# A Partial Memory Incremental Learning Methodology And Its Application To Computer Intrusion Detection

**Marcus A. Maloof and Ryszard S. Michalski**

**MLI 95-2**

**March 1995**

# A Partial Memory Incremental Learning Methodology and its Application to Computer Intrusion Detection

Marcus A. Maloof and Ryszard S. Michalski
Center for Machine Learning and Inference
George Mason University
Fairfax, VA 22030-4444
{maloof, michalski}@aic.gmu.edu

March 1995

# A Partial Memory Incremental Learning Methodology and its Application to Computer Intrusion Detection

## Abstract

This paper discusses work in progress and introduces a partial memory incremental learning methodology. The incremental learning architecture uses hypotheses induced from training examples to determine representative examples, which are maintained for future learning. Criticism and reinforcement from the environment or the user invoke incremental learning once the system is deployed. Such an architecture and development methodology is necessary for applications involving intelligent agents, active vision, and dynamic knowledge-bases. For this study, the methodology is applied to the problem of computer intrusion detection. Several experimental comparisons are made using batch and incremental learning between AQ15c, a feed-forward neural network, and *k*-nn. Experimental results suggest that AQ15c has several advantages over other methods in terms of predictive accuracy, incremental learning, learning and recognition times, the types of concepts induced by the method, and the types of data from which these methods can learn.

**Key words:** concept learning, incremental learning, intrusion detection, audit trail analysis.

## Acknowledgments

The authors thank Eric Bloedorn for many discussions on this work, reading drafts of the paper, and recommending Davis's work on characterizing time series. Thanks also goes to Dorothy Denning for providing pointers to SRI's work on intrusion detection.

This research was conducted in the Center for Machine Learning and Inference at George Mason University. The Center's research is supported in part by the Advanced Research Projects Agency under Grant No. N00014-91-J-1854, administered by the Office of Naval Research, and the Grant No. F49620-92-J-0549, administered by the Air Force Office of Scientific Research, in part by the Office of Naval Research under Grant No. N00014-91-J-1351, and in part by the National Science Foundation under Grants No. IRI-9020266 and DMI-9496192.

## 1 Introduction

This paper discusses work in progress and introduces a partial memory incremental learning methodology. The methodology is based on the AQ inductive learning algorithm and consequently uses Variable-Valued Logic ($VL_1$) as a representation language (Michalski 1969, 1972, 1973, 1980). The proposed methodology is incremental in that (a) static concepts are incrementally learned, and (b) incrementally changing concepts are learned. We conjecture that in both cases, completeness and consistency is maintained. The proposed methodology maintains and uses a partial memory model (Reinke and Michalski 1988), in which representative examples are maintained throughout the learning process that maximally expand and constrain learned concepts. This partial memory scheme is contrasted by no memory incremental learning (e.g., reinforcement learning), and full memory incremental learning (Hong et al. 1986; Reinke and Michalski 1988) Learned concepts aid in determining the set of representative concepts. This methodology is designed to support applications that require incremental learning of concepts and learning of concepts that incrementally change.

Traditional application of machine learning to intelligent systems development involves collecting a set of training examples, expressing those training examples using a representation language in a representation space that facilitates learning, and using a learning algorithm to induce a set of concepts from the codified training examples. These induced concepts are subsequently validated and incorporated into an inferential system and deployed into an environment. This paradigm dictates that learned concepts, once deployed in a system, are static and do not change with respect to the domain, the user, or the environment. If one or more of these factors does change, the entire development process is repeated to produce a system capable of coping with the new scenario.

New applications, such as intelligent agents (e.g., Maes 1994) and active vision (e.g., Ballard and Brown 1993), require autonomous or semi-autonomous functioning and adaptation to changes in the domain, the environment, or the user. Such requirements suggest that incremental learning, as opposed to batch learning, is needed. As experimental results presented here demonstrate, when compared to batch learning, partial memory incremental learning yields faster learning times and reduced memory requirements at the expense of slightly lower predictive accuracy. Although incremental learning is needed in application areas such as intelligent agents and active vision, the application considered here is a dynamic knowledge-based system for computer intrusion detection (Denning 1987).

To apply the proposed partial memory incremental learning methodology to the problem of detecting intruders in a computer system, training examples were extracted from a Unix audit file and represented using $VL_1$. Symbolic *use profiles* for the computer system's users were learned using the AQ inductive learning algorithm . Experimental comparisons were made between AQ15c (Wnek 1994), the most recent implementation of the AQ algorithm, a feed-forward neural network (Zurada 1992), a non-symbolic learning algorithm, and *k*-nearest neighbor (Weiss and Kulikowski 1992), a statistical pattern recognition technique. These results demonstrate that AQ has distinct advantages over neural networks and *k*-nearest neighbor (*k*-nn). Furthermore, the advantages of AQ extend beyond predictive accuracy and learning and recognition times to issues of coping with symbolic data and incremental learning, which are discussed in detail.

Global connectivity of computer systems has elevated concerns about computer security and intrusion detection. Passwords, gateways, and firewalls are all designed to protect computing and information resources from unauthorized access. Audit trails serves as a valuable tool to system administrators when detecting probes and attacks. Intruders often exploit holes in the operating system or crack password files to gain access to the computer system and masquerade as a legitimate user. At this point, detection of an intruder becomes increasingly difficult, especially with computer systems that have a large number of users.

Quite a bit of research has been conducted in attempts to statistically model user behavior (Smaha 1988; Lunt et al. 1989; Vaccaro 1989; Anderson et al. 1994b). Statistical models, or profiles, once acquired, are subsequently used to verify that a user's recent behavior is consistent with past behavior. While a statistical approach to this problem is certainly valid, there are advantages to the machine learning approach taken here. These advantages include using both statistical and logical information when learning use profiles, learning symbolic concepts which can be inspected and understood by humans, and finally, because learned concepts are directly accessible, learning incrementally is possible, which allows the system to adapt to changes in a user's behavior over time.

The organization of this paper is as follows. Section 2 provides details of the $VL_1$ representation language, the AQ algorithm, and recognition through flexible matching. It also presents a taxonomy for incremental learning and briefly reviews related work in intrusion detection. Section 3 introduces the incremental learning architecture and methodology based on $VL_1$ and the AQ algorithm. Section 4 describes comparative experimental results between AQ15c, *k*-nn, and a feed-forward neural network. The paper concludes with a discussion of the results and directions for future work.

## 2 Background

This section provides background material on the $VL_1$ representation language, the AQ algorithm, recognition through flexible matching, presents a taxonomy for incremental learning, and briefly reviews related work in intrusion detection.

### 2.1 Representation Language and AQ-Based Learning Method

In our approach, a concept description is induced from a set of pre-classified training examples within the context of background knowledge. The induced concept description can then be used to classify or recognize unknown observations. In this approach, a few epistemological assumptions are made. First, each training example is descriptive of the concept to be learned. Second, given training examples for a concept are sufficient to learn a concept description. Sufficiency is either assumed or guaranteed by an expert. Third and finally, future unknowns will be from the same classes from which the training examples were drawn.

Examples, background knowledge, and concept descriptions must be expressed using a representation language, preferably a symbolic representation language. Symbolic representation languages are stressed because of the importance of human understandability of learned concepts. Human understandability of learned concepts is important because if a system could act in a way that is harmful to humans, then the concepts responsible for this behavior require modification. Symbolic concepts allow for this inspection and modification. To accomplish this, we must able to determine the boundaries of decision regions so we know under which conditions the system might change its decision and consequently its behavior. Mapping the boundaries of decision regions formed by neural network learning is made more difficult, since learned concepts are non-symbolic.

Each potential representation language for expressing examples and learned concepts carries a representational bias. This bias ultimately affects what knowledge can be represented and learned. The AQ15c concept learning system (Wnek 1994) uses an attributional representation language based on Variable-Valued Logic, or $VL_1$ (Michalski 1973), to learn decision rules from training examples. We begin by defining a representation space for the problem which is formed from a finite set of relevant attributes and finite attribute domains. Each training example is expressed as a vector of attribute values, which is a particular instantiation of the relevant attributes using values from the attribute domains. Hence, the assumption is made that all examples and concepts can be represented as a set of attribute-value pairs expressed in disjunctive normal form. Hypotheses induced by AQ have the form:

*D1* <:: *C1*

where

| | |
|---|---|
| *C1* | is a concept description or cover consisting of a disjunction of complexes, |
| *D1* | is the decision class assigned by *C1*, and |
| <:: | denotes a class assignment operator. |

Complexes or rules are conjunctions of selectors or conditions, each having the form:

'[' <*referee*> <*relation*> <*referent*> ']'

where

| | |
|---|---|
| <*referee*> | is a member of the finite set of relevant domain attributes, |
| <*relation*> | is a relational operator (=, <>, >, <, >=, <=), and |
| <*referent*> | is a subset of the finite domain for <*referee*>. |

Under strict matching conventions, a decision class is assigned if one of the rules in the associated cover is true. A rule is true if all of its selectors are true. Figure 5.1 shows two rules induced using AQ15c (Wnek 1994).

The objective is to induce a set of hypotheses that describes the training examples in a maximally general way. The difficulty arises with the term *maximally general*. When phrased in this manner, the problem of finding a maximally general description is an instance of the set covering problem. That is, find a minimal set of attribute-value pairs that covers (or explains) all the positive concept examples while not covering any of the negative concept examples. Negative concept examples either are explicitly labeled as such, or in a multiple concept learning context, are formed by the remaining training examples for all other concepts.

The set covering problem is known to be NP-complete, but the AQ algorithm (Michalski 1969) solves the set covering problem in a quasi-optimal manner. Briefly, the AQ algorithm randomly selects one of the positive training examples (referred to as the *seed*) and uses this example as the basis to compute a maximally general description with respect to the negative examples (referred to as the *bounded star*). A preference criterion is used to select the most preferable description from the bounded star. If this description results in the coverage all positive examples, then the final concept description is the disjunction of all descriptions selected from bounded stars. If positive events remain uncovered by the description, then the algorithm is repeated until all positive examples are covered. The AQ algorithm guarantees completeness and consistency of learned concepts. Completeness means that a learned concept covers all positive examples. Consistency means that a learned concept does not cover any negative examples. AQ15c (Wnek 1994) is the most recent implementation of the AQ algorithm.

## 2.2 Recognition Through Flexible Matching

After learning and validation, induced concepts can be incorporated into a system and deployed. As new observations come to the system, it must classify these events. Under strict matching conventions, if the example's attribute values satisfy all conditions of a rule, then the decision class to which the rule belongs is assigned.

Conceptually, rules carve out decision regions in an event space. An event or representation space is defined by the finite set of domain attributes and the values these attributes take. Referring to Figure 2.1, the concept description $C_1$ associated with decision class $D_1$ and the concept description $C_2$ associated with decision class $D_2$ both carve out decision regions in the event space E. Concept $C_1$ consists of three covers (illustrated by overlapping rectangles), while concept $C_2$ consists of two. Regardless of whether matching is strict or flexible, if an unknown example, such as example $e_1$, falls within a decision region, it is assigned the decision class associated with that region. Example $e_1$ would be assigned to decision class $D_1$.

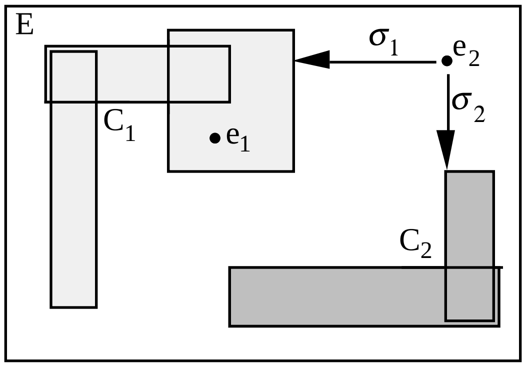

Figure 2.1: Examples and concepts in an event space.

On the other hand, if an example does not fall within a decision region, as is the case with example $e_2$, then a strict matching algorithm would not be able to classify the event and it would be left as unclassified or unknown. Conversely, a flexible matching algorithm will compute the distance from the example to all the concepts in the example space and assign the decision class with the greatest degree of match. Thus, a flexible matching system will always provide the best possible decision. Flexible matching alleviates the brittleness usually associated with rule-based reasoning systems.

Two aspects of rule induction necessitate flexible matching. First, AQ is a heuristic algorithm, so although AQ is capable of generating optimal descriptions, this cannot be guaranteed in all cases. Second, for sufficiently complex representation spaces, it is practically impossible to gather all training examples associated with a concept. Consequently, even if we had an admissible rule induction algorithm, our inability to gather all training examples pertaining to a concept makes it impossible to learn the optimal concept.

Several flexible matching schemes exist to calculate the degree of match between examples and concepts (see Wnek 1994). The method used for experiments presented in this paper is computed as follows. The degree of match $\sigma_i$ between the example $e_2$ and the concept description $C_i$ consisting of *n* complexes is given by:

$$\sigma_i = \sum_{j=1}^{n} \frac{\alpha_{ij}}{\beta_{ij}} - \prod_{j=1}^{n} \frac{\alpha_{ij}}{\beta_{ij}} \qquad (1)$$

where

$\alpha_{ij}$ is the number of selectors in rule *j* of concept $C_i$ satisfied by example $e_2$, and
$\beta_{ij}$ is the total number of selectors in rule *j* of concept $C_i$.

Formula 1 yields a real number in the range [0, 1] where 0 represents no match and 1 represents complete match.

## 2.3 Taxonomy for Incremental Learning

Incremental learning is the direct or indirect iterative modification or refinement of concepts from sets of training examples distributed over time (Reinke and Michalski 1988). Referring to Figure 2.2, incremental learning can be accomplished using one of three memory models: no memory, partial memory, and full memory.

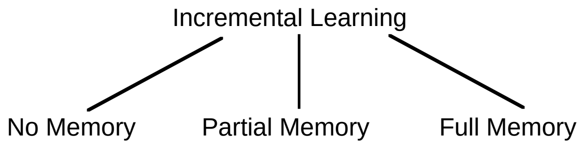

Figure 2.2: Taxonomy for incremental learning.

With a no memory model, concepts are refined based solely on the current set of new training examples, which are discarded after the incremental learning step is complete. Reinforcement learning is an example of no memory incremental learning. A partial memory model involves storing a portion of training examples, usually in some representative or generalized form, for future learning steps. Partial memory learning uses previously learned concepts to aid in judging which training examples are representative. The work presented in this paper falls into this category and is discussed further in section 3. Finally, under a full memory model, all training examples are maintained and concepts are refined in response to new training examples, but are modified within the context of all training examples seen. Examples of full memory systems include AQ15 (Hong et al. 1986) and the system described by Bhandaru and Murty (1991). Note that these systems are not intended to learn incrementally changing concepts.

The primary disadvantage to full memory incremental learning is the storage requirements. All training examples collected must be retained. While many may argue that storage is inexpensive, few users are willing to devote large amounts of storage space to an intelligent agent, for example, that performs relatively simple tasks. Furthermore, memory requirements are great concerns for builders of autonomous vehicles.

## 2.4 Related Work on Intrusion Detection

Approaches to intrusion detection fall primarily into two categories: statistical approaches and machine learning approaches. While both statistics and machine learning seek simple descriptions for collections of data, the approaches differ primarily in the nature of these simple descriptions. Statistical approaches typically use analog models and produce numeric results. Machine learning, on the other hand, uses symbolic or logic-based models to produce symbolic results, which may be complemented by numerics. Based on these definitions, neural network models, for example, are sometimes classified as either a statistical approach or a machine learning method. Unfortunately, the boundaries between machine learning and statistics are sometimes faint, and overlap and intermingle, but we will attempt to place surveyed approaches into one of these categories with justifications and caveats.

## Statistical Approaches

Denning's (1987) seminal paper laid the foundations for the Intrusion Detection Expert System (IDES) which uses a statistical component for anomaly detection and a rule-based component for detecting known intruder behaviors (Lunt et al. 1989). The IDES system later evolved into the Next Generation Intrusion Detection Expert System (NIDES), both of which were developed at SRI International (Anderson et al. 1994a). Figure 2.3 shows the NIDES architecture.

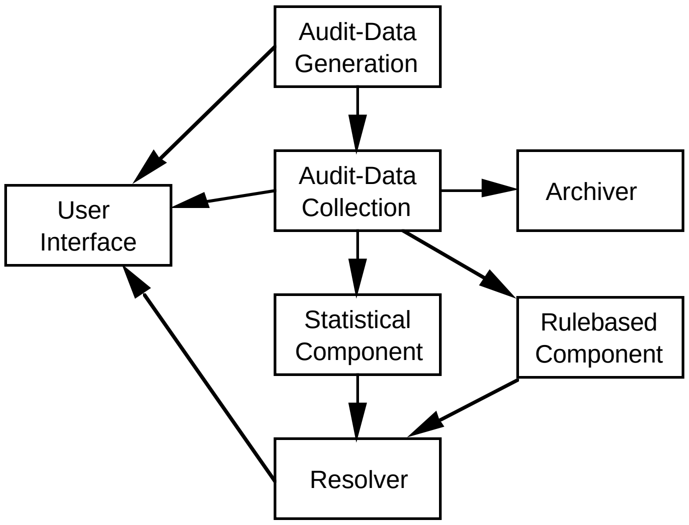

Figure 2.3: NIDES Architecture (Anderson et al. 1994a).

The NIDES statistical component (Javitz and Valdes 1994) detects anomalous behaviors in user's, groups of users, and in the global computing system by developing profiles, which are histories of long-term computing activity. Each audit record generated is expressed as a vector of measures, which NIDES analyzes and returns a numeric representing the degree of abnormality. If the number of abnormal records exceeds some pre-set threshold over a determined period of time, then an alert is sent to system administrators.

The abnormality metric is computed from four classes of measures: intensity, audit record distribution, categorical, and counting measures. These measures are aged using an exponential aging function so that recent activity is weighted more heavily. Intensity measures capture absolute levels of activity on varying time scales. Examples of intensity measures would be the number of audit records generated over a one minute period and over a one hour period.

Audit record distribution measures divide, or distribute, short periods of audit records into categories of activity. These categories of activity are compared with historical data to see if current use is similar to past use. For example, for a given period, 60% of a user's audit records pertain to CPU usage, while 30% pertain to disk access. This short term behavior would be compared with the user's historical data to see if this recent activity is suspicious.

Categorical measures profile non-numeric aspects of a user's behavior, such as the user's terminal, filenames accessed, and the like. Each individual categorical measure consists of a finite domain. As a user's profile is developed, frequency counts are made and a discrete probability density function is computed for this finite domain. For example, a user is most likely to login from the terminal in her office. Occasionally, she logs in from the conference room. If she were to login from her boss's terminal, then she would not match her historical profile, which would be a factor when NIDES computes the user's abnormality metric.

Finally, counting measures are simply counts of various system measures, such as number of CPU seconds and number of characters read or written. Again, these measures are compared to historical use.

Other statistically-based intrusion detection systems include Wisdom & Sense (Vaccaro 1989) and Haystack (Smaha 1988). Haystack is also the core of the Distributed Intrusion Detection System (Mukherjee et al. 1994).

## Neural Network Approaches

Debar et al. (1992) describe a recurrent neural network component for detecting anomalous user behavior by learning Unix command sequences (see Figure 2.4). Each command from a pre-defined set of commands was assigned to an input neuron of the network. The presence of a command at time *t* is represented as a 1; the absence is represented as 0. The output neurons, ranging from 0 to 1, predict the next likely command in the sequence. A knowledge-based system for neural network analysis and control manages recurrent learning and uncertainty. An expert system analyzes neural network results within the context of a security policy and produces a decision to the user. Experimental results were presented for eight command sequences for one user. Over the eight command sequences, the recurrent neural network predicted the next command with an average predictive accuracy of 82%. Results were not as strong for predicting the second and third commands in the sequence.

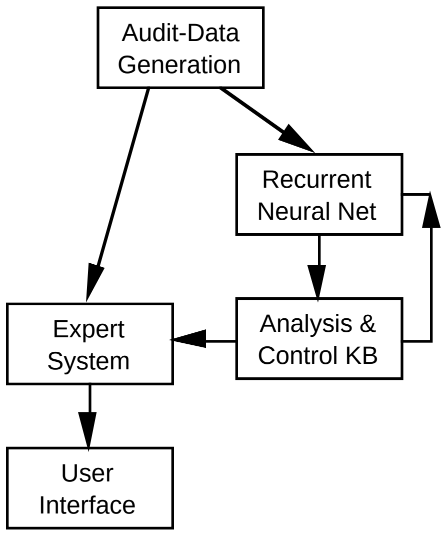

Figure 2.4: Neural network architecture for intrusion detection.

**Rule-Based Approaches**

Teng et al. (1990) describe an approach for inductively learning rules from temporal patterns, or sequences, using the Time-based Inductive Machine (TIM). Induced rules are subsequently used for anomaly detection (see Figure 2.5). Security events consisted of a time stamp and descriptive information, such as user name, event type, and process identifier. A discrete and linear model of time is assumed. Rules generated by TIM have strictly ordered, deterministic sequences as conditional elements with probabilistic consequents. For example, consider the simple sequence given by Teng et al. (1990):

A B C S T S T A B C A B C

From this sequence, TIM induces the following rule-base:

| | |
|---|---|
| R1: | A, B $\rightarrow$ (C, 100%) |
| R2: | C $\rightarrow$ (S, 50%; A 50%) |
| R3: | S $\rightarrow$ (T, 100%) |
| R4: | T $\rightarrow$ (A, 50%; S, 50%) |

Rules are evaluated and selected based on an entropy measure. In the above example, rules R2 and R4 would be discarded, while rules R1 and R3 would remain to form a profile because they are stronger since they cover more events in the sequence and their consequents are more certain. No experimental results were reported, although the authors did comment that "about 9.5% of the rules generated with an entropy value of less than 0.25 could explain or cover more than 63.5% of the security events over a given period of time" (Teng et al. 1990, p. 28).

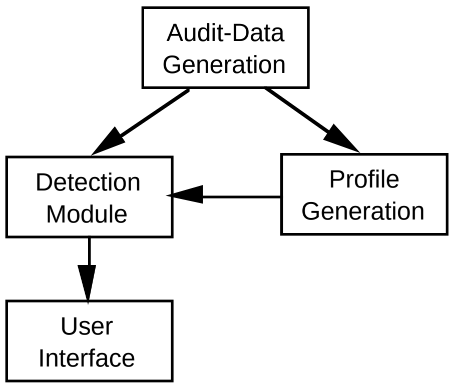

Figure 2.5: Intrusion detection architecture based on TIM (Teng et al. 1990).

## 3 Methodology

Viewing an intrusion detection application as a concept learning problem gives rise to two distinct phases (see Figure 3.2). The first phase, or the development phase, involves collecting an initial set of training examples such that a sufficient concept can be learned and will provide the system with enough inferential capability to be useful in its intended environment. The development phase equates to the historical and traditional paradigms of concept learning or learning from examples.

The second phase, or the deployment phase, involves installing the system in its environment where it should function in a semi-autonomous fashion. During the deployment phase, the system must incrementally learn and adapt to changes in the environment or in the behaviors of users. Incremental learning is required when an observation is misclassified, which is determined by feedback from the user or from the environment.

A partial memory incremental learning approach uses learned concepts to determine which training examples establish the outer bounds of a concept. Referring to Figure 3.1, assume that for some event space or representation space *E*, we have a collection of positive and negative examples and that we have learned some concept *c* that is complete and consistent. That is, the concept covers all of the positive examples and none of the negative examples. The representative positive examples of a concept are those examples that lie at the boundaries of the concept in the event space (e.g., the outlined plus signs), while the representative negative examples are those that constrain the concept in the event space (e.g., the outlined minus signs). All other training examples can be discarded.

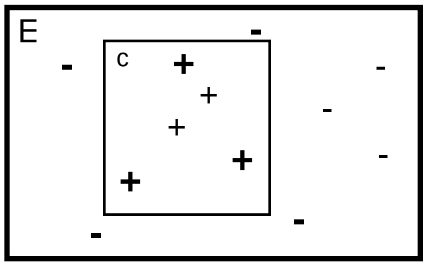

Figure 3.1: Partial memory incremental learning.

After the initial concepts have been learned in the development phase and the system has been deployed, the learned concepts are used for inference. The system will receive reinforcement or criticism from its environment, its user, or both. If the system makes a wrong decision, then this is a signal that the system's concepts require refinement. The misclassified training example is included with the existing representative examples and inductive learning takes place. Once new concepts are learned, the training examples are evaluated and those determined to be representative are selected to form the new set of representative examples. The new concepts are used for inference and the new representative training examples are stored. This process repeats indefinitely. Figure 3.2 illustrates the architecture for a partial memory incremental learning system.

A partial memory incremental learning scheme would be difficult, if not impossible, to implement for memory-based and neural network learning methods. With *k*-nn, for example, there is no hypothesis formation, which is a necessary requirement for partial memory incremental learning. To incrementally learn concepts using *k*-nn, additional training examples are simply memorized. To learn incrementally changing concepts, the same memorizing mechanism is exploited, but additional aging processes must be implemented so old concepts are forgotten.

Neural network learning, on the other hand, clearly formulates generalized hypotheses, but does so in a non-symbolic manner. Consequently, a neural network's learned concepts are not available for human inspection or interpretation, which makes mapping concept boundaries difficult, if not impossible. Not only is such mapping necessary for partial memory incremental learning, but also direct manipulation of learned concepts is required for no memory and full memory incremental learning. Consequently, incremental learning,

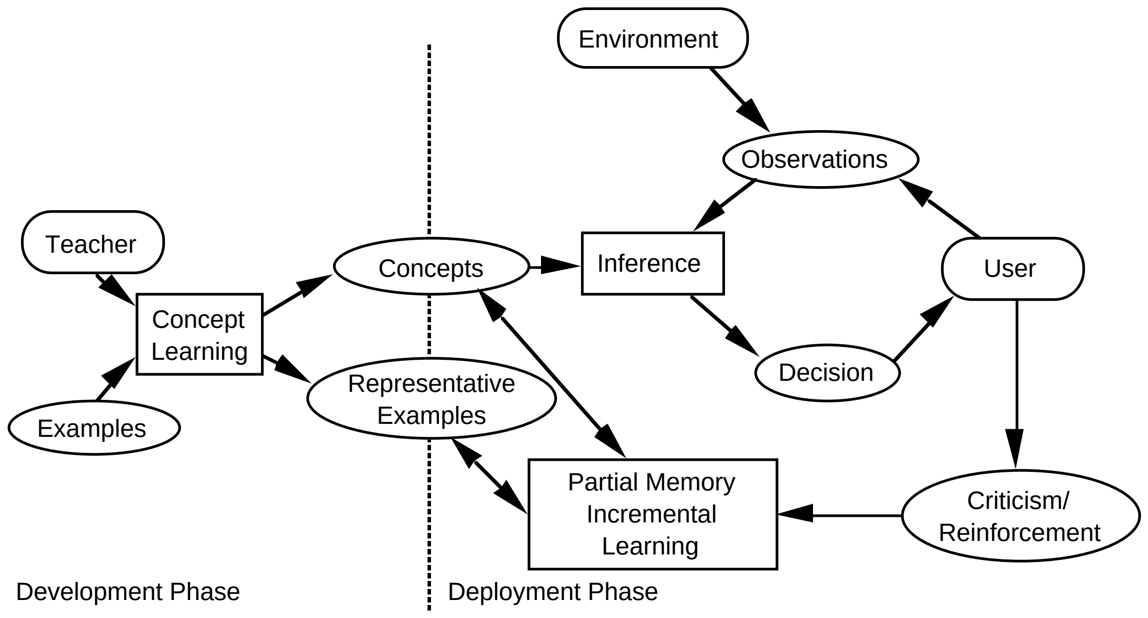

Figure 3.2: Partial memory incremental learning architecture and methodology.

especially as is described by Reinke and Michalski (1988), would be difficult, if not impossible to implement in neural networks. Since neural network learning involves converging to a global minimum, learning incrementally changing concepts is also difficult. Because of this required convergence, learning incrementally changing concepts can only be accomplished through batch learning, which can require long training periods.

Several issues must be addressed when applying a partial memory incremental learning methodology to a problem. The first concerns how representative examples are chosen. The second involves how these representative examples are maintained throughout the incremental learning process. Regarding the first issue of how representative examples are chosen, for this study maximally general examples were used that either expanded or constrained concepts in the representation space. Specifically, the attribute values for the attributes appearing in learned rules were retained, while other attribute values were changed to unknown or don't care values. The set of representative examples was the set union of these maximally general examples, to eliminate redundancies. For instance, the training examples

```
daffy-event
avgcpu mincpu maxcpu avgblks minblks maxblks
12     6      14     50      8       75

coyote-event
avgcpu mincpu maxcpu avgblks minblks maxblks
34     32     36     30      20      50
```

might yield the decision rules

```
daffy-rule
[avgblks = 48..52]

coyote-rule
[mincpu = 30..33]
```

in which case, the original training examples would be translated into the following representative examples:

```
daffy-event
avgcpu mincpu maxcpu avgblks minblks maxblks
?      6      ?      50      ?       ?

coyote-event
avgcpu mincpu maxcpu avgblks minblks maxblks
?      32     ?      30      ?       ?
```

The second issue involves how representative examples are maintained throughout the incremental learning phase. This is an important consideration since examples are generalized and what is considered representative early in incremental learning may not remain representative throughout the incremental learning phase. Consequently, attributes whose values are discarded early may prove to be better in later stages. On the other hand, fewer generalized examples are likely to be kept than specialized examples. One method is to eliminate old representative examples as what is considered 'representative' changes. Another method, the method used here, is to keep all past representative examples regardless of how the representative examples change. The first method would probably be suitable for rapidly changing domains, whereas the second method would yield better results in more stable domains.

## Application to Intrusion Detection

The proposed incremental learning architecture and methodology was applied to the problem of computer system intrusion detection. Typically, an intruder masquerades as one of the legitimate system users. If machine learning could be used to learn *use patterns* for the users of the computer system, then these patterns could be used to detect intruders. This research is most similar to the work of Teng et al. (1990) in the sense that we are inductively learning symbolic rules. It differs in that we do not learn sequences of actions.

In a traditional concept learning scenario, we divide training examples into classes, express the training examples in a representation space that facilitates learning and assert that the examples themselves are sufficient for learning the intended concept. For this application, we might be tempted to divide patterns into classes relating to a "legitimate user" and "intruder", but collecting examples of an intruder's behavior would be a difficult task, since intrusions are a relatively infrequent events. Rather, we should learn use patterns for classes of users or for individual users on the system. In all legitimate uses, the user's login name should match the decision given by the system. If the system's decision does not match the user's login name, then the user is possibly an intruder and appropriate security actions can be taken. These would include making a entry in a systems log file or even forcing the suspect user off the system.

Because of AQ's flexible matching algorithm, the intrusion detection system not only produces a decision or classification (i.e., a user's identity), but it also provides a measure of certainty using the degree of match. The degree of match functions similarly to the abnormality measure in NIDES (Javitz and Valdes 1994). For example, assume we have two users, daffy and coyote, and that daffy's session has been logged and sent to the system for interpretation. Three decisions are possible:

1. the user is daffy (i.e., daffy's degree of match is higher than coyote's),
2. the user is coyote (i.e., coyote's degree of match is higher than daffy's), or
3. the user could be either daffy or coyote (i.e., the degree of match for both users is equal).

With the first decision, the system would assume that everything is normal. However, we can look at the relative difference between the degrees of match to infer additional information. If for example with the first case, daffy's degree of match is 1.0, while coyote's is 0.0, then perhaps no additional action is needed. On the other hand, if the relative distance between the degrees of match were smaller, then perhaps this incident should be logged. The thresholds to take various actions in response to close matches or close misses can be parameters set by the system administrator.

The second decision indicates either a problem with the learned concepts or with the person using daffy's account. With the former case, the system administrator would give feedback to the system regarding the misclassification, at which time, an incremental learning process would begin to modify the system's existing concepts and representative training examples. With the latter case, the system administrator could set the system to log the activity or to autonomously take action. Again, the relative distance of the degrees of match can give the administrator guidance on how to proceed. In this case, the greater the distance between the degrees of match the greater the possibility of an intruder masquerading as daffy.

The third decision represents a plausible situation in which the degrees of match for the two users are equivalent. More than likely, this would be a cue to the system administrator to invoke the incremental learning process to update existing concepts and representative examples by providing criticism to the system.

## 4 Experimental Results

A series of experiments were conducted using Unix *acctcom* audit data and three learning methods: AQ15c, a feed-forward neural network, and *k*-nn. Accounting data was collecting for a period of three weeks yielding over 11,200 audit records. The first set of experiments involved batch learning of use patterns for nine of the system's active users. Performance comparisons were made between AQ15c, a feed-forward neural network, and *k*-nn. The second set of experiments involved partial memory incremental learning from two of the system's active users. Only two users were selected, since the bulk of the experimentation was carried out manually. Performance comparisons are made between AQ15c using batch learning and partial memory incremental learning.

### 4.1 Data Preparation

The first task involved extracting training examples for each user. A *session* is defined as a contiguous period of activity bounded by a gap in activity of 20 minutes or more. This includes idle time and logouts. Audit data produced by the Unix *acctcom* command was parsed into sessions by user. Attributes were then computed from the various data fields in the audit file. Several different representation spaces were computed from the audit data using the following schemes:

1. in a time frequency domain with and without log function scaling (e.g., number of characters transferred per minute),
2. in a command frequency domain with and without log function scaling (e.g., number of characters transferred per command),
3. in both time and command frequency domains with and without log function scaling,
4. using average, minimum and maximum values computed over a session.

The first six representation spaces (items 1–3) did not produce satisfactory predictive accuracies with any of the selected learning algorithms. The final representation space (item 4) produced the best results, which are presented in this paper.

Essentially, each numeric metric in an audit file is a time series. Davis (1981) characterized time series data for symbolic learning by taking the minimum, maximum, and average values of a time series over a window. For this application, a session is a window of activity. Figure 4.1 illustrates how a time series is parsed into sessions and how attributes are extracted. Average, maximum, and minimum computations were made for seven metrics in the audit file: the real time, CPU time, user time, characters transferred, blocks read and written, the CPU factor, and the hog factor. Consequently, each training example, which was derived from a single user session, consisted of 21 continuous or real-valued attributes. All total, there were 239 training examples distributed over 9 classes, which correspond to the 9 selected users.

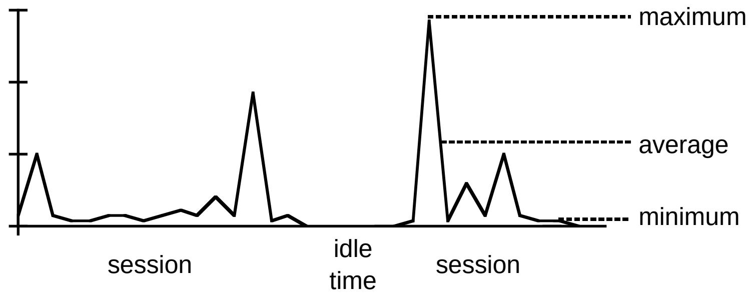

Figure 4.1: Session parsing and attribute extraction from audit data.

Training data for the feed-forward neural network and AQ15c required scaling. For the feed-forward neural network, the training data was scaled using a uniform interval method which involved mapping the real range of each attribute into the range [0, 1].

AQ15c requires discrete-valued attributes, so the SCALE implementation (Bloedorn et al. 1993) of the ChiMerge algorithm (Kerber 1992) was used. The ChiMerge algorithm merges real-valued attributes into discrete intervals using the chi-square statistic to correlate intervals to classes. For example, the `minchar` attribute, which is the minimum number of characters transferred during a session, ranged from 0.0 to 18747.28. ChiMerge determined that only seven discrete levels were needed for this attribute. Table 4.1 shows the ranges ChiMerge chose for the seven intervals. Notice how the ranges pertaining to the discrete attributes are not uniform; for instance, compare interval 1 with interval 4. After ChiMerge scaling, attribute levels for the AQ15c training data ranged between 3 and 77.

| Interval | Range |
|---|---|
| 0 | 0.00…8.99 |
| 1 | 9.00…26.99 |
| 2 | 27.00…63.99 |
| 3 | 64.00…74.99 |
| 4 | 75.00…418.99 |
| 5 | 419.00…1029.99 |
| 6 | 1030.00…18747.28 |

Table 4.1: ChiMerge scaling for the `minchar` attribute.

## 4.2 Batch Learning Experimental Method

Our experimental method for the three learning methods was as follows. After scaling the original training data, it was split into a training set (66%) and a testing set (34%). Several learning runs were conducted to determine acceptable learning parameters for each of the learning methods. Once these parameters were determined, 100 runs were conducted using a 2-fold cross validation methodology, as prescribed by Weiss and Kulikowski (1992). Each run involved splitting the original training data evenly into a training set and a testing set. The training set was given to a learning algorithm and a concept was learned. The learned concept was then tested using the testing set and a predictive accuracy was computed. The predictive accuracy is the percentage of correctly classified testing examples. This process was repeated 100 times. For each 100 run experiment, four metrics were computed: the average predictive accuracy, the best single performance, the average learning time, and the average recognition time.

The average predictive accuracy is the average of all predictive accuracies over the 100 learning and recognition runs. The best single performance is the highest predictive accuracy the learning algorithm achieved during the 100 run experiment. The average learning time is the average amount of CPU time the learning method spent learning during the 100 run experiment. The average recognition time is the average amount of CPU time required to recognize the testing examples using learned concepts during the 100 run experiment. Sun SPARC 2s were used for all experiments. Table 4.2 summaries the experimental results for the three learning methods. Finally, learning curves were computed for each learning method, which demonstrates how well the learning methods performed on varying amounts of training data (see Figure 4.2).

### Batch Learning Experimental Results

### *k*-nn Results

*k*-nn has historically been considered a statistical pattern recognition technique (Weiss and Kulikowski 1992). In the machine learning community it is referred to as a memory-based learning technique — although, some have labeled it as a neural network model (*DARPA Neural Network Study* 1988). Regardless of its labeling, *k*-nn relies on a collection of classified training examples. When classifying an unknown, *k*-nn computes the distance between the unknown and the training examples using either an absolute or Euclidean distance measure. For $k = 1$, the classification of the closest training example to the unknown is used as the classification for the unknown. For $k > 1$, a voting procedure occurs and the class with the most closest neighbors to the unknown is assigned as the class for the unknown. Typically $k$ is odd to prevent ties.

Experimental runs were conducted using $k$ = 1, 3, 5, 7, 9, 11, 13, and 15 using a Euclidean distance measure. $k$ = 1 produced the best predictive accuracy. For 100 runs, $k$-nn achieved an average predictive accuracy of 83% with a best performance of 88%. Average learning and recognition times were 0.7 seconds and 1.43 seconds, respectively.

## Feed-Forward Neural Network Results

An artificial neural network (ANN) is a computational model inspired by the neuronal architecture of the human brain. The class of multi-layer feed-forward networks are capable of learning statistical regularities inherent in training data. They are considered non-symbolic machine learning models because the concepts that neural networks induce are represented in a non-literal fashion as a collection of real-valued weights distributed throughout the network's connections. The model used here was trained with the Quickprop algorithm (Fahlman 1986), which is a fast variant of the back-propagation algorithm (Rumelhart et al. 1986).

The network architecture consisted of 21 continuous inputs, one for each attribute, 15 hidden units, and 9 linear output units, one for each class. The learning rate and momentum were set at 0.4 and 0.0, respectively, and control how fast the learning algorithm converges to a potential solution. The error constraint was set at 0.2, but the net was allowed to train for a maximum number of 2000 epochs. Note that the constraint error may seem high, but the best performing net did so at a constraint error of 0.6. Often it is not best to allow a network to train to a low, tight constraint, because better generalization occurs at a higher network error value. Of course, finding this point during training is difficult, although we have some ideas about a network training methodology that would address this problem, especially when running numerous repeated learning experiments. This is discussed in future work.

The average predictive accuracy the neural network achieved for a 100 run experiment was 85%. The best predictive accuracy during experiment was 94%. Average CPU time for network learning was 580 seconds, although recognition took on average 0.17 seconds.

## AQ15c Results

AQ15c (Wnek 1994) is the latest implementation of the AQ algorithm (Michalski 1969). The AQ algorithm heuristically solves the set covering problem to learn a symbolic concept expressed in $VL_1$ using both logical and statistical information. The best performing parameters for AQ were as follows. Maxstar was set at 10. Maxstar is a parameter that determines how many intermediate solutions AQ maintains during concept formation. AQ was set to generate intersecting covers, meaning that covers or rules in the representation space are not simply disjoint, but may overlap or intersect. General rules were generated, as opposed to specific rules. Finally, any ambiguous examples (i.e., duplicate examples) were regarded as positive, although this parameter did not affect predictive accuracy during early parameter tuning.

After parameters were determined, an experiment consisting of 100 learning and recognition runs was conducted, as described previously. AQ achieved an average predictive accuracy of 88% with a best predictive accuracy of 96%. The average CPU time AQ spent learning during this experiment was 68.8 seconds, while recognition took, on average, 0.3 seconds. Table 4.2 summaries these results and compares them to the results from the other learning methods. Figure 4.2 shows the learning curve for AQ and the other methods.

| Learning Method | Average Predictive Accuracy (%) | Best Predictive Accuracy (%) | Average Learning Time (seconds) | Average Recognition Time (seconds) |
|---|---|---|---|---|
| *k*-nn | 83 | 89 | 0.7 | 1.43 |
| ANN | 85 | 94 | 580.0 | 0.17 |
| AQ15c | 88 | 96 | 68.8 | 0.3 |

Table 4.2: Comparative summary of results.

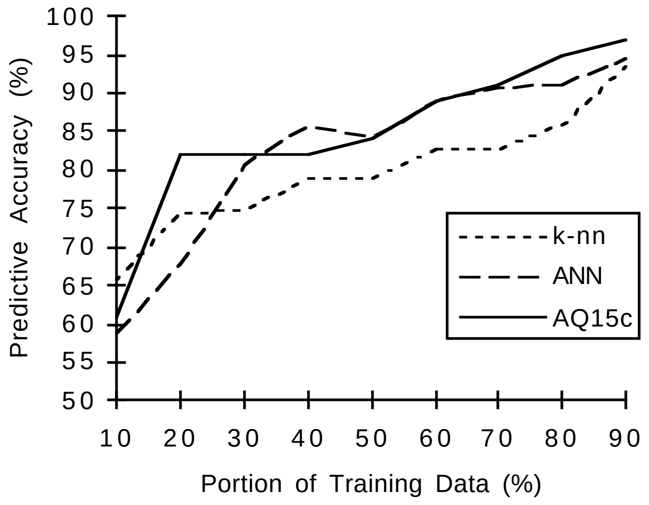

Figure 4.2: Learning curves for each learning method.

## Incremental Learning Experimental Method

Two classes were selected from the original training data extracted for the batch learning experiments. The training data for these classes were partitioned into 10 sets and used for partial memory incremental learning experiments, which were compared to AQ15c batch learning. The batch learning experiment involved accumulating the training examples from the 10 data partitions for learning. Partial memory incremental learning was carried out using the following algorithm:

```
Given data partitions $DATA_i$, for $i = 1..10$
0. $i = 1$
1. $TRAIN_i = DATA_i$
2. $CONCEPTS_i = Learn(TRAIN_i)$
3. $REPRESENTATIVE_i = FindRepresentativeExamples(CONCEPTS_i, TRAIN_i)$
4. $MISSED_i = FindMissedNewExamples(CONCEPTS_i, DATA_{i+1})$
5. $TRAIN_{i+1} = REPRESENTATIVE_i \cup MISSED_i$
6. $i = i + 1$
7. go to step 2
```

The counter *i* represents a temporal counter that signifies the passage of time in the system's environment. When $i = 1$, steps 1–3 relate to the development phase of Figure 3.2. The remaining steps of the algorithm for $i = 1$ and under different values of *i* relate to the deployment phase. Representative examples are found in step 3 using the method described previously. In step 4, missed examples are found by testing new observations, which are represented by $\text{DATA}_{i+1}$, using the current set of learned concepts $\text{CONCEPTS}_i$. Those missed examples are identified by a user and are communicated to the system as criticism. No feedback is viewed as reinforcement. The set of representative examples and the set of misclassified examples are given back to the learning algorithm that incrementally learns new concepts. For these experiments, this algorithm was executed manually for each of the 10 data partitions.

## Incremental Learning Experimental Results

Several experimental comparisons were made between AQ15c batch learning and partial memory incremental learning using a variety of metrics. Figure 4.3 illustrates how AQ15c's predictive accuracy varies with respect to the portion of training data under batch and partial memory incremental learning. Although the difference in predictive accuracy is large early in the learning process, toward the end of learning, predictive accuracy differs by only 2%.

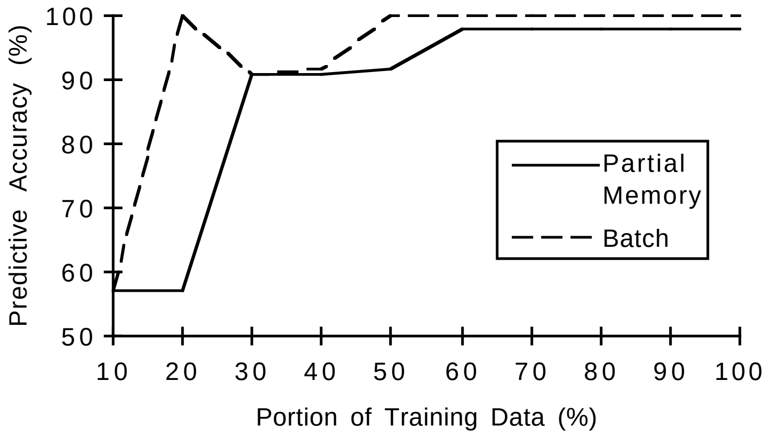

Figure 4.3: Learning curves for partial memory incremental learning.

Figure 4.4 provides a learning time comparison between AQ15c batch and partial memory incremental learning. After the first learning step, incremental learning time was consistently less than 0.1 CPU seconds.

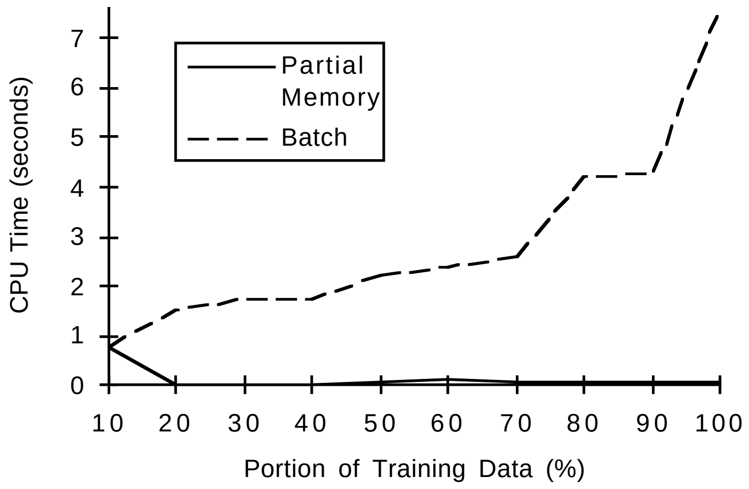

Figure 4.4: Learning times for partial memory incremental learning.

Figure 4.5 demonstrates the number of examples each approach, partial memory and batch learning, required. Although batch learning required a linearly increasing quantity examples produced only a moderate increase in predictive accuracy of 2% over partial memory incremental learning.

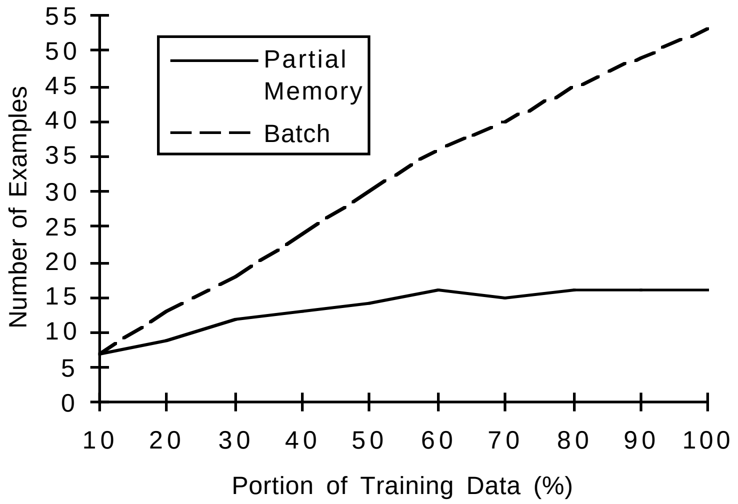

Figure 4.5: Number of examples for partial memory incremental learning.

The final comparison, illustrated by Figure 4.6, shows how batch learning and partial memory incremental learning compare with respect to rule complexity or the number of conditions in the learned concepts. Batch learning produced less complex rules than partial memory learning, but this is possibly an artifact of how partial memory learning was carried out, which was manually. There are probably ways to optimize rules based on the representative examples that will yield simpler rules. This notion will be investigated during implementation.

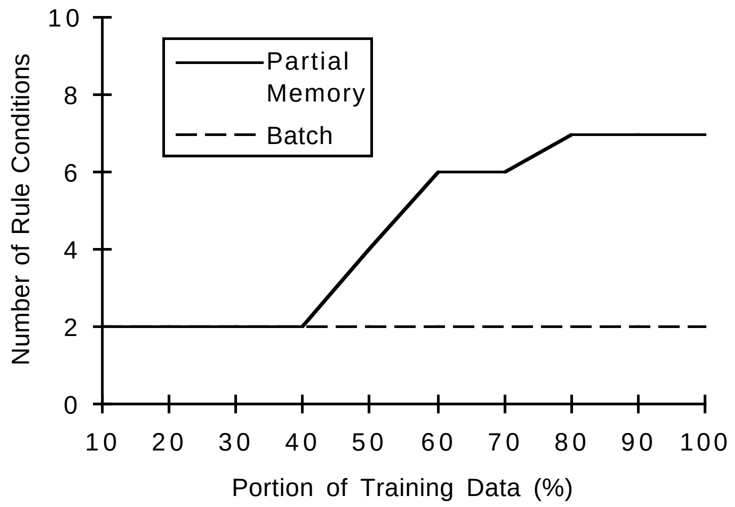

Figure 4.6: Rule complexity for partial memory incremental learning.

Note that these results are consistent with the full memory incremental learning results reported by Reinke and Michalski (1988). Future work will compare partial memory incremental learning with AQ15 full memory incremental learning.

## 5 Discussion and Future Work

Batch learning experimental results suggest that at least in the intrusion detection domain considered here, AQ15c performed better than a feed-forward neural network and *k*-nn. *k*-nn's fast learning time cannot mitigate its poor predictive accuracy. And although the neural network had faster recognition times, AQ rules can serve as the basis for a neural network architecture that will achieve equally fast recognition times (Bala et al. 1994).

There are other issues to be considered, however, especially for systems that must adapt to a changing environment. The first issue relates to the types of information that can be learned, which include continuous, linear, symbolic, and structured data. The second involves the ability of the system to incrementally learn, which is necessary for adaptive systems.

While neural networks are well-suited for continuous and linear data, symbolic and structured data must be coded into a non-literal representation, such as binary, bipolar, or continuous. While this is not an insurmountable problem, others exist that put neural networks at a severe disadvantage and stem from how neural networks represent learned concepts. Concepts in a neural network are represented in a non-symbolic fashion; specifically, concepts are represented as real-valued weights distributed throughout the network's connections. Consequently, learned concepts are not available for direct human inspection, which is crucially important for certain applications, and consequently makes incremental learning difficult, if not impossible.

The problem with *k*-nn is in how it recognizes unknown observations. It does so using either an absolute or an Euclidean distance measure, consequently, using *k*-nn on data that is not either linear or continuous creates a few philosophical conundrums. Nominally valued attributes (i.e., symbolic attributes) having only two values can be handled by mapping the values to 0 or 1, but for nominally valued attributes having three or more values, we again get into some problems. For instance, almost anyone would agree that a pick-up truck and a sport-utility vehicle are closer to each other than each is to a sports car, but how do we develop a function to measure this closeness based on inherent characteristics? What we need, unfortunately, is a domain-specific semantic distance measure — something that is elusive if not impossible to specify.

AQ15c, in contrast to neural networks and *k*-nn, is able to naturally handle a variety of data types and because learned concepts can be manipulated directly, partial memory incremental learning is possible. Figure 5.1 shows two AQ15c rules induced for users daffy and elmer. These rules were taken from the best performing rule set that achieved a predictive accuracy of 96% on testing data. The rule that characterizes daffy's computer use is simple, consisting of one complex and one condition. The rule says that if the maximum real time for a user is equal to 16, then this user is daffy. Note that the value 16 represents a range of real time values because of ChiMerge scaling. The t-weight and u-weight describe the strength of the complex. The t-weight indicates the total number of training examples this rule covers, while the u-weight indicates how many of those examples are unique (i.e., not covered by other complexes).

```
daffy-rule
#    complex
1    [maxreal=16]   (t-weight:16, u-weight:16)

elmer-rule
#    complex
1    [maxsys=8..18] & [maxchar=10..20]   (t-weight:7, u-weight:2)
2    [minreal=0..3] & [maxreal=0..11] & [avgblks=14..48] & [avgcpu=1..26] &
     [avghog=13..35]   (t-weight:6, u-weight:1)
```

Figure 5.1: AQ15c rules for daffy and elmer.

The rule for elmer's use, on the other hand, is more complex, consisting of 2 complexes and a total of 7 conditions. The first complex, for example, covers a total of 7 training examples, but only two of those examples are unique. The second complex covers a total of 6 examples, but only one of these is unique. This implies that there is great overlap among these two complexes. Referring again to the first complex of elmer's rule, in order for this complex to be true, under strict matching conventions, the maximum system time must be inclusively between 8 and 18, and the maximum number of characters transferred must be inclusively between 10 and 20. Again note that these discrete intervals relate to much larger real ranges, but were abstracted by the ChiMerge algorithm. If these two conditions are true, then the symbol "elmer" will be assigned as the decision class. Otherwise, the second complex will be tested in much the same manner.

Experimental results suggest that partial memory incremental learning is beneficial. Although these experiments demonstrate that partial memory incremental learning resulted in slightly higher rule complexity and slightly lower predictive accuracy than batch learning, significant decreases were seen in the CPU time spent learning and in the number of examples maintained over time. Future work will be on the implementation of these ideas, which will allow larger experiments to determine how this approach scales up to more complex problems. Comparisons will also be made to AQ15 full memory incremental learning described by Reinke and Michalski (1988).

One difficulty with neural network training is that it is not a deterministic process. Networks cannot be optimally trained simply by iterating for some fixed number of epochs or until they reach some error constraint. This difficulty arises especially when running numerous learning and recognition runs, as was done in this paper. Presently, we stop training when the error constraint or a maximum number of epochs is reached. Perhaps a better network training methodology would be to start accumulating testing results when network stops making bit errors, although the error could still be quite high. If using a binary or linear representation, a bit error occurs when an actual neuron output does not match its desired output. During the remainder of the training cycle in which the network error descends to the error constraint, the best predictive accuracy will be saved. If necessary, the network's weights can be saved as well. At the end of the training cycle, when the network error has reached the error constraint, we will have saved the best performing network. This training methodology will find the best generalizing network and is immune to over-training, although it does increase overall training time.

To evaluate this work as an approach to intrusion detection, notice that the architectures of the systems surveyed in Section 2 require modules to fuse decisions made by the statistical or neural network component with decisions made by the rule-based components. By virtue of learning symbolic descriptions, any expert-elicited domain knowledge can be represented using the same representational formalism used to represent symbolic concepts. Consequently, there is no representational distinction between the learned anomaly detection knowledge and expert-provided knowledge. And hence, no need for special modules to fuse decisions from two different reasoning systems.

Clearly, more behavioral factors need to be included into this system to increase viability and predictive accuracy. Future work will involve incorporating symbolic data (e.g., terminal names, command names) and structured data (e.g., command hierarchies), and if temporal trends exist (e.g., daffy always works in the afternoon and evening, but not in the morning), investigating how they be exploited to yield higher predictive accuracy.

The incremental learning ideas expressed in this report will serve as the foundation for an implementation, which will allow better opportunities for experimentation. Directions with regards to an implementation include optimization of incrementally learned rules and investigating Baconian support mechanisms (Cohen 1977) for rule conflict resolution. Baconian support mechanisms should be investigated because during inference, ties often result between two or more classes when the degrees of match for the decision classes are equal. In this situation, AQ employs an ad hoc mechanism to select a decision class. Baconian mechanisms, or some other evidential reasoning mechanism, could be used to resolve conflicts between ties. As the system interacts with its environment, accumulated support based on the number of observations, or the number of examples used to induce various concepts, would be used to select a decision class in the event of a tie. Concept refinement operators should also be investigated.

## 6 Conclusions

This paper presents work in progress and introduces a partial memory incremental learning methodology, which is applied to the problem of computer intrusion detection. Such an incremental learning architecture and methodology is important for applications involving intelligent agents, active vision, and computer intrusion detection because of the need for the system to interact with users and the environment and adapt its behavior over time. Experimental comparisons suggest that AQ15c has advantages over feed-forward neural networks and *k*-nn in terms of predictive accuracy and learning and recognition times. Further arguments can be made to support AQ15c on grounds of the types of data from which the method learns and the types of concepts learned by the method. For the purposes of partial memory incremental learning, AQ15c also proved superior because of the need for symbolically learned concepts in judging representative examples. Partial memory learning yielded significant improvements over batch learning in terms of the number of examples maintained and learning time at the cost of slightly lower predictive accuracy and higher rule complexity. Computer implementation of the ideas presented in this paper will allow more extensive experimentation.

## References

Anderson, D.; Frivold, T.; Tamaru, A.; and Valdes, A. (1994a) *Next Generation Intrusion Detection Expert System (NIDES): software design, product specification, and version description document.* SRI International Software Design Document A002 Version Description Document A005. SRI International, Menlo Park, CA.

Anderson, D.; Frivold, T.; and Valdes, A. (1994b) *Next Generation Intrusion Detection Expert System (NIDES): final technical report*. SRI International Final Technical Report A008. SRI International, Menlo Park, CA.

Bala, J. W.; Michalski, R. S.; and Pachowicz, P. W. (1994) Progress on vision through learning at George Mason University. *Proceedings of the 1994 Image Understanding Workshop*, 191–207.

Ballard, D., and Brown, C. (1993) Principles of animate vision. In Aloimonos, Y., ed., Active Perception. 245–282. Hillsdale, NJ: Lawrence Erlbaum Associates.

Bhandaru, M. K., and Murty, M. N. (1991) Incremental learning from examples using HC-expressions. *Pattern Recognition* 24.4:273–282.

Bloedorn, E.; Wnek, J.; Michalski, R. S.; and Kaufman, K. (1993) AQ17 — A multistrategy learning system: the method and user’s guide. *Reports of the Machine Learning and Inference Laboratory*, MLI 93–12. Center for Machine Learning and Inference, George Mason University, Fairfax, VA.

Cohen, L. J. (1977) *The probable and provable*. Oxford: Oxford University Press.

*DARPA Neural Network Study* (1988) Fairfax, VA: AFCEA International Press.

Davis, J. H. (1981) CONVART: a program for constructive induction on time dependent data. Master’s Thesis. Department of Computer Science, University of Illinois at Urbana-Champaign, Urbana, IL.

Debar, H.; Becker, M.; and Siboni, D. (1992) A neural network component for an intrusion detection system. *Proceedings of the 1992 IEEE Computer Society Symposium on Research in Security and Privacy*, 240–250.

Denning, D. E. (1987) An intrusion-detection model. *IEEE Transactions on Software Engineering*. SE-13.2 (February) 222–232.

Fahlman, S. E. (1988) *An empirical study of learning speed in back-propagation networks*. Technical Report CMU-CS-88-182. Department of Computer Science, Carnegie-Mellon University, Pittsburg, PA.

Hong, J.; Mozetic, I.; and Michalski, R. S. (1986) AQ15: incremental learning of attribute-based descriptions from examples, the method and user's guide. *UIUCDCS-F-86-949*. Department of Computer Science, University of Illinois at Urbana-Champaign, Urbana, IL.

Javitz, H. S., and Valdes, A. (1994) *The NIDES Statistical Component: description and justification*. SRI International Annual Report A010. SRI International, Menlo Park, CA.

Kerber, R. (1992) ChiMerge: discretization of numeric attributes. *AAAI-92*. 123–128.

Lunt, T. F.; Jagannathan, R.; Lee, R.; Whitehurst, A.; and Listgarten, S. (1989) Knowledge-based intrusion detection. *Proceedings of the Annual Artificial Intelligence Systems in Government Conference*, 102–107.

Maes, P. (1994) Agents that reduce work and information overload. *Communications of the ACM* 37.7 (July) 31–40.

Michalski, R. S. (1969) On the quasi-minimal solution of the general covering problem. *Fifth International Symposium on Information Processing*, A3:125–128.

Michalski, R. S. (1972) A variable-valued logic system as applied to picture description and recognition. *IFIP Working Conference on Graphic Languages*, 21–47.

Michalski, R. S. (1973) AQVAL/1 — computer implementation of a variable-valued logic system $VL_1$ and examples of its application to pattern recognition. *First International Joint Conference on Pattern Recognition*, 3–17.

Michalski, R. S. (1980) Pattern recognition as rule-guided inductive inference. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 2.4:349–361.

Mukherjee, B.; Heberlein, L. T.; and Levitt, K. N. (1994) Network intrusion detection. *IEEE Network* 8.3 (May-June) 26–41.

Reinke, R. E., and Michalski, R. S. (1988) Incremental learning of concept descriptions: a method and experimental results. In Hayes, J. E.; Michie, D.; and Richards, J., eds., *Machine Intelligence 11*, 263–288. Oxford: Clarendon Press.

Rumelhart, D. E.; Hinton, G. E.; and Williams, R. J. (1986) Learning internal representations by error propagation. In Rumelhart, D. E., and McClelland, J. L., eds., *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, Vol. 1, 318–362. Cambridge MA: MIT Press.

Smaha, S. E. (1988) Haystack: an intrusion detection system. *Proceedings of the Fourth Aerospace Computer Security Applications Conference*, 37–44.

Teng, H. S.; Chen, K.; and Lu, S. C-Y. (1990) Security audit trail analysis using inductively generated predictive rules. *Proceedings of the Sixth Conference on Artificial Intelligence Applications*, 24–29.

Vaccaro, H. S. (1989) Detection of anomalous computer session activity. *Proceedings of the 1989 IEEE Symposium on Research in Security and Privacy*, 280–289.

Weiss, S. M. and Kulikowski, C. A. (1992) *Computer systems that learn: classification and prediction methods from statistics, neural nets, machine learning and expert systems*. San Mateo, CA: Morgan Kaufmann.

Wnek, J. (1994) A fast implementation of the AQ-based inductive learning program for large datasets: AQ15c. *Reports of the Machine Learning and Inference Laboratory*, MLI 94–3. Center for Machine Learning and Inference, George Mason University, Fairfax, VA (to appear).

Zurada, J. M. (1992) *Introduction to artificial neural systems*. St. Paul, MN: West Publishing.