Machine Learning in Molecular Biology Sequence Analysis

Philip K. Chan

CUCS-041-91

Department of Computer Science
Columbia University
New York, NY 10027
pkc@cs.columbia.edu
(212) 854-8100

November 27, 1991

Abstract

To investigate how human characteristics are inherited, molecular biologists have been analyzing chemical sequences from DNA, RNA, and proteins. To facilitate this process, sequence analysis knowledge has been encoded in computer programs. However, translating human knowledge to programs is known to be problematic. Machine Learning techniques allow these systems to be generated automatically. This article discusses the application of learning techniques to various analysis tasks. It is shown that the learned systems constructed to date are often more accurate than human-designed systems. Moreover, learning can form plausible new hypotheses, which potentially lead to discovering new knowledge.

1 Introduction

To fully understand how human characteristics are inherited, molecular biologists have been investigating the human cell nuclei at the molecular level. In particular, researchers have been studying the structure and functions of DNA and protein molecules, and their interactions. It is known that DNA carries genetic information and this information dictates the production of proteins, the very building blocks of life. However, we do not know the functions of most of the DNA or protein segments. With the advancement of technology, scientists have been able to map DNA and proteins to their building blocks: nucleotides and amino acids. From this structural information, more functional information can be discovered. Since the building blocks are linked in a sequence, this process is commonly known as *sequence analysis*.

Various computer systems have been built to facilitate the process of analyzing sequences (von Heijne, 1987; Bishop and Rawlings, 1987). However, most of the systems require translating analysis techniques developed by humans to programs. It is well known that this process, *knowledge engineering*, can be lengthy and problematic (Buchanan et al., 1983; Boose, 1986).

Machine learning is an artificial intelligence technique which allows systems to be generated automatically by finding patterns and causal relationships in the provided data. That is, it is theoretically possible that sequence-analysis systems can be built automatically and directly from exemplar sequence information without obtaining and translating human expertise. Furthermore, machine learning techniques allow the possibility of discovering patterns and concepts unknown to the experts. As we will see later, some of these systems generated by learning techniques outperform *human-designed* systems.

This paper focuses on the application of machine learning techniques to generating sequence-analysis systems. Descriptions of human-designed systems can be found elsewhere (e.g., (von Heijne, 1987)). For each learning application, the approach is identified and details of the algorithm used are then discussed. Experiments run on the systems are described and their performance results are presented. If there is more than one learning application for a sequence analysis task, results from different systems, including some human-designed systems, are compared. One important theme to keep in mind is the inevitable explosion of new stores of information that will require efficient implementations to process massive amounts of data. Much of the existing work reported here for the most part is not concerned with issues of computational efficiency.

The remainder of this paper is organized as follows. We first overview the two main areas of sequence analysis. We then overview the different approaches of machine learning. Section 4 discusses learning applications in amino acid sequence analysis while Section 5 describes nucleotide sequence analysis. We conclude with a discussion of the importance and implications of the results achieved to date.

2 Sequence Analysis

Molecular biologists have been focusing on analyzing sequences obtained from proteins, DNA (DeoxyriboNucleic Acid), and RNA strands (RiboNucleic Acid). These sequences are divided into two groups: amino acid sequences and nucleotide sequences. These two groups are briefly discussed in the following two subsections.

2.1 Amino Acid Sequence

Proteins are fundamental and instrumental in every aspect of human biological function even though they are quite chemically simple in structure. Each protein is a sequence of amino acids linked together in a linear chain. There are a total of twenty different amino acids. A protein segment can thus be represented as a sequence of symbols, where each symbol signifies a distinct amino acid. For example, PIVDTGSVAP is a segment of ten amino acids in Haemoglobin V (Qian and Sejnowski, 1988). The order of amino acids determines the 3-D shape of a protein, which largely determines the protein's function. (Imagine a rope with a series of knots, each knot representing an amino acid. Our imaginary protein rope can be twisted and folded into a globule exposing some knots externally while hiding others internally. The exposed external knots largely determine the protein's biochemical behavior.) The number of distinct proteins, however, is essentially unbounded. The purpose of analyzing amino acid sequences is to gain information about proteins both structurally and functionally.

Consider, for example, amino acids in a protein interacting with each other. These interactions induce regular secondary structures, which are important in determining the function of the protein (more details are provided in Section 4.1). Scientists have been trying to find ways to predict the secondary structures of particular amino acid sequences so that they can learn more about the functional properties of proteins.

2.2 Nucleotide Sequence

The basic building blocks of human genetics are nucleotides. There are four different kinds of nucleotides in DNA (adenine, cytosine, guanine, and thymine) and RNA (adenine, cytosine, guanine, and uracil). That is, a DNA or RNA segment can be represented as a sequence of symbols, where each symbol denotes one of the four nucleotides. For example, GGGACG-GUCC is a segment of ten nucleotides of the U1 RNA (Nakata et al., 1985). DNA is double stranded in double helix form whereas RNA is single stranded.

Human characteristics and functions are controlled by proteins, whose production is regulated by the information encoded in the nucleotide sequences of DNA in the cell nucleus. This genetic information is basically the order of nucleotides in the DNA. Proteins are not directly produced from the information on our DNA. Instead, information on DNA segments, *genes*, is copied to another type of nucleotide sequence, RNA, whose nucleotide order is used to produce proteins. As in analyzing amino acid sequences, we can gain more structural and functional information about DNA and RNA from studying their nucleotide sequences.

For example, the decoding process of producing a protein always starts at a certain location in an RNA segment called the *promoter site* (more details in are provided Section 5.1). Molecular biologists try to identify the initiation region in a given RNA sequence so that they can understand more about the interactions between RNA and protein production.

3 Machine Learning

The goal of machine learning is to make a system "improve" by itself. Improvements generally fall into two categories: gaining knowledge and enhancing computational efficiency. More specifically, learning includes the tasks of forming concepts by generalizing data, compiling knowledge into a compact form for efficient execution and access, finding useful explanations for valid concepts, clustering data to form new classes, and many others.

Machine learning can be roughly divided into four paradigms (Carbonell, 1989):

- *Inductive learning* concerns forming concepts from data without a lot of knowledge from the domain (e.g., *learning from examples* (Michalski, 1983) and *conceptual clustering* (Michalski and Stepp, 1983)).

- *Analytic learning* involves the use of existing knowledge to derive new useful concepts (e.g., *explanation-based learning* (Mitchell et al., 1986) and certain forms of *analogical* (Carbonell, 1983) and *case-based* (Slade, 1991) learning methods).

- *Connectionist learning methods* use artificial neural networks to search for and represent concepts (Rumelhart and McClelland, 1986; Hinton, 1989).

- *Genetic algorithms* utilize the *Darwinism* metaphor, "survival of the fittest," to search for the most effective concept (Holland, 1975; Booker et al., 1989).

It is important to note that the first two paradigms characterize the nature of learning tasks whereas the latter two represent two particular methods to perform different learning tasks. The first two paradigms are commonly referred to as symbolic approaches, which generally exclude the connectionist and genetic algorithm approaches.

Inductive, connectionist, and genetic learning methods are typically used in tasks that are data-oriented and provide little or no knowledge of the domain. These methods primarily concentrate on finding similarities and differences among data. On the other hand, analytic learning methods rely on the presence of background knowledge to infer or derive new knowledge. That is, analytic methods are suitable for tasks that need to build knowledge from existing knowledge. In addition, due to the nature of data representation in connectionist methods, they can conveniently represent numeric/binary data. However, inductive and genetic methods are more suitable for data abstracted as symbolic or nominal features. In other words, connectionist methods are generally more suitable in working with low-level data, inductive and genetic methods with feature-level data, and analytic methods with logic-level data.

The learning approaches used in the sequence analysis tasks discussed in this article fall into the inductive learning and connectionist paradigms. However, the learning problem is roughly the same for the different tasks: given some positive and negative examples, learning is achieved by forming concepts to distinguish the two types of examples in seen and unseen instances. Since the classifications of the training set are known beforehand, this is also known as *learning from examples* or *supervised learning.* In this classification task, a learning algorithm is presented with a set of examples with their appropriate classifications. The algorithm then tries to form concepts based on these examples; this is the *training* phase. To evaluate the concepts learned by the algorithm, instances are presented for classification and accuracy is measured; this is the *testing* phase. To truly evaluate the accuracy of the learned concepts, and hence the effectiveness of the learning algorithm, the training and testing instances are usually disjoint. In practice, given a set of examples, a random subset is used for training and the rest are used for testing. To assure randomness, experiments are usually run multiple times with different sets of random training examples and the accuracies are averaged. One of the more common techniques used in evaluating learned concepts is *cross-validation* (Breiman et al., 1984). In this technique, a subset of the examples is used for testing and the rest are used for training. This is repeated for a different subset of the examples as the test instances until the examples are exhausted, and the accuracies are averaged.

The following subsections give a brief overview of the four learning paradigms and techniques. The overview is intended to provide readers unfamiliar with machine learning a broad view of the different approaches. Readers who want to learn more about these techniques are directed to the references cited in respective sections. Introductory readings in machine learning can be found in (Carbonell, 1989; Cohen and Feigenbaum, 1982; Michalski et al., 1983).

3.1 Inductive Learning

Inductive learning (also known as *empirical learning* or *similarity-based learning*) involves forming concepts by finding similarities and differences among data. This approach is best suited for tasks where a considerable amount of data is available and knowledge about the domain is scarce. This method does not work effectively when there is insufficient data since it relies on finding patterns among data. With insufficient amounts of data, too little information is available to infer patterns with high degree of confidence. The following overview discusses approaches in supervised learning, where examples are classified before they are presented to the learning system. Discussions of unsupervised learning approaches, where data are grouped or clustered according to the algorithms, can be found in (Michalski and Stepp, 1983; Fisher and Langley, 1985).

3.1.1 Concept Learning

One form of inductive learning, called *concept learning*, is to generate concepts by generalizing the data. A concept can be regarded as a semantically meaningful structured object. These concepts essentially characterize patterns in the data. Learning is achieved by searching the hypothesis (descriptor) space and locating descriptors that best identify or differentiate patterns in the data. Some of the common representations for the generated concepts are decision trees, rules, and version spaces. Decision trees are used in ID3 (Quinlan, 1986), where each concept is represented as a conjunction of terms on a path from the root of a tree to a leaf. Rules in CN2 (Clark and Niblett, 1987) are if-then expressions, where the antecedent is the pattern and the consequent is the classification. Each version space learned in VS (Mitchell, 1982) defines the most general and specific description boundaries of a concept.

For example, given the descriptions of cups and non-cups in Table 1, a simple concept learning algorithm which only gathers common descriptors present in the cup examples and

Table 1: Descriptions of cups

Example	COLOR	MATERIAL	HANDLE	LIFTABLE	CONCAVE	CUP
A	RED	METAL	YES	YES	YES	YES
B	BLUE	PLASTIC	NO	YES	YES	YES
C	RED	CERAMIC	YES	YES	YES	YES
D	WHITE	PLASTIC	NO	YES	YES	YES
E	BROWN	METAL	NO	NO	NO	NO
F	WHITE	CERAMIC	YES	NO	YES	NO

absent from the non-cup examples would form the following concept for cups:

$$(\text{LIFTABLE} = \text{YES}) \text{ and } (\text{CONCAVE} = \text{YES}) \rightarrow \text{CUP}$$

That is, only the "liftable" and "concave" features are present in all the examples of cups and therefore an instance is a cup if it is liftable and concave.

3.1.2 Exemplar-based Learning

Another form of inductive learning, called *exemplar-based learning*, involves storing all or a large subset of the examples in memory. Given an unclassified instance, a distance metric, which encodes the notion of "similarity" and is adjusted during training, is used to find the "closest" example(s) in the memory relative to the test instance. The classification of the test instance is then based on this selected example(s). Each exemplar in memory is associated with a weight that reflects its effectiveness in correct prediction and is incorporated into the distance metric. This weight is adjusted during training (Cost and Salzberg, 1990b).

For example, using the cup and non-cup descriptions in the previous section, we can transform Table 1 to Table 2. The binary substitutions are: no = 0 and yes = 1. (COLOR and MATERIAL, which are not binary descriptors, are omited from Table 2 for simplicity. Section 4.1.2 describes a method for dealing with non-binary descriptors.) A simple distance metric is the sum of absolute differences in all descriptors. For simplicity, after training, all the examples have equal weights. Given the instance [HANDLE = 0, LIFTABLE = 1, and CONCAVE = 1], the closest examples are B and D (since the distances for these two are zeros, while the distances of the other examples are one or two); therefore, it is classified as a cup. (Ties are broken arbitrarily if the closest neighbors do not agree.)

This approach is related to *rote learning*, where all the examples are memorized and an instance can only be classified when the instance exists in the memory. That is, there is no explicit reasoning or generalization in rote learning.

3.2 Analytic Learning

Explanation-based learning (EBL) (Mitchell et al., 1986) is a form of analytic learning. It involves forming explanations of why an example of a coarsely defined concept belongs to

Table 2: Binary descriptions of cups

Example	HANDLE	LIFTABLE	CONCAVE	CUP	Distance from (0 1 1)
A	1	1	1	YES	1
B	0	1	1	YES	0
C	1	1	1	YES	1
D	0	1	1	YES	0
E	0	0	0	NO	2
F	1	0	1	NO	2

that concept. A *domain theory* (facts and rules about the domain) forms the basis for explanations and an *operationality criterion* restricts the language of explanations to a useful form. Moreover, explanations are generalized by a form of goal regression (Waldinger, 1977; Nilsson, 1980) to find a sufficient condition for the explanations. That is, a most general concept is formed from an example(s) according to the operationality criterion that is consistent with the domain theory and the initial concept.

For example, consider the cup recognition problem in (Mitchell et al., 1986) with the following goal concept:

$$\text{CUP(x)} \leftrightarrow \text{LIFTABLE(x) and STABLE(x) and OPEN-VESSEL(x)}$$

Some entries in the domain theory are:

IS(x, LIGHT) and PART-OF(x, y) and ISA(y, HANDLE) $\rightarrow$ LIFTABLE(x)
PART-OF(x, y) and ISA(y, BOTTOM) and IS(y, FLAT) $\rightarrow$ STABLE(x)
PART-OF(x, y) and ISA(y, CONCAVITY) and IS(y, UPWARD-POINTING) $\rightarrow$
OPEN-VESSEL(x)

The operationality criterion is to express the concept in terms of structural features like LIGHT, HANDLE, and FLAT. Given the training example:

OWNER(OBJ1, EDGAR) and PART-OF(OBJ1, CONCAVITY-1) and IS(OBJ1, LIGHT)
and ...

an EBL algorithm can form an explanation as shown in Figure 1. By changing constants to variables, the explanation can be generalized to:

PART-OF(x, xc) and ISA(xc, CONCAVITY) and IS(xc, UPWARD-POINTING) and
PART-OF(x, xb) and ISA(xb, BOTTOM) and IS(xb, FLAT) and
PART-OF(x, xh) and ISA(xh, HANDLE) and IS(x, LIGHT) $\rightarrow$ CUP(x)

which represents a generalized concept for cups.

The major difference between EBL and inductive learning is that EBL is more knowledge intensive and can generalize from one example. However, EBL requires a domain theory, which is not necessary in inductive learning. Indeed, that is the intent of this learning paradigm. A thorough survey of EBL approaches can be found in (Ellman, 1989).

Figure 1: An explanation tree for the cup problem (Mitchell et al. 1986)

3.3 Connectionist Methods

Neural networks (Lippmann, 1987; Vemuri, 1988) were originated from *perceptrons* introduced by Rosenblatt (1962) and Minsky and Papert (1969). Connectionism is an attempt to mimic how information is processed in the human brain, which has likely many billions of neurons and connections among them.

A neural network usually consists of layers of units (*neurons*) and links between units in adjacent layers. There are other networks that are not layered. A typical layered neural network has an input layer, an output layer, and zero or more hidden layers (e.g., Figure 2 is a three-layer neural network). A perceptron is a special case of a neural network. It has input and output layers, but there are no hidden layers. That is, the input layer is directly connected to the output layer. In addition, there is only one output unit in perceptrons. It is well known that the absence of hidden layers prevents perceptrons from learning anything but linearly separable concepts. That is, in a two-dimensional event space, a concept is learnable by perceptrons if a straight line can be drawn in the event space separating positive examples from negative ones.

In a fully connected network, each unit in a layer is connected to every unit in the adjacent layer. A *feed-forward network* propagates values from the input layer, through the links, to the output layer. The output of a node at each layer is determined by an output function, f, a threshold, θ, inputs from the previous layer, x, and weights on the links connected to the previous layer, w (Figure 3). The output, y, of a unit is typically (Lippmann, 1987):

$$y = f\left(\sum_{i=1}^{n} w_i x_i - \theta\right) \tag{1}$$

where n is the number of links from the previous layer to this unit. f is typically a threshold or sigmoid function with values from 0 to 1.

Learning is achieved by adjusting weights and thresholds in the network by propagating, from the output layer back to the input layer, the difference between the desired output and

Figure 2: A three-layer neural network

Figure 3: A unit in a neural network

Figure 4: Learning the OR function with a neural net

the actual output at each unit. A well-known adjustment algorithm is the back-propagation algorithm introduced by Rumelhart et al. (1986). Usually, examples are repeatedly presented to the network until certain criteria are met; for example, the error rate on the training examples is within a certain limit or a certain amount of processing time has been consumed.

For example, consider a network with two input units and one output unit, and the network attempts to learn the OR function. The output unit has a threshold of 1 and the initial weights are randomly set at 0 or 1. A simple weight adjustment algorithm is to add the inputs to the weights when the output is too low and subtract when the output is too high. The four combinations of inputs are repeatedly presented to the network until all the inputs produce the correct output.

Figure 4 shows how the OR function can be learned. The two weights in the network are first initialized to one and zero. The first input pair is 00. Since the weighted sum of the inputs is 0 ($0 \times 1 + 0 \times 0$) and is lower than 1 (the threshold), the output is zero. The output is correct and hence the weights are not adjusted. When the second input pair 01 is presented to the network, the weighted sum is 0 ($0 \times 1 + 1 \times 0$) and is still lower than the threshold and hence the output is zero. Because the output is lower than the correct output, 1, the inputs are added to the corresponding weights. That is, the weights become 1 and 1. Similarly, after four more pairs of inputs, the weights converge to two and two, which correctly characterize the OR function.

3.4 Genetic Algorithms

The idea of genetic algorithms was introduced by Holland (1975). It is based on Darwin's theory on evolution, *Darwinism*, which essentially posits that the offspring of organisms are generally different and only the offspring with characteristics suitable for the environment can survive and reproduce. Hence, "good" characteristics are being passed on through generations and "bad" ones are lost.

In genetic algorithms there is a set of rules, or *classifiers*, and a message list (Holland, 1986; Booker et al., 1989). If the antecedent of a classifier matches with the messages on the list, the classifier will *bid* to post its consequence onto the message list. A *bid* is a probability of how likely the message will actually be posted and is determined by the *strength* of the rule and *specificity* of the rule's antecedent. The strength of a rule represents its usefulness in the task and the specificity of a rule represents its relevance in the current iteration. If the message of a rule is actually posted, the strength of the posting rule is decreased by the bid and the strength of the rules which posted the messages, in the previous iteration, matched by the posting rule is increased by sharing the bid. That is, the classifiers in the previous iteration are "credited" if their posted messages contributed to the successful posting classifier in the current iteration. Similarly, the current posting classifier will also be "credited" if its posted message is matched by a posting classifier in the next iteration. Consequently, this creates a chain effect which benefits all the classifiers that are in the solution path. At each iteration, all the messages from the previous iteration are replaced by the ones from the current iteration. At the end only strong classifiers survive and are the ones that contributed to the process.

To adjust the classifiers, the notion of evolution is employed. Genetic operators are applied to classifiers to generate offspring. Common genetic operators are: *crossover* (exchanging segments between two classifiers), *inversion* (reversing the order of two segments in the same classifier), and *mutation* (changing a segment of a classifier to something different). The strength of an offspring is derived from their parents. The offspring then compete with the original classifiers and can only survive if they replace weaker classifiers—the number of classifiers remains the same. The rate of applying each operator and the number of reproducing classifiers are based on user-specified parameters.

Wilson (1987) presents a specialized genetic algorithm for learning from examples. The antecedent of classifiers is a template for matching inputs and the consequent is the class prediction of the matched input. When an input is presented to the classifiers, the matching classifiers form the *match set*, M. From M, a classifier is selected using the probability distribution over the strengths in M and the prediction of the chosen classifier is the system output.

All the classifiers with the same output from M constitute the *action set*, A, and the rest of M form the *not-action set*, $notA$. During the strength-adjustment (credit-assignment) process, a fraction of the strengths of all the classifiers in A are deducted. A payoff R is added to the strength of each classifier in A if the system output is correct and R', where $0 \leq R' < R$, is added when the output is wrong. A fraction of the strengths in $notA$ is then deducted. To generate new classifiers and replace weak classifiers, crossovers and mutations are performed according to user-specified rates and probabilities.

During training, examples are presented to the system repeatedly until convergence is achieved or certain criteria are met. The strengths of classifiers are adjusted, new classifiers are generated, and weak ones are replaced according to the algorithm described above.

For example, Figure 5 illustrates an evolution of classifiers using a genetic algorithm to learn the OR function. A simplified version of Wilson's algorithm is adopted to show the working of genetic algorithms. The strength-adjustment process only consists of adding 1 to the strength of classifiers that match the training instance and predict the correct classification. The initial strength of each classifier is 1. The strength of the two offspring

```
a)    1/00/0      2/00/0      2/00/0                       2/00/0
b)    1/#1/1  00  1/#1/1  01  2/#1/1  crossover(c,d)   2/#1/1  10
c)    1/0#/1  --> 1/0#/1  --> 2/0#/1       -->         2/0#/1  -->
d)    1/10/0      1/10/0      1/10/0                   1.5/1#/1

      2/00/0         2/00/0                2/00/0       3/00/0
      2/#1/1  11     3/#1/1  mutation(c)  3/#1/1  00    3/#1/1
      2/0#/1  -->    2/0#/1      -->      2/00/1  -->   2/00/1
    2.5/1#/1       3.5/1#/1             3.5/1#/1      3.5/1#/1
```

Figure 5: Learning the OR function with a genetic algorithm

generated by a crossover is half the sum of their parents' strengths. In our example, there are four classifiers (a–d); each classifier is represented as *strength/pattern/classification.* (The "#" in a pattern is a wild card.)

After two examples, a crossover operation is applied to classifier `c` and `d`. The crossover point is in the middle of the pattern (i.e. 2/0≀#/1 and 1/1≀0/0). Therefore, the offspring are 1.5/1#/1 and 1.5/00/0. Since 1/10/0 (one of the two parents) is weaker than the offspring, it is replaced by 1.5/1#/1 (arbitrarily chosen from the two equally strong offspring). A mutation operation is then applied to classifier `c` after four examples, the "#" in the pattern is replaced by a "0." After five training examples, the strongest three classifiers succinctly describe the OR function.

4 Amino Acid Sequence Analysis

We now turn our attention to a description of the various machine learning approaches employed in various amino acid sequence analysis tasks. Each of the following subsections discusses a different task and the applied learning techniques.

4.1 Protein Secondary Structures

Recall that due to the physical and chemical interactions among amino acids, proteins do not appear as linear ropes. Interacting segments create twists and turns (called *protein folding*) which make proteins appear globular. Scientists have identified structural patterns in proteins and classified three structural levels. The primary structure is the sequence of amino acids, a linear chain of specific acids. The main secondary structures are three-dimensional structures formed from this linear sequence called *α-helix*, *β-sheet*, and *coil.* Groups of secondary structures produce tertiary structures.

There have been quite a number of research attempts to use machine learning techniques to identify protein secondary structures from amino acid sequences. The task is to learn the rules governing the formation of, say an α-helix, given a particular amino acid sequence. All the techniques described below use a *windowing* technique for generating training sequences. Each training sequence consists of a fixed number of amino acids in sequence and a *window*, a fixed number of amino acids considered as a subsequence. The window slides over the protein sequence, one amino acid at a time, to generate different training sequences. The window size varies according to the method applied in different tasks. The systems are described in two subsections, one discusses the neural network approaches and the other discusses the symbolic learning approaches. A third subsection summarizes the results of various learning approaches and compares these approaches to results obtained from human-designed systems.

4.1.1 Neural Network Approaches

Qian and Sejnowski (1988) used neural networks to learn the rules of secondary structure formation. They varied the number of groups, or window size, in the input layer from 1 to 21 and empirically found 13 to be the most effective. The input groups represent a sequence of amino acids. Each input group has 21 units, which encode 20 amino acids and a "spacer." They attempted 0 to 60 units in the hidden layer and determined that a layer with 40 units was the most appropriate. It is not clear from the paper why why these settings worked best. There are three units in the output layer that encode the three secondary structure classes. For a given sequence of amino acids at the input units, the output units represent the structure classification of the center amino acid in the input sequence. During training, the weights between nodes are adjusted using the back-propagation learning algorithm (Rumelhart et al., 1986). The highest accuracy obtained was 62.7%.

They also attempted two connected networks, called *cascaded networks*, where the output of the first network is the input of the second. The first network is the same one described above. The second network has 13 input groups with 3 units each. The 13 input groups represent a sequence of 13 outputs from the first network. The second network still has a hidden layer of 40 units and an output layer of three units. With the cascaded networks, they achieved an accuracy of 64.3%. This means that the learned system is correct in predicting secondary structures 64.3% of the time.

Holley and Karplus (1989) independently used a very similar neural network approach. Based on the evidence of high statistical correlation with secondary structure and 8 amino acids on either side of a prediction point (Garnier et al., 1978), the input layer has 17 groups, each with 21 units. The hidden layer has only 2 units, but it is unclear why 2 was chosen. The output layer also has two units and the secondary structures are encoded as follows: (1,0) = helix, (0,1) = sheet, and (0,0) = coil. They used the same back-propagation algorithm Qian and Sejnowski used for training the network. Since the output units generate real numbers between 0 and 1, a threshold is used to determine the class represented by the outputs. In addition, domain knowledge is also incorporated in the classification process. Helix is assigned to any group of four or more contiguous amino acids having helix outputs greater than sheet outputs and greater than the threshold. Similarly, sheet is assigned to any group of two or more contiguous amino acids having sheet outputs greater than helix outputs and greater than the threshold. The rest are assigned as coil. The threshold of .37 was found to be the best achieving an accuracy of 63%, roughly the same as Qian and Sejnowski's result. It is unknown, however, whether these two systems predicted correctly on the same instances.

4.1.2 Inductive Learning Approaches

Concept Learning King (1987) used a concept learning approach to generate rules for secondary structure prediction. Amino acids are grouped according to their chemical properties as presented in (Taylor, 1986). Conjunctions and disjunctions of descriptors in the antecedent are allowed as long as they are consistent with the chemical properties so that the search space is limited. The antecedents of rules can also be specialized and generalized according to a generalization/class lattice. A generalization lattice is a graph, where each node is a set of descriptors and its child nodes are subsets of the parent node but are not subsets of other child nodes. It is not a tree because a node can have more than one parent node. A parent node is, in a sense, more "general" than its child nodes. This is similar to a generalization/class hierarchy that may be traversed to find the appropriate descriptors (Utgoff, 1986).

The learned rules are of the form:

$$Descriptor_1, Descriptor_2, \ldots, Descriptor_n \rightarrow SecondaryStructureType$$

where $Descriptor_i$ is the descriptor of an amino acid in a segment and all the amino acids in the segment have the same $SecondaryStructureType$. n is not fixed.

The search operators employed in this system included adding disjunctive or conjunctive descriptors and generalizing or specializing existing descriptors in the antecedent of a rule. Using a best-first search, the system looks for a better rule until the search operators cannot generate rules better than the current one. The evaluation function consists of two parts: *coverage* and *accuracy*. Coverage measures the number of amino acids covered by the rule in the examples. Accuracy measures the percentage of correct classifications by the rule. During the rule selection process, only rules with an accuracy greater than 60% and a coverage of more than 80 amino acids are selected. These thresholds were chosen arbitrarily by King. When more than one rule passes the threshold, the rule with a higher coverage is preferred.

According to the article, only rules for α-helices and β-sheets are generated. A sample rule from (King, 1987) is:

$$TinyOrPolar, Large, AromaticOrM, Large, LargeAndNotNegative \rightarrow Helix$$

which matches five amino acids in sequence. Classification is performed by matching the instances with all the rules. If conflicts occur (both helix and sheet rules match), helix is assigned. If none of the rule matches, coil is assigned. (The decision is based on the probabilities of the three structures: helix = .26, sheet = .20, and coil = .54.) King achieved an accuracy of 60%.

Seshu et al. used a constructive induction approach (Seshu et al., 1989). New features are built from primitive features using construction operators (constructors). These constructors can be domain-dependent or domain-independent. Domain-dependent constructors combine features according to domain knowledge. For example, in the secondary structure task, there is a sequence constructor which takes a pattern and returns a feature which counts the number of occurrences of that pattern in the database. Domain-independent constructors include boolean operators like conjunctions and disjunctions. An optimization algorithm

(Seshu et al., 1989) is then applied to select a subset of constructed features to be added to the original feature set. The new feature set is then used in the PLS1 induction system (Rendell, 1983) to generate rules. If the accuracy of the generated rules is not satisfactory, the *construct-select-induce* process is repeated until an acceptable accuracy is obtained.

Unlike King's rules, which match different numbers of amino acids, Seshu et al.'s rules are fixed to match nine amino acids and the structure predicted by the rule applies to the center amino acid, similar to the approach used in the neural network methods. That is, a rule is of the following form:

$$Descriptor_1, \ldots, Descriptor_5, \ldots, Descriptor_9 \rightarrow SecondaryStructureTypeAtDescriptor_5$$

where $Descriptor_i$ is applied to amino acid categories, six in total, instead of the amino acids themselves, which is similar to King's approach.

Using the same data as in other work, they achieved an accuracy of about 60.6%. Seshu et al. (1989) also mentioned that they achieved 53.7% accuracy with two induction systems: PLS1 (Rendell, 1983) and ID3 (Quinlan, 1986).

It is important to note that there is a major difference between King's and Seshu et al.'s approaches besides the representation of rules. King uses a greedy approach to generate rules to cover the examples while Seshu et al. keep generating a new set of rules based on an "enhanced" set of features until accuracy cannot be improved.

Exemplar-based Learning Cost and Salzberg (1990a) used an exemplar-based learning approach. Based on the examples, they use the *value difference metric* (Stanfill and Waltz, 1986) to generate distance tables for each symbolic feature. This metric provides a numeric distance measure between two values of a symbolic feature and is defined as:

$$\delta(V_1, V_2) = \sum_{i=1}^{n} \left| \frac{C_{1_i}}{C_1} + \frac{C_{2_i}}{C_2} \right|^k \tag{2}$$

where V_1 and V_2 are two values of a feature; for example, two amino acids. The distance between two values are summed over all n classes; in this case, $n = 3$ (*helix*, *sheet*, and *coil*). C_{1_i} is the number of times V_1 is classified as class i, C_1 is the number of times V_1 occurs, and k is fixed at 1 (Cost and Salzberg, 1990b).

During the classification process, these tables are consulted to determine the distance between two instances. The distance between two instances is defined as:

$$\Delta(A, B) = w_A \sum_{i=1}^{N} \delta(a_i, b_i)^r \tag{3}$$

where A is an exemplar in the memory and B is the new instance. a_i and b_i are amino acids of A and B at the i^{th} position, in a window size of N. w_A is the ratio of the number of uses of A to the number of correct uses of A. Basically, Δ calculates the weighted sum of distance between the two instances at each attribute. If every time an exemplar A is picked to be the nearest neighbor to an instance B and the class of A matches that of B, w_A is close to 1. Otherwise, w_A is greater than 1, which means A is not very reliable and extra distance is added when A is used. During training, w_A is constantly adjusted. r is set to 1 (Manhattan distance) for this task (Cost and Salzberg, 1990b).

Table 3: Summary of Secondary Structure Prediction Accuracy

Method	Accuracy (%)	Type
Lim	50	Human-designed
Garnier-Robson	53	
Chou-Fasman	50	
Qian-Sejnowski	62.7	Neural Networks
Qian-Sejnowski (cascaded)	64.3	
Holly-Karplus	63	
King	60	Concept Learning
PLS1 or ID3	53.7	
Seshu et al.	60.6	
Cost-Salzberg	71.0	Exemplar-based Learning

The classification of the nearest instance in the memory becomes the prediction of the new instance. The classification is then adjusted according to the minimal sequence length restrictions used by Holley and Karplus (1989). These restrictions state that a sheet must span at least two amino acids and a helix must span at least four, as mentioned in Section 4.1.1. The highest accuracy reported was 71.0% with a window size of 19.

4.1.3 Summary of Prediction Accuracy

The first three methods in Table 3 are human-designed systems and are the work of Lim (1974), Garnier and Robson (Garnier et al., 1978), and Chou and Fasman (1978). The accuracies of these methods are summarized in (Qian and Sejnowski, 1988). The rest used machine learning techniques, which are described in previous subsections. As we can see from the table, systems generated by machine learning techniques are often more accurate than human-designed systems. Since there are no strong indications that all the systems used the same data and the same method for measuring accuracy, the above table can only be treated as a rough comparison among systems. These issues are further discussed in Section 6.1.

The highest accuracy achieved was only 71%, which is rather low for practical purposes. Hunter (1991) suggests two possible reasons that attribute the low accuracy. One reason is that all the systems only use local information (adjacent amino acids) to make predictions; more distant amino acids might play a significant role in determining the structure. The other reason is that helices, sheets, and coils might not be the appropriate level of description for forming concepts; this problem is related to the representation issues that will be discussed in Section 6.1. In addition, the amount of training data might not be sufficient for the systems to generate accurate concepts.

4.2 Signal Sequences of Exported Proteins

Gascuel and Danchin (1986) tried to differentiate prokaryotes' (*E. coli*) exported protein signal sequences from eukaryotes' (*H. sapiens*). They used a symbolic concept learning

Figure 6: Amino acid class hierarchy (Gascuel and Danchin, 1986)

approach to investigate if there exist rules for the differentiation. This differentiation can indicate a difference in the mechanism of how proteins are exported through membranes.

Gascuel and Danchin used 18 bacterial and 22 human sequences as data. Amino acids are grouped into classes and are arranged in a hierarchy with more general classes at the top (for example, Figure 6). Primitive descriptors are functions that are applied to nodes (amino acid classes or amino acids) in the hierarchy. The primitive descriptors include the *number of amino acids of a given class*, *barycenter of a given amino acid*, *distribution of amino acids*, *position from the start*, *minimum distance between two amino acids*, and *presence of a pattern* (Gascuel and Danchin, 1986). Descriptors are composed from primitive descriptors according to a grammar provided by the user. Two sample descriptors are:

the signal sequence contains at least one C
and
the barycenter of aromatic amino acids is greater than or equal to 7.

Each descriptor is applied to all the nodes in the class hierarchy, which forms a space for that descriptor. For example, applying the NUMBER descriptor (number of amino acids) to Figure 6 yields Figure 7. During learning, for each descriptor, its space is searched in a top-down manner. The most general instance of the descriptor and its children are evaluated. At each evaluation cycle, if an instance is below the threshold, its children in the hierarchy will be evaluated. When the search ends, the descriptor instance with the highest score is selected. The evaluation is based on a contingency table with a chi-square test.

Totally, 17 descriptors were found to pass the threshold, which include the two sample descriptors mentioned in the previous paragraph. When the descriptors are used to classify a sequence, each descriptor provides a score of 0, 1, or .5, where 0 is more bacterial, 1 is more human, and .5 is undecided. The score of the 17 descriptors are then summed. According to the scores from the training set (40 sequences), a score of 6.5 was reported to be the best differentiation point. For a test set with 14 sequences, the accuracy rate was 64%.

5 Nucleotide Sequence Analysis

Another type of sequence analysis is investigated for nucleotide sequences from DNA and RNA. The following subsections discuss how learning techniques were applied to nucleotide

Figure 7: Hierarchy for descriptor NUMBER (Gascuel and Danchin, 1986)

sequence analysis.

5.1 Promoters

Promoters are DNA regions where *transcription* begins. Transcription is the process of copying information from a gene on DNA to mRNA (messenger RNA). For all the systems described below, nucleotide sequences from *E. coli* were used.

5.1.1 Neural Networks

Towell et al. (1990) used neural networks with guidance from general knowledge about promoters. Given a domain theory, a set of rules about the domain, their system, called Kbann (Knowledge-Based Artificial Neural Networks), translates the rules to a neural network. The rules are used to generate the topology as well as the weights and thresholds of an initial neural network. It is important to note that, unlike most networks, the network generated by Kbann is not necessarily symmetrical or uniform; that is, there might be connections between non-adjacent layers (a semi-layered network). This is due to the fact that the number of connections from an input unit to an output unit relies on the dependencies of terms in the rules, and so there is no guarantee that all the output units have the same distance from the input units. In addition, input units corresponding to features that do not appear in the rules are added. Connections between units are also added to explore dependencies not expressed in the rules; the weights for these connections are initialized to zero. As a last step in constructing the initial network, near-zero random numbers are added to the weights and thresholds to avoid symmetry-breaking problems (Rumelhart et al., 1986). For the promoter recognition task, each sequence has 57 nucleotides, and hence the input layer has 57 input groups, each has four units to represent a nucleotide in the DNA sequence. In the output layer, there is only one unit to indicate if the input sequence is a promoter. Figure 8 contains part of the domain theory used in Towell et al.'s study and the initial neural network built from the domain theory.

Towell et al. used 53 promoter sequences and 53 non-promoter sequences for training. Each sequence has 57 nucleotides. The non-promoter sequences were generated from a DNA segment that is believed not to contain any promoter sites. In their experiments they

promoter ← contact, conformation
contact ← minus_35, minus_10

Figure 8: Partial domain theory and initial neural network for promoter recognition (Towell et al., 1990)

achieved an error rate of 4/106 (96% accuracy).

5.1.2 Exemplar-based Learning

Cost and Salzberg (1990) used an exemplar-based learning technique to determine the location of promoters. They used the same approach as in their protein secondary structure task which is described in Section 4.1.2 except that they set r to 2 in Equation 3 (Euclidean distance) (Cost and Salzberg, 1990b). They used the same data Towell et al. used—53 promoter and 53 non-promoter sequences, each with 57 nucleotides. They obtained an error rate of 4/106 (96% accuracy), same as Towell et al.'s result. But, it is not clear if both systems predicted incorrectly on the same instances.

5.1.3 Summary of Prediction Error Rate

Table 4 summarizes the error rates for various promoter prediction systems. The first system is due to O'Neill (1989) and is the most accurate human-designed system according to Towell et al. (1990). The results for a neural network with the standard backpropagation algorithm and ID3 are from Towell et al. (1990). Although all the learned systems used the same data, it is not clear if O'Neill's system did. Hence, again, the above table can only provide a rough comparison among the systems (see Section 6.1). In general, except ID3, learned systems were more effective than the human-designed systems.

Although the promoter prediction accuracies demonstrated by these system are not perfect, they are far much better than those in protein secondary structure prediction (Section 4.1.3). This might be attributed to the larger amount of information exhibited by nucleotides near the promoter site. That is, the performance results suggest that adjacent

Table 4: Summary of Promoter Prediction Error Rate

Method	Error Rate	Type
O'Neill	12/106	Human-designed
Standard Backpropagation	8/106	Neural Networks
Towell et al.	4/106	Neural Networks with domain knowledge
ID3	19/106	Concept Learning
Cost-Salzberg	4/106	Exemplar-based Learning

nucleotides to the promoter site are indicative of the presence of such a site.

5.2 RNA Splice Junctions

In eukaryotes' DNA, there are interrupted genes. That is, some regions of a gene do not encode protein information. During transcription, these non-protein-encoding regions are passed to the RNA. These regions on an RNA are called *introns* and are sliced off during *translation*, the process of decoding information on an RNA to generate proteins. Before translation begins, the regions that encode protein information, *exons*, are spliced together after the introns are removed.

Lapedes et al. (1990) used perceptrons to determine if DNA sequence segments containing the dinucleotides "AG" or "GT" are transcribed to RNA splice junctions. Input sequences with 11, 21, and 41 nucleotides were used in different experiments. Each input group has four units, which represent the four different nucleotides. All known splice junctions are divided into *donor* sites (the boundary between an intron and an exon) and *acceptor* sites (the boundary between and an exon and an intron). For the donor group, DNA segments were selected with the highly conserved dinucleotide "AG" in the middle. Similarly, for the acceptor group, the dinucleotide "GT" was used. All but 50 DNA segments were used as positive or negative training examples and the rest were used in testing. They obtained an accuracy of 91.2% (41 input nucleotides) in the acceptor group and 94.5% (11 and 21 input nucleotides) in the donor group.

Lapedes et al. (1990) also reports that they achieved a slightly lower accuracy with ID3. With ID3, the input sequence length was 20. In the generated decision tree, each node contains a descriptor specifying a nucleotide at a particular location. Figure 9 contains the top two levels of an ID3-generated tree.

5.3 Protein Coding Regions

Lapedes et al. (1990) tried to determine whether certain DNA sequence segments are translated to protein. Not all DNA segments encode proteins, some encode "control" information; for example, regions that signify the initiation and termination of protein production.

Lapedes et al. used perceptrons to determine protein coding regions in eukaryotic (*H. sapiens*) and prokaryotic (*E. coli*) DNA sequences. However, they used higher-order perceptrons introduced by Lee et al. (1986) to determine if the input representation can be

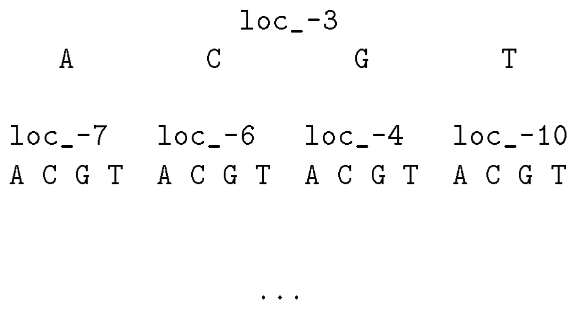

Figure 9: Partial decision tree for RNA splice region recognition (Lapedes et al., 1990)

improved before the regular (first-order) perceptron algorithm is used. In higher-order perceptrons, instead of summing the weighted single inputs at the output unit (i.e., $\sum_i w_i x_i$, Equation 1), weighted products of multiple inputs are summed (i.e., $\sum_{ijk} w_{ijk} x_i x_j x_k$ for a third-order perceptron).

Using a third-order perceptron, Lapedes et al. decided that the *codon* input representation (a sequence of three nucleotides which encodes an amino acid) is better than the nucleotide input representation. For the codon input representation, 64 units are used in each input group for the perceptrons. Input sequence lengths were varied from 15 to 270 for *E. coli* and 45 to 270 for *H. sapiens* in different experiments. Training examples for *E. coli* were obtained from the first 50 entries in GenBank, a genetic database, and the rest were used for testing. Training examples for *H. sapiens* were obtained from known coding sequences in liver cells. The highest accuracies Lapedes et al. obtained were 99.4% (180 and 270 input nucleotides) for *E. coli* and 98.4% (270 input nucleotides) for *H. sapiens*.

Furthermore, in the course of investigation, Lapedes et al. predicted a previously unannotated protein-coding region in the GenBank entry including the 5' UTR of *E. coli* fhuA gene, which was later confirmed experimentally to be the *pon* B gene (Lapedes et al., 1990). This demonstrates the viability of machine learning in helping molecular biologists discover new knowledge by observing patterns in data that humans have otherwise missed.

5.4 Translational Initiation Sites in mRNA

A translational initiation site or ribosome binding site is a region of mRNA (messenger RNA) where a ribosome binds to the mRNA and starts producing a protein according to the nucleotide sequence on the mRNA.

Stormo et al. (1982) used perceptrons to identify translational initiation sites. Input sequence lengths were 51, 71, and 101 in different experiments. Each input group has four units, which represent the four different nucleotides. In addition, the output layer has only one unit. The output unit has a threshold of 0 to determine whether the input sequence is a translational initiation site. 124 known sites and 167 non-sites were used for training. The trained perceptrons were then tested on the entire mRNA library, which had over 78,000 sites. The perceptrons trained with 71 and 101 nucleotides were able to identify the 124 known sites correctly. The perceptron trained with 51 nucleotides could not reach a state to differentiate the sites from the non-sites.

6 Discussion

A variety of machine learning techniques have been applied to automating the development of different sequence-analysis systems. Learning techniques range from symbolic to neural network approaches. It has been demonstrated that these generated systems are often more effective and accurate than their human-designed counterparts. This strongly indicates that learning techniques are viable alternatives in developing knowledge-based systems. It also implies the learning techniques are no longer confined to "toy" applications and can be successfully applied to "real-world" problems.

Besides building more accurate sequence-analysis systems efficiently, learning techniques are capable of unearthing previously unknown information in molecular biology. As mentioned in Section 5.3, a protein coding region was identified by a learned system and was subsequently verified by experiments. This capability is particularly important to molecular biologists especially when they have to analyze vast amounts of data. Learning systems can help scientists to process the data efficiently and develop hypotheses, which can then be verified or refuted by experiments. In other words, learning systems can speed up and contribute to the process of discovering new knowledge.

6.1 Issues

Domain Knowledge In some of the systems discussed in this paper, a substantial amount of knowledge in molecular biology has been incorporated into the symbolic inductive learning and neural network systems. For example, chemical properties of amino acids are taken into account in King's work (Section 4.1.2) and promoter facts are used to build the initial neural network in Towell et al.'s work (Section 5.1.1). This deviates from a number of similar systems which mainly rely on identifying patterns in the data without reliance upon background knowledge. The incorporation of knowledge helps searching algorithms avoid exploring many implausible hypotheses and vastly reduces the size of the search space, and hence shortens the overall execution time. However, the use of knowledge in learning systems might potentially prevent discovery outside the search space delimited by domain knowledge. Therefore, the use of knowledge in inductive systems has a delicate trade-off between execution time and discovery potential.

Knowledge Representation Knowledge representation is a well-known fundamental problem in artificial intelligence. It is not easy to find the appropriate representation for a particular task and in almost all cases, there is no single effective and agreed upon representation for a wide variety of tasks. Learning systems are no exceptions. In symbolic learning, researchers have been using various techniques to augment and enrich the language used in the hypothesis space. For example, Gascuel and Danchin use a domain-dependent grammar to generate descriptors (Section 4.2), King uses a domain-dependent generalization lattice to provide different levels of abstractions for the descriptors (Section 4.1.2), and Seshu et al. provide domain-dependent constructors to introduce new descriptors (Section 4.1.2). In neural networks, most systems experiment with different alternatives to find the "right" number of input and hidden units (trial and error). In most cases the ranges of these numbers are conveniently and randomly chosen. However, some work has been done to make a more reasonable choice of representation. For example, a high-order neural network is used to evaluate two different representations in Lapedes et al.'s work (Section 5.3) and promoter facts are employed to initialize a neural network topology in Towell et al.'s work (Section 5.1). Although more work needs to be done, this is a step toward the right direction in providing more effective knowledge representations for learning systems.

Comprehensibility Symbolic concepts are generally more comprehensible than concepts represented in neural nets. In some applications connectionist approaches might generate more effective concepts than symbolic approaches, but it might be worthwhile to sacrifice some accuracy for comprehensibility. One plausible solution to this problem is to extract symbolic concepts from a connectionist representation (Towell et al., 1991).

Speed The issue of efficiency was not a concern in the work discussed above. Most of the current research in machine learning has been focused on improving the accuracy of learning algorithms on relatively small data sets. Due to the initiation of the *Human Genome Project* (DeLisi, 1988), large amounts of genetic data will be generated in the coming years. That is, the current learning technology will not be adequate in efficiently processing genetic databases of potentially massive size; current learning systems might not be able to scale up.

Choice of Parameters How to choose parameters (thresholds, number of input units, etc.) optimally for learning systems is another concern. In most systems described here, parameters are selected in an ad hoc way. One would prefer to have a systematic way of determining these parameters instead of experimenting with them. One approach might be to perform sensitivity analyses on how parameter settings affect prediction accuracy. These analyses might shed some light on which parameter causes which type of errors.

Standardization It might be deceiving to proclaim one learning system is better than the other when their accuracies are based on different data sets. In the machine learning community, researchers addressed this problem by setting up archives for data sets (for example, the one at University of California at Irvine) so that the same data sets can be used for comparison. In addition, they encourage fellow colleagues to make their private data sets publicly available so that other researchers can reproduce their results. In addition, there are no standardized methods for measuring accuracy. Comparative performance results might be misleading if different methods are used. Fortunately, cross-validation techniques are gaining popularity and are generally accepted as a standardized method for measuring accuracy.

6.2 Directions

Although a variety of learning techniques have been used in the sequence analysis tasks discussed in this article, they are restricted to the inductive learning and connectionist paradigms. The tasks are data-oriented and genetic algorithms are suitable and should be explored. We believe, however, that techniques in analytic learning are not appropriate since they are knowledge-intensive and are for high-level inference. However, there might be other sequence-analysis tasks that are knowledge-intensive and hence appropriate for analytic learning.

As mentioned above, the efficiency of current learning systems will be challenged by the enormous amount of data generated by the Human Genome Project. One approach to remedy this problem is to devise more efficient learning algorithms. This area has been receiving much attention in recent years and learning theorists have been building formal computational learning models (Valiant, 1984) and developing more efficient learning algorithms based on these models. In fact, an annual workshop devoted to learning theory was commenced in 1988 (Haussler and Pitt, 1988). Another approach is to utilize parallel processing. However, unfortunately, there has not been much attention in this area, especially in the symbolic learning paradigm. NSF's Scientific Database Initiative has attempted to focus interest here.

Different learning algorithms inherently possess different *inductive biases* (Mitchell, 1980). That is, the way a hypothesis space is searched varies among learning systems. Different systems have different heuristics, and hence different ways to search their hypothesis spaces. Therefore, a learning system's inductive bias impacts its effectiveness in dissimilar tasks. As mentioned above, different knowledge representations are appropriate for different tasks. To take advantage of this diversity among algorithms, multiple learning systems with different biases and representations can be used on the same task at the same time. Furthermore, in the same learning task, different learning systems can generate different concepts and hence make different mistakes. If these mistakes do not overlap, the concepts can complement each other to produce more accurate results. We believe that a group of learning systems can be at least as effective as the system with the most appropriate bias and representation, and the highest accuracy. Hence, the quality of learned concepts can potentially be improved by complementing the systems. These systems can run independently and their results might then be coalesced by another algorithm, which can be another learning algorithm. They can also run cooperatively using *distributed artificial intelligence* (DAI) techniques (Bond and Gasser, 1988), where knowledge is exchanged among individual *agents* while working on the same task. Since these different systems can be run concurrently, parallelism, in this case, can potentially improve both the speed and quality of learning. In our view this is also an area ripe for exploration with the coming generation of distributed and parallel computing systems.

7 Concluding Remarks

This paper surveys how machine learning techniques have been applied to generating sequence-analysis systems. It has been demonstrated that these techniques can be successfully applied to different analysis tasks and the prediction accuracies are often higher than those obtained from human-designed systems. In addition to being more accurate, learning systems are capable of forming verifiable new hypotheses. That is, learning techniques are viable alternatives in developing sequence-analysis systems as well as aiding molecular biologists in discovering new knowledge.

To efficiently process the anticipated massive influx of data from the Human Genome Project, we believe that utilizing parallel processing can be more beneficial than improving existing sequential algorithms in the long run. The reason is that genetic databases might grow to a point that the fastest sequential algorithm will still not be efficient enough. In addition to speed, parallelism also allows multiple learning systems to run concurrently and potentially improve the quality of learning. It is our conjecture that parallelism is vital in coping with future needs in the sequence analysis community as well as the machine learning community.

References

Bishop, M. and Rawlings, C. (1987). *Nucleic Acid and Protein Sequence Analysis.* IRL Press, Oxford, England.

Bond, A. and Gasser, L., editors (1988). *Readings in Distributed Artificial Intelligence.* Morgan Kaufmann, San Mateo, CA.

Booker, L., Goldberg, D., and Holland, J. (1989). Classifier systems and genetic algorithms. *Artificial Intelligence*, 40:235–282.

Boose, J. (1986). *Expertise Transfer for Expert System Design.* Elsevier, Amsterdam, Netherlands.

Breiman, L., Friedman, J. H., Olshen, R. A., and Stone, C. J. (1984). *Classification and Regression Trees.* Wadsworth, Belmont, CA.

Buchanan, B., Barstow, D., Bechtel, R., Bennett, J., Clancey, W., Kulikowski, C., Mitchell, T., and Waterman, D. (1983). Constructing an expert system. In Hayes-Roth, F., Waterman, D., and Lenat, D., editors, *Building Expert Systems*, pages 127–167. Addison-Wesley, Reading, MA.

Carbonell, J. (1983). Learning by analogy: Formulating and generalizing plans from past experience. In Michalski, R., Carbonell, J., and Mitchell, T., editors, *Machine Learning: An Artificial Intelligence Approach*, pages 137–161. Morgan Kaufmann, Los Altos, CA.

Carbonell, J. (1989). Introduction: Paradigms for machine learning. *Artificial Intelligence*, 40:1–9.

Chou, P. and Fasman, G. (1978). Prediction of the secondary structure of proteins from their amino acid sequence. *Adv. Enzymol. Relat. Areas Mol. Biol.*, 47:45–148.

Clark, P. and Niblett, T. (1987). The CN2 induction algorithm. *Machine Learning*, 3:261–285.

Cohen, P. and Feigenbaum, E., editors (1982). *The Handbook of Artificial Intelligence*, volume 3, chapter XIV, pages 323–511. William Kaufmann, Los Altos, CA.

Cost, S. and Salzberg, S. (1990a). Exemplar-based learning to predict protein folding. In *Proc. Sym. Comp. Appl. Medical Care*, Washington, DC.

Cost, S. and Salzberg, S. (1990b). A weighted nearest neighbour algorithm for learning with symbolic features. Technical Report JHU-90/11, Department of Computer Science, Johns Hopkins University, Baltimore, MD.

DeLisi, C. (1988). The human genome project. *American Scientist*, 76:488–493.

Ellman, T. (1989). Explanation-based learning: A survey of programs and perspectives. *Computing Surveys*, 21:163–221.

Fisher, D. and Langley, P. (1985). Approaches to concetual clustering. In *Proc. IJCAI-85*, pages 691–697.

Garnier, J., Osguthorpe, D., and Robson, B. (1978). Analysis of the accuracy and implications of simple methods for predicting the secondary structure of globular proteins. *J. Mol. Biol.*, 120:97–120.

Gascuel, O. and Danchin, A. (1986). Protein export in prokaryotes and eukaryotes: Indications of a difference in the mechanism of exportation. *J. Mol. Evol.*, 24:130–142.

Haussler, D. and Pitt, L., editors (1988). *Proceedings of the 1988 Wrokshop on Computational Learning Theory*, San Mateo, CA. Morgan Kaufmann.

Hinton, G. (1989). Connectionist learning procedures. *Artificial Intelligence*, 40:185–234.

Holland, J. (1975). *Adaptation in Natural and Artificial Systems.* University of Michigan Press, Ann Arbor, MI.

Holland, J. (1986). Escaping brittleness: The possiblilities of general-purpose learning algorithms applied to parallel rule-based systems. In Michalski, R., Carbonell, J., and Mitchell, T., editors, *Machine Learning: An Artificial Intelligence Approach (Vol. 2)*, pages 593–623. Morgan Kaufmann, Los Altos, CA.

Holley, L. H. and Karplus, M. (1989). Protein secondary structure prediction with a neural network. *Proc. Natl. Acad. Sci. USA*, 86:152–156.

Hunter, L. (1991). Artificial intelligence and molecular biology. *AI Magazine*, 11(5):27–36.

King, R. (1987). An inductive learning approach to the problem of predicting a protein's secondary structure from its amino acid sequence. In *Progress in Machine Learning: Proc. Second European Working Session on Learning*, pages 230–250.

Lapedes, A., Barnes, C., Burks, C., Farber, R., and Sirotkin, K. (1990). Application of neural networks and other machine learning algorithms to DNA sequence analysis. In Bell, G. and Marr, T., editors, *Computers and DNA: The Proceedings of the Interface between Computation Science and Nucleic Acid Sequencing Workshop*, pages 157–182. Addison-Wesley, Redwood City, CA.

Lee, Y. and et al. (1986). Machine learning using a higher order correlation network. *Physica*, 22D:276–306.

Lim, V. (1974). Algorithms for prediction of α-helical and β-structural regions in globular proteins. *J. Mol. Biol.*, 88:873–894.

Lippmann, R. (1987). An introduction to computing with neural nets. *IEEE ASSP Magazine*, 5(2):4–22.

Michalski, R. (1983). A theory and methodology of inductive learning. In Michalski, R., Carbonell, J., and Mitchell, T., editors, *Machine Learning: An Artificial Intelligence Approach*, pages 83–134. Morgan Kaufmann, Los Altos, CA.

Michalski, R., Carbonell, J., and Mitchell, T., editors (1983). *Machine Learning: An Artificial Intelligence Approach*. Morgan Kaufmann, Los Altos, CA.

Michalski, R. and Stepp, R. (1983). Learning from observation: Conceptual clustering. In Michalski, R., Carbonell, J., and Mitchell, T., editors, *Machine Learning: An Artificial Intelligence Approach*, pages 331–363. Morgan Kaufmann, Los Altos, CA.

Minksy, M. and Papert, S. (1969). *Perceptrons: An Introduction to Computation Geometry*. MIT Press, Cambridge, MA.

Mitchell, T. (1982). Generalization as search. *Artificial Intelligence*, 18:203–226.

Mitchell, T., Keller, R., and Kedar-Cabelli, S. (1986). Explanation-based generalization: A unifying view. *Machine Learning*, 1:47–80.

Mitchell, T. M. (1980). The need for biases in learning generalizaions. Technical Report CBM-TR-117, Department of Computer Science, Rutgers University.

Nakata, K., Kanchisa, M., and DeLisi, C. (1985). Prediction of splice junctions in mRNA sequences. *Nucl. Acids Res.*, 13:5327–5340.

Nilsson, N. (1980). *Principles of Articficial Intelligence*. Tioga, Palo Alto, CA.

O'Neill, M. (1989). Esherichia coli promoters: II. A spacing class-dependent promoter search protocol. *J. Biol. Chem.*, 264:5531–5534.

Qian, N. and Sejnowski, T. (1988). Predicting the secondary structure of globular proteins using neural nework models. *J. Mol. Biol.*, 202:865–884.

Quinlan, J. R. (1986). Induction of decision trees. *Machine Learning*, 1:81–106.

Rendell, L. (1983). A new basis for state-space learning systems and a successful implementation. *Artificial Intelligence*, 20:369–392.

Rosenblatt, F. (1962). *Principles of Neurodynamics.* Spartan, New York, NY.

Rumelhart, D., Hinton, G., and Williams, R. (1986). Learning internal representations by error propagation. In Rumelhart, D. and McClelland, J., editors, *Parallel Distributed Processing*, volume 1, pages 318–362. MIT Press, Cambridge, MA.

Rumelhart, D. and McClelland, J., editors (1986). *Parallel Distributed Processing*, volume 1. MIT Press, Cambridge, MA.

Seshu, R., Rendell, L., and Tcheng, D. (1989). Managing constructive induction using optimization and test incoopration. In *Proc. Fifth Intl. Conf. Art. Intell. Appl.*, pages 191–187.

Slade, S. (1991). Case-based reasoning: A research paradigm. *AI Magazine*, 12(1):42–55.

Stanfill, C. and Waltz, D. (1986). Toward memory-based reasoning. *Comm. ACM*, 29(12):1213–1228.

Stormo, G., Schneider, T., Gold, L., and Ehrenfeucht, A. (1982). Use of the 'perceptron' algorithm to distinguish translational initiation sites in *E. coli*. *Nucl. Acids Res.*, 10:2997–3011.

Taylor, W. (1986). The classification of amino acid conservation. *J. Theor. Biol.*, 119:205–221.

Towell, G., Graven, M., and Shavlik, J. (1991). Constructive induction in knowledge-based neural networks. In *Proc. Eighth Intl. Workshop Machine Learning*, pages 213–217.

Towell, G., Shavlik, J., and Noordewier, M. (1990). Refinement of approximate domain theories by knowledge-based neural networks. In *Proc. AAAI-90*, pages 861–866.

Utgoff, P. (1986). *Machine Learning of Inductive Bias.* Kluwer Academic, Norwell, MA.

Valiant, L. (1984). A theory of the learnable. *Comm. ACM*, 27:1134–1142.

Vemuri, V., editor (1988). *Artificial Neural Networks: Theoretical Concepts.* IEEE, Washington, DC.

von Heijne, G. (1987). *Sequence Analysis in Molecular Biology.* Academic Press, San Diego, CA.

Waldinger, R. (1977). Achieving several goals simultaneously. In Elcock, E. and Michie, D., editors, *Machine Intelligence 8*, pages 94–136. Ellis Horwood.

Wilson, S. (1987). Classifier systems and the animat problem. *Machine Learning*, 2:199–228.