This paper is reprinted from *Computational Learning Theory and Natural Learning Systems*, vol. 3, T. Petsche, S. Hanson and J. Shavlik (eds.), 1995. Copyrighted 1995 by MIT Press.

Investigating the Value of a Good Input Representation

Mark W. Craven
Jude W. Shavlik
Computer Sciences Department
University of Wisconsin
1210 West Dayton St.
Madison, Wisconsin 53706
U. S. A.
phone: (608) 263-0475
email: {craven, shavlik}@cs.wisc.edu

Abstract

The ability of an inductive learning system to find a good solution to a given problem is dependent upon the representation used for the features of the problem. A number of factors, including training-set size and the ability of the learning algorithm to perform constructive induction, can mediate the effect of an input representation on the accuracy of a learned concept description. We present experiments that evaluate the effect of input representation on generalization performance for the real-world problem of finding genes in DNA. Our experiments that demonstrate that: (1) two different input representations for this task result in significantly different generalization performance for both neural networks and decision trees; and (2) both neural and symbolic methods for constructive induction fail to bridge the gap between these two representations. We believe that this real-world domain provides an interesting challenge problem for the machine learning subfield of constructive induction because the relationship between the two representations is well known, and because conceptually, the representational shift involved in constructing the better representation should not be too imposing.

1 Introduction

The ability of an inductive learning system to find a good solution to a given problem is often dependent upon the representation used for the features of the problem (Flann & Dietterich, 1986). However, there are a number of factors that can mediate the effect of an input representation on the accuracy of a learned concept description. These factors include the training-set size and the ability of the learning algorithm to perform constructive induction (Michalski, 1983). Constructive induction involves automatically constructing new features from given ones, thereby changing the problem representation.

In this paper we investigate, for a real-world problem, the difference in generalization performance that results from using two different input representations. We evaluate the efficacy of both connectionist and symbolic methods for constructive induction in bridging the gap between these two representations. Our experiments indicate that common feature-construction approaches are not able to compensate for the weaker input representation. We suggest that this real-world problem provides an interesting challenge problem for systems that perform constructive induction.

The real-world task that we discuss is the problem of identifying genes in DNA sequences. As part of the Human Genome Project, many biological laboratories are now in the process of ascertaining the DNA sequence of humans and several other organisms. One of the primary challenges of these efforts is to distinguish the more interesting parts of the DNA sequences from the parts with little or no functionality. Several researchers have addressed this problem using neural networks and have found that the input representation has a significant impact on the generalization performance of the trained networks (Craven & Shavlik, 1993; Farber et al., 1992; Lapedes et al., 1989). In this paper we further investigate two input representations in particular. The performance difference between these two representations is especially intriguing because the relationship between the representations is well known. Furthermore, the representational shift involved in going from the weaker representation to the stronger one involves only forming an appropriate set of conjunctions.

We present two sets of experiments. We first demonstrate the effect of input representation on generalization performance for this problem domain, using both a symbolic learning method (Quinlan's C4.5) and a neural learning technique (Rosenblatt's perceptrons) which lacks the ability to construct features. The second set of experiments evaluates two constructive-induction approaches on this problem. The first approach to constructive induction that we consider involves simply adding hidden units to the networks used in the previous experiment. One of the touted virtues of multi-layer artificial neural networks is that their hidden units are able to construct new features from the given input features (Rumelhart et al., 1986). The other system that we investigate is Citre (Matheus, 1990a), which performs constructive induction on decision trees. In our experiments, neither of these approaches are able to construct the features needed for good generalization in the gene-finding domain.

The next section provides a brief introduction to the biology underlying the problem of finding genes in DNA. Section 3 describes the two input representations used in our experiments, and presents an experiment that demonstrates the effect of input representation on generalization performance for this problem domain. In Section 4 we argue our case for why gene finding is a challenging problem for constructive-induction research. The following

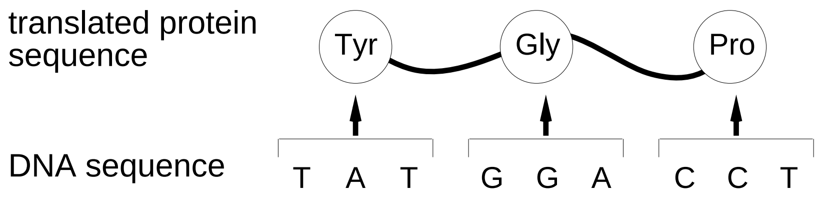

Figure 1: **DNA and protein translation.** Depicted here is a DNA sequence and the amino-acid sequence (protein) that results from its translation. Each bracket delineates a codon; collectively, the brackets show the reading frame for this sequence.

two sections investigate constructive-induction approaches to the gene-finding problem: Section 5 investigates using hidden units to construct features in neural networks, and Section 6 explores using Citre to construct features for a decision-tree learning system. The final section discusses the significance of these experiments and provides conclusions.

2 The Problem Domain

This section provides a brief description of the problem that serves as a testbed for our experiments. A more thorough treatment of the biology underlying the problem can be found elsewhere (Watson et al., 1987).

A DNA strand is a linear sequence of *nucleotides* composed from the alphabet {A, G, T, C}. A DNA molecule comprises two strands organized as a double helix. Certain subsequences of a DNA strand, called *genes*, encode *proteins*. Proteins are important because they provide most of the structure, function, and regulatory mechanisms of cells. Interspersed between the genes are areas, termed *noncoding regions*, that do not encode proteins. An important problem in biology is to be able to distinguish the coding from the noncoding regions of DNA sequences.

Proteins are also linear sequences; they are composed from the 20-character alphabet of *amino acids*. As illustrated in Figure 1, each consecutive string of three nucleotides in a gene encodes a single amino acid (e.g., "GGA" encodes glycine). The nucleotide triplets are called *codons* and the mapping from codons to amino acids is called the *genetic code*. The genetic code is almost universal across species and is well known. The process of translating a gene into protein involves grouping nucleotides into codons and inserting the amino acid encoded by each codon into the protein chain being synthesized.

In order to determine the amino-acid sequence encoded by a given DNA sequence, it is necessary to know the *reading frame* of the sequence. The *reading frame* refers to how the nucleotides of a DNA sequence are grouped into triplets as a gene is translated. As an analogy, consider trying to decode a bit stream that contains a message encoded in ASCII. Unless the bits are correctly grouped into bytes, the decoded message will be nonsense.

The problem that we address is the following: given a relatively small, fixed-length "window" on a DNA sequence, predict whether or not the sequence is part of a gene that is "in-frame " with the window. The window is considered to be in-frame with a gene when the leftmost nucleotide in the window is the first nucleotide in a codon of the translated gene. Thus the problem involves classifying input sequences into two classes: *coding* and

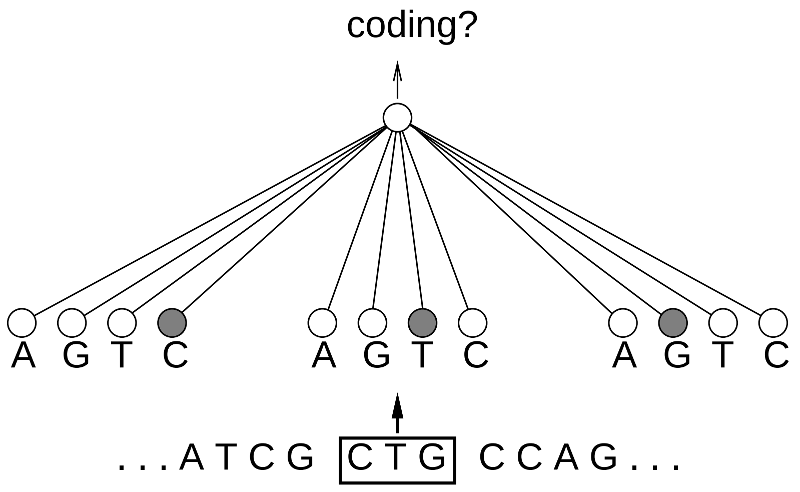

Figure 2: **Representing the input window as nucleotides using a local encoding.** The nucleotides in the window determine the activations of the input units. Shaded input units have activations of 1, the other input units have activations of 0. In this figure, the input window is three nucleotides wide.

noncoding. The classifiers need to learn to recognize coding regions only in one frame; all out-of-frame sequences are treated as *noncoding* examples. Other researchers have also investigated neural-network approaches (Farber et al., 1992; Lapedes et al., 1989; Uberbacher & Mural, 1991) to the problem of finding genes in DNA sequences.

3 The Effect of Input Representation

As part of the Wisconsin *E. coli* Genome Project (Daniels et al., 1992), we have been investigating the use of neural networks to find genes in DNA sequences of the bacterium *E. coli* (Craven & Shavlik, 1993). In the course of this research, we have found that the choice of input representation has a significant effect on how well the task is learned; other researchers have reached the same conclusion (Lapedes et al., 1989). It was this finding that motivated us to explore the effect of input representation on generalization performance. In this section we describe the two different representations that are used for DNA sequences in our experiments.

Both representations that we investigate represent DNA sequences as feature-value pairs. The first approach uses a binary encoding of the nucleotides that are present in the input window. This input representation requires four features for every nucleotide position in the input window, where each feature represents one of the four nucleotides that could occupy the position. We will refer to this as the *nucleotides* representation. The second approach involves representing the codons that are present in the window and are in-frame with respect to the window. This input representation involves sixty-four binary features for every codon position in the window, where each feature represents one of the codons that could occupy the position. We will refer to this as the *codons* representation. Figure 2 and Figure 3 depict artificial neural nets using the *nucleotides* and *codons* representations respectively.

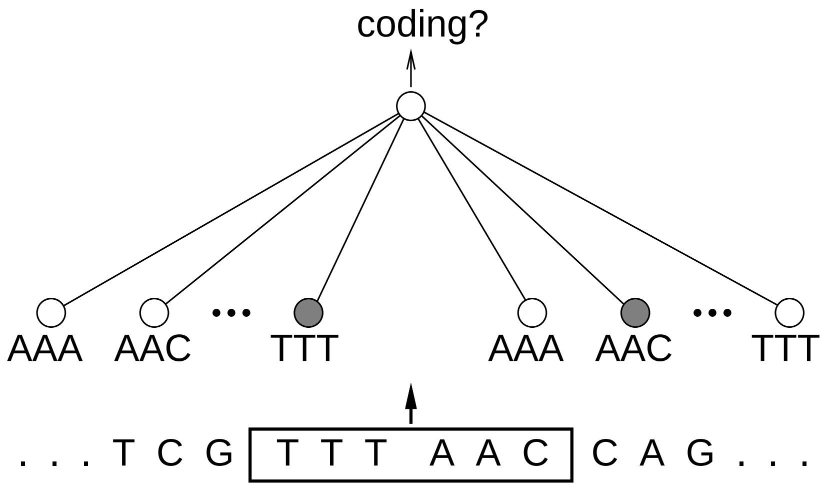

Figure 3: **Representing the input window as codons.** There are sixty-four input units for each codon in the window. In this figure, the input window is two codons wide.

In order to evaluate the effect of input representation for the problem of recognizing genes in DNA, we construct generalization curves for both the *nucleotides* and the *codons* representations. A generalization curve plots test-set error on the y-axis against training-set size on the x-axis. We construct these curves using the C4.5 decision tree algorithm (Quinlan, 1993), and perceptrons (Rosenblatt, 1958). A perceptron is a neural network with no hidden units and a linear-threshold output unit.

A window size of 15 nucleotides is used to determine the input features for both representations. For both representations, classifiers are trained on example sets that range from 100 to 10,000 examples. For a given run, each successive training set is a superset of the previous training set. The classifiers are tested on a disjoint set of 5,000 examples after learning each training set, and the accuracies on this set are plotted. The results are averaged over four runs for each training-set size. All training and testing sets contain approximately 50% *coding* sequences and 50% *noncoding* sequences.

The networks are actually trained using sigmoidal output units instead of linear threshold units, so technically, they are not perceptrons during the learning process. When we use the networks to classify instances, however, we treat the output units as threshold units, and hence the networks can be considered to be perceptrons. We train networks until convergence using a conjugate-gradient algorithm (Kramer & Sangiovanni-Vincentelli, 1989). Conjugate-gradient learning obviates the need for learning-rate and momentum parameters. A *tuning set* consisting of 10% of each training set is used to determine when the network weights are saved so that networks do not "overfit" the training data. Overfitting means that a network has represented too much about specific examples in the training set and not enough about the general characteristics of the training set. Error is not backward-propagated and weights are not updated for members of the tuning set; they are simply classified by the network in order to estimate the accuracy of the network on unseen examples. For the decision trees, pessimistic pruning (Quinlan, 1993) is used to avoid overfitting.

Figure 4 shows the observed generalization curves for perceptrons using the *nucleotides* and *codons* input representations. Figure 5 shows the observed generalization curves for

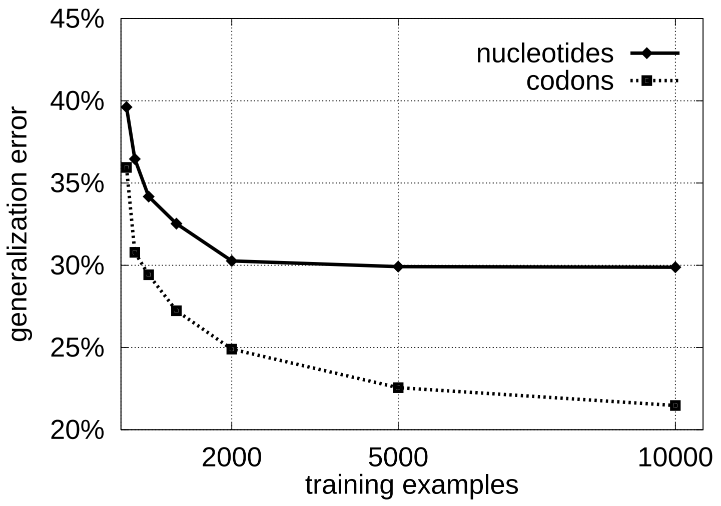

Figure 4: **Perceptron generalization curves for the *nucleotides* and the *codons* representations.** The solid line shows the observed generalization curve for the *nucleotides* representation. The dashed line shows the observed generalization curve for the *codons* representation. *Generalization error* is the percentage of misclassified test-set examples.

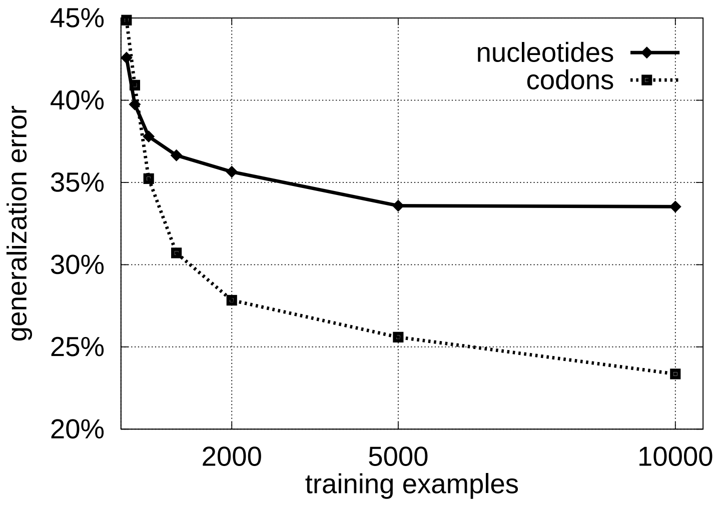

Figure 5: **C4.5 generalization curves for the *nucleotides* and the *codons* representations.** The solid line shows the observed generalization curve for the *nucleotides* representation. The dashed line shows the observed generalization curve for the *codons* representation. *Generalization error* is the percentage of misclassified test-set examples.

decision trees using both representations. For both learning methods, using the *codons* representation results in classifiers that generalize substantially better than those using the *nucleotides* representation.

4 Gene Recognition and Constructive Induction

Because the defining characteristics of genes are more readily apparent at the codon level than the nucleotide level, the *codons* representation of DNA sequences is obviously a better representation for the task of recognizing genes. However, what if we did not know *a priori* that the *codons* representation was a reasonable input representation to try for this task? Would this representation be discovered by state-of-the-art learning systems that perform constructive induction?

There are several reasons why the problem of finding genes provides an interesting testbed for constructive-induction research. First, it is a real-world problem that can be "reverse-engineered." We know that codons provide a good representation for the problem, and the mapping between nucleotides and codons is known. Second, the representational shift involved in going from nucleotides to codons is neither trivial nor overly-complex. The features of the *codons* representation, namely the codons themselves, are simply ternary conjunctions of adjacent nucleotides. Thus we would expect that a fairly general feature-construction algorithm would be able to construct codon features from the nucleotide features. Third, the process of finding the *codons* representation is, to some extent, a problem of scientific discovery. The process of "cracking" the genetic code in the early 1960's involved discovering the mapping between sequences of nucleotides and sequences of amino acids. Two parts of this discovery, in particular, are manifested in the process of learning the *codons* representation: (1) determining that each consecutive nucleotide triplet encodes a single amino acid; and (2) determining that the code is non-overlapping (i.e., , codons do not overlap each other).

In the following sections we describe experiments that involve applying both neural and symbolic constructive-induction approaches to the problem of recognizing genes. The question that motivates these experiments is whether or not general methods for constructive induction are able to discover a good representation (e.g., the *codons* representation) for this problem when given only the *nucleotides* representation?

5 Constructing Features in Neural Networks

The neural networks used in the previous experiment were perceptrons, meaning that they did not have any hidden units. When used for classification tasks, such as the gene-recognition problem, perceptrons are able only to make linear discriminations in their input space. For many problems, a linear boundary may be inadequate for separating instances of different classes. The role of hidden units in a neural network is to transform the input space into a different representation – one in which two classes may be separated by a single linear boundary. Thus, the concepts represented by hidden units can be thought of as constructed features. The question that we address in this section is whether or not neural networks using the *nucleotides* representation, when given a sufficient number of hidden units, are

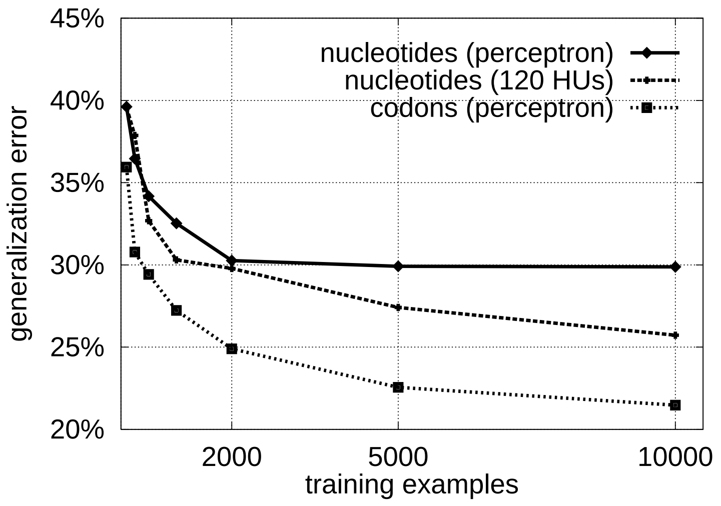

Figure 6: **Generalization curves for multi-layer networks and perceptrons.** The curves show observed generalization for 120-hidden-unit networks using the *nucleotides* representation and for perceptrons using the *nucleotides* and the *codons* representations.

able to construct features that enable them to approach the generalization performance of networks using the *codons* input representation. It is important to note that we are not particularly concerned with exactly what features the networks learn to encode with their hidden units. It is possible that the hidden units could learn a local encoding of codons, or they may learn a distributed representation (Hinton et al., 1986) of codons, or some entirely different, but useful, set of features.

In order to investigate this question, we construct a set of generalization curves using networks with different numbers of hidden units. The methodology and the data sets used to train and test the multi-layer networks are the same as in the previous experiment. The networks that we test have 5, 10, 20, 40, 80, 120 and 160 hidden units. The hidden units are arranged in a single layer and each unit is fully-connected to the set of input units. The number of free parameters (weights + biases) in a neural network gives a rough indication of the *capacity* of the network (Baum & Haussler, 1989). The 40-hidden-unit networks used in this experiment have 2481 parameters. In contrast, it would take only 1601 parameters to encode by hand a network that had a layer of 320 hidden units hard-wired to represent each of the 64 codons in each of the five codon positions in the window. Thus, although the networks in this experiment do not provide a topology that enables them to form a local representation of the codons in the window, they should have sufficient complexity to form an alternative, distributed encoding of similar features.

Figure 6 shows the generalization curve for the networks with 120 hidden units as well as the generalization curves for the perceptrons used in the first experiment. The networks with 120 hidden units provided the best test-set performance of the multi-layer networks evaluated in this experiment. Although networks with hidden units offer an improvement over the generalization performance of perceptrons using the *nucleotides* representation, this improvement is not enough to match the performance of the perceptrons using the *codons* representation. This result indicates that the fully-connected networks are failing to represent codons with their hidden units.

6 Constructing Features in Decision Trees

A number of algorithms have been developed to perform feature construction using decision trees as the concept description language (Matheus & Rendell, 1989; Pagallo & Haussler, 1990). The CITRE system (Matheus & Rendell, 1989) provides a general approach for constructive induction on decision trees. In this section we describe an experiment in which we train decision trees using the *nucleotides* representation, and then use CITRE to construct features on these trees. The motivation for this experiment is to see if a symbolic system for constructive induction, such as CITRE, is able to bridge the generalization gap between the *nucleotides* representation and the *codons* representation.

The CITRE approach to feature construction involves an iterative cycle of learning a decision tree, constructing new features, and then learning a new tree using the constructed and original features. CITRE uses a learned decision tree to suggest constituent features to be used in the constructed features. As outlined by Matheus, the CITRE approach has four aspects:

1. the *detection* of when new features are required
2. the *selection* of relationships used to define new features
3. the *generalization* of new features
4. the global *evaluation* of constructed features

Below we discuss how we address these four aspects in our experiment.

Matheus' criterion for *detection* is that feature construction should be performed whenever disjunctive regions, as evidenced by the presence of more than one positively-labelled leaf, are detected in a decision tree (for our purposes, a positively-labelled leaf is one labelled *coding*). We believe that for many real-world tasks this is probably too stringent of a criterion because it requires the concept to eventually be represented as a conjunction. In our experiment, we finesse the issue of defining a good detection criterion; instead we just try to estimate a lower bound on the generalization error over a fixed number of iterations of tree building and feature construction.

The *selection* process involves forming conjunctions of pairs of Boolean features. Matheus describes a number of ways in which pairs of features can be selected (Matheus, 1990a). In this experiment we use the *adjacent* method, which selects all adjacent pairs of tests that occur on decision-tree branches that lead to positively-labelled leaves. For example, consider a branch in a decision tree that leads to a leaf labelled *coding*. If one node along this branch tests if the first nucleotide in the window is 'A', and the next node tests if the third nucleotide is 'C', then the *adjacent* operator would construct a new Boolean feature that is true when the first nucleotide is 'A' and the third nucleotide is 'C'. The selection process can also exploit domain knowledge to narrow the set of candidate constructed features. In this

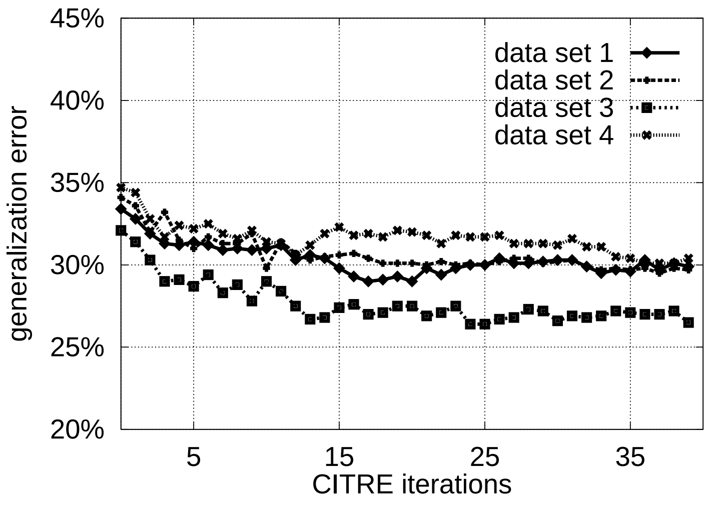

Figure 7: **Generalization for the CITRE trees.** The initial set of features is simply the *nucleotides* representation. The x-axis represents the number of feature-construction iterations, and the y-axis represents the test-set error for a decision tree induced using the features of a given iteration. Fixed-size training sets of 5,000 examples are used to generate the trees.

experiment we do not use any domain knowledge in this process because we are interested in determining how well we can do without using such information.

The *generalization* step enables Citre to generalize the features constructed in the selection process. For example, constructed features could be generalized by introducing disjunctions or by dropping terms from conjunctions. (Obviously, such generalization operators should be restricted so that they do not simply undo steps that were just performed by the *selection* process.) In this experiment we do not perform any generalization of constructed features.

The *evaluation* process serves to limit the number of new features constructed. The evaluation criterion we use is the information gained when an individual feature is used to partition the entire training set. For this experiment we limit the number of constructed features to 320 at any given time. Since there are 60 original features, the total number of features never exceeds 380. When the number of features is restricted in this manner, Citre can be thought of as performing a beam search through the space of constructed features. Note that 320 features is the number necessary to represent each codon in each position in the input window.

In order to evaluate the ability of Citre to construct useful features, we test the feature-construction process starting only with nucleotides as features. In this experiment we use four fixed-size training sets of 5,000 examples. The training and test sets are the same 5,000-example sets used in the previous experiment. For each training set, we run Citre for 40 iterations of feature construction.

Figure 7 shows the test set error for each of the four training sets on each iteration of Citre. From this figure it can be seen that the generalization performance fluctuates as

Table 1: **Generalization error for the decision tree and neural network approaches.** Reported values are averages for four training sets of 5,000 examples each. The CITRE result is an average of the values from the iterations with the *best* performance for each data set.

approach	% error
C4.5 with *nucleotides*	33.6
CITRE starting with *nucleotides*	28.8
C4.5 with *codons*	25.6
perceptron with *nucleotides*	29.5
120 HU network with *nucleotides*	26.7
perceptron with *codons*	22.6

the feature set changes. Over the course of the feature-construction process, however, test set accuracies do improve somewhat. Table 1 shows the generalization performance of the CITRE-induced trees relative to the performance of C4.5 trees using both representations. The numbers in this table represent averages over all four test sets. The CITRE error rate was determined by taking the tree from the iteration with the *best* performance for each data set. Thus, the error rate for the CITRE approach is a lower bound on what it would be if we used, say, a tuning set to decide when to stop iterating. All differences in this table (except CITRE vs. the *nucleotides* perceptrons and CITRE vs. 120-hidden unit networks) are significant to at least the 0.025 level using a paired, 1-tailed t-test.

The results of this experiment indicate that CITRE, when given no domain knowledge, is able to improve upon the performance of a decision tree using only the *nucleotides* representation, but that this improvement is not enough to match the performance of trees using the *codons* representation. We have also conducted this experiment using the *fringe* feature selection method (Pagallo & Haussler, 1990), and *competitive evaluation* of features (Matheus, 1990b). However, we found that the *adjacent* selection method and information-based evaluation provided the best results. Further experimentation needs to be conducted to explore the effects of using domain knowledge, feature-generalization operators, and more iterations of feature construction.

7 Conclusions

We have investigated the effect of input representation on generalization performance for the real-world problem of finding genes in DNA sequences. Our experiments demonstrate that two different input representations result in significantly different generalization performance for neural networks and decision trees trained using them. We investigated the ability of learning algorithms that perform constructive induction to achieve the generalization performance of the *codons* representation given the weaker *nucleotides* representation. Our experiments indicate that neither multi-layer neural networks nor the symbolic CITRE algorithm perform the representational shift necessary to bridge the generalization gap between the two representations.

We believe that the task of finding genes in DNA is an interesting problem for research in representation and constructive induction for several reasons. First, it embodies the complexity inherent in an important real-world problem. Second, two different, yet natural, input representations result in significantly different generalization performance for neural networks and decision trees trained using them. Finally, the relationship between the two representations is well known, and the representational shift involved in constructing the better one is not too imposing.

An interesting question to address in future research is to determine what is necessary to enable multi-layer neural networks and CITRE to successfully construct features, such as codons, that provide a good representation for the gene-finding problem. Domain knowledge, changes to the feature-construction algorithms, and topological hints are among the issues that could be investigated. One obvious piece of domain knowledge that could be exploited is information about the sequential nature of DNA. For example, in a neural network we could connect hidden units so that each one is connected only to a spatially-local part of the input window. Similarly, we could bias CITRE so that conjunctions of neighboring nucleotides are preferred. Although we know the mapping between nucleotides and codons, it is important to find feature-construction methods which are as general as possible so that they can be extended to problems in which a good representation is not known.

The issue of input representation is an important one in machine learning. Often it is not obvious what the best input representation is for a given problem. It is therefore important to develop learning techniques that are able construct better representations when they are initially given weak representations. We believe that the problem of finding genes in DNA can help guide research in constructive induction towards its goal of finding effective domain-independent biases for feature construction.

8 Acknowledgements

Insightful comments by Chris Matheus contributed to this research and its presentation. Dave Opitz and two anonymous reviewers provided helpful comments on an earlier version of this paper. This work was partially supported by Department of Energy Grant DE-FG02-91ER61129, National Science Foundation Grant IRI-9002413, and Office of Naval Research Grant N00014-90-J-1941.

References

Baum, E. B. & Haussler, D. (1989). What size net gives valid generalization? *Neural Computation*, 1:151–160.

Craven, M. W. & Shavlik, J. W. (1993). Learning to predict reading frames in *e. coli* DNA sequences. In *Proceedings of the 26th Hawaii International Conference on System Sciences*, (pp. 773–782), Maui, HI. IEEE Computer Society Press.

Daniels, D. L., Plunkett III, G., Burland, V. D., & Blattner, F. R. (1992). Analysis of the Escherichia coli genome: DNA sequence of the region from 84.5 to 86.5 minutes. *Science*, 257:771–778.

Farber, R., Lapedes, A., & Sirotkin, K. (1992). Determination of eucaryotic protein coding regions using neural networks and information theory. *Journal of Molecular Biology*, 226:471–479.

Flann, N. S. & Dietterich, T. G. (1986). Selecting appropriate representations for learning from examples. In *Proceedings of the Fifth National Conference on Artificial Intelligence*, (pp. 460–466), Philadelphia, PA. Morgan Kaufmann.

Hinton, G. E., McClelland, J. L., & Rumelhart, D. E. (1986). Distributed representations. In Rumelhart, D. E. & McClelland, J. L., editors, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition.* MIT Press, Cambridge, MA.

Kramer, A. H. & Sangiovanni-Vincentelli, A. (1989). Efficient parallel learning algorithms for neural networks. In Touretzky, D., editor, *Advances in Neural Information Processing Systems (volume 1).* Morgan Kaufmann, San Mateo, CA.

Lapedes, A., Barnes, C., Burks, C., Farber, R., & Sirotkin, K. (1989). Application of neural networks and other machine learning algorithms to DNA sequence analysis. In Bell, G. & Marr, T., editors, *Computers and DNA, SFI Studies in the Sciences of Complexity, vol. VII.* Addison-Wesley.

Matheus, C. J. (1990a). Adding domain knowledge to SBL through feature construction. In *Proceedings of the Eighth National Conference on Artificial Intelligence*, (pp. 803–808), Boston, MA. MIT Press.

Matheus, C. J. (1990b). *Feature Construction: An Analytic Framework and an Application to Decision Trees.* PhD thesis, University of Illinois at Urbana-Champaign.

Matheus, C. J. & Rendell, L. A. (1989). Constructive induction on decision trees. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, (pp. 645–650), Detroit, MI.

Michalski, R. S. (1983). A theory and methodology of inductive learning. *Artificial Intelligence*, 20:111–161.

Pagallo, G. & Haussler, D. (1990). Boolean feature discovery in empirical learning. *Machine Learning*, 5:71–99.

Quinlan, J. R. (1993). *C4.5: Programs for Machine Learning.* Morgan Kaufmann, San Mateo, CA.

Rosenblatt, F. (1958). The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65(6):386–407.

Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986). Learning internal representations by error propagation. In Rumelhart, D. E. & McClelland, J. L., editors, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition.* MIT Press, Cambridge, MA.

Uberbacher, E. C. & Mural, R. J. (1991). Locating protein coding regions in human DNA sequences by a multiple sensor – neural network approach. *Proceedings of the National Academy of Sciences*, 88:11261–11265.

Watson, J. D., Hopkins, N. H., Roberts, J. W., Steitz, J. A., & Weiner, A. M. (1987). *Molecular Biology of the Gene (volume I).* Benjamin/Cummings, Menlo Park, CA, fourth edition.