algorithm, ID3 finds the tree which correctly classifies the examples in the set \( S \), by building from the attributes in the set \( V_k \). The find-features procedure then constructs new features from the tree. If no new
Algorithm Fringe

Fringe($V : \text{set of attributes}, S : \text{examples}, M : \text{positive integer}$)
1. set $k = 0, V_1 = V$
2. repeat
   $k = k + 1$
   $T_k = \text{ID3}(S, V_k)$
   $F = \text{find-features}(T_k)$
   $V_{k+1} = V_k \cup F$
3. until ( $V_{k+1} = V_k$ or $|V_{k+1}| \geq M$)
4. return $T_k, V_k$

<table>
<thead>
<tr>
<th>Figure 3. Algorithm Fringe</th>
</tr>
</thead>
</table>

features are found or if the number of features exceeds some predetermined integer $M$, then the algorithm stops and outputs the tree $T_k$ and the features $V_k$.

The find-features procedure forms simple new features by using the conjunction and negation operators. The procedure chooses the features at the fringe of the ID3 decision tree and constructs new features from them. As the negation and conjunction operators are complete, this algorithm has the capability to learn arbitrary boolean functions. Figure 5 shows examples of the new features created by procedure find-features.

Fringe was found to work very well for learning various boolean functions like DNF functions, Multiplexor functions, etc. The expert-gate algorithm is also intended to overcome the replication problem by the very nature of the algorithm. A comparative study of Fringe and the Expert-Gate algorithm is done in Chapter 3.
1.3 Thesis Outline

Chapter 2 describes the Expert-Gate algorithm along with some of the theoretical background for it. Various modifications were done to the algorithm as the working of the algorithm was tested. Some of these modifications were necessary for execution of the algorithm while some were done to see the overall effect on the execution. All these different versions of the algorithm are discussed in this chapter.

The working version of the algorithm was tested on various artificially generated problems. As Fringe is one of the most successful of the inductive learning algorithms found so far, the Expert-Gate algorithm was compared with Fringe to really give an idea as to the quality of its performance. The testing was done on sets of DNF and CNF functions. Also, examples were generated from randomly generated combinations of experts and gate and the algorithms were tested. All these results are discussed in Chapter 3.

Chapter 3 also describes testing the Expert-Gate algorithm in a real life domain. The NETtalk problem and the results of the algorithm for this problem are discussed.

The topic of Chapter 4 is how the algorithm can be further modified to improve its performance and how it can be further tested. Some directions for future work are pointed out. The viability of the Expert-Gate algorithm for real life applications is discussed.
Procedure Find-features

find-features (T)
1. set $F = \text{empty set}$
2. for each positive leaf $n$ at depth $\geq 2$ in $T$
   let $p$ and $g$ be the parent and grandparent nodes of $n$
   if ($n$ is on the right subtree of $p$) then
     if ($p$ is on the right subtree of $g$) then
       feature = $p \cdot g$
     else
       feature = $p \cdot \bar{g}$
     else
       if ($p$ is on the right subtree of $g$) then
         feature = $\bar{p} \cdot g$
       else
         feature = $\bar{p} \cdot \bar{g}$
       add feature to $F$
3. end
4. return $F$

Figure 4. Procedure Find-features
Figure 5. An example of new features created by procedure find-features

New features defined:

\[ f_1 = x_7 \cdot x_4 \]
\[ f_2 = \overline{x_4} \cdot \overline{x_6} \]
\[ f_3 = \overline{x_4} \cdot x_3 \]
\[ f_4 = x_2 \cdot \overline{x_1} \]
Chapter 2

The Expert-Gate Algorithm

2.1 Motivation

Popular learning algorithms like ID3, Fringe etc. build decision trees for classification of given examples. Each path in the decision tree ends in a leaf which has a class as the label. The algorithms build the trees without trying to use the structure inherent in the problem. In many cases, it may happen that, the examples can be divided into subsets where all examples from a subset have either a part of the path in the decision tree in common or belong to similar subtrees on different paths. In plain terms, the domain may be such that there are a number of experts with each expert good at recognizing a subset of the examples. This idea was put forward by Jacobs and Hinton at the Connectionist Summer School in Pittsburgh in 1988. It was used in a learning procedure in the paper by Jacobs et al. [JJNH91]. They proposed a supervised learning procedure for systems composed of many separate networks, each of which learned to recognize a subset of examples. They presented it as an associative version of competitive learning.

If we know in advance that the domain can be divided into a number of subsets that correspond to distinct subtasks, then we can try to train a number of expert networks and a gating network that decides which of the expert networks should be used for each training case. Jacobs et al. used feedforward multilayer networks...
as the experts and a feedforward multilayer network with normalized output as a gating network. Each of the examples is fed to all the expert networks and they produce output. The gating expert is also fed the example and its output is the expert which is chosen. In a sense the gate acts like a switch directing which of the expert’s output should appear as the final output. Jacobs et al. used an error function which encouraged localization to each expert and also made the experts become competitive rather than cooperative.

They applied the above described learning procedure to the task of multispeaker vowel recognition. They found that the learning procedure attains the same accuracy faster than the standard backpropagation networks. However, they don’t discuss whether the accuracy increases if the procedure is run for more time. In a sense it is not clear whether this is a good algorithm if we want a highly accurate prediction, although it may give a reasonably good estimate within shorter time.

The idea presented in the above paper was modified and used in the algorithm presented in the next section.

2.2 The Algorithm

In simple language, the algorithm can be stated as follows;

1. Assign examples randomly to experts. (Initialization)
2. Learn the gate.
3. repeat
   Assign examples to experts according to the gate.
   Learn the experts.
   Assign each example to the expert that best classifies it.
   Learn the gate.
4. until (sufficient accuracy is attained or the maximum number of iterations is exceeded)
5. Output the gate and the experts.
The input to the algorithm is $S$, a set of preclassified examples. There are essentially two modes of learning within the algorithm. The gate is learned from a set of feature-vector/expert pairs whereas the experts are learned from set of feature-vector/class pairs. This means that the terminal nodes in the gate decisions tree are numbers denoting the experts. The classification of any test example can be then done as follows. Go down the gate tree and find the expert corresponding to the terminal node. Go down that expert tree and find the class corresponding to the terminal node. This class is the classification of the test example.

**Expert-Gate Algorithm**

Expert-Gate ($S$ : set of training examples, $K$ : the number of experts)

1. Let $A$ = Minimum accuracy desired.
2. Let $M$ = Maximum number of iterations allowed.
3. Assign the examples randomly to the $K$ experts.
   
   Let $G^{(0)}(S_i)$ be the expert assigned to example $S_i$.
4. Train gate on examples $G^{(0)}(S)$.
   
   The gate learns the approximation $\hat{G}^{(0)}(S)$.
5. Set $m = 0$.
6. repeat
   
   (a) For $j = 1, \ldots, K$, train expert $j$ on $\{S_i | \hat{G}^{(m)}(S_i) = j\}$
   
   (b) $\forall i \in S$ assign $G^{(m+1)}(S_i) = j$ if expert $j$ best classifies example $S_i$.
   
   (c) Train gate on examples $G^{(m+1)}(S)$.
   
   The gate learns the approximation $\hat{G}^{(m+1)}(S)$.
   
   (d) Set $m = m + 1$
7. while (accuracy $< A$ and $m < M$)
8. Output Gate and the Experts.

**Figure 6. The Expert-Gate Algorithm**
Learning the gate and the experts is an iterative process. Initially, the examples are randomly assigned to experts. The gate is learned from these example-expert pairs. Then repeatedly the following is done until sufficient accuracy is attained. The examples are assigned to the experts according to the label on the terminal node in the gate to which they go. Each expert is now learned with these example-class pairs assigned to it. The examples are assigned to the newly learned expert that best classifies it. The gate is now learned with these example-expert pairs. This process is repeated. The final output of the algorithm is the gate and the experts.

$A$ is the minimum training sample accuracy desired from the algorithm and $M$ is the maximum number of iterations desired. Naturally, one of the issues related to the execution of the algorithm is how to choose these numbers. These issues and other issues encountered while coding and testing the algorithm are discussed in detail in the next section.

2.3 Decisions Regarding the Algorithm

2.3.1 Choice of Constants

1. Minimum training sample accuracy desired from the algorithm: In the initial versions of the algorithm this criterion was used as the training sample accuracy did not always become 1.0. After some modifications to the algorithm, discussed in the following sections, were done, this criterion was no longer necessary as the training sample accuracy was always 1.0.

2. Maximum number of iterations: This is another criterion used for stopping the algorithm if it is found that no further improvement is possible. This ensures that the algorithm produces a result within a fixed time. Fixing the value of the maximum number of iterations is an implementation decision and was left for that stage.
2.3.2 Choosing The Best Expert

In the algorithm, the second step inside the loop assigns each example to the expert that best classifies it. A decision has to be made as to which expert best classifies a given example. Two different issues arise in this respect.

1. When more than one expert classifies the example correctly, the example can be assigned to any of the experts. Assigning the example to all the correct experts is also a possible choice. When this algorithm was coded and tested on a number of small problems, three choices were tried. A possible choice that was not tried would be to use a probability estimate of confidence in the classification for choosing the best expert. This is discussed in further detail in the last chapter.

(a) Assigning the example to all correct experts: When this was tested it was found that the algorithm degenerated to show ID3-like results. Before the start of the loop, the examples are assigned randomly to the experts. Naturally, the experts can correctly classify randomly distributed examples irrespective of the distribution in the domain. Hence, in the first iteration, for most of the examples more than one expert is correct. This means that the number of examples for each expert increases. In the following iterations, this trend continues. After only a few iterations, the gate degenerates to a single node with a particular expert as the label, and that expert doing all the classification. As can be seen from the above, assigning the example to all the correct experts is not a good choice.

(b) Randomly choosing a correct expert: Out of all the correct experts one is chosen randomly and the example is awarded to it. It was found that in almost all the problems for which this was tried, the algorithm failed to stop. In each iteration, each example was assigned
to a randomly-chosen correct expert. This meant that even though some examples really belonged to a certain subset which the algorithm was trying to capture (to be classified by a single expert), the examples were almost never assigned to the same expert in a single iteration. Even when the algorithm converged to a stable expert-gate configuration, the accuracy of this configuration was very low.

(c) Assigning the example to the numerically lowest expert: The experts were labeled with integers from 1 to $K$. If experts 2, 5 and 7 correctly classified an example, then the example was assigned to expert 2. This had the advantage of consistency. If one expert classified an example correctly and was assigned that example, then in the next iteration if the same expert classified that example correctly, it would be assigned that example again. This meant that the algorithm would converge very quickly. Actual runs on problem data also showed good results for this choice.

2. When no expert classifies a given example correctly, we must select an expert the example should be assigned to. Again, two choices were tried.

(a) Assign the example to a randomly-chosen expert: As with the previous argument against choosing an expert randomly when there is more than one correct expert, reasons can be given against this choice. The examples are almost never assigned to the right expert. The algorithm fails to stop because it never reaches a stable expert-gate configuration. The algorithm gets into a cycle with the same configuration repeated after every few iterations.

(b) Assign the example to the numerically lowest expert: Again, this has the advantage of consistency. Any example which is not classified correctly by any expert will be always assigned to the lowest expert. This can also be seen as follows. Each expert can be seen as the search
by the algorithm for a rule for a subset of the examples. The examples which are not classified by any expert are the exceptions to these rules. These exceptions are always assigned to the numerically lowest expert so that expert learns to classify all these special cases.

2.3.3 Number of Experts

1. The number of experts has to be chosen as an integer greater than the number of classes. (If the number of classes is two then choosing two experts might mean that each expert stands for a class, and the algorithm may degenerate to the usual inductive learning algorithm (ID3), with each expert being just a node and the gate being the full decision tree.)

2. The number of experts should not be too large. It was observed during the runs that in general only a few experts were really left in the final configuration. As mentioned earlier, if the example was classified correctly by more than one expert then it was assigned to the numerically first expert. This tended to make the algorithm converge to a very few experts very quickly.

3. Another change in the code for the algorithm had to be made because of this characteristic. At some point in execution, it may happen that an expert is assigned only a few examples. This may lead to a trivial decision tree for that expert. (For example, given a task of classifying human beings, the expert would always say "Tall", no matter what it saw). It was observed that when such an expert was created, the algorithm would degenerate, with the gate doing all the classification (full decision tree) and the experts being just class labels. A choice was made to discard these trivial experts when they were created. (Also, when two experts became identical, one of them was discarded, as it would otherwise become a trivial expert in the next iteration.)
4. This means that, the number of experts should be an integer between 3 and 10.

2.3.4 Gate/Expert Depth

1. Gate Depth: The idea behind the algorithm is that, the gate should decide which expert should classify the given example. The algorithm is supposed to build the gate so that it picks out the subset to which the example belongs. This means, the depth of the gate decision tree should be small compared to the number of relevant attributes. Otherwise, we may get a very complex gate and trivial experts. In the extreme case, the algorithm may degenerate to simple ID3, with experts being just class labels. To prevent this from happening, the gate should be restricted to a certain maximum depth. In practice runs, it was found that a gate depth of approximately half the number of relevant attributes worked the best.

2. Expert Depth: After fixing the gate depth, we have three choices for learning the experts.

(a) The experts can be learned with no restriction on the expert depth or on the attributes the learning algorithm can use. When this was tried, it was found out the Expert-Gate algorithm rarely degenerated to ID3.

(b) The expert depth can be fixed as the number of attributes minus the gate depth. No restriction is placed on the attributes, the learning algorithm (ID3) can use.

(c) The attributes, the expert learning procedure can use can be restricted to some fixed set. Consider that we have learned the gate and we want to find which attributes we can use for learning an expert 'e'. Let A be the set of attributes on all the paths in the gate, from the root to those leaves with e as the label. The attributes allowed for the learning of
the expert are then \((V - A)\), where \(V\) is the set of all the attributes. In this case, we place no restriction on the depth to which the expert can grow.

The reasoning behind this choice was as follows. By restricting the set of attributes, we try to isolate the attributes into two classes. Those attributes used in the gate for finding the correct expert and those used for classification by the expert.

It was found through experimental tries that all the three approaches were equally good. It was observed that placing no restriction on the experts worked well for all the problems, hence this choice was used. Further testing on some real life problems is needed to ascertain the relative merits of these three choices.

### 2.3.5 Gate/Expert Learning Algorithms

The ID3 algorithm uses all the attributes for building the decision tree. In the Expert-Gate algorithm we need a slightly different algorithm for learning the gate and the experts. There are two minor changes which have to be made.

1. The ID3 algorithm presented in Figure 1 handles input examples with only two classes. The Gate learning algorithm needs to be able to handle more than two classes. The labels of leaves in a gate are the experts and as there are in general more than two experts, it is necessary to have a multi-class learning algorithm. ID3 can be easily extended for more than two classes as given in Quinlan's paper [Qui86].

Only the calculation of the information gain needs to be changed to make ID3 work on a multiple class problem. The rest of the ID3 algorithm remains the same.

Let us assume that the number of classes is \(m\). The number of examples of class \(i\) is \(n_i\). The information gain for attribute \(A\) is then,
\[ \text{gain}(A) = I(n_1, n_2, \ldots, n_m) - E(A). \]

\[ I(n_1, n_2, \ldots, n_m) \] is the information gain for the correct classification of the examples and is given by the formula,

\[ I(n_1, n_2, \ldots, n_m) = - \sum_{i=1}^{m} \frac{n_i}{\sum_{i=1}^{m} n_i} \log_2 \frac{n_i}{\sum_{i=1}^{m} n_i}. \]

Let us assume that there are \( v \) different possible outcomes of the test on the attribute \( A \) (the attribute \( A \) takes \( v \) different values.) The number of examples of class \( i \) for outcome \( j \) be \( n_{ij} \).

\[ n_i = \sum_{j=1}^{v} n_{ij} \]

\( E(A) \), the expected information required for the tree with \( A \) as the root is given by the formula,

\[ E(A) = \sum_{j=1}^{v} \frac{\sum_{i=1}^{m} n_{ij}}{\sum_{i=1}^{m} n_i} I(n_1, n_2, \ldots, n_m) \]

The test with the maximum information gain is chosen as the best test.

2. The ID3 algorithm calls itself recursively on subsets of decreasing size until all the examples belong to the same class or the information gain is not significant. This means, all the attributes may be used as nodes before the algorithm reaches a leaf of the decision tree. In learning either expert or gate, we need to either restrict the attributes that can be used for building the tree or restrict the depth of the tree. This means two different versions of ID3 have to be used.

(a) ID3 with a restriction on the attributes that can be used: This is an easy task as the ID3 algorithm lends itself easily to this modification. In ID3, as an attribute is chosen and next recursive call is made with
a subset of the original examples, a list of attributes is passed. The attributes, that have been already used on the path from the root to the current node need not be considered for choosing the best attribute in the subsequent calls. When the attributes are passed to the next call, the attribute chosen currently is marked as used.

To place a restriction on the attributes that can be used for learning an expert/gate, all we need to do is as follows. Make a list of attributes in which the disallowed attributes are marked used. Call ID3 with this new list instead of the complete list of attributes.

(b) ID3 with restriction on the depth of the learned tree: The task in this case is to learn the best tree which has a depth no more than the limit. ID3 was modified as follows. When the current path reaches a depth equal to the limit, if all the examples do not belong to the same class, then the majority class is chosen as the label for the leaf. (Another approach was also tried: if the depth was within one of the limit and there was a majority class among the examples then that class was chosen as label. No significant difference was noticed between these two approaches.)

2.4 Summary

This chapter presented the Expert-Gate algorithm and discussed the various issues involved. ID3 with a few modifications is used as the inductive learning algorithm for learning the gate and the experts. These changes are discussed in this chapter. Also, the Expert-Gate algorithm as presented in section 2.2 is just the simple theoretical idea behind the actual program. Some changes had to be made for the actual working of the algorithm. Although, the bounds on the values of the variables and the constants have been discussed in this chapter, the actual optimal values were only discovered when the algorithm was tested on various problems.
In the next chapter the results of the test runs on different problems are presented.
**Expert-Gate** ($S$ : set of training examples, $K$ : Number of experts)

1. Initialize examples to set $S$.
2. Initialize expexs[i] to nil, for i = 1 to $K$.
3. Initialize all-features to set of all attributes.
4. Assign examples randomly to experts. (Initialization)
   
   for example-number = 1 to $|S|$ 
   
   make the class of examples[example-number] = random(1,$K$)
5. Learn the gate.
   
   gate = ID3-ltd(all-features, examples)
6. repeat
   
   (a) Assign examples to experts according to the gate.
      
      for example-number = 1 to $|S|$ 
      
      expert-number = find-class(gate, example-number)
      
      add example to expexs[expert-number]
   
   (b) Learn the experts.
      
      for expert-number = 1 to $K$
      
      features = get-features(gate, expert-number)
      
      expert = ID3(features, expexs[expert-number])
   
   (c) Assign each example to the expert that best classifies it.
      
      for example-number = 1 to $|S|$ 
      
      expert-number = best-expert(example-number)
      
      make the class of examples[example-number] = expert-number
   
   (d) Learn the gate.
      
      gate = ID3-ltd(all-features, examples)
7. until (sufficient accuracy is attained or
      
      the maximum number of iterations is exceeded)
8. Output the gate and the experts.

**Figure 7. The Expert-Gate Algorithm (modified)**
Chapter 3

Performance Evaluation

This chapter evaluates the expert-gate algorithm by testing the algorithm on various problems and by comparing the accuracy of the algorithm to that of Fringe. Section 3.1 presents results on test cases showing how variation of the algorithm parameters, such as, the number of experts, gate depth etc. affects the accuracy and what choices are the best. In Section 3.2, we give the absolute results for different sets of problems and compare them with Fringe. In Section 3.3, we present the results for the NETtalk domain.

3.1 Parameter Selection

As described in the previous chapter, the performance of the algorithm depends greatly on the selection of various parameters, such as, expert/gate depth, expert features, method for choosing the best expert etc. In this section, we justify our final choices by presenting results corroborating them.

1. Criterion for Terminating The Algorithm.
   In the detailed algorithm, presented in Chapter 2, we said that the outer loop in the algorithm is exited when sufficient accuracy is attained, or some maximum number of iterations is exceeded. There are two issues that are related here.
Table 1. Iterations vs. Accuracy

(a) **Maximum number of iterations:** The algorithm was tested on many problems. Even in the initial stages, it was found that if at all the algorithm converged to a stable experts-gate configuration, it did so within a few (not more than 20) iterations. As can be seen from the results in Table 1, the training sample accuracy becomes 1.0 within two iterations. The results in the next sections also show that only a few iterations are required before the accuracy becomes 100 percent. This would mean that a limit of 40 would suffice to ensure that the algorithm can be safely terminated if it was found not to converge earlier.

(b) **Attaining sufficient accuracy:** The algorithm usually attained accuracy of 100 percent on the training sample. The question that arises is whether further iterations would improve the test sample accuracy. If the accuracy becomes 100 per cent and the gate-expert configuration
stabilizes, (that is there is no further change in either the gate or the experts decision trees) then there is no point in continuing further. It was observed that this criterion of exiting the loop when the training sample accuracy became 1.0 and there was no further change in either the gate or the experts worked the best.

Table 1 presents results for two test problems. The target concept for the problem 1 was an expert-gate configuration and the target concept for problem 2 was a DNF function. As can be seen in both the cases, the training sample accuracy became 1.0 in the second iteration. The gate-expert configuration attained a steady state on the 7th and 9th iteration respectively for the two test problems. The test sample accuracy, in fact, increased from the second iteration. For problem 1, the best test sample accuracy occurred after the fourth iteration, but the difference between that and the accuracy after the last iteration is almost negligible. For problem 2, the best test sample accuracy occurred after the last iteration. In fact, it was observed that, in most cases the best test sample accuracy occurred when the algorithm stopped, as was the case for problem 2.

2. Number of Experts: Table 2 presents results for two test problems. The columns in the table are for the number of experts and the test sample accuracy. The target concept in problem 1 was a DNF expression and the target concept in problem 2 was an Expert-Gate configuration. The number of features was 24 in both the cases. For each problem, the training examples and the test examples were held fixed and the number of experts was varied from 1 to 18. To minimize the effects of the random initialization, the same experiment was repeated 20 times and the average was taken. The results in Table 2 show that the accuracy does not vary much as the number of experts is changed. The test sample accuracy is not affected by the number
of experts as long as there are more than a minimum number of experts. The reasons for this are explained in the previous chapter.

3. **Gate Depth:** Table 3 shows the gate-depth and the test sample accuracy for two test problems. In both the cases, the training sample and the test sample were created by using a DNF function as a target concept. The number of features was 24 in both the cases. The training sample and the test sample was held fixed and the gate-depth was varied from 1 to 24. To minimize the effects of random initialization, the same experiment was repeated 20 times and the average was taken. As can be seen from the table, the best test sample accuracy occurs for gate depths of 12 and 4 respectively.

The results on test problems showed that it is not possible to guess the gate depth that would give the best test sample accuracy. The gate depth that gave best results even with the same number of features varied widely from problem to problem, although, it was found that a gate depth equal to about
<table>
<thead>
<tr>
<th>Depth</th>
<th>Acc</th>
<th>Depth</th>
<th>Acc</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.800</td>
<td>13</td>
<td>0.817</td>
</tr>
<tr>
<td>2</td>
<td>0.800</td>
<td>14</td>
<td>0.835</td>
</tr>
<tr>
<td>3</td>
<td>0.800</td>
<td>15</td>
<td>0.836</td>
</tr>
<tr>
<td>4</td>
<td>0.800</td>
<td>16</td>
<td>0.805</td>
</tr>
<tr>
<td>5</td>
<td>0.800</td>
<td>17</td>
<td>0.787</td>
</tr>
<tr>
<td>6</td>
<td>0.790</td>
<td>18</td>
<td>0.799</td>
</tr>
<tr>
<td>7</td>
<td>0.800</td>
<td>19</td>
<td>0.798</td>
</tr>
<tr>
<td>8</td>
<td>0.800</td>
<td>20</td>
<td>0.826</td>
</tr>
<tr>
<td>9</td>
<td>0.805</td>
<td>21</td>
<td>0.821</td>
</tr>
<tr>
<td>10</td>
<td>0.813</td>
<td>22</td>
<td>0.803</td>
</tr>
<tr>
<td>11</td>
<td>0.822</td>
<td>23</td>
<td>0.839</td>
</tr>
<tr>
<td>12</td>
<td>0.840</td>
<td>24</td>
<td>0.806</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Depth</th>
<th>Acc</th>
<th>Depth</th>
<th>Acc</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.897</td>
<td>13</td>
<td>0.899</td>
</tr>
<tr>
<td>2</td>
<td>0.897</td>
<td>14</td>
<td>0.901</td>
</tr>
<tr>
<td>3</td>
<td>0.897</td>
<td>15</td>
<td>0.901</td>
</tr>
<tr>
<td>4</td>
<td>0.911</td>
<td>16</td>
<td>0.901</td>
</tr>
<tr>
<td>5</td>
<td>0.906</td>
<td>17</td>
<td>0.899</td>
</tr>
<tr>
<td>6</td>
<td>0.909</td>
<td>18</td>
<td>0.899</td>
</tr>
<tr>
<td>7</td>
<td>0.908</td>
<td>19</td>
<td>0.901</td>
</tr>
<tr>
<td>8</td>
<td>0.906</td>
<td>20</td>
<td>0.901</td>
</tr>
<tr>
<td>9</td>
<td>0.900</td>
<td>21</td>
<td>0.902</td>
</tr>
<tr>
<td>10</td>
<td>0.899</td>
<td>22</td>
<td>0.901</td>
</tr>
<tr>
<td>11</td>
<td>0.898</td>
<td>23</td>
<td>0.900</td>
</tr>
<tr>
<td>12</td>
<td>0.897</td>
<td>24</td>
<td>0.900</td>
</tr>
</tbody>
</table>

Table 3. Gate depth vs. Accuracy

half of the number of features worked well for almost all the problems. Actually, no significant difference in the accuracies was noticed for gate depths within the range of 0.4 to 0.6 of the number of features when the number of features was very large.

4. Expert Depth/ Expert Features: From the results on the test problems, it was not obvious which of the three choices had some relative merit. The three choices were,

(a) No restriction on the expert learning algorithm.

(b) Expert tree-depth restricted to number of features minus gate depth.

(c) Expert learning restricted to a set of features (this was explained in the previous chapter.)
The final gate-expert configuration was usually not affected by whichever of the three choices was selected. The random initialization procedure affected the final result more than this choice. In the actual program used for problems presented in the next section as well as in the next chapter, no restriction was placed on the expert learning algorithm.

3.2 Artificially Generated Target Concepts

In this section, we discuss the results for several artificially generated test problems. Section 3.2.1 describes how the test problems were created.

We present the results for the DNF target concepts in Section 3.2.2 and the results for the CNF target concepts in Section 3.2.3. Also, some selected configurations of experts and gate were used as target concepts and these results are presented in Section 3.2.4. We compare these results with the results for Fringe.

3.2.1 Creating Test Problems

The test problems were created as follows. The target concept (DNF/ CNF function) was created using a random number generator. The upper and the lower bounds on the number of terms in the function were specified. The actual number of terms, within these bounds, was then created using a random number generator. For each term, a feature appeared in that term with a probability (1 / Number of terms). Each feature in a term was negated with a probability of 0.5.

The training sample and the test sample were also selected using the same random number generator. An upper bound on the sample size was specified. These many examples were randomly selected from all the examples, assuming an uniform distribution. Two thirds of the sample were then randomly picked and made the training sample. The rest one third made the test sample. Each selected example was then classified according to the generated function. These samples with their classifications, were used for training and testing the algorithm.
3.2.2 DNF Target Concepts

As stated in Section 3.2.1, the DNF target concepts were generated. For each target concept, the same experiment was run 10 times with the same data set to minimize the effects of random initialization. Average test sample accuracy over these 10 runs was taken. The results in Table 4 show the target concept (a description of these DNF functions can be found in Appendix A), the number of features, and the test sample accuracy of the Expert-Gate algorithm and Fringe.

<table>
<thead>
<tr>
<th>Target Concept</th>
<th>Features</th>
<th>EG acc</th>
<th>Fringe acc</th>
</tr>
</thead>
<tbody>
<tr>
<td>dnf1</td>
<td>14</td>
<td>0.996</td>
<td>0.999</td>
</tr>
<tr>
<td>dnf2</td>
<td>16</td>
<td>0.999</td>
<td>1.000</td>
</tr>
<tr>
<td>dnf3</td>
<td>18</td>
<td>1.000</td>
<td>1.000</td>
</tr>
<tr>
<td>dnf4</td>
<td>20</td>
<td>1.000</td>
<td>1.000</td>
</tr>
<tr>
<td>dnf5</td>
<td>22</td>
<td>0.999</td>
<td>0.997</td>
</tr>
<tr>
<td>dnf6</td>
<td>24</td>
<td>0.999</td>
<td>1.000</td>
</tr>
<tr>
<td>dnf7</td>
<td>26</td>
<td>0.949</td>
<td>1.000</td>
</tr>
<tr>
<td>dnf8</td>
<td>28</td>
<td>0.987</td>
<td>0.998</td>
</tr>
</tbody>
</table>

Table 4. Accuracy for DNF target concepts

As can be seen from results in Table 4, the Expert-Gate algorithm performs almost as well as Fringe for the DNF target concepts. This is expected, because, the replication problem is very well handled by the Expert-Gate algorithm. The expert-gate algorithm attains almost 100 percent accuracy in almost all the cases.

We could also compare the execution times of both algorithms. Both the algorithms use ID3 to build the classifying tree, hence we could compare the number of calls made to ID3. The expert-gate algorithm usually terminates within a few (3 to 10) iterations of the outer loop. In each iteration, first the gate is learned and then the experts are learned. On average, 4 experts are learned in each iteration.
Target Concept (DNF) : \( x_1 \overline{x}_{12} x_4 \overline{x}_5 + x_3 x_2 x_6 + x_8 \overline{x}_9 \overline{x}_{10} + x_7 x_11 x_9 \)

Number of features : 12
Number of training examples : 552
Number of test examples : 270

<table>
<thead>
<tr>
<th>Expert Gate Algorithm</th>
<th>Fringe</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of calls to ID3 : 26</td>
<td>Number of calls to ID3 : 4</td>
</tr>
<tr>
<td>Number of iterations : 6</td>
<td>Number of new features defined : 48</td>
</tr>
<tr>
<td>Run time : 11.72 seconds</td>
<td>Run time : 2.96 seconds</td>
</tr>
<tr>
<td>Accuracy : 0.951</td>
<td>Accuracy : 1.0</td>
</tr>
</tbody>
</table>

**Figure 8.** Comparison of run times on a DNF target concept

That means about 5 calls to ID3 per iteration or 15 to 50 calls in all. Fringe on the other hand makes only 4 or 5 calls to ID3 (for 10 features to start with) but the number of features keeps on increasing in every iteration. This means that the later calls to ID3 take much longer than the earlier ones. The comparison in Figure 8 shows that, the EG algorithm made 26 calls to ID3 and Fringe made only 4 calls.

### 3.2.3 CNF Target Concepts

We present the results for the CNF target concepts in Table 5. (A description of these CNF target concepts can be found in Appendix B.) For each target concept, the experiment was repeated 10 times with the same data set to minimize the effects of random initialization. It can be easily seen that Expert-Gate algorithm works well for CNF concepts too. For the target concept cnf7, Fringe ran out of memory. For this run, the dynamic memory limit was changed to 4000 segments or 40 MB. Even with such large dynamic memory Fringe was unable to terminate. A
<table>
<thead>
<tr>
<th>Target Concept</th>
<th>Features</th>
<th>EG acc</th>
<th>Fringe acc</th>
</tr>
</thead>
<tbody>
<tr>
<td>cnf1</td>
<td>14</td>
<td>0.997</td>
<td>0.995</td>
</tr>
<tr>
<td>cnf2</td>
<td>16</td>
<td>1.000</td>
<td>0.997</td>
</tr>
<tr>
<td>cnf3</td>
<td>18</td>
<td>0.995</td>
<td>0.995</td>
</tr>
<tr>
<td>cnf4</td>
<td>20</td>
<td>1.000</td>
<td>1.000</td>
</tr>
<tr>
<td>cnf5</td>
<td>22</td>
<td>0.999</td>
<td>1.000</td>
</tr>
<tr>
<td>cnf6</td>
<td>24</td>
<td>0.997</td>
<td>0.999</td>
</tr>
<tr>
<td>cnf7</td>
<td>26</td>
<td>0.967</td>
<td>***</td>
</tr>
<tr>
<td>cnf8</td>
<td>28</td>
<td>0.986</td>
<td>0.993</td>
</tr>
</tbody>
</table>

Table 5. Accuracy for CNF target concepts

A comparison of run times is made in Figure 9 for a particular CNF target concept. It can be seen that, although EG algorithm made 39 calls to ID3 and Fringe made only 16 calls to ID3, EG algorithm took less time than Fringe. The reason for this was the large number of new features defined.

### 3.2.4 Expert-Gate Target Concepts

The algorithm uses the idea of experts and gate to build the final classifier, so we wanted to test the algorithm’s performance on a target concept made of a configuration of experts and gate. We used 5 chosen experts-gate configurations to test the performance of EG algorithm and Fringe. The experiments were repeated 10 times to minimize the effects of random initialization.

The results in Table 6 again show that Expert-Gate algorithm works remarkably well. For target concept eg2, fringe ran out of memory when the limit on dynamic memory was placed at 4000 segments or 40 MB.
Target Concept (CNF) : 
\( (x_2 + x_{13} + \bar{x}_4 + x_{12}) \cdot (\bar{x}_5 + \bar{x}_{11} + x_6 + x_8) \cdot (x_1 + x_3 + x_7 + x_{19}) \cdot (x_9 + \bar{x}_{14} + x_{15} + x_{16}) \cdot (\bar{x}_{18} + \bar{x}_{19}) \)

Number of features : 20
Number of training examples : 993
Number of test examples : 496

<table>
<thead>
<tr>
<th>Expert Gate Algorithm</th>
<th>Fringe</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of calls to ID3 : 39</td>
<td>Number of calls to ID3 : 16</td>
</tr>
<tr>
<td>Number of iterations : 7</td>
<td>Number of new features defined : 466</td>
</tr>
<tr>
<td>Run time : 63.23 seconds</td>
<td>Run time : 442.02 seconds</td>
</tr>
<tr>
<td>Accuracy : 0.879</td>
<td>Accuracy : 0.844</td>
</tr>
</tbody>
</table>

Figure 9. Comparison of run times on a CNF target concept

3.3 NETtalk

Although, the EG algorithm performs remarkably well on artificial problems, it is necessary to evaluate how the algorithm would perform in real life domains. The data from the NETtalk domain [SR87] was used to test the expert-gate algorithm. In this section, we describe the NETtalk domain and then present the results of our algorithm on this domain.

3.3.1 NETtalk Domain Description

The overall goal in the NETtalk domain is to learn to pronounce English words. We are given input data made of 3-tuples of word, its phoneme representation, and its stress representation. The aim is to predict the pronunciation of words not seen in the input data. Table 7 gives an example of the NETtalk data. There is a one-to-one mapping from the characters in the word to the characters in the phoneme and stress string. This representation tries to be as close as possible to
### Table 6. Accuracy for configurations of Experts-Gate as target concepts

<table>
<thead>
<tr>
<th>Target Concept</th>
<th>Features</th>
<th>EG acc</th>
<th>Fringe acc</th>
</tr>
</thead>
<tbody>
<tr>
<td>eg1</td>
<td>10</td>
<td>1.000</td>
<td>1.000</td>
</tr>
<tr>
<td>eg2</td>
<td>20</td>
<td>0.896</td>
<td>***</td>
</tr>
<tr>
<td>eg3</td>
<td>10</td>
<td>0.991</td>
<td>0.997</td>
</tr>
<tr>
<td>eg4</td>
<td>10</td>
<td>1.000</td>
<td>1.000</td>
</tr>
<tr>
<td>eg5</td>
<td>10</td>
<td>1.000</td>
<td>1.000</td>
</tr>
</tbody>
</table>

The one chosen by Sejnowski and Rosenberg [SR87].

### Table 7. An example of NETtalk data

<table>
<thead>
<tr>
<th>Word</th>
<th>Phoneme string</th>
<th>Stress string</th>
</tr>
</thead>
<tbody>
<tr>
<td>born</td>
<td>bcrn</td>
<td>&gt; 1 &lt;&lt;</td>
</tr>
<tr>
<td>led</td>
<td>lEd</td>
<td>&gt; 1 &lt;</td>
</tr>
<tr>
<td>tradition</td>
<td>trxdIS-xn</td>
<td>&gt;&gt; 0 &gt; 1 &lt; 0 &lt;&lt;</td>
</tr>
<tr>
<td>april</td>
<td>epr-L</td>
<td>1 &gt;&gt; 0 &lt;</td>
</tr>
<tr>
<td>dwelled</td>
<td>dwEl-d</td>
<td>&gt;&gt; 1 &lt;&lt;&lt;&lt;</td>
</tr>
</tbody>
</table>

To learn the pronunciation, two steps are performed.

1. Pronunciation is learned letter by letter. Of course, the pronunciation of a letter is dependent on the neighbor letters in the word context. Therefore, a learning-example consists of a window of 7 letters, with the letter to pronounce in the middle. Blanks are added if a letter is at the border of a word.

If W is a k-letter word from the input data, then k examples,

\[(w_1 p_1 s_1), (w_2 p_2 s_2), \ldots, (w_k p_k s_k)\]
are generated. \( w_i \) is a 7 letter window with the \( i \)-th letter of \( W \) in the middle position. \( p_i \) is the phoneme information and \( s_i \) stress information of the \( i \)-th letter of \( W \) respectively.

For example, if the tuple is (born bcrn " > 1 <<"), then we generate 4 windows,

(a) (___born b >)
(b) (___born_ c 1)
(c) (___born__ r <)
(d) (born___ n <)

2. Every example \( (w,p,s) \) is then converted to a binary vector representation. We employed the distributed representations developed by Sejnowski and Rosenberg [SR87] for the phonemes and the stresses.

(a) The 7 letter window \( w \) is converted into a 7*29 = 203-bit vector, each one of the 7 letters is represented by 29 bits.

(b) The phoneme \( p \) is converted into a 21-bit representation.

(c) The stress \( s \) is converted into a 5-bit representation.

For each letter from every word in the data, we now have a 203-bit feature vector and a 26-bit phoneme-stress classification vector. Every bit of the phoneme-stress information is learned independently and separately. That means we have to learn 26 different decision trees. For each phoneme/stress bit we learn the decision tree from a data set \( (x,B[i]) \), where \( x \) is the 203-bit vector representation of the 7-letter window and \( B(i) \) is a boolean value representing the \( i \)-th bit of the 26-bit phoneme-stress representation.
3.3.2 NETtalk Results

The 26 bits of the phoneme-stress representation are each learned separately. We used 1000 training words and 1000 test words, randomly picked from a dictionary of 20,002 words, made available by Sejnowski and Rosenberg. The accuracy of the Expert-Gate algorithm for the phoneme bits is given in Table 8 and that for the stress bits is given in Table 9.

<table>
<thead>
<tr>
<th>P-bit</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accuracy</td>
<td>96.4</td>
<td>93.9</td>
<td>98.2</td>
<td>95.6</td>
<td>91.4</td>
<td>97.3</td>
<td>99.2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>P-bit</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accuracy</td>
<td>99.6</td>
<td>99.6</td>
<td>99.9</td>
<td>100.0</td>
<td>98.8</td>
<td>94.4</td>
<td>98.6</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>P-bit</th>
<th>15</th>
<th>16</th>
<th>17</th>
<th>18</th>
<th>19</th>
<th>20</th>
<th>21</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accuracy</td>
<td>95.3</td>
<td>95.7</td>
<td>91.6</td>
<td>96.2</td>
<td>100.0</td>
<td>100.0</td>
<td>96.2</td>
</tr>
</tbody>
</table>

**Table 8. Accuracy for phoneme bits**

<table>
<thead>
<tr>
<th>S-bit</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accuracy</td>
<td>91.3</td>
<td>92.5</td>
<td>89.3</td>
<td>93.0</td>
<td>100.0</td>
</tr>
</tbody>
</table>

**Table 9. Accuracy for stress bits**

For each test word, the 26 predicted bits are concatenated to form the predicted classifying vector. This 26-bit predicted vector may not be same as any legal phoneme/stress binary representation vector. In that case, we consider the nearest neighbor to this vector which is legal. This legal predicted vector is then compared with the actual classification to find out whether the phoneme and stress classification is accurate. The overall phoneme/stress accuracy is calculated by adding the number of correct phoneme/stress classifications. The phoneme/stress accuracy results are given in Table 10 and Table 11.
Table 10. Phoneme accuracy on NETtalk data

For each letter, there is a phoneme and stress pair. If the classification of both phoneme and stress is correct then the classification of the letter is correct. Also, the letters make up the words, and if all the letters in the word are classified correctly then the word is said to be classified correctly. Figure 10 presents accuracy results for the 1000 test words.

In Table 12, we compare the accuracy of the EG algorithm with several other algorithms: nearest neighbor, RBF (Radial Basis Function networks)[WD92], ID3

Table 11. Stress accuracy on NETtalk data
Number of Training words : 1000
Number of Test words : 1000
Percentage of words correctly classified : 10.9
Number of test letters : 7242
Percentage of letters correctly classified : 68.5
Percentage of correctly classified phonemes : 82.3
Percentage of correctly classified stress : 78.2
Percentage of correctly classified phoneme bits : 97.03
Percentage of correctly classified stress bits : 93.23

Figure 10. Results for a data set from the NETtalk domain problem

[Qui86], backpropagation, Wolpert’s [Wol90] HERBIE algorithm, and ID3 with error-correcting output codes [DB91]. The results in Table 12 were taken from [WD92]. As can be seen from the table, the accuracy of the EG algorithm is comparable to that of the backpropagation algorithm.

3.4 Summary

This chapter presented the results for the actual application of the expert-gate algorithm. Section 3.1 deals with the selection of the parameter values and we present results to corroborate our selection. These results were very useful in deciding the actual program values.

In Section 3.2, we presented the results for a number of artificially created test problems. The target concepts in these problems were DNF functions, CNF functions or expert-gate configurations. The algorithm gave excellent results for all the test problems. The accuracy of the EG algorithm was comparable to that of Fringe. For some test problems, like cnf7 or eg2, Fringe used too much dynamic memory as it defined too many unnecessary features. On such test problems EG
Algorithm | Percent Correct
---|---:---:---:---
Nearest Neighbor | 3.3 | 53.1 | 61.1 | 74.0
RBF | 3.7 | 57.0 | 65.6 | 80.3
ID3 | 9.6 | 65.5 | 78.7 | 77.2
Expert-Gate | 10.9 | 68.5 | 82.3 | 78.2
Backpropagation | 13.6 | 70.6 | 80.8 | 81.3
Wolpert | 15.0 | 72.2 | 82.6 | 80.2
ID3 + 127-bit ECC | 20.0 | 73.7 | 85.6 | 81.1

Table 12. Comparison of the EG algorithm with other algorithms

algorithm works definitely better than Fringe.

We also tested the algorithm on the NETtalk problem, the results for which were presented in Section 3.3. It can be seen that the test sample accuracy of the Expert-gate algorithm is comparable to the backpropagation algorithm on the NETtalk problem. The Wolpert algorithm and the ID3 + Error Correcting codes approach perform better than the EG algorithm. Further testing is necessary to see how the EG algorithm performs on other real life problems. The results are encouraging to believe that this could be a viable approach for other real life problems.

In the next chapter we summarize our discussion and also discuss some enhancements to the EG algorithm that can be tried.
Chapter 4

Modifications and Conclusions

The expert-gate algorithm was presented in this thesis. Chapter 1 presented the necessary theoretical groundwork. Chapter 2 presented the algorithm and Chapter 3 presented the results. In this chapter, we discuss possible modifications to the algorithm which may improve its performance.

4.1 Future Work

Many different approaches can be tried in order to improve the performance of the expert-gate algorithm. We present a few in this section which we believe may be advantageous.

4.1.1 Choosing the Best Expert

The Expert-Gate algorithm is presented in the second chapter (Figure 7). In each iteration of the algorithm, we assign the examples to the best expert (Step 6(c)). We describe the method used for choosing the best expert in Section 2.3.2. When more than one expert classified the example correctly, the example was assigned to the numerically lowest correct expert. When no expert classified the example correctly, we always assigned that example to the numerically lowest expert. This approach had the advantage of consistency. In this section, we discuss some
other methods for choosing the best expert which may improve the algorithm's performance.

1. **Probability assignment to the expert classification:** We can use some model for assigning a probability estimate of the classification being correct for a given pair of decision tree and example. When more than one expert classifies an example correctly, we can use this model to choose the expert with the highest probability estimate. This might help to assign examples which belong to the same subgroup to the same expert and reduce the effect of the replication problem even further.

This approach will also solve the problem of assigning examples which are not classified correctly by any expert. We can always choose an expert which classifies the given example correctly with some positive probability. This means, there won't be any exceptions (examples which are not classified correctly by any expert).

2. **Assigning examples to multiple experts:** When more than one expert classifies the example correctly, the example could be assigned to all the experts. The gate is then learned with sets of experts appearing as the labels at the leaves. The classification of the example is then the majority opinion of the experts at the gate leaf.

3. **Handling exceptions:** When no expert classifies the example correctly, the current version of the algorithm always assigns such an example to the numerically lowest expert. Such examples can be treated as, exceptions to the concepts we are trying to capture as expert trees. These exceptions can be assigned to a separate expert created only for handling the exceptions. This approach may help in handling the exceptions better. Assigning the exceptions to the particular expert should be done only after the expert-gate configuration stabilizes a little, that is after some structure is built out of the
random initialization.

In addition, we can even create exception handlers for each expert. When an example is classified correctly by an expert ‘i’ and in the next iteration is not classified correctly by any expert, then that example is assigned to the exception handler ‘i’. This could further reduce the effect of the exceptions on the decision trees.

4.1.2 Pruning to an Optimal Tree

Minimal cost-complexity pruning [BFOS84] can be used to find the best decision tree that fits the given data. The cost-complexity of a decision tree for a data set is defined in terms of the resubstitution estimate (the cost of the tree) and the number of leaves in the tree (the complexity of the tree).

$$R_\alpha(T) = R(T) + \alpha n(T)$$

Here, $R(T)$ is the resubstitution estimate and $n(T)$ is the number of leaves. For a given value of the multiplying constant $\alpha$, we can find the subtree that minimizes the cost-complexity parameter. By successively increasing the value of $\alpha$, starting from zero, we get a sequence of trees. We decide the best tree by using either test sample estimate or cross-validation estimate. This tree is returned as the best pruned subtree.

We can use this technique in two ways to modify the algorithm.

1. In each iteration of the outer loop of the algorithm, we learn the gate and the experts. For expert/gate, the decision tree can be grown and the best pruned subtree found using the minimal cost complexity technique can be used. The disadvantage of this approach is that it may be too time consuming as we have to find the best pruned tree for each expert/gate in each iteration.

2. After the outer loop is exited, the algorithm outputs the gate decision tree and the expert decision trees. We can form a single decision tree from this
gate-experts configuration by simply replacing each leaf of the gate decision tree by the corresponding expert decision tree. The minimal cost-complexity technique can then be used to find the best pruned subtree of this decision tree. This subtree can be then used for classification.

4.1.3 Improving the Learning Algorithms

The Expert-Gate algorithm is independent of the learning algorithms used for learning the Gate/Experts. Any other learning algorithm can be also used in place of ID3. In fact, as we can see from Table 12, ID3 with Error-correcting Codes gave the best results for the NETtalk domain. If we use ID3 with Error-correcting Codes as the multiclass learning algorithm in the Expert-Gate algorithm then there is a good chance that the performance of the algorithm will improve. We believe that this is a definite direction for improving the performance of the algorithm.

4.2 Observations

The results in Section 3.2 show that the expert-gate algorithm works remarkably well for DNF and CNF target concepts. Some peculiar things were observed about the gate and the expert decision trees. The example in Figure 11 will demonstrate this very well.

As can be seen in Figure 11, both the gate and expert trees are unbalanced. At most of the nodes, one of the children is a leaf and the other one is an interior node. Also, there are only two experts, although as many as 8 were allowed to grow at the start. One of the experts is just a classifying leaf whereas the other is a decision tree using all the features. This same kind of structure was observed for other DNF and CNF target concepts with small numbers of features. When the number of features became larger (more than 25), the same kind of structure was observed with a little change. The number of experts sometimes became 3 or 4. No
direct relationship was observed between the number of features and the number of experts.

A possible reason for the occurrence of the type of expert trees shown in Figure 11 could be the way the best expert is selected in the algorithm. When no expert classifies an example correctly, the numerically lowest expert is assigned that example. This could lead to an unbalanced decision tree for expert 1.

If the expert decision trees are substituted in the gate decision tree and the redundant nodes are removed, then it can be seen that the final decision tree is the same as the correct decision tree (although it is slightly bigger than the correct optimal decision tree.) The same thing was observed for other DNF and CNF
target concepts.

4.3 Summary

This thesis presented the Expert-Gate algorithm. The main aim of the algorithm was to handle the replication problem more effectively. In this goal we were successful as the Expert-Gate algorithm showed very good results for DNF or CNF target functions. When tried on a real domain (NETtalk), the algorithm gave favorable results though not comparable to the best.

Further modifications suggested in this chapter, need to be done to improve the performance of the algorithm. With a little improvement, the algorithm does seem a good candidate to try in other real life domains.
BIBLIOGRAPHY


Appendix A

Description of DNF Target Concepts

dnf1 : \( \bar{x}_1 x_7 \bar{x}_6 \bar{x}_5 + \bar{x}_5 x_1 + \bar{x}_14 x_12 x_{11} \bar{x}_{10} x_9 \bar{x}_7 \bar{x}_4 x_3 \bar{x}_2 \)
dnf2 : \( \bar{x}_9 + \bar{x}_10 \bar{x}_3 + \bar{x}_5 \bar{x}_4 + x_1 + \bar{x}_16 x_10 x_8 x_7 x_6 \bar{x}_2 x_1 \)
dnf3 : \( x_{16} x_{15} x_{14} x_{13} x_3 x_1 + x_{18} \bar{x}_{15} x_{12} x_4 \bar{x}_3 \)
\(+ x_{16} \bar{x}_7 x_2 + x_{16} x_{15} x_{10} + \bar{x}_{14} \bar{x}_9 x_5 x_2 \)
dnf4 : \( x_{17} \bar{x}_{16} \bar{x}_{13} x_8 \bar{x}_4 x_2 + x_{20} x_{16} x_8 x_6 \bar{x}_4 \bar{x}_3 x_2 \)
\(+ x_{20} \bar{x}_{18} \bar{x}_{16} x_{12} x_{11} \bar{x}_9 x_3 x_2 \)
dnf5 : \( x_{22} x_{20} x_{18} x_{17} \bar{x}_{16} \bar{x}_{12} x_6 x_5 x_1 + \bar{x}_{16} x_5 x_1 \)
\(+ x_{19} x_{17} x_{16} \bar{x}_{15} x_{14} \bar{x}_{11} x_8 \bar{x}_5 x_4 \bar{x}_3 x_1 \)
dnf6 : \( x_{24} \bar{x}_{20} x_{19} \bar{x}_{13} + \bar{x}_{14} \bar{x}_9 \bar{x}_2 + \bar{x}_{22} \bar{x}_{13} \)
\(+ \bar{x}_{24} x_{19} x_5 + x_{17} + \bar{x}_{23} \bar{x}_{14} \bar{x}_{11} x_5 \)
dnf7 : \( \bar{x}_5 \bar{x}_4 \bar{x}_1 + x_{21} \bar{x}_{17} x_{16} x_{10} \bar{x}_5 + \bar{x}_{25} x_{21} x_{16} x_9 \)
\(+ x_{26} \bar{x}_7 x_5 + x_{19} \bar{x}_{18} \bar{x}_{16} \bar{x}_{15} x_9 x_2 + \bar{x}_{22} \bar{x}_{13} \bar{x}_{11} \bar{x}_8 \)
dnf8 : \( \bar{x}_{25} \bar{x}_{14} x_6 + \bar{x}_{25} x_{15} + \bar{x}_{28} x_{24} x_{19} \bar{x}_{18} \bar{x}_{15} x_{11} \bar{x}_9 x_3 \)
\(+ \bar{x}_{28} \bar{x}_{21} \bar{x}_{18} x_{14} \bar{x}_{12} x_9 x_8 x_5 x_3 x_1 + \bar{x}_{24} \bar{x}_{19} \bar{x}_{17} \bar{x}_{14} x_1 \)
Appendix B

Description of CNF Target Concepts

cnf1: \[(\overline{x}_{13} + x_7 + x_6 + x_1) \cdot (\overline{x}_{14} + x_{12} + x_{11} + x_{10} + x_9 + x_7) \cdot (\overline{x}_4 + x_3 + x_2) \cdot (\overline{x}_5 + x_1)\]

cnf2: \[(\overline{x}_9) \cdot (\overline{x}_{10} + x_3) \cdot (\overline{x}_5 + x_4) \cdot (x_1) \cdot (\overline{x}_{16} + x_{10} + x_8 + x_7 + x_6 \overline{x}_2 + \overline{x}_1)\]

cnf3: \[(x_{16} + x_{15} + x_{14} + x_{13} + x_3 + x_1) \cdot (x_{16} + \overline{x}_7 + x_2) \cdot (x_{16} + x_{15} + x_{10}) \cdot (\overline{x}_{14} + \overline{x}_9 + x_5 + x_2) \cdot (x_{18} + \overline{x}_{15} + x_{12} + x_4 + \overline{x}_3)\]

cnf4: \[(x_{17} + \overline{x}_{16} + \overline{x}_{13} + x_6 + \overline{x}_4 + x_2) \cdot (x_{20} + \overline{x}_{18} + \overline{x}_{16} + x_{12} + x_{11} + \overline{x}_9 + x_3 + x_2) \cdot (x_{20} + x_{16} + x_8 + x_6 + \overline{x}_4 + \overline{x}_3 + x_2)\]

cnf5: \[(x_{22} + x_{20} + x_{18} + x_{17} + \overline{x}_{16} + \overline{x}_{12} + x_6 + x_5 + x_1) \cdot (\overline{x}_{16} + x_5 + \overline{x}_1) \cdot (x_{19} + x_{17} + x_{16} + \overline{x}_{15} + x_{14} \overline{x}_{11} + x_8 + \overline{x}_5 + x_4 + \overline{x}_3 + \overline{x}_1)\]

cnf6: \[(x_{24} + \overline{x}_{20} + x_{19} + \overline{x}_{13}) \cdot (\overline{x}_{14} + \overline{x}_9 + \overline{x}_2) \cdot (\overline{x}_{24} + x_{19} + \overline{x}_5) \cdot (x_{17}) \cdot (\overline{x}_{23} + \overline{x}_{14} + \overline{x}_{11} + \overline{x}_5) \cdot (\overline{x}_{22} + \overline{x}_{13})\]

cnf7: \[(\overline{x}_5 + \overline{x}_4 + \overline{x}_1) \cdot (x_{21} + \overline{x}_{17} + x_{16} + x_{10} + \overline{x}_5) \cdot (x_{26} + x_{17} + x_5) \cdot (x_{19} + \overline{x}_{18} + \overline{x}_{16} + \overline{x}_{15} + x_9 + x_2) \cdot (\overline{x}_{22} + \overline{x}_{13} + \overline{x}_{11} + \overline{x}_8) \cdot (\overline{x}_{25} + x_{21} + x_{16} + x_9)\]
cnf8 : 

\[(\overline{x}_{25} + \overline{x}_{14} + x_6). (\overline{x}_{25} + x_{15}). (\overline{x}_{24} + \overline{x}_{19} + \overline{x}_{17} + \overline{x}_{14} + x_1)\]

\[. (\overline{x}_{28} + \overline{x}_{21} + \overline{x}_{15} + x_{14} + \overline{x}_{12} + x_9 + x_8 + x_5 + x_3 + x_1)\]

\[. (\overline{x}_{28} + x_{24} + x_{19} + \overline{x}_{18} + \overline{x}_{15} + x_{11} + \overline{x}_{9} + x_3)\]
Appendix C

Description of Expert-Gate Target Concepts

<table>
<thead>
<tr>
<th>EG1</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>gate</td>
<td>(1 (2 2 (3 1 3)) (3 (4 1 2) (2 4 1)))</td>
</tr>
<tr>
<td>expert 1</td>
<td>(5 (6 0 (7 1 0)) (8 (9 (7 1 0) 0) (6 0 1)))</td>
</tr>
<tr>
<td>expert 2</td>
<td>(7 (6 1 (9 0 1)) 0)</td>
</tr>
<tr>
<td>expert 3</td>
<td>0</td>
</tr>
<tr>
<td>expert 4</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>EG2</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>gate</td>
<td>(1 (3 (4 (7 1 (2 3 1)) (5 (6 4 2) 5)) (9 (8 3 2) 5)) (7 (2 3 (8 5 2)) (3 (6 4 (5 1 2)) (9 5 (4 3 1))))</td>
</tr>
<tr>
<td>expert 1</td>
<td>(11 (16 0 (12 1 0)) (18 (13 (17 1 0) 0) (14 (15 1 0) 1)))</td>
</tr>
<tr>
<td>expert 2</td>
<td>(19 (16 (12 1 (14 0 1)) (11 0 1)) 0)</td>
</tr>
<tr>
<td>expert 3</td>
<td>0</td>
</tr>
<tr>
<td>expert 4</td>
<td>(13 (19 0 (14 1 0)) (15 (16 0 1) (17 1 0)))</td>
</tr>
<tr>
<td>expert 5</td>
<td>(17 (12 1 0) (11 0 1))</td>
</tr>
<tr>
<td>EG3</td>
<td></td>
</tr>
<tr>
<td>-------------</td>
<td>---------</td>
</tr>
<tr>
<td>gate</td>
<td>(1 2 4 4 2 3 1 3) 4 3 4 2 2 4 1)</td>
</tr>
<tr>
<td>expert 1</td>
<td>(5 6 0 7 1 0) 8 9 7 1 0 6 0 1)</td>
</tr>
<tr>
<td>expert 2</td>
<td>(7 6 1 9 0 1) 0</td>
</tr>
<tr>
<td>expert 3</td>
<td>(8 6 1 0 5 1 0)</td>
</tr>
<tr>
<td>expert 4</td>
<td>(9 5 8 0 1 0 1)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>EG4</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>gate</td>
<td>1</td>
</tr>
<tr>
<td>expert 1</td>
<td>(5 0 1)</td>
</tr>
<tr>
<td>expert 2</td>
<td>(7 6 1 9 0 1) 0</td>
</tr>
<tr>
<td>expert 3</td>
<td>(8 6 1 0 5 1 0)</td>
</tr>
<tr>
<td>expert 4</td>
<td>(9 5 8 0 1 0 1)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>EG5</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>gate</td>
<td>(2 1 2)</td>
</tr>
<tr>
<td>expert 1</td>
<td>(5 0 1)</td>
</tr>
<tr>
<td>expert 2</td>
<td>(7 6 1 0) 0</td>
</tr>
<tr>
<td>expert 3</td>
<td>(8 6 1 0 5 1 0)</td>
</tr>
<tr>
<td>expert 4</td>
<td>(9 5 8 0 1 0 1)</td>
</tr>
</tbody>
</table>