Representation Problems in Machine Learning: A Proposal

Paul E. Utgoff, Sharad Saxena
James P. Callan, Tom E. Fawcett
Department of Computer and Information Science
University of Massachusetts
Amherst, MA 01003 U.S.A.

COINS Technical Report 89-23
March 15, 1989

Contents

Abstract

The project investigates methods for automatically constructing or shifting representations for machine learning. The representation in which generalizations must be expressed is a fundamental bias that strongly influences the inferences that a learning program makes. For effective learning, a program needs an appropriate representation. Currently, learning programs are not able to find an appropriate representation. Instead, a person must hand-craft a representation and then assess the effectiveness of the learning for that. An intelligent learning program should be able to perform this activity autonomously.

Issues addressed include use of multiple representations, concept formation, feature generation, feature discovery, representation evaluation, and evaluation function learning. Expected results include improved ability to construct hybrid representations, to construct nonlinear evaluation functions from qualitative criticism, to evaluate and select representations with domain-independent metrics, to generate and select plausible features, and to discover features via interestingness heuristics during problem solving.

1 Issues

A longstanding goal of the research in Machine Learning is to understand how to build a problem solving system that improves with experience. A problem solving system is given an initial state, a set of operators for transforming states, and a definition for a goal state. The problem solving task is to find a sequence of operators that would successively transform the initial state into a goal state. To build an efficient problem solving system, one needs to build an efficient search algorithm for finding a solution sequence. It follows that to build a problem solving system that improves with experience, one needs to build a mechanism that improves the efficiency of the search algorithm, based on experience.

There are several broad issues in learning from problem solving experience. One concerns *what* is to be learned. There exist many problem solving algorithms. Each makes control decisions based on available knowledge, intermediate states that are produced during search, and a strategy for deciding what to do next. Whatever determines problem solving behavior must be changed so that problem solving behavior improves. A second issue concerns *how* the learning will be accomplished. Fortunately, there are many existing applicable learning algorithms. Unfortunately, all require an appropriate hand-crafted representation for instance descriptions and concept descriptions. With an appropriate representation, the existing learning algorithms are generally very effective. Without an appropriate representation, useful concept descriptions are not found. This leads to a third issue: given that a learning algorithm is not learning effectively, how can a better representation be found so that learning will be improved. Very little work has addressed this critical issue. The problem of finding an appropriate representation is the focus of the research project.

The remainder of this section has four parts. The first describes the relationship between appropriate representation and effective problem solving. The second reviews several learning systems, noting how the representation problem was handled in each case. The third discusses the existing work on finding an appropriate representation. Finally, the principal issues are summarized.

1.1 Representation and Learning Efficacy

When searching for a general rule (concept) that is consistent with the observations (instances), the space of possible rules is determined by the describable attributes of the instances and the describable combinations of those attributes. A learner assumes that the observed instances are representative of a larger population. The learner tries to find a rule that is correct for the observed instances and that extrapolates to the entire population. Consider a simple example. The observed instances are English words, some of which satisfy an unstated rule, called the *target concept*, that is to be inferred. The positive instances (examples) are {"dog", "turkey", "porpoise"}, and the negative instances (counterexamples) are {"block", "water", "curtain"}. People already know many attributes that can be applied to words. This kind of prior knowledge illustrates how the learning process is (and needs to be) biased. One rule that is consistent with the instances is "a word that indicates an animate object". Another rule is "a word containing a letter extending below the line". These and many other rules are consistent with the observed instances, but which should be chosen? This raises the issue of inductive bias.

Any factor that causes a learner to prefer one consistent hypothesis (rule) to another is *bias* (Mitchell, 1980; Utgoff & Mitchell, 1982). A learning program can only consider rules that it can describe, so the provided description language imposes a bias. For example, if a program can evaluate the "animate" attribute for a given word, then it can consider a rule based on that attribute. On the other hand, if a program cannot evaluate the "letter extending below the line" attribute, then it may be difficult or impossible for the program to infer the correct rule. Note, it might infer an isomorphic rule like "a word containing at least one of the letters {g,j,p,q,y}". In general, one needs an adequate set of attributes for describing instances. How a program could obtain such attributes automatically is an open question. The state of the art remains that attributes are hand-crafted. A well known illustration of the problem is Quinlan's effort of two man-months to find a good set of attributes for describing a certain class of chess positions. After he had settled on a set of meaningful attributes, his ID3 algorithm found a correct rule in a matter of minutes (Quinlan, 1983).

Further bias is possible for deciding among the describable rules. Michalski allows a human user to specify a bias, based on meta-attributes about concept descriptions, such as minimizing the number of disjuncts (Michalski, 1983). Rendell calls this "paramaterized bias" (Rendell et al, 1987). Other researchers build in this and other domain-independent biases. A major bias embraced by many is toward simpler rules, based on the principal of Occam's Razor. In the above example, the "animate object" rule appears simpler than the "some letter below the line" rule, but a precise simplicity metric is needed to make such decisions. Quinlan's ID3 employs a domain-independent bias based on information theory that leads to small decision trees. For his work, simplicity is equated with the size of the induced decision tree.

The connectionist learning methods also must be biased if the learner is to generalize from the observed instances (Lippman, 1987; Rumelhart & McClelland, 1986). In connectionist networks, bias is a function of the network architecture, the activation, propagation, and update rules, and various numeric learning parameters. For example, a network architecture with too many units will tend to learn by rote, while an architecture with too few units will generalize quickly but cease learning, not having found the target concept. Few (Gallant, 1986; Utgoff, 1988b) have attempted to address how network architecture can be altered dynamically

Bias is essential to learning, viz. (Mitchell, 1980; Utgoff & Mitchell, 1982; Watanabe, 1985). Otherwise, there is no guidance in choosing from among the possible consistent hypotheses. No guidance would mean that people would be unable to extrapolate from the observed instances, and people wouldn't survive long if that were true. The essence of concept learning is to be able to capture useful regularities. An important research problem is to identify automatic methods that enable a program to shift representation (bias) as necessary so that useful regularities are captured and learning remains effective. This idea is exemplified in Shaefer's ARGOT system, which dynamically alters scale and precision of numeric parameters it is optimizing (Shaefer, 1988). Shift of representation means changing the set of describable attributes or describable combinations of attributes. Such change is also known as *constructive induction* (Dietterich & Michalski, 1981). People do not know how they do this themselves, so there is no methodology or introspective model to follow.

1.2 Learning from Problem Solving Experience

This section observes how the representation problem was addressed in several learning systems.

1.2.1 Learning State Preference

The earliest example of a program that learns from problem solving experience is Samuel's original checkers player (Samuel, 1959; Samuel, 1967). It used a numeric evaluation function $h(x)$, which maps a state x to a numeric value. The value is an estimate of the given state's strategic and tactical merit. Samuel's evaluation function was a linear combination of a weight vector $\mathbf{w}$ and a feature vector of functions $\mathbf{f}(\mathbf{x})$, with each $f_i(x)$ handcoded by Samuel. The value of a state x was computed as $h(x) = \mathbf{w} \cdot \mathbf{f}(x)$. Various strategies were provided for assigning credit or blame to individual states, from which $\mathbf{w}$ was altered so that the $h()$ would be in better agreement with such criticism. The tuning of the weights worked well, but Samuel found that a linear combination of his features was not sufficient to represent a correct evaluation function. To alleviate this problem, he tried two extensions. The first was a method for adding new features that were products of two primitive features. The second, called *signature tables*, divided the range of each primitive feature into a smaller set of discrete values. This coarse coding of the feature vector was then used to index a multi-dimensional table in which the value of the mapped state was stored. Both of these extensions were attempts to find a better representation so that the learning could have greater effectiveness.

Rendell's PLS1 (Rendell, 1983) learns an evaluation function based on penetrance. Penetrance is the proportion of nodes on the found solution path to total nodes produced in the search. The program learns an association between the feature values and expected penetrance. The performance element prefers to expand states with higher expected penetrance. The program includes a method for clustering groups of feature values into classes with similar expected penetrance. The features are hand-crafted and remain static. The system finds a locally optimal evaluation function based on penetrance, and is able to solve randomly generated 15-puzzle problems.

Lee and Mahajan built Bill (Lee & Mahajan, 1988), which learns an evaluation function for the game of Othello. The evaluation function is represented as a descriminant for won and lost positions. The learning algorithm is an implementation of Bayesian Learning (Duda & Hart, 1973), using four hand-crafted features that are known to be predictive of winning (Rosenbloom, 1982). The work is elegant, but points out that its contribution is on how to combine features, not how to discover or generate them.

The work on learning a preference predicate (Utgoff and Saxena, 1987a) showed that an evaluation function is a special case of a preference predicate. If $P(x, y)$ is a predicate that is $TRUE$ if and only if state x is preferred to state y, then $P(x, y) \leftrightarrow h(x) > h(y)$ is an implementation of $P()$ in terms of an evaluation function $h()$. By mapping an ordered pair of states (x, y) to a single instance of state preference i, it is then possible to apply concept learning methods to the problem of learning state preference. The technique was demonstrated for best-first search, applied to the eight-puzzle. The approach shows how concept learning techniques can be applied to learning state preference, thereby adding to the set of available representations and methods for learning state preference.

1.2.2 Learning Operator Preference

The principal contribution of LEX (Mitchell et al, 1983) was the idea that one could apply concept learning algorithms to the problem of learning control heuristics. For each problem solved, LEX would decompose the solution graph into individual steps that either led toward the solution or away from the solution. Each step along the solution path produced a positive instance for the operator that was applied in the step. Each step leading away from the solution path produced a negative instance for the associated operator. Problem solving improved because, as a result of refining the applicability conditions of the operators through learning, fewer operators were applicable to a given state. The correct applicability conditions were not completely learned because they were not representaable in the concept description language. The STABB subsystem (Utgoff, 1986), described below, was able to revise the concept description language, by adding useful new terms, but such changes could not capture concepts that were nowhere present in the concept description language or operator language. For example, one operator might be preferable to another because it is less expensive to apply. Distinctions based on expense could not be captured explicitly, because they were only implicit in the labelling of training instances by the Critic module (Keller, 1987). Other concepts from integral calculus were simply missing.

Minton's PRODIGY (Minton et al, 1987; Minton & Carbonell, 1987; Minton, 1988) learns control rules for a Strips-style problem solver. The learning method is principally explanation-based learning. In addition to an integration of problem solving and explanation-based learning, the system is able to evaluate the utility of the control rules it generates. This allows the system to discard rules that do not pay their way. This attention to utility was motivated by Minton's earlier experience with a system named MORRIS that became mired in control rules (Minton, 1984). The explanations that are produced are expressed in terms of the primitive concepts that are given to the system. Minton observes that these primitive concepts bias the explanations that PRODIGY can produce (see page 234 of (Minton & Carbonell, 1987)).

The SOAR program (Laird et al, 1986) is a problem solving model that includes a chunking learning mechanism. The problem solver selects from among the available problem spaces, operators, and states, and then applies the indicated operator to the state in the problem space. The decision is made by determining applicable preferences. Whenever an impasse in making such a decision is reached, a subgoal is created to resolve the impasse. Whenever a problem is solved at any level, the successful preferences are chunked. Chunks are generalized in two ways. First, constants can be replaced by variables. Second, the chunk only refers to what was referenced during the problem solving. The authors point out that this kind of generalization is bounded by the hand-provided representation for objects.

1.3 Representation Change

This section reviews representative research that has addressed selecting or changing representation for learning.

1.3.1 Multiple Representations

One approach to representation change is to provide a program with several possible representations and a control strategy for selecting or mixing representations. Flann and Dietterich have studied how to mix structural and functional representations (Flann and Dietterich, 1986). An important point they make is that a representation must be picked that is appropriate for how the concept will be used. If a concept is functional, then a functional representation is appropriate. Their WYL system converts structural descriptions to functional form, generalizes the functional form, and converts the generalization back to structural form. The functional form is appropriate for generalization and the structural form is appropriate for efficient recognition during problem solving.

Schlimmer's STAGGER mixes three representations (Schlimmer, 1987a; Schlimmer, 1987b). One representation is for boolean terms over the instance attributes. A second can create a new boolean term for a numeric-valued attribute by determining a constant threshold against which the attribute value is to be compared. Such a term is boolean because the numeric comparison of the attribute value to the threshold is true or false. The third representation is a numerical weighting of boolean terms constructed in either of the first two representations.

Utgoff's perceptron tree is either a linear threshold unit, or a decision node with branches to perceptron trees (Utgoff, 1988b). The algorithm attempts to represent a concept at a node as a hyperplane. If the instances at a node do not appear to be linearly separable, then the algorithm uses Quinlan's attribute selection heuristic to choose an attribute to place at the node. In subsequent training, new branches are grown as necessary. The approach makes it possible to learn a tree structured concept description incrementally, without saving training instances.

1.3.2 Finding Useful Classes

A second approach to representation change is to provide a program with the ability to construct useful new classes of instances. Such classes can be viewed as subconcepts, and are often denoted by a new name or term. Utgoff's STABB subsystem (Utgoff, 1986) was able to generate useful new classes by either of two methods. One method inserted new terms into a language hierarchy so that concepts that would otherwise need to be described as disjunctions could be described directly. The second method used a form of goal regression, and a solution sequence, to deduce the correct preimage for each tail of the operator sequence.

Each such preimage was a set that should be describable in the concept description language. For preimages that were not easily described, new terms were constructed. The system was able to create the terms "even integer" and "odd integer" in the domain of integral calculus.

Fisher's COBWEB system (Fisher, 1987) performs concept clustering by measuring the utility of possible partitions of the instance space. The system is incremental, updating and possibly revising its concept hierarchy as new concepts become available. An intriguing use of the system is to ignore the special status of the class name for each instance, and to treat it simply as another attribute value pair. This makes it possible to guess values for missing attributes by treating such attributes as the class name, i.e. the dependent variable.

1.3.3 Finding useful Features

A third approach to representation change is to provide a program with the ability to construct useful new attributes for describing instances. Pagallo & Haussler include a find-features procedure as part of their FRINGE procedure for constructing compact concept descriptions (Pagallo and Haussler, 1988). FRINGE uses ID3 (Quinlan, 1983; Quinlan, 1986) to construct a decision tree; then it calls find-features to generate new attributes. New attributes are always conjunctions of two existing attributes, but since $\frac{n^2-n}{2}$ combinations are possible, they cannot all be explored. Instead, find-features only combines the last two variables that appear in a path to a positive leaf node in the decision tree. The new attributes are then added to every training instance, and the process of generating a decision tree and finding attributes is repeated. ID3 determines the worth of each attribute when it builds a decision tree, so neither FRINGE nor find-features need to test the utility of the new attributes. FRINGE terminates when find-features is unable to find any new attributes.

Muggleton's DUCE (Muggleton, 1987) addresses the problem of proposing new terms that help produce compact generalizations of the examples expressed in propositional logic. The approach is to apply a set of transformation rules that propose generalizations of the examples. A typical transformation rule is to replace common sub-expressions in a set of examples by a single propositional variable. The user, called the oracle, is presented with these generalizations. If the generalization is valid, the oracle certifies it and gives a name for any new terms that the generalization may contain. For example, if replacing the common sub-expression in a set of examples by a propositional variable is a valid generalization of the examples, then the oracle would name the newly introduced propositional variable. In this manner, the utility of the new terms is assessed by the user and only useful new terms are accepted. In CIGOL (Muggleton & Buntine, 1988) these transformation rules have been extended for first order predicate logic. This approach was also suggested earlier in (Russell, 1986). The drawback of this approach is that each proposed new term has to be evaluated by an oracle. This makes it necessary to have an oracle who knows the target concept and at each stage can correctly identify a useful new term.

Rendell's probablistic learning system PLS0 constructs features in three stages (Rendell, 1985). In the first stage, a set of *primitive patterns* is constructed. A primitive pattern is an ordered pair (SS,UD), where SS is a subset of the primitive features and UD describes the utility for each combination of feature values. Because there are exponentially many SSs, clustering is used to focus attention on a subset of them (typically, about $2^{\frac{n}{3}}$). In the second stage, primitive patterns with similiar UD's are merged, resulting in a pattern class.

In the third stage, each pattern class is generalized. Transformations are applied to its SS's in order to fill in "missing" SS's. The result is called a pattern group. Features are formed directly from either a pattern class or a pattern group. A feature is evaluated for an object by projecting the object onto the feature's SS's and then summing the resulting UD's. The utility of a feature is reflected by its UD's.

1.3.4 Theoretical Work

In addition to the above three approaches, there has been some initial theoretical work. Analysis by Haussler and his colleagues (Haussler, 1986; Haussler, 1987; Blumer, et. al., 1987) of various classes of concept description languages has led to the quantification of the bias of that class. An example of a class, analyzed by these methods would be the class of k term, pure conjunctive concepts of n attributes. The bias of a concept description language is expressed in terms of its *Vapnik-Chervonenkis dimension* or simply its *dimension* (Vapnik and Chervonenkis, 1971; Pearl, 1978). The dimension of a concept description language is a combinatorial parameter that characterizes the number of concepts that can be expressed in a particular language. A lower value of the dimension implies that the concept description language has a stronger bias. The number of examples required to learn a concept, in the Valiant sense (Valiant, 1984), belonging to a particular concept description language increases with increasing dimension of the concept description language. This characterization of the bias of a concept description language makes it possible to determine which particular concept description language will be more suited for learning a concept in the Valiant sense.

Grosof has proposed that bias be represented declaratively with defaults and preferences (Russell & Grosof, 1987). Shift of bias would then be modelled by non-monotonic reasoning. In the same paper, Russell showed that Mitchell's Candidate Elimination Algorithm can be expressed as standard logical inference. This make it possible to incorporate background knowledge cleanly, especially determinations. A determination is an inference rule that defines how a generalization of one property can be inferred from the generalization of another (Russell, 1986). Russell's example is: if person a is from Italy and person b is from Italy and person a speaks Italian then person b speaks Italian. Thus, nationality determines language. By using a priori knowledge about which features of an instance are useful, determinations can select additional related features (Russell, 1988). Russell shows that the structure of the derivation of useful features can be used to form a tree-structured bias. Instead of learning a concept over the flat set of chosen instance features, a program can learn subconcepts over the features as indicated by the tree-structured bias. Such a decomposition of the learning problem is guided by the derivation through the determinations.

1.4 Summary

This section summarizes several important issues that were identified above. These issues are the foundation of the proposed research, and are to be addressed as subprojects as described below in section 2.

1. In general, the state of the art in constructing, selecting, and shifting representation is primitive. The representation problem is the major bottleneck in constructing autonomous learning systems. Currently, a human must iterate through representations,

noting the performance of the learning system so that improvements in representation can be determined for the next iteration. The process needs to be automated.

2. There is little methodology for selecting an appropriate representation. There are two avenues for progress. First, develop strategies for employing multiple representations, as in (Utgoff, 1988b; Schlimmer, 1987b). It would be useful to understand better how to pick a representation for a subconcept and how to combine representations into a hybrid scheme. Second, develop specific criteria for measuring the appropriateness of a representation for a given learning task. This involves investigation of fundamental mechanisms for detecting and measuring regularity.

3. Mechanisms for discovering or synthesizing attributes (features) are desparately needed. For too long, it has been accepted practice to hunt for good features manually. Researchers are now quite capable of building a program that will find a good concept in terms of the provided features, but quite incapable of having a program discover such features. For example, no program has yet discovered the concept of faulty pawn structure in chess. Mechanisms capable of such discovery of features are essential to effective learning, and are fundamental to intelligent behavior.

2 Approaches and Expected Results

This section proposes five subprojects that address the issues summarized in section 1.4.

2.1 Hybrid Representations and Algorithms

A recent paper (Utgoff, 1988b) shows how two representations can be combined effectively into a hybrid. The work was motivated by the requirements of learning a preference predicate, principally that the algorithm must be incremental and it must be able to handle a heavy stream of training instances. It was noted that the two best candidate representations, decision trees and linear threshold units, each had some shortcoming. A hybrid representation, called a *perceptron tree*, and a hybrid learning algorithm were constructed. A perceptron tree is either a linear threshold unit, or a decision node with branches to perceptron trees. The algorithm makes a choice whether to represent the concept at a node with a linear threshold unit or, instead, to replace the linear threshold unit with a decision node and branches to new linear threshold units. The bias for a linear threshold unit is that the concept must be represented as a hyperplane. The bias for attribute selection is the entropy measure used by Quinlan.

The current largest shortcoming of the representation is that all attributes are treated as nominal-valued. This means that numeric-valued attributes are not handled as well as they could be. Given that numeric values are ordered, unlike nominal values, additional inference is possible. For example, if high values of an attribute are predictive of one class, and low values are predictive of another, then one can draw inferences about unseen values, based on whether they are high or low. It is a mistake to ignore the natural ordering of numeric values. There is a general issue here of how to mix nominal-valued and numeric-valued attributes during concept learning.

Three approaches to improving the hybrid representation are to be considered. First, instead of encoding each numeric attribute-value pair as a binary feature (present or absent), simply use the numeric value directly as a numeric-valued feature. This is standard practice for perceptron learning. The problem is that numeric-valued attributes are difficult to handle in the decision tree formalism. It doesn't make sense to have a value branch for each possible value of a numeric-valued attribute. An approach taken by Quinlan (Quinlan, 1986), Schlimmer (Schlimmer, 1987a; Schlimmer, 1987b) and others is to compute a numeric threshold against which the value of the given attribute is to be compared. Thus, a binary feature (above or below) is constructed, which can be handled routinely in the decision tree formalism. Adding such a mechanism to the perceptron tree formalism is one possible fix.

Second, simply never place a numeric-valued attribute at a decision node. This is workable, except that the convergence guarantee of the existing perceptron tree algorithm would no longer apply. That is because there is no guarantee that the decision surface over the numeric-valued attributes is describable by a hyperplane. It would be necessary to provide a learning algorithm that is guaranteed to find a separating surface even if it must be non-linear. This could be handled by a generalized polynomial descriminant (Duda & Hart, 1973). A prototype algorithm for learning a polynomial descriminant has been implemented separately as part of the work described below in section 2.2.

Finally, discard the decision tree formalism in favor of an entirely numerical formalism. Instead of coercing numeric-valued attributes into the decision tree formalism, it may be possible to look for a best ordering on nominal values, assign numeric values according to the ordering, and then find a separating surface numerically. Along these lines, it may be possible to construct a tree with partially correct separating surfaces at each decision node, as suggested in (Breiman et al, 1984) and (Gallant, 1986).

A second shortcoming of the work to date is that no formal analysis or empirical comparisons, especially to ID5R (Utgoff, to appear) have yet been reported. Initial informal experiments indicate that the perceptron tree algorithm is somewhat slower than ID5R in terms of total training time. This is mitigated because ID5R retains the instances on which it has trained, and a perceptron tree does not. A good analysis of learning rate in linear threshold units is given in (Hampson and Volper, 1986). A question is how to characterize the overall learning rate in the hybrid formalism. The partitioning of the instance space at a decision node via an attribute test produces subspaces of reduced dimensionality at the leaves. The linear threshold unit at a leaf node need not include any features based on the ancestor attribute tests, because the features based on those attributes are constant at a given node. A decision node reduces the dimensionality of each subspace by the number of values the attribute can take on. One surprising finding from initial informal experiments is that a single linear threshold unit is often sufficient to represent a concept that would otherwise require a large decision tree.

Expected results are:

- An improved perceptron tree formalism.
- Analysis of learning rate when using the perceptron tree formalism.
- Empirical comparison of the perceptron tree algorithm to existing concept learning algorithms.

- Improved understanding of the relationship between nominal-valued and numeric-valued attributes in concept learning.
- Improved understanding of relationships between numerical approaches and symbolic approaches to concept learning.

2.2 Learning Evaluation Functions from Qualitative Criticism

The subproject addresses the problem of inducing a numeric evaluation function from qualitative criticism. For evaluation function learning, each piece of qualitative criticism asserts that the value of a given state a should be greater than the value of a given state b, i.e. $h(a) > h(b)$. This differs from quantitative criticism, in which a given value v is asserted to be the correct function value for a given state a, i.e. $h(a) = v$. Learning from qualitative criticism corresponds to the problem of learning a preference predicate, as described above in section 1.2.1.

The principal advantage of learning from qualitative criticism is that it is easier to infer than quantitative criticism. Any method that allows inferring state preference makes it possible to provide qualitative criticism. For example, from a correct solution path, it is possible to infer the node selection that best-first search should have made, and hence the relative preference of that state to all others on the existing open list (Utgoff and Saxena, 1987a).

In addition to the many search methods that would allow such inferences, the approach also facilitates capturing human expertise. Consider an example. A medical doctor caring for a patient makes decisions regarding which tests to conduct and which treatments to apply. The patient-care process can be modelled as hill-climbing in state space defined by attributes of the patient. One can infer state preference directly from observation of the state variables (patient attributes) and the doctor's decisions. In such a mode, a learning program is operating as an apprentice, acquiring the decision making capability of the expert. Of course, successful performance depends on paying attention to the right attributes, which are not perfectly known within medical science.

Learning an evaluation function from qualitative criticism can be viewed as learning to descriminate better states from worse states. Such a descriminant is equivalent to an evaluation function because it returns a positive value when $P(x, y)$ and a negative value when $P(y, x)$. For connectionist learning, classification learning tasks can be viewed as solving a set of linear inequalities (Ho and Kashyap, 1965). Each positive training instance is represented as a feature vector $\mathbf{f}(x)$ whose inner product with a weight vector $\mathbf{w}$ should be greater than 0. Similarly, each negative training instance is represented as an inner product that should be less than 0. The learning problem is solved by finding a $\mathbf{w}$ that satisfies the set of inequalities. Now consider how a piece of qualitative criticism can be represented by a linear inequality. Given a pair of states x, y, for which x should have a higher evaluation than y, the inequality is $\mathbf{W} \cdot \mathbf{f}(x) > \mathbf{W} \cdot \mathbf{f}(y)$, which can be rewritten $\mathbf{W} \cdot (\mathbf{f}(x) - \mathbf{f}(y)) > 0$. Because $\mathbf{f}(x) - \mathbf{f}(y)$ is a constant vector, the learning problem is solved by finding a $\mathbf{w}$ that satisfies the set of inequalities, as in the standard formulation for finding a descriminant.

The subproject will focus on automatic methods for learning descriminants, particularly in the context of learning an evaluation function that descriminates better from worse. The main issue is how to alter network architecture and learning parameters dynamically, without human intervention. The research addresses the inductive bias issue as manifested in connectionist learning. A new algorithm for learning a polynomial descriminant (Duda & Hart, 1973) has been implemented. The algorithm automatically decides when to add new higher order terms. Ideas developed in (Caudill, 1988; Ivakhnenko, 1971) are to be investigated. The current algorithm has a clean interface. It exists as a set of C-library routines and is easily incorporated within a performance program. There are no learning parameters to set; it is entirely self-controlled. An algorithm that builds and alters a layered network architecture is planned.

Expected results are:

- Show utility of inferring and using qualitative criticism to build a numeric evaluation function, trained by learning to descriminate better from worse.
- Improved understanding of bias issues in connectionist learning, relationship to same issues for non-numerical approaches.
- Algorithm for learning numeric evaluation function from qualitative criticism.
- Algorithm to dynamically alter a layered connectionist network.

2.3 Evaluating Alternative Representations

The subproject addresses the problem of evaluating which, among a given set of alternative representations of a problem, is best suited for learning from examples. The thesis is that a representation that leads to a simpler function of the input features is better suited for learning. There is evidence to suggest that fewer examples are required to learn a simpler function (Tesauro and Janssens, 1988; Rendell and Cho, 1988; Hampson and Volper, 1987; Hampson and Volper, 1986). Also a simpler function would be easier to understand for a user than a more complex function. Here an algorithm for estimating the complexity of the function, from a set of examples, is proposed. Preliminary experiments with the algorithm for evaluating alternative representations show that, in two different cases, the algorithm correctly identifies the better of the two provided representations.

The efficacy of algorithms for learning from examples varies significantly with different representations for the same problem. Often, it is possible to conceive a variety of representations for a particular problem (Amarel, 1968; Nadel, 1987; Korf, 1980), but methods for determining which is best for learning do not exist. A representation well suited for a particular task not only leads to faster learning but also results in compact expressions that are good generalizations of the examples. It would be useful to be able to identify which among a given set of representations is best suited for learning.

Consider the problem of learning a description that discriminates among a fixed set of categories. Such descriptions are called *discriminant descriptions* (Michalski, 1983). Further assume that each example is expressed as a set of attribute value pairs, where each value is from a finite set. Any such problem can be encoded as a problem of learning a boolean function. Such a common encoding makes it possible to compare representations on the basis of the complexity of the resulting boolean function. The complexity of a boolean function is given by the size of the smallest circuit that implements the function (Wegener, 1987). Unfortunately, there are no known algorithms to determine the complexity of an arbitrary boolean function. A method of approximating the complexity of a boolean function uses the representation of a boolean function in terms of its Walsh coefficients (Karpovsky, 1976; Hurst, Miller, and Muzio, 1985). The advantage of using this method is that it enables the separation of the complexity of the function into various components.

There are various techniques for specifying a boolean function. One is to give the truth table of the function. Another is to select a set of simple functions and determine how much each of these simple functions contributes to the function under consideration. The advantage of this representation of the function is that there is a simple expression for the complexity of a boolean function in terms of its Walsh coefficients.

To be able to estimate the complexity of the function that results from different representations, two questions remain to be resolved. First, the fastest known algorithms to compute the Walsh transform of a boolean function of n variables take take $O(n2^n)$ steps (Karpovsky, 1976; Hurst, Miller, and Muzio, 1985). The exponential dependence of these algorithms on the number of variables makes them unsuitable for problems with large number of input variables. It is proposed that this problem be addressed by computing the contribution to complexity of a fixed number of spectral coefficients. This provides an estimate of the complexity. Experiments with multiplexer tasks of various sizes and the eight puzzle problem suggest that the first few spectral coefficients are good indicators of the function complexity. The reason for this could be that for most natural representations of a problem, the function being learned is simple enough that the first few spectral terms contribute the most to the complexity.

The second problem is that in a learning situation the value of the function is known at only a few of the possible inputs. That is, the boolean function is incompletely specified. The computation of a spectral value requires the knowledge of the function at all possible inputs. It is proposed that the value of the function at the unknown inputs be encoded by 0, with *true* encoded as -1 and *false* encodeed as 1. As a result, the contribution to the complexity due to the unknown inputs is ignored.

The effect of these modifications on the estimate of the complexity obtained needs to be evaluated. Initial experiments with alternative representations of the multiplexer problem and the eight puzzle problem indicate that the above algorithm is effective in estimating the complexity of the function that would result from employing various representations. Consider some initial results from evaluating alternative representations of the multiplexer problem.

In the multiplexer problem the goal is to learn a rule that correctly classifies all the instances. The address bits address one of the data bits. The classification of the instance is the value of the data bit addressed. Consider two alternative representations for this problem. One representation is:

$$a_1, a_2, \cdots, a_n, d_1, d_2, \cdots, d_{2^n}$$

here $a_1, a_2, \cdots, a_n$ are the address bits. $d_1, d_2, \cdots, d_{2^n}$ are the data bits. The value of the bit addressed by the address bits gives the classification of the example (0 or 1). An alternative representation for the multiplexer problem is

$$x_1, x_2, \cdots, x_{2^n}, a_1, a_2, \cdots, a_n, d_1, d_2, \cdots, d_{2^n}$$

Largest order computed	Complexity of the Representation 1	Complexity of the Representation 2	Number of such coefficients
1	256	1	10
2	1280	13	55
3	2048	58	175
4	2048	138	385
5	2048	213	637
6	2048	249	847
7	2048	256	967
10	2048	256	1023

Table 1: Estimate of complexity of the multiplexer using a subset of spectral coefficients.

here x_i is 1 if and only if i is the bit addressed by the address bits. In this representation the $x_1, x_2, \cdots, x_{2^n}$ directly give the data bit that is addressed, rather than it being decoded from the address bits.

The second representation is results in a simpler function. Results obtained from using the spectral method, as shown in table 1, correctly identify the second representation as resulting in a simpler function. These results are for the multiplexer problem with 2 address bits and 4 data bits. Similar results have been obtained for multiplexer problems with 1 and 3 address bits.

The results obtained on alternative representations of the multiplexer problem and the eight puzzle problem indicate that estimating the complexity of the function that results from employing the alternative representations provides a method of determining which representation is better suited for learning. In particular, using the spectral representation and estimating the complexity based on a subset of spectral coefficients was correct in evaluating the alternative representations of these problems.

Expected results include:

- Improved understanding of why one representation is better than another. This will lead to a better understanding of how to generate good representations.
- A computationally tractable algorithm for identifying which of a given set of representations will produce the most effective learning.

2.4 Plausible Feature Generation

The most important part of an instance description language is the set of features than can be evaluated with respect to an instance. The instance features provide the basis on which instances are to be grouped into useful classes. Effective learning depends upon features that allow groupings that are useful for the given learning task. The ability to find appropriate features is essential. One approach is to transform available information, particularly the problem specification, into plausible features. A problem specification is a good source of information about the problem-specific constraints. The intent is for the new features to make these relationships explicit to the learning algorithm.

As an example, in the n-queens problem, one must place each of the n queens so that it is not threatening any other queen. The statement of the problem may define "threat" as a predicate. It is not difficult to extend the idea of "is square k threatened?" to "how many threats are there to square k?" The number of threats at each square can then participate as a feature of each state. Michalski's *counting arguments* rule (Michalski, 1983) is similar, but it is traditionally applied to predicates in either of the instance or concept description languages. Other differences are discussed below. If neither description language contains the predicate "threat", the number of threats cannot be counted. Indeed, most approaches to feature generation depend upon some characteristic of a learning algorithm or its description languages (Rendell, 1985; Schlimmer, 1987b; Utgoff, 1986; Pagallo and Haussler, 1988). The approach proposed here is unusual because it does not. Instead, it depends upon other sources of domain-specific information, for example a problem specification.

As discussed above, learning algorithms are often applied to learning search control knowledge. Plausible feature generation is easily applied to improving the representation for search control knowledge due to the availability of a problem specification. One way to learn search control knowledge is for a problem solver to train a learning algorithm while it searches. Each training instance is a set of features that describes a search state and its classification (on or off of the solution path). This approach is appealing because it is simple, but it is not always effective. Sometimes the descriptions that are easiest for the problem solver to produce are difficult for the learning algorithm to use. For example, (Flann and Dietterich, 1986) demonstrated that, for one particular problem, learning performance was improved when the problem solver's representation of a search state was translated into a different (functional) representation.

New features can be generated from a search problem specification with a two step process. The first step is to generate a set of simpler problems by splitting the original problem statement along *every* conjunct within a set of parentheses. This process is recursively applied to the resulting simpler problems until no more simpler problems can be generated. For example, problem statement $s_1 \wedge (s_2 \wedge s_3)$ yields two new statements: s_1 and $(s_2 \wedge s_3)$. Statement $(s_2 \wedge s_3)$ then yields statements s_2 and s_3. In this case, 4 simpler problems result from the original problem statement. In general, problem statements containing c conjuncts yield between $c+1$ and $2c$ simpler problems. This procedure for generating simpler problems is also truth-preserving, since the conjunction of the set of simpler problems is equivalent to the original problem. Gaschnig proposed a similar method of generating simpler problems (Gaschnig, 1979), except that it was manually guided and any combination of conjuncts or disjuncts could be produced. A human needed to identify which of the $2^{t+1} - 2$ combinations should be explored (t is the number of connectives in the problem statement). In contrast, the method proposed here requires no manual intervention because it yields fewer and simpler problems.

The second step is to transform the simpler problems into new features. Each simpler problem can yield zero or more new features, depending upon how many transformations can be applied to it. Initial research on this problem has identified a set of transformations, some already known, for generating plausible features from the simpler problem statements. For example, one transformation counts the ways a statement with unassigned variables can be satisfied. Michalski proposed a set of similar *counting arguments* rules, but they count the ways in which the quantified variables satisfy some arbitrary condition (Michalski, 1983). Using the problem statement, instead of an arbitrary condition, simplifies the automatic generation of new features. New transformations include turning inequalities into numeric features that measure the magnitude of the inequality.

Each of the transformations has been effective at generating useful features for the n-queens problem. In initial tests, one of the features identified a solution path 63% of the time. Random decisions would have only identified one 46% of the time.

While the methods of plausible feature generation appear promising, several problems remain to be addressed. First, the methods are sensitive to how the domain-specific information is represented. Some representations enable more or better new features than other representations. This problem can be alleviated either by stating domain-specific information carefully, or by enlarging the set of transformations that generate features. Both additional transformations and guidelines for stating domain-specific information are goals of this research. A second problem is detecting and eliminating useless new features. Useless features are undesirable because they increase the learner's workload without providing any benefit. This problem can be addressed either by measuring the information content of each feature (Quinlan, 1986), or by observing whether the learning algorithm is actually using each feature.

Expected results include:

- Methods of generating features from problem specifications.
- Guidelines for specifying a problem to facilitate feature generation.
- Methods for identifying useless features in an instance description language.
- Improved performance of a learning algorithm in learning search control knowledge.

2.5 Feature Discovery

A learning system often exists as a component of a problem solving system, in order to improve the problem solving performance in some way. Within such a system, one source of features is the commonly occuring states and subgoals of the problem solver. Therefore, a learning system may attempt to discover such features by observing the problem solving process. Features of this type cannot be determined from the problem description alone, because they are a product of the interaction of problem solving goals within the domain. For example, many board games have the concept of *fork*, as being the coexistence of mutually exclusive protection goals; and the concept of *piece mobility*, as being the relative number of future potential moves. Through experience, these patterns become elevated to features of a game state itself.

This process differs from plausible feature generation, described above, which produces features from a problem description. Although feature generation and discovery are not mutually exclusive, feature discovery is experience-driven and opportunistic, because the features come about through the exericnce of solving problems in a domain. As such, the process is a combination of empirical and analytical reasoning.

This project investigates feature discovery in the domain of board games. Such domains are attractive for two reasons. First, a game is completely defined by a succinct sets of rules. This allows a simple implementation, which can serve as a source of numerous, cheap learning examples. Second, board games are difficult problem solving tasks, with search spaces larger than many real-world planning domains. For example, a typical Othello game has a search space of approximately 10^{50} nodes (Rosenbloom, 1982).

2.5.1 When to learn: Noticing interesting phenomena

Because the features created will represent the significant phenomena of a domain, the first major issue is deciding when a problem-solving event or episode is significant. This is similar to the issue of *interestingness* in machine learning (DeJong, 1983; Lebowitz, 1984). In the domain of board games, certain events are universally significant, such as losing the game or having no legal moves available. However, even if these events can be easily recognized, their usefulness is usually limited. Recognizing that the game is about to be lost usually comes too late to make a difference in the outcome. Other inherently significant events, such as losing a queen in chess, are similarly catastrophic.

Interestingness heuristics must be able to detect more subtle events. Also, as Lebowitz points out (Lebowitz, 1984), interestingness may be expected to change over time. A learning system should be able to start with a small set of interesting general events, and refine and extend them through experience. A number of heuristics that will allow the system to do this are:

- *Precondition satisfaction*: If a move is interesting, then the conditions that enabled it are interesting as well. For example, the loss of a valuable piece via a trap is interesting; therefore, the moves leading up to a trap are in turn interesting. A system could learn to avoid such situations itself, or to force them on its opponent.

- *Violated expectations*: Every move or group of moves can have associated with it a set of expectations about its effects. If a situation violates an expectation, the situation is judged interesting. The simplest expectation of a move sequence is that it achieve the goal for which it is applied; if this expectation is violated, *i.e.* the plan fails, the event is deemed interesting and analysis is carried out to determine why the failure occurred. This is implicit in most of the work on learning from planning failures (Mostow & Bhatnagar, 1987; Minton, 1984; Tadepalli, 1988; Hammond, 1986).

2.5.2 What to learn: Characterizing phenomena

A second major issue is how to describe a phenomenon as a feature that can be recognized in future game states. One approach is to use an inductive learning technique, such as decision trees (Quinlan, 1986), to cluster the interesting board states by their existing instance features. However, feature-based clustering ignores two important sources of knowledge. First, it does not consider the reasons for which the events were interesting (and so is goal-insensitive). Second, it ignores the rules of the game by which the board states came about.

Because the rules of the game are readily available and form a complete and succinct theory of the game, explanation-based generalization, a form of Explanation-Based Learning (EBL), is applicable (DeJong, 1983; Mitchell et al, 1986; Minton, 1988). However, existing EBL methods have significant limitations that make them difficult to apply directly. First, they can only generate *complete* explanations of events. For example, a move may have catastrophic consequences that are not immediately obvious because they do not come about for several more moves. An explanation would typically involve all of the intervening moves, and thus would be very specific to the particular game situation. Second, existing explanation-based methods generate *exact* conditions. However, a feature may be valuable even if it correlates inexactly with a particular game situation. Since efficiency is a consideration in game-playing and in many planning domains, it is desirable to create simpler features at the expense of accuracy, and existing explanation-based methods are incapable of making this tradeoff. These two problems are characteristic of *intractable* domains (Mitchell et al, 1986), and no general method has yet been devised for overcoming them.

This project will use a discovery system architecture (Lenat, 83; Langley et al, 1983) extended with an explanation-based learning component. The goal of the discovery component is to derive useful features not mentioned in the initial set of rules. In its simplest form, the discovery module would be able to provide perturbed examples (Araya, 1984; Kibler & Porter, 1983) that would be used to generalize concepts acquired through EBL. By performing constrained generalization of these concepts, the discovery system can produce features of greater generality than could be produced by an EBL component alone. Furthermore, since the discovery component can perform directed experimentation, less actual game-playing experience should be needed.

To be able to investigate theses issues, a prototype feature discovery system is being implemented using the game of Othello. The prototype is currently able to recognize general traps in Othello games and produce an explanation of the moves involved. The system learns a feature representing the situation leading up to the trap, and learns to avoid making similar moves in the future. Ultimately, this explanation component will become part of the larger system for discovering and synthesizing features not present in its initial set of rules. The next goal is to model the discovery of the concept of stable discs and the feature of stability.

The expected results of this project include:

- Increased understanding of how useful features can be discovered. This includes understanding of how some phenomena come to be noticed and how they become developed into features through inductive and analytical means.
- The use of expectations and interestingness heuristics to guide feature discovery.
- An implemented discovery system that is capable of synthesizing a variety of useful features for the game of Othello.

3 Conclusion

The project will investigate a closely related set research ideas that focus on the problem of enabling learning programs to improve their representations automatically. Getting humans out of the loop is, by definition, necessary for contructing intelligent autonomous learning programs. At the moment, only humans are able to construct appropriate representations for learning.

The long term goal of the research is to advance the technology for providing problem solving systems that improve as a result of problem solving experience. The largest obstacle to building such systems is that good methods do not exist by which a program can find an appropriate representation so that learning remains effective. Progress on this front will provide significant payoffs.

References

Amarel, S. (1968). On representations of problems of reasoning about actions. In Michie (Ed.), *Machine Intelligence*, pp. 131-171, Edinburgh University Press.

Araya, A. (1984). Learning problem classes by means of experimentation and generalization. In *Proceedings of the Fourth National Conference on Artificial Intelligence* (pp. 11-15), University of Texas at Austin: Morgan Kaufmann.

Breiman, L., Friedman, J. H., Olshen, R. A., and Stone, C. J. (1984). *Classification and regression trees*. Belmont, CA: Wadsworth International Group.

Blumer, A., Ehrenfeuct. A., Haussler, D., Warmuth, M. K., (1987). *Learnability and the Vapnik-Chervonenkis Dimension*. (Technical report UCSC-CRL-87-20), University of California, Santa Cruz, CA 95064, USA.

Caudill, M. (1988). The polynomial ADALINE algorithm. *Computer Language*, (53-59), December.

DeJong, G. (1983). An approach to learning from observation. In *Proceedings of the Second International Machine Learning Workshop* (pp. 171-176). Champaign-Urbana, Ilinois.

Inductive learning of structural description. *Artificial Intelligence 16(3)*, 257-294, North-Holland.

Duda, R. O. and Hart, P. E. (1973). *Pattern Classification and Scene Analysis*. Wiley.

Ellman, T. (1988). Explanation-directed search for simplifying assumptions. In *Symposium on Explanation-based Learning* (pp. 95-99). Stanford University, Stanford, CA.

Fisher, D. H. (1987). Knowledge acquisition via incremental conceptual clustering. *Machine Learning 2(2)*, 139-172, Kluwer.

Flann, N. S. and Dietterich, T. G. (1986). Selecting appropriate representations for learning from examples. In *Proceedings of the Fifth National Conference on Artificial Intelligence, Vol 1* (pp. 460-466), Philadelphia, PA: Morgan Kaufmann.

Flann, N. (1988). Improving problem solving performance by example-guided reformulation of knowledge. In *Proceedings of the First International Workshop on Change of Representation and Inductive Bias* (pp 14-28). Sponsored by Philips Laboratories.

Gallant, S. I. (1986) Optimal linear discriminants. In *Proceedings of the International Conference on Pattern Recognition* (pp. 849-852). IEEE Computer Society Press.

Gaschnig, John. (1979). A problem similarity approach to devising heuristics: First results. In *Proceedings of IJCAI-79, Vol 1.* pp 301-307.

Hammond, Kristian. (1986). Learning to anticipate and avoid planning problems through the explanation of failures. In *Proceedings of the Fifth National Conference on Artificial Intelligence, Vol. 1* (pp. 556-560). Philadelpha, PA: Morgan Kaufmann.

Hampson, S. E. and Volper, D. J. (1986) Linear function neurons: Structure and training. In *Biological Cybernetics, 53,* 203-217. Springer-Verlag.

Hampson, S. E., Vopler, D. J., (1987). "Disjunctive Models of Boolean Category Learning", *Biological Cybernetics 55.*

Haussler, D., (1987) Bias, Version Spaces and Valiant's learning framework. *Proceedings of the Fourth Machine learning Workshop*, (pp 324-336).

Haussler, D. (1986). Quantifying the inductive bias in concept learning. *Proceddings of the Fifth National Conference on Artificial Inteliigence*, (pp. 485-489), Morgan Kaufmann.

Ho, Y. C. and Kashyap, R. L. (1965). An algorithm for linear inequalities and its applications. *IEEE Transactions on Electronic Computers, EC-14(5)*, pp. 683-688.

Hurst, S. L., Miller, D.M., Muzio, J.C. (1985). *Spectral Techniques in Digital Logic*, Academic Press Inc.

Ivakhnenko, A. G. (1971). Polynomial theory of complex systems. *IEEE Transactions on Systems, Man, and Cybernetics SMC-1(4)*, (364-378).

Karpovsky, M. G., *Finite Orthogonal Series in the Design of Digital Devices*, John Wiley & Sons, 1977.

Keller, R. M. (1987). Concept learning in context. *Proceedings of the Fourth International Workshop on Machine Learning* (pp. 91-102). Morgan Kaufmann.

Kibler D. and Porter, B. (1983). Perturbation: A means for guiding generalization. In *Proceedings of the Eighth International Joint Conference on Artificial Intelligence-83.* Karlsruhe, Germany: Morgan Kaufmann.

Korf, R. E. (1980). Towards a model of representation change. *Artificial Intelligence 14*, pp 41-78, North-Holland.

Laird, J. E., Rosenbloom, P. S., & Newell, A. (1986). Chunking in SOAR: the anatomy of a general learning mechanism. *Machine Learning 1(1)*, 11-46, Kluwer.

Langley, P., Bradshaw, G., and Simon, H. (1983). Rediscovering Chemistry with the BACON system. In Michalski, Carbonell, & Mitchell (Eds.) *Machine Learning: An artificial intelligence approach*, Morgan Kaufmann, Pp. 307-329.

Lebowitz, M. (1984). Interestingness and predictability: Deciding what to learn, when to learn (Technical Report). New York, New York: Columbia University.

Lee, K. F. and Mahajan, S. (1988). A pattern classification approach to evaluation function learning. *Artificial Intelligence 36(1)*, 1-25, North-Holland.

Lenat, D. (1983). The role of heuristics in learning by discovery: Three case studies. In Michalski, Carbonell, & Mitchell (Eds.) *Machine Learning: An artificial intelligence approach*, Morgan Kaufmann, Pp. 243-306.

Lippman, R. P. (1987). An introduction to computing with neural nets. *IEEE ASSP Magazine*, (pp. 4-22), April.

Michalski, R. S. (1983). A theory and methodology of inductive learning. *Machine Learning: An Artificial Intelligence Approach*, Michalski, Carbonell, Mitchell (Eds.), Morgan Kaufman.

Michalski, R. S. (1986). Understanding the nature of learning: Issues and research directions. *Machine Learning: An Artificial Intelligence Approach, volume II*, Michalski, Carbonell, Mitchell (Eds.), Morgan Kaufmann.

Minton, S. (1984). Constraint-based generalization: Learning game-playing plans from single examples. In *Proceedings of the Fourth National Conference on Artificial Intelligence* (pp. 251-254). University of Texas at Austin: Morgan Kaufmann.

Minton, S., Carbonell, J. G., Etzioni, O., Knoblock, C. A. and Kuokka, D. R. (1987). Acquiring effective search control rules: explanation-based learning in the PRODIGY system. *Proceedings of the Fourth International Workshop on Machine Learning*, Morgan Kaufman, pp. 122-133.

Minton, S. and Carbonell, J. G. (1987). Strategies for learning search control rules: an explanation-based approach. *Proceedings of the Tenth International Joint Conference on Artificial Intelligence*, (pp. 228-235), Morgan Kaufman.

Minton, S. (1988). Quantitative results concerning the utility of explanation-based learning. In *Proceedings of the Seventh National Conference on Artificial Intelligence, Vol. 2* (pp. 564-569). Saint Paul, Minnesota: Morgan Kaufmann.

Mitchell, T. M. (1980). The need for biases in learning generalizations. Report CBM-TR-117, Rutgers University.

Mitchell, T. M., Utgoff, P. E. and Banerji, R. B. (1983). Learning by experimentation: acquiring and refining problem-solving heuristics. In Michalski, Carbonell, & Mitchell (Eds.) *Machine Learning: An artificial intelligence approach*, Morgan Kaufmann, 163-190.

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

Mostow, J. and Bhatnagar, N. (1987). Failsafe - a floor planner that uses EBG to learn from its failures. *Proceedings of the Tenth International Joint Conference on Artificial Intelligence*, (pp. 249-255), Morgan Kaufman.

Muggleton, S. (1987). Duce, an oracle based approach to constructive induction. *Proceedings of the Tenth International Joint Conference on Artificial Intelligence*, (pp. 287-292), Morgan Kaufman.

Muggleton, S. and Buntine, W. (1988). Machine invention of first-order predicates by inverting resolution. *Proceedings of the Fifth International Conference on Machine Learning* (pp. 339-352). Morgan Kaufman.

Nadel, B. (1987). Representation selection for constraint satisfaction problems: a case study using n-queens. Technical Report DCS-TR-208, Department of Computer Science, Rutgers University.

Pagallo, G. and Haussler, D. (1988). Feature discovery in empirical learning. Dept. of Computer and Information Science, Univ. of Calif., Santa Cruz.

Pearl, J. (1978). On the connection between the complexity and credibility of inferred models. *International Journal of General Systems*, 4, (pp. 116-126)

Quinlan, J. R. (1983). Learning efficient classification procedures and their application to chess end games. In Michalski, Carbonell, & Mitchell (Eds.) *Machine Learning: An artificial intelligence approach*, Morgan Kaufmann, 463-482.

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

Rendell, L. (1983). A new basis for state-space learning systems and a successful implementation. *Artificial Intelligence 20(4)*, pp. 369-392, North-Holland.

Rendell, L. (1985). Substantial constructive induction using layered information compression: Tractable feature formation in search. In *Proceedings of the Ninth International Joint Conference on Artificial Intelligence-85, Vol 1* (pp. 650-658). University of California at Los Angeles: Morgan Kaufmann.

Rendell, L. (1986). A general framework for induction and a study of selective induction. *Machine Learning 1(2)*, 177-226, Kluwer.

Rendell, L., Seshu, R. and Tcheng, D. (1987). Layered concept learning and dynamically variable bias management. *Proceedings of the Tenth International Joint Conference on Artificial Intelligence*, pp. 308-314, Morgan Kaufman.

Rendell, L. and Cho, H. (1988). Empirical concept learning as a function of data sampling and concept character. Report No. UIUCDCS-R-88-1410, Department of Computer Science, University of Illinois at Urbana-Champaign.

Rosenbloom, P. (1982). A world-championship-level Othello program. *Artificial Intelligence, 19*, pp. 279-320, North-Holland.

Rumelhart, D. E. and McClelland, J. L. (1986). *Parallel Distributed Processing*, MIT Press. (two volumes).

Russell, S. J. (1986). Preliminary steps toward the automation of induction. In *Proceedings of the Fifth National Conference on Artificial Intelligence* (pp. 477-484), Morgan Kaufmann.

Russell, S. J. and Grosof, B. N. (1987). A declarative approach to bias in concept learning. *Proceedings of the Sixth National Conference on Artificial Intelligence* (pp. 505-510). Morgan Kaufmann.

Russell, S. J. (1988). Tree-structured bias. *Proceedings of the Seventh National Conference on Artificial Intelligence* (pp. 641-645). Morgan Kaufmann.

Samuel, A. (1959). Some studies in machine learning using the game of checkers. *IBM Journal of Research and Development 3(3)*, 211-229.

Samuel, A. (1967). Some studies in machine learning using the game of checkers II: recent progress. *IBM Journal of Research and Development 11(6)*, 601-617.

Saxena, S. & Utgoff, P. E. (1988). A relationship between classification accuracy and search quality. COINS Technical Report 88-104, Department of Computer and Information Science, University of Massachusetts, Amherst, MA, December. (Submitted to the Eleventh International Joint Conference on Artificial Intelligence)

Schlimmer, J. C. (1987a). Incremental adjustment of representations. *Proceedings of the Fourth International Workshop on Machine Learning*, Morgan Kaufman, pp. 79-90.

Schlimmer, J. C. (1987b). Learning and representation change. *Proceedings of the Sixth National Conference on Artificial Intelligence* (pp. 511-515). Morgan Kaufmann.

Shaefer, C. (1988). The ARGOT strategy. *Proceedings of the First International Workshop in Change of Representation and Inductive Bias*, Philips Laboratories, New York.

Tadepalli, P. (1988). Lazy explanation-based learning: A solution to the intractable theory problem. Unpublished.

Tesauro, G., Janssens, R. (1988). Scaling relationships in back-propogation learning: dependence on predicate order. Technical Report CCSR-81, Center for Complex Systems Research, University of Illinois at Urbana-Champaign.

Utgoff, P. E. & Mitchell, T. M. (1982). Acquisition of appropriate bias for inductive concept learning. In *Proceedings of the Second National Conference on Artificial Intelligence* (pp. 414-417). Pittsburgh, PA: Morgan Kaufmann.

Utgoff, P. E. (1986). *Machine Learning of Inductive Bias*, Kluwer, Hingham MA, 168 pages.

Utgoff, P. E. and Saxena, S. (1987). Learning a preference predicate *Proceedings of the Fourth International Workshop on Machine Learning*, Morgan Kaufman, pp. 115-121.

Utgoff, P. E. and Saxena, S. (1987). A perfect lookup table evaluation function for the eight-puzzle. COINS Technical Report 87-71, Department of Computer and Information Science, University of Massachusetts, Amherst, MA.

Utgoff, P. E. (1988a). ID5: an incremental ID3. *Proceedings of the Fifth International Conference on Machine Learning*, Morgan Kaufman, pp. 107-120.

Utgoff, P. E. (1988b). Perceptron trees: a case study in hybrid concept representations. *Proceedings of the Seventh National Conference on Artificial Intelligence* (pp. 601-606). Saint Paul, Minnesota: Morgan Kaufmann.

Utgoff, P. E. (to appear). Incremental induction of decision trees. *Machine Learning*, Kluwer.

Valiant, L. G. (1984). A theory of the learnable. *Communications of the Association for Computing Machinery*, 27(11), pp. 1134-1142.

Vapnik, V. N. & Chervonenkis, A. Y. (1971). On the uniform convergence of relative frequencies of events to their probabilities. *Theory of Probability and its Applications*, 16(2), (pp. 264-80).

Watanabe, S. (1985). *Pattern Recognition: Human and Mechanical.* Wiley & Sons, New York.

Wegner, I., *The Complexity of Boolean Functions*, John Wiley & Sons, 1987.