MACHINE LEARNING

TOM M. MITCHELL

Machine Learning

Tom M. Mitchell

Product Details

- **Hardcover:** 432 pages ; Dimensions (in inches): 0.75 x 10.00 x 6.50

- **Publisher:** McGraw-Hill Science/Engineering/Math; (March 1, 1997)

- **ISBN:** 0070428077

- **Average Customer Review:** ⭐⭐⭐⭐ Based on 16 reviews.

- **Amazon.com Sales Rank:** 42,816

- **Popular in:** Redmond, WA (#17) , Ithaca, NY (#9)

Editorial Reviews

From Book News, Inc. An introductory text on primary approaches to machine learning and the study of computer algorithms that improve automatically through experience. Introduce basics concepts from statistics, artificial intelligence, information theory, and other disciplines as need arises, with balanced coverage of theory and practice, and presents major algorithms with illustrations of their use. Includes chapter exercises. Online data sets and implementations of several algorithms are available on a Web site. No prior background in artificial intelligence or statistics is assumed. For advanced undergraduates and graduate students in computer science, engineering, statistics, and social sciences, as well as software professionals. *Book News, Inc.®, Portland, OR*

Book Info: Presents the key algorithms and theory that form the core of machine learning. Discusses such theoretical issues as How does learning performance vary with the number of training examples presented? and Which learning algorithms are most appropriate for various types of learning tasks? DLC: Computer algorithms.

Book Description: This book covers the field of machine learning, which is the study of algorithms that allow computer programs to automatically improve through experience. The book is intended to support upper level undergraduate and introductory level graduate courses in machine learning

PREFACE

The field of machine learning is concerned with the question of how to construct computer programs that automatically improve with experience. In recent years many successful machine learning applications have been developed, ranging from data-mining programs that learn to detect fraudulent credit card transactions, to information-filtering systems that learn users' reading preferences, to autonomous vehicles that learn to drive on public highways. At the same time, there have been important advances in the theory and algorithms that form the foundations of this field.

The goal of this textbook is to present the key algorithms and theory that form the core of machine learning. Machine learning draws on concepts and results from many fields, including statistics, artificial intelligence, philosophy, information theory, biology, cognitive science, computational complexity, and control theory. My belief is that the best way to learn about machine learning is to view it from all of these perspectives and to understand the problem settings, algorithms, and assumptions that underlie each. In the past, this has been difficult due to the absence of a broad-based single source introduction to the field. The primary goal of this book is to provide such an introduction.

Because of the interdisciplinary nature of the material, this book makes few assumptions about the background of the reader. Instead, it introduces basic concepts from statistics, artificial intelligence, information theory, and other disciplines as the need arises, focusing on just those concepts most relevant to machine learning. The book is intended for both undergraduate and graduate students in fields such as computer science, engineering, statistics, and the social sciences, and as a reference for software professionals and practitioners. Two principles that guided the writing of the book were that it should be accessible to undergraduate students and that it should contain the material I would want my own Ph.D. students to learn before beginning their doctoral research in machine learning.

A third principle that guided the writing of this book was that it should present a balance of theory and practice. Machine learning theory attempts to answer questions such as "How does learning performance vary with the number of training examples presented?" and "Which learning algorithms are most appropriate for various types of learning tasks?" This book includes discussions of these and other theoretical issues, drawing on theoretical constructs from statistics, computational complexity, and Bayesian analysis. The practice of machine learning is covered by presenting the major algorithms in the field, along with illustrative traces of their operation. Online data sets and implementations of several algorithms are available via the World Wide Web at http://www.cs.cmu.edu/~tom/mlbook.html. These include neural network code and data for face recognition, decision tree learning code and data for financial loan analysis, and Bayes classifier code and data for analyzing text documents. I am grateful to a number of colleagues who have helped to create these online resources, including Jason Rennie, Paul Hsiung, Jeff Shufelt, Matt Glickman, Scott Davies, Joseph O'Sullivan, Ken Lang, Andrew McCallum, and Thorsten Joachims.

ACKNOWLEDGMENTS

In writing this book, I have been fortunate to be assisted by technical experts in many of the subdisciplines that make up the field of machine learning. This book could not have been written without their help. I am deeply indebted to the following scientists who took the time to review chapter drafts and, in many cases, to tutor me and help organize chapters in their individual areas of expertise.

> Avrim Blum, Jaime Carbonell, William Cohen, Greg Cooper, Mark Craven, Ken DeJong, Jerry DeJong, Tom Dietterich, Susan Epstein, Oren Etzioni, Scott Fahlman, Stephanie Forrest, David Haussler, Haym Hirsh, Rob Holte, Leslie Pack Kaelbling, Dennis Kibler, Moshe Koppel, John Koza, Miroslav Kubat, John Lafferty, Ramon Lopez de Mantaras, Sridhar Mahadevan, Stan Matwin, Andrew McCallum, Raymond Mooney, Andrew Moore, Katharina Morik, Steve Muggleton, Michael Pazzani, David Poole, Armand Prieditis, Jim Reggia, Stuart Russell, Lorenza Saitta, Claude Sammut, Jeff Schneider, Jude Shavlik, Devika Subramanian, Michael Swain, Gheorgh Tecuci, Sebastian Thrun, Peter Turney, Paul Utgoff, Manuela Veloso, Alex Waibel, Stefan Wrobel, and Yiming Yang.

I am also grateful to the many instructors and students at various universities who have field tested various drafts of this book and who have contributed their suggestions. Although there is no space to thank the hundreds of students, instructors, and others who tested earlier drafts of this book, I would like to thank the following for particularly helpful comments and discussions:

> Shumeet Baluja, Andrew Banas, Andy Barto, Jim Blackson, Justin Boyan, Rich Caruana, Philip Chan, Jonathan Cheyer, Lonnie Chrisman, Dayne Freitag, Geoff Gordon, Warren Greiff, Alexander Harm, Tom Ioerger, Thorsten

Joachim, Atsushi Kawamura, Martina Klose, Sven Koenig, Jay Modi, Andrew Ng, Joseph O'Sullivan, Patrawadee Prasangsit, Doina Precup, Bob Price, Choon Quek, Sean Slattery, Belinda Thom, Astro Teller, Will Tracz

I would like to thank Joan Mitchell for creating the index for the book. I also would like to thank Jean Harpley for help in editing many of the figures. Jane Loftus from ETP Harrison improved the presentation significantly through her copyediting of the manuscript and generally helped usher the manuscript through the intricacies of final production. Eric Munson, my editor at McGraw Hill, provided encouragement and expertise in all phases of this project.

As always, the greatest debt one owes is to one's colleagues, friends, and family. In my case, this debt is especially large. I can hardly imagine a more intellectually stimulating environment and supportive set of friends than those I have at Carnegie Mellon. Among the many here who helped, I would especially like to thank Sebastian Thrun, who throughout this project was a constant source of encouragement, technical expertise, and support of all kinds. My parents, as always, encouraged and asked "Is it done yet?" at just the right times. Finally, I must thank my family: Meghan, Shannon, and Joan. They are responsible for this book in more ways than even they know. This book is dedicated to them.

Tom M. Mitchell

CONTENTS

CHAPTER

1

INTRODUCTION

Ever since computers were invented, we have wondered whether they might be made to learn. If we could understand how to program them to learn—to improve automatically with experience—the impact would be dramatic. Imagine computers learning from medical records which treatments are most effective for new diseases, houses learning from experience to optimize energy costs based on the particular usage patterns of their occupants, or personal software assistants learning the evolving interests of their users in order to highlight especially relevant stories from the online morning newspaper. A successful understanding of how to make computers learn would open up many new uses of computers and new levels of competence and customization. And a detailed understanding of information-processing algorithms for machine learning might lead to a better understanding of human learning abilities (and disabilities) as well.

We do not yet know how to make computers learn nearly as well as people learn. However, algorithms have been invented that are effective for certain types of learning tasks, and a theoretical understanding of learning is beginning to emerge. Many practical computer programs have been developed to exhibit useful types of learning, and significant commercial applications have begun to appear. For problems such as speech recognition, algorithms based on machine learning outperform all other approaches that have been attempted to date. In the field known as data mining, machine learning algorithms are being used routinely to discover valuable knowledge from large commercial databases containing equipment maintenance records, loan applications, financial transactions, medical records, and the like. As our understanding of computers continues to mature, it

seems inevitable that machine learning will play an increasingly central role in computer science and computer technology.

A few specific achievements provide a glimpse of the state of the art: programs have been developed that successfully learn to recognize spoken words (Waibel 1989; Lee 1989), predict recovery rates of pneumonia patients (Cooper et al. 1997), detect fraudulent use of credit cards, drive autonomous vehicles on public highways (Pomerleau 1989), and play games such as backgammon at levels approaching the performance of human world champions (Tesauro 1992, 1995). Theoretical results have been developed that characterize the fundamental relationship among the number of training examples observed, the number of hypotheses under consideration, and the expected error in learned hypotheses. We are beginning to obtain initial models of human and animal learning and to understand their relationship to learning algorithms developed for computers (e.g., Laird et al. 1986; Anderson 1991; Qin et al. 1992; Chi and Bassock 1989; Ahn and Brewer 1993). In applications, algorithms, theory, and studies of biological systems, the rate of progress has increased significantly over the past decade. Several recent applications of machine learning are summarized in Table 1.1. Langley and Simon (1995) and Rumelhart et al. (1994) survey additional applications of machine learning.

This book presents the field of machine learning, describing a variety of learning paradigms, algorithms, theoretical results, and applications. Machine learning is inherently a multidisciplinary field. It draws on results from artificial intelligence, probability and statistics, computational complexity theory, control theory, information theory, philosophy, psychology, neurobiology, and other fields. Table 1.2 summarizes key ideas from each of these fields that impact the field of machine learning. While the material in this book is based on results from many diverse fields, the reader need not be an expert in any of them. Key ideas are presented from these fields using a nonspecialist's vocabulary, with unfamiliar terms and concepts introduced as the need arises.

1.1 WELL-POSED LEARNING PROBLEMS

Let us begin our study of machine learning by considering a few learning tasks. For the purposes of this book we will define learning broadly, to include any computer program that improves its performance at some task through experience. Put more precisely,

> *Definition*: A computer program is said to **learn** from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E.

For example, a computer program that learns to play checkers might improve its performance *as measured by its ability to win* at the class of tasks involving *playing checkers games*, through experience *obtained by playing games against itself*. In general, to have a well-defined learning problem, we must identity these

- Learning to recognize spoken words.

  All of the most successful speech recognition systems employ machine learning in some form. For example, the SPHINX system (e.g., Lee 1989) learns speaker-specific strategies for recognizing the primitive sounds (phonemes) and words from the observed speech signal. Neural network learning methods (e.g., Waibel et al. 1989) and methods for learning hidden Markov models (e.g., Lee 1989) are effective for automatically customizing to individual speakers, vocabularies, microphone characteristics, background noise, etc. Similar techniques have potential applications in many signal-interpretation problems.

- Learning to drive an autonomous vehicle.

  Machine learning methods have been used to train computer-controlled vehicles to steer correctly when driving on a variety of road types. For example, the ALVINN system (Pomerleau 1989) has used its learned strategies to drive unassisted at 70 miles per hour for 90 miles on public highways among other cars. Similar techniques have possible applications in many sensor-based control problems.

- Learning to classify new astronomical structures.

  Machine learning methods have been applied to a variety of large databases to learn general regularities implicit in the data. For example, decision tree learning algorithms have been used by NASA to learn how to classify celestial objects from the second Palomar Observatory Sky Survey (Fayyad et al. 1995). This system is now used to automatically classify all objects in the Sky Survey, which consists of three terrabytes of image data.

- Learning to play world-class backgammon.

  The most successful computer programs for playing games such as backgammon are based on machine learning algorithms. For example, the world's top computer program for backgammon, TD-GAMMON (Tesauro 1992, 1995), learned its strategy by playing over one million practice games against itself. It now plays at a level competitive with the human world champion. Similar techniques have applications in many practical problems where very large search spaces must be examined efficiently.

TABLE 1.1
Some successful applications of machine learning.

three features: the class of tasks, the measure of performance to be improved, and the source of experience.

A checkers learning problem:

- Task T: playing checkers
- Performance measure P: percent of games won against opponents
- Training experience E: playing practice games against itself

We can specify many learning problems in this fashion, such as learning to recognize handwritten words, or learning to drive a robotic automobile autonomously.

A handwriting recognition learning problem:

- Task T: recognizing and classifying handwritten words within images
- Performance measure P: percent of words correctly classified

- Artificial intelligence

   Learning symbolic representations of concepts. Machine learning as a search problem. Learning as an approach to improving problem solving. Using prior knowledge together with training data to guide learning.

- Bayesian methods

   Bayes' theorem as the basis for calculating probabilities of hypotheses. The naive Bayes classifier. Algorithms for estimating values of unobserved variables.

- Computational complexity theory

   Theoretical bounds on the inherent complexity of different learning tasks, measured in terms of the computational effort, number of training examples, number of mistakes, etc. required in order to learn.

- Control theory

   Procedures that learn to control processes in order to optimize predefined objectives and that learn to predict the next state of the process they are controlling.

- Information theory

   Measures of entropy and information content. Minimum description length approaches to learning. Optimal codes and their relationship to optimal training sequences for encoding a hypothesis.

- Philosophy

   Occam's razor, suggesting that the simplest hypothesis is the best. Analysis of the justification for generalizing beyond observed data.

- Psychology and neurobiology

   The power law of practice, which states that over a very broad range of learning problems, people's response time improves with practice according to a power law. Neurobiological studies motivating artificial neural network models of learning.

- Statistics

   Characterization of errors (e.g., bias and variance) that occur when estimating the accuracy of a hypothesis based on a limited sample of data. Confidence intervals, statistical tests.

TABLE 1.2
Some disciplines and examples of their influence on machine learning.

- Training experience E: a database of handwritten words with given classifications

A robot driving learning problem:

- Task T: driving on public four-lane highways using vision sensors
- Performance measure P: average distance traveled before an error (as judged by human overseer)
- Training experience E: a sequence of images and steering commands recorded while observing a human driver

   Our definition of learning is broad enough to include most tasks that we would conventionally call "learning" tasks, as we use the word in everyday language. It is also broad enough to encompass computer programs that improve from experience in quite straightforward ways. For example, a database system

that allows users to update data entries would fit our definition of a learning system: it improves its performance at answering database queries, based on the experience gained from database updates. Rather than worry about whether this type of activity falls under the usual informal conversational meaning of the word "learning," we will simply adopt our technical definition of the class of programs that improve through experience. Within this class we will find many types of problems that require more or less sophisticated solutions. Our concern here is not to analyze the meaning of the English word "learning" as it is used in everyday language. Instead, our goal is to define precisely a class of problems that encompasses interesting forms of learning, to explore algorithms that solve such problems, and to understand the fundamental structure of learning problems and processes.

1.2 DESIGNING A LEARNING SYSTEM

In order to illustrate some of the basic design issues and approaches to machine learning, let us consider designing a program to learn to play checkers, with the goal of entering it in the world checkers tournament. We adopt the obvious performance measure: the percent of games it wins in this world tournament.

1.2.1 Choosing the Training Experience

The first design choice we face is to choose the type of training experience from which our system will learn. The type of training experience available can have a significant impact on success or failure of the learner. One key attribute is whether the training experience provides direct or indirect feedback regarding the choices made by the performance system. For example, in learning to play checkers, the system might learn from *direct* training examples consisting of individual checkers board states and the correct move for each. Alternatively, it might have available only *indirect* information consisting of the move sequences and final outcomes of various games played. In this later case, information about the correctness of specific moves early in the game must be inferred indirectly from the fact that the game was eventually won or lost. Here the learner faces an additional problem of *credit assignment*, or determining the degree to which each move in the sequence deserves credit or blame for the final outcome. Credit assignment can be a particularly difficult problem because the game can be lost even when early moves are optimal, if these are followed later by poor moves. Hence, learning from direct training feedback is typically easier than learning from indirect feedback.

A second important attribute of the training experience is the degree to which the learner controls the sequence of training examples. For example, the learner might rely on the teacher to select informative board states and to provide the correct move for each. Alternatively, the learner might itself propose board states that it finds particularly confusing and ask the teacher for the correct move. Or the learner may have complete control over both the board states and (indirect) training classifications, as it does when it learns by playing against itself with no teacher

present. Notice in this last case the learner may choose between experimenting with novel board states that it has not yet considered, or honing its skill by playing minor variations of lines of play it currently finds most promising. Subsequent chapters consider a number of settings for learning, including settings in which training experience is provided by a random process outside the learner's control, settings in which the learner may pose various types of queries to an expert teacher, and settings in which the learner collects training examples by autonomously exploring its environment.

A third important attribute of the training experience is how well it represents the distribution of examples over which the final system performance P must be measured. In general, learning is most reliable when the training examples follow a distribution similar to that of future test examples. In our checkers learning scenario, the performance metric P is the percent of games the system wins in the world tournament. If its training experience E consists only of games played against itself, there is an obvious danger that this training experience might not be fully representative of the distribution of situations over which it will later be tested. For example, the learner might never encounter certain crucial board states that are very likely to be played by the human checkers champion. In practice, it is often necessary to learn from a distribution of examples that is somewhat different from those on which the final system will be evaluated (e.g., the world checkers champion might not be interested in teaching the program!). Such situations are problematic because mastery of one distribution of examples will not necessary lead to strong performance over some other distribution. We shall see that most current theory of machine learning rests on the crucial assumption that the distribution of training examples is identical to the distribution of test examples. Despite our need to make this assumption in order to obtain theoretical results, it is important to keep in mind that this assumption must often be violated in practice.

To proceed with our design, let us decide that our system will train by playing games against itself. This has the advantage that no external trainer need be present, and it therefore allows the system to generate as much training data as time permits. We now have a fully specified learning task.

A checkers learning problem:

- Task T: playing checkers
- Performance measure P: percent of games won in the world tournament
- Training experience E: games played against itself

In order to complete the design of the learning system, we must now choose

1. the exact type of knowledge to be learned
2. a representation for this target knowledge
3. a learning mechanism

1.2.2 Choosing the Target Function

The next design choice is to determine exactly what type of knowledge will be learned and how this will be used by the performance program. Let us begin with a checkers-playing program that can generate the *legal* moves from any board state. The program needs only to learn how to choose the *best* move from among these legal moves. This learning task is representative of a large class of tasks for which the legal moves that define some large search space are known a priori, but for which the best search strategy is not known. Many optimization problems fall into this class, such as the problems of scheduling and controlling manufacturing processes where the available manufacturing steps are well understood, but the best strategy for sequencing them is not.

Given this setting where we must learn to choose among the legal moves, the most obvious choice for the type of information to be learned is a program, or function, that chooses the best move for any given board state. Let us call this function *ChooseMove* and use the notation *ChooseMove* : $B \rightarrow M$ to indicate that this function accepts as input any board from the set of legal board states B and produces as output some move from the set of legal moves M. Throughout our discussion of machine learning we will find it useful to reduce the problem of improving performance P at task T to the problem of learning some particular *target function* such as *ChooseMove*. The choice of the target function will therefore be a key design choice.

Although *ChooseMove* is an obvious choice for the target function in our example, this function will turn out to be very difficult to learn given the kind of indirect training experience available to our system. An alternative target function— and one that will turn out to be easier to learn in this setting—is an evaluation function that assigns a numerical score to any given board state. Let us call this target function V and again use the notation $V : B \rightarrow \Re$ to denote that V maps any legal board state from the set B to some real value (we use $\Re$ to denote the set of real numbers). We intend for this target function V to assign higher scores to better board states. If the system can successfully learn such a target function V, then it can easily use it to select the best move from any current board position. This can be accomplished by generating the successor board state produced by every legal move, then using V to choose the best successor state and therefore the best legal move.

What exactly should be the value of the target function V for any given board state? Of course any evaluation function that assigns higher scores to better board states will do. Nevertheless, we will find it useful to define one particular target function V among the many that produce optimal play. As we shall see, this will make it easier to design a training algorithm. Let us therefore define the target value $V(b)$ for an arbitrary board state b in B, as follows:

1. if b is a final board state that is won, then $V(b) = 100$
2. if b is a final board state that is lost, then $V(b) = -100$
3. if b is a final board state that is drawn, then $V(b) = 0$

4. if *b* is a not a final state in the game, then $V(b) = V(b')$, where b' is the best final board state that can be achieved starting from *b* and playing optimally until the end of the game (assuming the opponent plays optimally, as well).

While this recursive definition specifies a value of $V(b)$ for every board state *b*, this definition is not usable by our checkers player because it is not efficiently computable. Except for the trivial cases (cases 1–3) in which the game has already ended, determining the value of $V(b)$ for a particular board state requires (case 4) searching ahead for the optimal line of play, all the way to the end of the game! Because this definition is not efficiently computable by our checkers playing program, we say that it is a *nonoperational* definition. The goal of learning in this case is to discover an *operational* description of *V*; that is, a description that can be used by the checkers-playing program to evaluate states and select moves within realistic time bounds.

Thus, we have reduced the learning task in this case to the problem of discovering an *operational description of the ideal target function V*. It may be very difficult in general to learn such an operational form of *V* perfectly. In fact, we often expect learning algorithms to acquire only some *approximation* to the target function, and for this reason the process of learning the target function is often called *function approximation*. In the current discussion we will use the symbol $\hat{V}$ to refer to the function that is actually learned by our program, to distinguish it from the ideal target function *V*.

1.2.3 Choosing a Representation for the Target Function

Now that we have specified the ideal target function *V*, we must choose a representation that the learning program will use to describe the function $\hat{V}$ that it will learn. As with earlier design choices, we again have many options. We could, for example, allow the program to represent $\hat{V}$ using a large table with a distinct entry specifying the value for each distinct board state. Or we could allow it to represent $\hat{V}$ using a collection of rules that match against features of the board state, or a quadratic polynomial function of predefined board features, or an artificial neural network. In general, this choice of representation involves a crucial tradeoff. On one hand, we wish to pick a very expressive representation to allow representing as close an approximation as possible to the ideal target function *V*. On the other hand, the more expressive the representation, the more training data the program will require in order to choose among the alternative hypotheses it can represent. To keep the discussion brief, let us choose a simple representation: for any given board state, the function $\hat{V}$ will be calculated as a linear combination of the following board features:

- x_1: the number of black pieces on the board
- x_2: the number of red pieces on the board
- x_3: the number of black kings on the board
- x_4: the number of red kings on the board

- x_5: the number of black pieces threatened by red (i.e., which can be captured on red's next turn)
- x_6: the number of red pieces threatened by black

Thus, our learning program will represent $\hat{V}(b)$ as a linear function of the form

$$\hat{V}(b) = w_0 + w_1x_1 + w_2x_2 + w_3x_3 + w_4x_4 + w_5x_5 + w_6x_6$$

where w_0 through w_6 are numerical coefficients, or weights, to be chosen by the learning algorithm. Learned values for the weights w_1 through w_6 will determine the relative importance of the various board features in determining the value of the board, whereas the weight w_0 will provide an additive constant to the board value.

To summarize our design choices thus far, we have elaborated the original formulation of the learning problem by choosing a type of training experience, a target function to be learned, and a representation for this target function. Our elaborated learning task is now

Partial design of a checkers learning program:

- Task T: playing checkers
- Performance measure P: percent of games won in the world tournament
- Training experience E: games played against itself
- *Target function: $V:Board \rightarrow \Re$*
- *Target function representation*

$$\hat{V}(b) = w_0 + w_1x_1 + w_2x_2 + w_3x_3 + w_4x_4 + w_5x_5 + w_6x_6$$

The first three items above correspond to the specification of the learning task, whereas the final two items constitute design choices for the implementation of the learning program. Notice the net effect of this set of design choices is to reduce the problem of learning a checkers strategy to the problem of learning values for the coefficients w_0 through w_6 in the target function representation.

1.2.4 Choosing a Function Approximation Algorithm

In order to learn the target function $\hat{V}$ we require a set of training examples, each describing a specific board state b and the training value $V_{train}(b)$ for b. In other words, each training example is an ordered pair of the form $\langle b, V_{train}(b) \rangle$. For instance, the following training example describes a board state b in which black has won the game (note $x_2 = 0$ indicates that red has no remaining pieces) and for which the target function value $V_{train}(b)$ is therefore +100.

$$\langle \langle x_1 = 3, x_2 = 0, x_3 = 1, x_4 = 0, x_5 = 0, x_6 = 0 \rangle, +100 \rangle$$

Below we describe a procedure that first derives such training examples from the indirect training experience available to the learner, then adjusts the weights w_i to best fit these training examples.

1.2.4.1   ESTIMATING TRAINING VALUES

Recall that according to our formulation of the learning problem, the only training information available to our learner is whether the game was eventually won or lost. On the other hand, we require training examples that assign specific scores to specific board states. While it is easy to assign a value to board states that correspond to the end of the game, it is less obvious how to assign training values to the more numerous *intermediate* board states that occur before the game's end. Of course the fact that the game was eventually won or lost does not necessarily indicate that *every* board state along the game path was necessarily good or bad. For example, even if the program loses the game, it may still be the case that board states occurring early in the game should be rated very highly and that the cause of the loss was a subsequent poor move.

Despite the ambiguity inherent in estimating training values for intermediate board states, one simple approach has been found to be surprisingly successful. This approach is to assign the training value of $V_{train}(b)$ for any intermediate board state b to be $\hat{V}(Successor(b))$, where $\hat{V}$ is the learner's current approximation to V and where $Successor(b)$ denotes the next board state following b for which it is again the program's turn to move (i.e., the board state following the program's move and the opponent's response). This rule for estimating training values can be summarized as

Rule for estimating training values.

$$V_{train}(b) \leftarrow \hat{V}(Successor(b)) \tag{1.1}$$

While it may seem strange to use the current version of $\hat{V}$ to estimate training values that will be used to refine this very same function, notice that we are using estimates of the value of the $Successor(b)$ to estimate the value of board state b. Intuitively, we can see this will make sense if $\hat{V}$ tends to be more accurate for board states closer to game's end. In fact, under certain conditions (discussed in Chapter 13) the approach of iteratively estimating training values based on estimates of successor state values can be proven to converge toward perfect estimates of V_{train}.

1.2.4.2   ADJUSTING THE WEIGHTS

All that remains is to specify the learning algorithm for choosing the weights w_i to best fit the set of training examples $\{\langle b, V_{train}(b) \rangle\}$. As a first step we must define what we mean by the *best fit* to the training data. One common approach is to define the best hypothesis, or set of weights, as that which minimizes the squared error E between the training values and the values predicted by the hypothesis $\hat{V}$.

$$E \equiv \sum_{\langle b, V_{train}(b) \rangle \in \ training\ examples} (V_{train}(b) - \hat{V}(b))^2$$

Thus, we seek the weights, or equivalently the $\hat{V}$, that minimize E for the observed training examples. Chapter 6 discusses settings in which minimizing the sum of squared errors is equivalent to finding the most probable hypothesis given the observed training data.

Several algorithms are known for finding weights of a linear function that minimize E defined in this way. In our case, we require an algorithm that will incrementally refine the weights as new training examples become available and that will be robust to errors in these estimated training values. One such algorithm is called the least mean squares, or LMS training rule. For each observed training example it adjusts the weights a small amount in the direction that reduces the error on this training example. As discussed in Chapter 4, this algorithm can be viewed as performing a stochastic gradient-descent search through the space of possible hypotheses (weight values) to minimize the squared error E. The LMS algorithm is defined as follows:

LMS weight update rule.

For each training example $\langle b, V_{train}(b) \rangle$

- Use the current weights to calculate $\hat{V}(b)$
- For each weight w_i, update it as

$$w_i \leftarrow w_i + \eta \ (V_{train}(b) - \hat{V}(b)) \ x_i$$

Here η is a small constant (e.g., 0.1) that moderates the size of the weight update. To get an intuitive understanding for why this weight update rule works, notice that when the error $(V_{train}(b) - \hat{V}(b))$ is zero, no weights are changed. When $(V_{train}(b) - \hat{V}(b))$ is positive (i.e., when $\hat{V}(b)$ is too low), then each weight is increased in proportion to the value of its corresponding feature. This will raise the value of $\hat{V}(b)$, reducing the error. Notice that if the value of some feature x_i is zero, then its weight is not altered regardless of the error, so that the only weights updated are those whose features actually occur on the training example board. Surprisingly, in certain settings this simple weight-tuning method can be proven to converge to the least squared error approximation to the V_{train} values (as discussed in Chapter 4).

1.2.5 The Final Design

The final design of our checkers learning system can be naturally described by four distinct program modules that represent the central components in many learning systems. These four modules, summarized in Figure 1.1, are as follows:

- The **Performance System** is the module that must solve the given performance task, in this case playing checkers, by using the learned target function(s). It takes an instance of a new problem (new game) as input and produces a trace of its solution (game history) as output. In our case, the
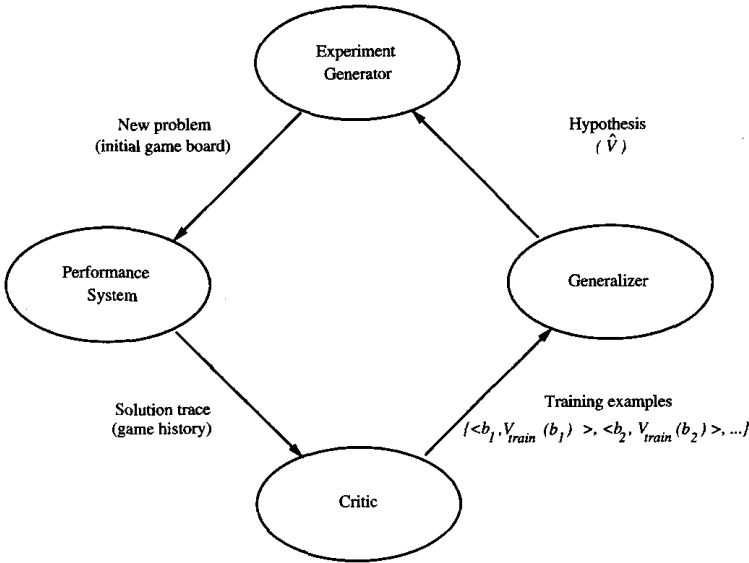
FIGURE 1.1
Final design of the checkers learning program.

strategy used by the Performance System to select its next move at each step
is determined by the learned $\hat{V}$ evaluation function. Therefore, we expect
its performance to improve as this evaluation function becomes increasingly
accurate.

- The **Critic** takes as input the history or trace of the game and produces as
output a set of training examples of the target function. As shown in the
diagram, each training example in this case corresponds to some game state
in the trace, along with an estimate V_{train} of the target function value for this
example. In our example, the Critic corresponds to the training rule given
by Equation (1.1).

- The **Generalizer** takes as input the training examples and produces an output
hypothesis that is its estimate of the target function. It generalizes from the
specific training examples, hypothesizing a general function that covers these
examples and other cases beyond the training examples. In our example, the
Generalizer corresponds to the LMS algorithm, and the output hypothesis is
the function $\hat{V}$ described by the learned weights $w_0, \ldots, w_6$.

- The **Experiment Generator** takes as input the current hypothesis (currently
learned function) and outputs a new problem (i.e., initial board state) for the
Performance System to explore. Its role is to pick new practice problems that
will maximize the learning rate of the overall system. In our example, the
Experiment Generator follows a very simple strategy: It always proposes the
same initial game board to begin a new game. More sophisticated strategies

could involve creating board positions designed to explore particular regions of the state space.

Together, the design choices we made for our checkers program produce specific instantiations for the performance system, critic, generalizer, and experiment generator. Many machine learning systems can be usefully characterized in terms of these four generic modules.

The sequence of design choices made for the checkers program is summarized in Figure 1.2. These design choices have constrained the learning task in a number of ways. We have restricted the type of knowledge that can be acquired to a single linear evaluation function. Furthermore, we have constrained this evaluation function to depend on only the six specific board features provided. If the true target function V can indeed be represented by a linear combination of these
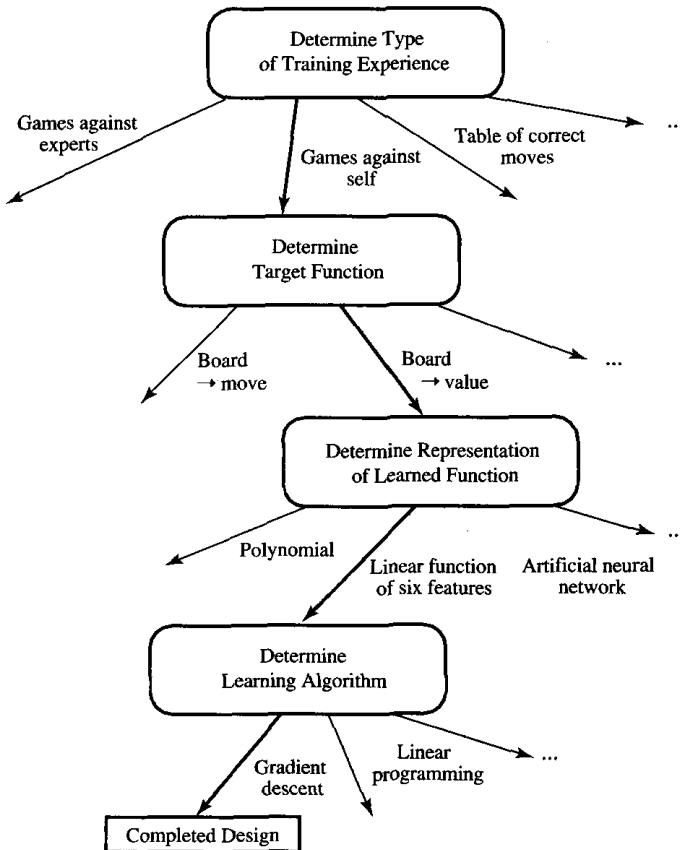
FIGURE 1.2
Summary of choices in designing the checkers learning program.

particular features, then our program has a good chance to learn it. If not, then the best we can hope for is that it will learn a good approximation, since a program can certainly never learn anything that it cannot at least represent.

Let us suppose that a good approximation to the true V function can, in fact, be represented in this form. The question then arises as to whether this learning technique is guaranteed to find one. Chapter 13 provides a theoretical analysis showing that under rather restrictive assumptions, variations on this approach do indeed converge to the desired evaluation function for certain types of search problems. Fortunately, practical experience indicates that this approach to learning evaluation functions is often successful, even outside the range of situations for which such guarantees can be proven.

Would the program we have designed be able to learn well enough to beat the human checkers world champion? Probably not. In part, this is because the linear function representation for $\hat{V}$ is too simple a representation to capture well the nuances of the game. However, given a more sophisticated representation for the target function, this general approach can, in fact, be quite successful. For example, Tesauro (1992, 1995) reports a similar design for a program that learns to play the game of backgammon, by learning a very similar evaluation function over states of the game. His program represents the learned evaluation function using an artificial neural network that considers the complete description of the board state rather than a subset of board features. After training on over one million self-generated training games, his program was able to play very competitively with top-ranked human backgammon players.

Of course we could have designed many alternative algorithms for this checkers learning task. One might, for example, simply store the given training examples, then try to find the "closest" stored situation to match any new situation (nearest neighbor algorithm, Chapter 8). Or we might generate a large number of candidate checkers programs and allow them to play against each other, keeping only the most successful programs and further elaborating or mutating these in a kind of simulated evolution (genetic algorithms, Chapter 9). Humans seem to follow yet a different approach to learning strategies, in which they analyze, or explain to themselves, the reasons underlying specific successes and failures encountered during play (explanation-based learning, Chapter 11). Our design is simply one of many, presented here to ground our discussion of the decisions that must go into designing a learning method for a specific class of tasks.

1.3 PERSPECTIVES AND ISSUES IN MACHINE LEARNING

One useful perspective on machine learning is that it involves searching a very large space of possible hypotheses to determine one that best fits the observed data and any prior knowledge held by the learner. For example, consider the space of hypotheses that could in principle be output by the above checkers learner. This hypothesis space consists of all evaluation functions that can be represented by some choice of values for the weights w_0 through w_6. The learner's task is thus to search through this vast space to locate the hypothesis that is most consistent with

the available training examples. The LMS algorithm for fitting weights achieves this goal by iteratively tuning the weights, adding a correction to each weight each time the hypothesized evaluation function predicts a value that differs from the training value. This algorithm works well when the hypothesis representation considered by the learner defines a continuously parameterized space of potential hypotheses.

Many of the chapters in this book present algorithms that search a hypothesis space defined by some underlying representation (e.g., linear functions, logical descriptions, decision trees, artificial neural networks). These different hypothesis representations are appropriate for learning different kinds of target functions. For each of these hypothesis representations, the corresponding learning algorithm takes advantage of a different underlying structure to organize the search through the hypothesis space.

Throughout this book we will return to this perspective of learning as a search problem in order to characterize learning methods by their search strategies and by the underlying structure of the search spaces they explore. We will also find this viewpoint useful in formally analyzing the relationship between the size of the hypothesis space to be searched, the number of training examples available, and the confidence we can have that a hypothesis consistent with the training data will correctly generalize to unseen examples.

1.3.1 Issues in Machine Learning

Our checkers example raises a number of generic questions about machine learning. The field of machine learning, and much of this book, is concerned with answering questions such as the following:

- What algorithms exist for learning general target functions from specific training examples? In what settings will particular algorithms converge to the desired function, given sufficient training data? Which algorithms perform best for which types of problems and representations?

- How much training data is sufficient? What general bounds can be found to relate the confidence in learned hypotheses to the amount of training experience and the character of the learner's hypothesis space?

- When and how can prior knowledge held by the learner guide the process of generalizing from examples? Can prior knowledge be helpful even when it is only approximately correct?

- What is the best strategy for choosing a useful next training experience, and how does the choice of this strategy alter the complexity of the learning problem?

- What is the best way to reduce the learning task to one or more function approximation problems? Put another way, what specific functions should the system attempt to learn? Can this process itself be automated?

- How can the learner automatically alter its representation to improve its ability to represent and learn the target function?

1.4 HOW TO READ THIS BOOK

This book contains an introduction to the primary algorithms and approaches to machine learning, theoretical results on the feasibility of various learning tasks and the capabilities of specific algorithms, and examples of practical applications of machine learning to real-world problems. Where possible, the chapters have been written to be readable in any sequence. However, some interdependence is unavoidable. If this is being used as a class text, I recommend first covering Chapter 1 and Chapter 2. Following these two chapters, the remaining chapters can be read in nearly any sequence. A one-semester course in machine learning might cover the first seven chapters, followed by whichever additional chapters are of greatest interest to the class. Below is a brief survey of the chapters.

- Chapter 2 covers concept learning based on symbolic or logical representations. It also discusses the general-to-specific ordering over hypotheses, and the need for inductive bias in learning.

- Chapter 3 covers decision tree learning and the problem of overfitting the training data. It also examines Occam's razor—a principle recommending the shortest hypothesis among those consistent with the data.

- Chapter 4 covers learning of artificial neural networks, especially the well-studied BACKPROPAGATION algorithm, and the general approach of gradient descent. This includes a detailed example of neural network learning for face recognition, including data and algorithms available over the World Wide Web.

- Chapter 5 presents basic concepts from statistics and estimation theory, focusing on evaluating the accuracy of hypotheses using limited samples of data. This includes the calculation of confidence intervals for estimating hypothesis accuracy and methods for comparing the accuracy of learning methods.

- Chapter 6 covers the Bayesian perspective on machine learning, including both the use of Bayesian analysis to characterize non-Bayesian learning algorithms and specific Bayesian algorithms that explicitly manipulate probabilities. This includes a detailed example applying a naive Bayes classifier to the task of classifying text documents, including data and software available over the World Wide Web.

- Chapter 7 covers computational learning theory, including the Probably Approximately Correct (PAC) learning model and the Mistake-Bound learning model. This includes a discussion of the WEIGHTED MAJORITY algorithm for combining multiple learning methods.

- Chapter 8 describes instance-based learning methods, including nearest neighbor learning, locally weighted regression, and case-based reasoning.

- Chapter 9 discusses learning algorithms modeled after biological evolution, including genetic algorithms and genetic programming.

- Chapter 10 covers algorithms for learning sets of rules, including Inductive Logic Programming approaches to learning first-order Horn clauses.

- Chapter 11 covers explanation-based learning, a learning method that uses prior knowledge to explain observed training examples, then generalizes based on these explanations.

- Chapter 12 discusses approaches to combining approximate prior knowledge with available training data in order to improve the accuracy of learned hypotheses. Both symbolic and neural network algorithms are considered.

- Chapter 13 discusses reinforcement learning—an approach to control learning that accommodates indirect or delayed feedback as training information. The checkers learning algorithm described earlier in Chapter 1 is a simple example of reinforcement learning.

The end of each chapter contains a summary of the main concepts covered, suggestions for further reading, and exercises. Additional updates to chapters, as well as data sets and implementations of algorithms, are available on the World Wide Web at http://www.cs.cmu.edu/~tom/mlbook.html.

1.5 SUMMARY AND FURTHER READING

Machine learning addresses the question of how to build computer programs that improve their performance at some task through experience. Major points of this chapter include:

- Machine learning algorithms have proven to be of great practical value in a variety of application domains. They are especially useful in (a) data mining problems where large databases may contain valuable implicit regularities that can be discovered automatically (e.g., to analyze outcomes of medical treatments from patient databases or to learn general rules for credit worthiness from financial databases); (b) poorly understood domains where humans might not have the knowledge needed to develop effective algorithms (e.g., human face recognition from images); and (c) domains where the program must dynamically adapt to changing conditions (e.g., controlling manufacturing processes under changing supply stocks or adapting to the changing reading interests of individuals).

- Machine learning draws on ideas from a diverse set of disciplines, including artificial intelligence, probability and statistics, computational complexity, information theory, psychology and neurobiology, control theory, and philosophy.

- A well-defined learning problem requires a well-specified task, performance metric, and source of training experience.

- Designing a machine learning approach involves a number of design choices, including choosing the type of training experience, the target function to be learned, a representation for this target function, and an algorithm for learning the target function from training examples.

- Learning involves search: searching through a space of possible hypotheses to find the hypothesis that best fits the available training examples and other prior constraints or knowledge. Much of this book is organized around different learning methods that search different hypothesis spaces (e.g., spaces containing numerical functions, neural networks, decision trees, symbolic rules) and around theoretical results that characterize conditions under which these search methods converge toward an optimal hypothesis.

There are a number of good sources for reading about the latest research results in machine learning. Relevant journals include *Machine Learning, Neural Computation, Neural Networks, Journal of the American Statistical Association,* and the *IEEE Transactions on Pattern Analysis and Machine Intelligence.* There are also numerous annual conferences that cover different aspects of machine learning, including the International Conference on Machine Learning, Neural Information Processing Systems, Conference on Computational Learning Theory, International Conference on Genetic Algorithms, International Conference on Knowledge Discovery and Data Mining, European Conference on Machine Learning, and others.

EXERCISES

1.1. Give three computer applications for which machine learning approaches seem appropriate and three for which they seem inappropriate. Pick applications that are not already mentioned in this chapter, and include a one-sentence justification for each.

1.2. Pick some learning task not mentioned in this chapter. Describe it informally in a paragraph in English. Now describe it by stating as precisely as possible the task, performance measure, and training experience. Finally, propose a target function to be learned and a target representation. Discuss the main tradeoffs you considered in formulating this learning task.

1.3. Prove that the LMS weight update rule described in this chapter performs a gradient descent to minimize the squared error. In particular, define the squared error E as in the text. Now calculate the derivative of E with respect to the weight w_i, assuming that $\hat{V}(b)$ is a linear function as defined in the text. Gradient descent is achieved by updating each weight in proportion to $-\frac{\partial E}{\partial w_i}$. Therefore, you must show that the LMS training rule alters weights in this proportion for each training example it encounters.

1.4. Consider alternative strategies for the Experiment Generator module of Figure 1.2. In particular, consider strategies in which the Experiment Generator suggests new board positions by
  - Generating random legal board positions
  - Generating a position by picking a board state from the previous game, then applying one of the moves that was not executed
  - A strategy of your own design
  Discuss tradeoffs among these strategies. Which do you feel would work best if the number of training examples was held constant, given the performance measure of winning the most games at the world championships?

1.5. Implement an algorithm similar to that discussed for the checkers problem, but use the simpler game of tic-tac-toe. Represent the learned function $\hat{V}$ as a linear com-

bination of board features of your choice. To train your program, play it repeatedly against a second copy of the program that uses a fixed evaluation function you create by hand. Plot the percent of games won by your system, versus the number of training games played.

REFERENCES

Ahn, W., & Brewer, W. F. (1993). Psychological studies of explanation-based learning. In G. DeJong (Ed.), *Investigating explanation-based learning*. Boston: Kluwer Academic Publishers.

Anderson, J. R. (1991). The place of cognitive architecture in rational analysis. In K. VanLehn (Ed.), *Architectures for intelligence* (pp. 1–24). Hillsdale, NJ: Erlbaum.

Chi, M. T. H., & Bassock, M. (1989). Learning from examples via self-explanations. In L. Resnick (Ed.), *Knowing, learning, and instruction: Essays in honor of Robert Glaser*. Hillsdale, NJ: L. Erlbaum Associates.

Cooper, G., et al. (1997). An evaluation of machine-learning methods for predicting pneumonia mortality. *Artificial Intelligence in Medicine*, (to appear).

Fayyad, U. M., Uthurusamy, R. (Eds.) (1995). *Proceedings of the First International Conference on Knowledge Discovery and Data Mining*. Menlo Park, CA: AAAI Press.

Fayyad, U. M., Smyth, P., Weir, N., Djorgovski, S. (1995). Automated analysis and exploration of image databases: Results, progress, and challenges. *Journal of Intelligent Information Systems*, 4, 1–19.

Laird, J., Rosenbloom, P., & Newell, A. (1986). SOAR: The anatomy of a general learning mechanism. *Machine Learning*, 1(1), 11–46.

Langley, P., & Simon, H. (1995). Applications of machine learning and rule induction. *Communications of the ACM*, 38(11), 55–64.

Lee, K. (1989). *Automatic speech recognition: The development of the Sphinx system*. Boston: Kluwer Academic Publishers.

Pomerleau, D. A. (1989). *ALVINN: An autonomous land vehicle in a neural network*. (Technical Report CMU-CS-89-107). Pittsburgh, PA: Carnegie Mellon University.

Qin, Y., Mitchell, T., & Simon, H. (1992). Using EBG to simulate human learning from examples and learning by doing. *Proceedings of the Florida AI Research Symposium* (pp. 235–239).

Rudnicky, A. I., Hauptmann, A. G., & Lee, K. -F. (1994). Survey of current speech technology in artificial intelligence. *Communications of the ACM*, 37(3), 52–57.

Rumelhart, D., Widrow, B., & Lehr, M. (1994). The basic ideas in neural networks. *Communications of the ACM*, 37(3), 87–92.

Tesauro, G. (1992). Practical issues in temporal difference learning. *Machine Learning*, 8, 257.

Tesauro, G. (1995). Temporal difference learning and TD-gammon. *Communications of the ACM*, 38(3), 58–68.

Waibel, A., Hanazawa, T., Hinton, G., Shikano, K., & Lang, K. (1989). Phoneme recognition using time-delay neural networks. *IEEE Transactions on Acoustics, Speech and Signal Processing*, 37(3), 328–339.