This paper appeared in 1994 in *Machine Learning*, 15 (3): 251-277.

Toward an Ideal Trainer*

Susan L. Epstein
Department of Computer Science
Hunter College and The Graduate School of The City University of New York

Abstract

This paper demonstrates how the nature of the opposition during training affects learning to play two-person, perfect information board games. It considers different kinds of competitive training, the impact of trainer error, appropriate metrics for post-training performance measurement, and the ways those metrics can be applied. The results suggest that teaching a program by leading it repeatedly through the same restricted paths, albeit high quality ones, is overly narrow preparation for the variations that appear in real-world experience. The results also demonstrate that variety introduced into training by random choice is unreliable preparation, and that a program that directs its own training may overlook important situations. The results argue for a broad variety of training experience with play at many levels. This variety may either be inherent in the game or introduced deliberately into the training. Lesson and practice training, a blend of expert guidance and knowledge-based, self-directed elaboration, is shown to be particularly effective for learning during competition.

Key words: training, competition, game playing, reliability

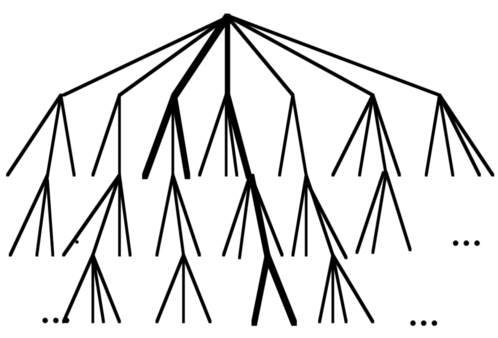
1. Introduction

To acquire expertise in an intractably-large problem space, a learner must somehow extract key knowledge inherent in the domain from a tiny fraction of it (Ericsson & Smith, 1991). A *trainer* guides the learner through the space, perhaps by providing an instruction sequence of positive and negative examples, or by serving as a model of behavior. The fragment of the problem space that the learning program experiences is influenced by the places to which its trainer leads it. After exposure to a limited portion of the search space, the learner is expected to function as if it were an expert in the entire space.

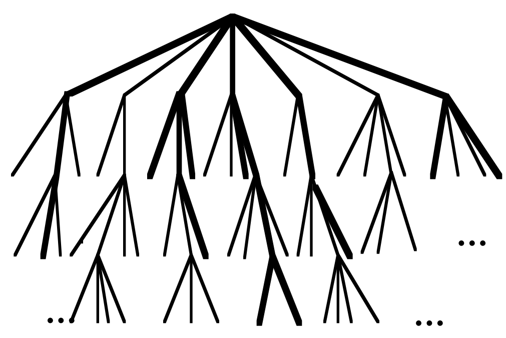
In particular, when a program learns to play a game, it is effectively trained by the nature of the opposition it encounters. Most of the recent game-learning programs develop their skill during training, a sequence of contests played against a fixed opponent (Epstein, 1992; Levinson & Snyder, 1991; Tesauro, 1992). The simulation of intelligence, however, requires that tasks that are easy for people are also easy for an intelligent machine, and that those more difficult for people are also more difficult for the machine (Turing, 1963). An intelligent game player, therefore, should be able not only to hold its own against other experts, but also to exploit its opposition's errors and to deal well with foolish moves. A program that learned to play a game extremely well against a perfect player or an expert player, but was readily and regularly defeated by a novice or a random player, would fail what is introduced here as the game players' Turing test. The central question addressed in this paper is what kind of training enables a game-learning program to become an intelligent expert, i.e., to pass the game players' Turing test.

The thesis of this paper is that the acquisition of absolute expertise in a competitive domain demands a broad variety of challenging experience as well as more thorough testing and more thoughtful evaluation than traditionally anticipated. The primary contribution of this paper is its empirical demonstration that a new general method, called lesson and practice training, is a better way to structure learning in a competitive environment than the ways commonly used. Lesson and practice training emphasizes the key portions of a very large search space: those that an expert would highlight (the lesson) and those that the learner selects and explores to expand its understanding of the task at hand (the practice). The secondary contribution of this paper is its formulation of an empirical structure and appropriate metrics to analyze the impact of different kinds of opposition on game learning. Because these experiments seek accurate measures of performance in a tractable and well-understood search space, the games examined are far simpler than chess.

An informal analysis explains the need for thoughtfully structured training and provides the intuition behind these experiments. A *perfect player* behaves as if it had searched the game tree exhaustively and minimaxed the result back up to the current state to select its next move (Nilsson, 1980). A perfect player that repeatedly played a game exactly the same (correct) way, for example, by always opening with the same move, would make no mistakes, but would also be unproductively rigid. If a perfect player is to offer a reasonable learning experience, it must provide a broad variety of high-quality play: make the best possible move and, when there is more than one best move, make different ones on different occasions. A perfect player used as a trainer for tic-tac-toe, for example, would open half its contests as X in the center and half in some randomly chosen corner.[1] A moment's consideration, however, makes it clear that a perfect player would not be a very good trainer. Figure 1(a) shows how the use of a perfect player as a trainer might direct learning within a hypothetical search tree whose most important nodes[2] lie in the center subtrees. Each darkened path in the tree represents a training experience. In the diagram, the perfect player leads the learning program through a narrow set of paths that reflect how an expert behaves, to form what is referred to here as the *perfect play skeleton*. This relative narrowness reflects the degree of skill involved and the difficulty of the task; the insight a highly skilled individual displays at a difficult task is to identify a relatively small fragment of the alternatives as the best ones (Ericsson, et al., 1991). When used as a trainer, the perfect player focuses the learner's attention on the places where the key ideas supporting expertise against another perfect player should somehow be found. Learning, while not entirely restricted to those paths, typically occurs only in their vicinity (Ericsson, et al., 1991). Once the program is set loose in the entire space, if forced to deviate broadly from those paths it is likely to be unprepared; the training experience has simply been too narrow. A program with a perfect player as its trainer is likely to fail the game players' Turing test when its opponent deviates from perfect play.



(a) The perfect play skeleton

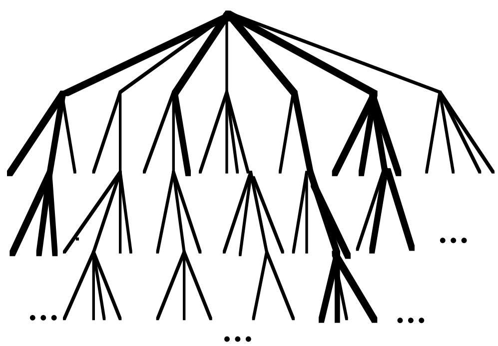


(b) Fallible training

---

[1]The center and a corner are roughly equally good openings. Although a perfect player for tic-tac-toe could open anywhere and go on to draw the contest, a wise competitor would choose the other openings because they offer greater opportunity to win if the other contestant makes a mistake. "A perfect player as trainer" is used in the sense of "a wise competitor" throughout this paper.

[2]The identification and relative scarcity in the search space of these "important places" are non-trivial. They are the central focus of work now in progress. For now, these nodes and the edges joining them are described merely as "most important," i.e., a learner must visit them to develop adequate expertise.



(c) Self-training (d) Lesson and practice training

Figure 1: Portions of a search space encountered during a variety of training: (a) training against a perfect player, (b) fallible training, (c) self-training, (d) lesson and practice training.

The obvious solution to this narrowness is the introduction of variety into the trainer, but how much variety to introduce, and how to introduce it, are not so clear. A *fallible trainer* adds some intermittent randomization to a perfect player to broaden the experience it offers the learner; it may be thought of as a perfect player with a stochastic opportunity (the noise parameter *e*) to make a mistake. 100 - *e*% of the time the fallible trainer makes the best move, or selects at random from among more than one equally good move; *e*% of the time the fallible trainer makes a randomly-chosen, legal, not necessarily imperfect move. Consider, for example, the perfect player for tic-tac-toe that opens half the time in the center and the rest of the time in a corner. A fallible trainer with $e = 10$ for that game would open 45% of the time in the center, 45% of the time in a randomly-chosen corner, and 10% of the time it will make any legal opening move, i.e., the center, a corner, or a side square.[3] In the spectrum of *e* values, $e = 0$ uses a perfect player as trainer and $e = 100$ uses a purely random trainer. The latter would be poor preparation because it is unlikely to explore a very large search space methodically. Some non-zero percentage of noise, however, as depicted in Figure 1(b), adds to the perfect play skeleton some scattered additional paths spread more widely about, suggesting a broader experience base. There is no guarantee that these additional paths pursue consistent ideas, i.e., represent variations likely to be encountered from intelligent, non-random opposition. A program with a fallible trainer is likely to fail the game players' Turing test when its opponent deviates from perfect play with consistent, non-random intent.

Self-training forces the learning program to play both sides in every contest. With self-training the program methodically determines and explores its own paths through the search space. Self-training is often suggested as a more natural environment, one that would ideally lead the learner through phases of random, novice, and expert performance. There is little guarantee, however, that the portions of the space investigated under self-training are the most significant ones. The areas explored may overlap very little with the perfect play skeleton. Self-training could instead resemble the diagram in Figure 1(c), where some paths are repeatedly broadened and thoroughly investigated, but not the most important ones. Thus a self-trained program is likely to fail the game players' Turing test when its opponent plays perfectly.

The best training experience would methodically broaden the perfect play skeleton in a non-random, consistent way to produce the kind of thoughtful elaboration represented in Figure 1(d). The result would be a program that learns not only the narrow paths of expertise, but also the reasonable, if less perfect, variations on them. *Lesson and practice training,* the new method introduced here, is intended to produce a pattern like Figure 1(d); its lessons impart high-quality

[3]Although a 10% noise level may seem high, the noise level must be considered within the context of the *branch factor*, the average number of legal alternative moves. A smaller branch factor decreases the likelihood of an error. For example, if there are on average three legal moves in a contest, only one of which is correct, a random selection still has a 33% chance of making the right one. Thus there is only a $.1(.67) = .067$ chance of making a mistake, even though $e = 10$.

information from the perfect play skeleton, and its practice provides non-random, self-chosen, relevant, methodical, knowledge-based elaboration on that framework.

This paper provides empirical support for the intuitions of Figure 1 in game learning. The research was done with a specific program called Hoyle. Although Hoyle has thus far learned to play 18 games like an expert, the study described here is restricted to in-depth experiments on three of them. The results, however, have been found applicable to the others in Hoyle's repertoire, and have intimations for other programs and other domains as well. The next section explains the experimental design. The study then begins, in Section 3, with a demonstration of the inadequacies of training against a perfect player, self-training, and fallible training. Section 4 produces the much stronger results for lesson and practice training, and Section 5 considers some variations on the experimental design. Subsequent sections discuss related work and conclusions.

2. Experimental Design

This section delineates what is to be learned and the standards to which that learning should be held. In this context, it is important to distinguish among a game, a contest, and a tournament. A *game* is an activity with a board, playing pieces, and rules. A *draw game* is one where two perfect players always tie against each other. A *contest* is an experience playing a game, beginning at an initial situation specified by the rules and ending when the rules declare a winner or a draw. A *tournament* is a sequence of contests at a specific game in which the participants alternately go first or second. For example, chess is a game at which two people might play a tournament of 16 contests.

In the experiments described here, a learning program called Hoyle is tested on three different draw games that present a range of challenges. The goal of training is to pass the game players' Turing test, as defined below. The program learns with a single trainer and is then evaluated against different kinds of opposition. Section 5 examines variations on this design.

2.1 The learning and testing cycle

A *trial* consists of a learning tournament followed by a testing tournament. At the beginning of a trial, the learning program has no specific knowledge about any of the games. A *learning experience* is determined by the choice of a game and a trainer. Once the game and the trainer for a trial are specified, the learning program plays a tournament of contests at the specified game with its designated trainer. After the program is judged to have learned to play, learning is turned off, and its skill is evaluated in a *testing experience,* a 20-contest tournament at the same game against each of four challengers (described below).

The learning experience is non-deterministic, with respect to both duration and move selection. Learning ends when the program meets a *behavioral standard*, i.e., when it draws or wins n consecutive contests against an external trainer, or when it draws n consecutive contests against itself.[4] Since the trainer or the learning program can make random legal move choices from time to time, a single run for a trial may not be representative of the trainer's impact on learning performance. To compensate for this uncertainty, each trial is run five times, and the results are averaged.

2.2 The learning program

Hoyle is based on FORR, a general architecture for learning and problem solving that postulates and capitalizes upon regularities (Epstein, To appear). Hoyle's domain is two-person, perfect information, finite board games. Given the definition of a new game, Hoyle begins as a rule-

---

[4]During self-training the program plays, and learns, as both contestants in every contest. If Hoyle defeats itself in a draw game, then it has certainly won on one side, but it has also made an error and lost on the other.

abiding novice that plays against an external, presumably expert, model. This model is only observed, never queried. As Hoyle plays, it gradually improves, often becoming expert or even perfect at a game.

Each game Hoyle can play is an instantiation of a general definition, a pre-specified, input instance, of its *game frame*. The only specific knowledge Hoyle has about a new game before playing is the values associated with the slots in the game frame. Some slots hold constants: the name of the game, the markers assigned to each participant, the initial state of the board before play, whether the board is two-dimensional or three-dimensional, how often to scroll the screen during play, which sets of locations on the board are considered adjacent in games where pieces may slide, which lines on the board are considered wins if the game is won by achieving a particular configuration. Other slots name LISP functions that display the current game state on the screen, read and filter input moves, generate and effect legal moves, detect the end of a contest and who has won, and transform the board back and forth between a list and a visual representation. These functions are very brief, typically a total of less than 100 lines of code per game.

Hoyle's *game-playing algorithm* is a script that provides pre-defined, uniform procedural direction to the program. The game-playing algorithm enables Hoyle to perform as if it were experienced in game playing, but without expertise at any particular game. This script detects when it is the program's turn to move, ensures that the participants alternately make legal moves, and announces the end of each contest, along with any winner. Given any valid game definition and the game-playing algorithm, Hoyle simulates a rule-abiding novice at the game, one that makes legal, if not astute, moves.

The game-playing algorithm also triggers Hoyle's Learner. The *Learner* is a set of algorithms for the discovery of *useful knowledge*, knowledge that is expected to be relevant and may be correct. Based on its playing experience, Hoyle computes and stores game-dependent useful knowledge. The Learner has at least one uniform, heuristic, game-independent learning procedure for each item of useful knowledge. There are useful knowledge slots to record average contest length, applicable two-dimensional symmetries, good openings, moves that expert opposition appears to have found valuable, relevant forks,[5] important contest histories, whether going first or second is an advantage, dangerous states, and significant states. A *dangerous state* is a situation that, while not yet exhaustively demonstrated to be a certain loss, appears to have losing repercussions. A *significant state* is a situation that results in an inevitable win or loss with perfect play. Significant loss states become situations to avoid; significant win states both improve the endgame and give insight into the middlegame.

The learning strategies vary from one procedure to the next. Average contest length is tabulated after a tournament. Applicable two-dimensional symmetries are induced from pre-tournament queries to rules of the form "If state A is a win, then is state B, created from state A by the symmetry *s*, also a win?" Good openings are the first *f*% of the moves in a contest where Hoyle moves second and loses, with the parameter *f* currently set at 10. Relevant forks are first computed from the topology of the board and then extracted by explanation-based learning (Epstein, 1990). Contest histories are retained in full if they produce significant states or an unexpected winner. Going first or second is calculated to be an advantage from tournament histories. Significant states are calculated from selective retrograde analysis (Epstein, 1992).

If the Learner were to retain everything Hoyle experiences, useful knowledge for an interesting game could quickly become unmanageably large. Therefore the learning algorithms generalize and are highly selective about what they retain. Once useful knowledge is acquired, however, there is no facility for forgetting. Thus, as long as novel portions of the search space are encountered, one would expect to see a strong correlation between the time devoted to learning and the storage space allocated for useful knowledge.

The application of learned useful knowledge is the task of Hoyle's Advisors. An *Advisor* is a

---

[5]Informally, a *fork* is a bipartite graph representation of intersecting multiple plans to win, and is applicable to states that are not symmetrically equivalent. See (Epstein, 1990) for details on forks and how they are learned.

heuristic procedure that makes comments about legal moves when it is the program's turn to play. A *comment* consists of the Advisor's name, a move, and a *strength*, an integer from 0 to 10, indicating an opinion somewhere between strong aversion (0) and enthusiastic support (10). Each Advisor constructs its comments based upon the current state and the useful knowledge for the current game. For example, the Advisor Victory compares useful knowledge with the current legal moves, and recommends with a strength of 10 each legal move that results in an immediate win.

Whenever it is Hoyle's turn to move, the game-playing algorithm provides the Advisors with the current game state, the legal moves, and any useful knowledge about the game already acquired. (If Hoyle has had little or no experience at this particular game, there may be no useful knowledge.) Based on the Advisors' comments, a simple arithmetic calculation selects a move that is forwarded to the game-playing algorithm for execution. This calculation, it will be seen, provides a knowledge-based method to introduce the reasonable, non-random variation that good training requires. Ties are broken by random selection among the most promising moves.

2.3 The evaluation criteria

The evaluation criteria used here reflect the central concerns of this work: to produce a powerful, reliable, robust player. Hoyle's post-learning skill is thoroughly tested against four kinds of opposition: a perfect challenger, an expert challenger, a novice challenger, and a random challenger. The four offer a broad variety of competitive experience. The *perfect challenger* is the perfect player described earlier. The *expert challenger* simulates a not-quite-perfect expert, equivalent to a fallible trainer with a 10% noise level: 90% of the time it plays flawlessly, 10% of the time it may err. The *novice challenger* simulates an experienced beginner, equivalent to a fallible trainer with a 70% noise level; only 30% of the time is it guaranteed to play perfectly.[6] Finally, the *random challenger* makes random legal moves, equivalent to a fallible trainer with a 100% noise level.

Table 1: Maximal power (percentage of wins) achieved by a perfect player against each of the non-perfect challengers at three games. Data was computed from tournaments of 10,000 contests each.

	Expert challenger	**Novice challenger**	**Random challenger**
Tic-tac-toe	17	79	93
Lose tic-tac-toe	16	66	74
Achi	45	99.7	99.98

A game-playing expert should be both *reliable* (make no errors, no matter how skilled the opposition) and *powerful* (fully exploit errors made by its opposition). At a draw game, by definition, a loss is a clear indication of a mistake on the part of the loser. *Reliability,* the ability to withstand the competition, can be measured for a draw game by the percentage of non-losses, i.e., of wins and draws. Similarly, a win at a draw game indicates successful exploitation of a fatal error made by the opposition. *Power*, the ability to exploit the opposition's errors, is measured for a draw game by the percentage of wins. Maximal achievable power is a function of the game played and the number of errors made by the opposition. Against a perfect player for a draw game, power is always 0%. If the opposition makes mistakes, the maximal achievable power can be measured empirically. Table 1 displays the empirical maximal achievable power (here called *maximal power*) as a function of a game and a challenger. Each value was computed from a 10,000 contest tournament at the game between a perfect player and the specified challenger.

A program is said to *meet the game players' Turing test* if and only if:

- it is within the 95% confidence limit of 100% reliability
- it is within the 95% confidence limit of maximal power

---

[6]Noise levels for both the expert and the novice were selected from laboratory observation of their resultant play quality.

• its reliability and power monotonically increase with the lack of skill of its opponent (the noise level in fallible training).

How nearly various trainers approach this goal is the focus of this study.[7] Throughout, Cox and Stuart's non-parametric binomial test for trend is used to decide if there is a correlation between two sequences of numbers (Conover, 1980). For example, a correlation between *e* value and power is tested by the application of this test to the sequence 0, 10,…,100 and the power achieved when *e* takes that value. All correlations cited are 93.75% or higher unless otherwise indicated.

2.4 The games

The three games in this study are tic-tac-toe, lose tic-tac-toe, and an African game called achi. *Tic-tac-toe* is played on a three-by-three grid. One participant has five X's, and the other has four O's. Initially the board is empty, and X moves first. A turn is placing one's letter in any empty square. The first participant to place three of the same letters in a row, vertically, horizontally, or diagonally, wins. There are eight such winning lines; play ends in a draw when there are no more empty squares. *Lose tic-tac-toe* is played the same way as tic-tac-toe, except that the first one to place three of the same markers in a row, vertically, horizontally, or diagonally, loses. *Achi* is played on the board in Figure 2. The first participant has, and may only move, four black markers; the second has, and may only move, four white ones. In the first stage of achi, the board is empty, black moves first, and a turn is placing a marker on the intersection of two or more lines; there are nine such positions. In the second stage, when all eight markers are on the board, a turn is moving a marker to the single empty position. The first one to place three of the same markers in a row, vertically, horizontally, or diagonally, wins. There are eight such winning lines. Play ends in a draw when it cycles through the same state for the fourth time.

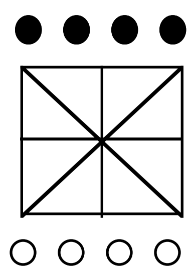


Figure 2. The initial state for achi.

These particular games were selected for several reasons. The three have certain commonalities that make comparison appropriate: their boards are isomorphic, each is known to be a draw game, and they do not impose any elements of chance (like dice) that would provide inherent variation. The games also differ from each other in ways that could be significant. Lose tic-tac-toe is distinguished from the others because its object is to *avoid* achieving a pattern, and because a randomly chosen move for X is likely to be a fatal error. Achi is distinguished by its two stages with different move types, its *cyclic* play (some states may occur more than once in the same contest), and its constant branch factor in the second stage. Finally, the construction of a perfect player and fallible trainers requires a complete and correct *perfect play theory* for a game, a real-time algorithm that identifies the best possible move from every possible game state. The construction of a thoughtful but partially flawed evaluation function (for some of the alternative

---

[7]Testing after training is not exhaustive. Although exhaustive testing (in every game state) would be feasible for the small search spaces used here, the intent is to see how learning manifests itself in play. The testing used here is based instead on contest outcome; it seeks effective play, rather than perfect play. It demands, however, that the participant take full advantage of the opposition's mistakes.

trainers described in Section 5) requires good human understanding of the features of a game. The games for which people have both a perfect play theory and such understanding are fairly simple games like these. One way to make the learning task more difficult in such a domain is to select games where the perfect play theory known to humans is not naturally expressible in the learning program's representation. Lose tic-tac-toe has a known complete and correct perfect play theory, one that is not quickly expressible in Hoyle's useful knowledge framework (Cohen, 1972).

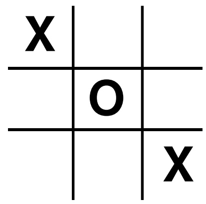


Figure 3: A difficult situation where Hoyle, and many people, will play incorrectly. O may believe that all is lost because of the potential forks in both the unoccupied corners. The correct move for O here is to any non-corner, empty space.

Because the three games present Hoyle with different challenges, one would expect different performance on each of them. Hoyle's Advisors are immediately able to support a fairly high level of play at tic-tac-toe. Indeed, against some trainers Hoyle never loses a tic-tac-toe contest. There is, however, a game state involving a simple fork from a corner opening, where Hoyle, before learning, will always make an incorrect move and lose the contest. This fork is shown is Figure 3 and explained in the caption. The fallible trainer with $e = 0$ periodically presents this game state; Hoyle fails, learns from its loss, and never makes a mistake in that state, or one symmetric to it, again. During training, however, there is no guarantee as to how soon the state in Figure 3 will arise, particularly with a fallible trainer where $e > 0$. Hoyle can learn many other things during training, but this particular piece of useful knowledge is essential for perfect reliability.

Lose tic-tac-toe is a game where relatively few moves are optimal, and even a single suboptimal move usually costs one the contest. For X there is typically exactly one correct move, including the single, non-intuitive correct opening in the center. Many people are able to learn the optimal defensive strategy for each side; the offensive strategy is non-trivial and much more difficult to acquire. As a result, the lose tic-tac-toe perfect play algorithm must be quite rigid and offers little opportunity to acquire breadth of experience during training. Table 1 indicates that, even so, it is more difficult to win at lose tic-tac-toe than at tic-tac-toe simply from an error by the other participant.

Achi is a game where most serious errors occur early, in the stage when markers are first placed on the board. A state isomorphic to Figure 3, for example, must be dealt with as it is in tic-tac-toe. Contests average 68 moves, offering ample opportunity for careless error while play cycles to a draw. Adjustment of the data in Table 1 for contest length indicates that random move selection at achi is about twice as likely to cost one the contest as an error at either of the other games. Achi has some unanticipated properties that, in retrospect, account for the speed and ease with which Hoyle learns achi in this study, compared to tic-tac-toe and lose tic-tac-toe. First, the other games average nine moves per contest, while achi averages 68 moves in a search space of roughly the same size. Thus the behavioral standard, measured in number of contests, exposes Hoyle to 7.6 times as many game states in achi as in the other two games. Second, the search space in achi is somewhat smaller, because there are no states in the game with more than eight markers on the board. Third, achi contests cycle through many of the same states to end in a draw; if the perfect player produces so little variation, there must be few non-losing states elsewhere in the game tree. The lack of useful knowledge necessary to play achi well makes this very clear.[8] These factors together make it

[8]In the sliding stage, a reviewer has pointed out that one can draw merely by not offering the other contestant a winning move on her next turn.

more likely that the program encounters a greater fraction of the important states when learning achi than when learning the other games, during ostensibly the same training.

3. The Preliminary Experiments

This section describes how training against a perfect player, self-training, and a range of fallible trainers (e = 10, 20, 30,…, 100) usually fail the game players' Turing test. With three games and 12 different trainers, it summarizes the results for 36 trials of five runs each. Most experiments are run with the behavioral standard n = 10.

Figures 4 through 7 provide a visual summary of the data in this section. Figures 4 and 5 for tic-tac-toe graph the impact of noise level on power and reliability, respectively, while Figures 6 and 7 do the same for lose tic-tac-toe. Graphs displaying reliability and power at achi have been omitted, because the variation in reliability and in power against all but the expert challenger produced by different training was minimal and without statistically significant trend. At achi, Hoyle achieved at least 97% reliability against every challenger with every kind of training, and often achieved 100% against all of them. It regularly achieved maximal power against all the challengers as well. For example, with e = 10, it was 100% reliable against all challengers, and achieved 100% power against the novice and random challengers, and 77% power against the expert challenger.

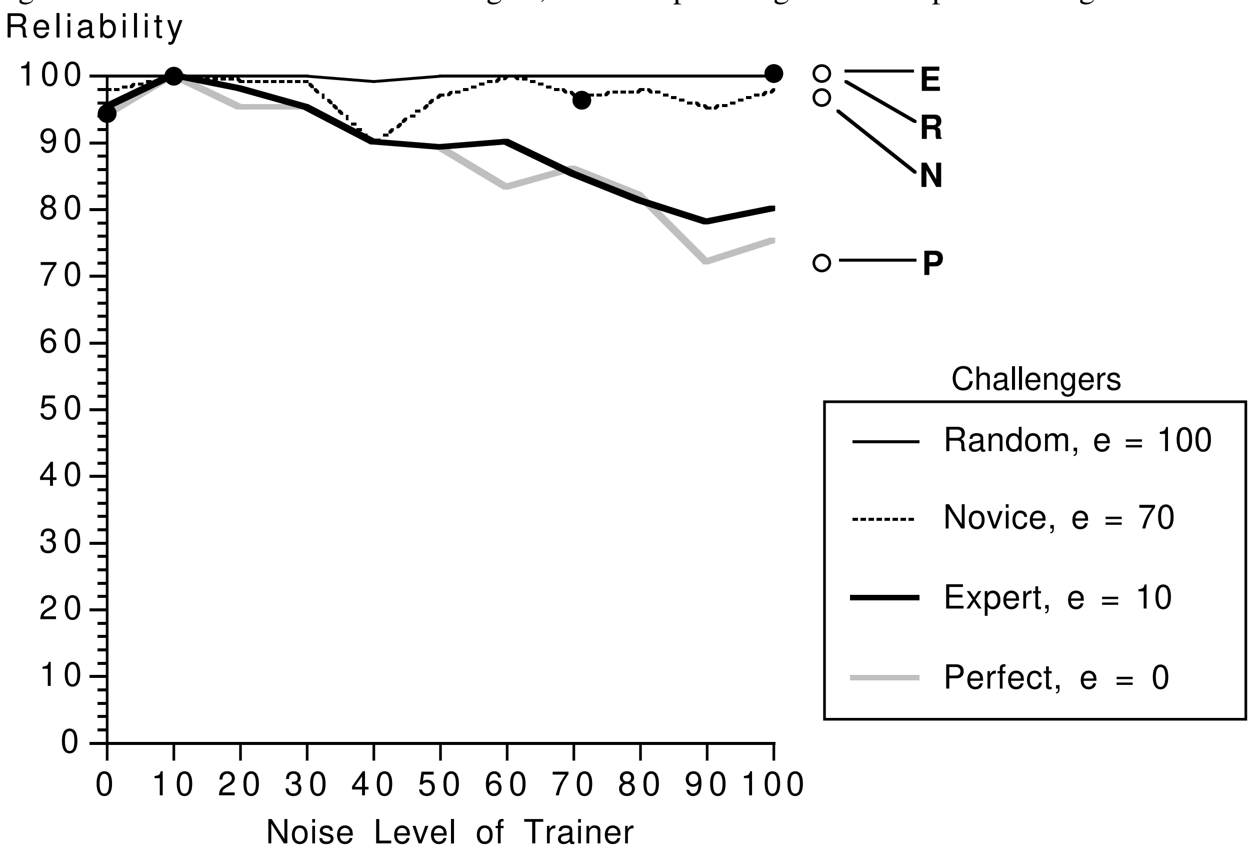


Figure 4: Reliability at tic-tac-toe measured against four different challengers after training. Training experiences have a noise level ranging from 0 to 100%. Points emphasized by black circles measure performance against a challenger exactly like the trainer. White circles labeled with letters denote values for self-training.

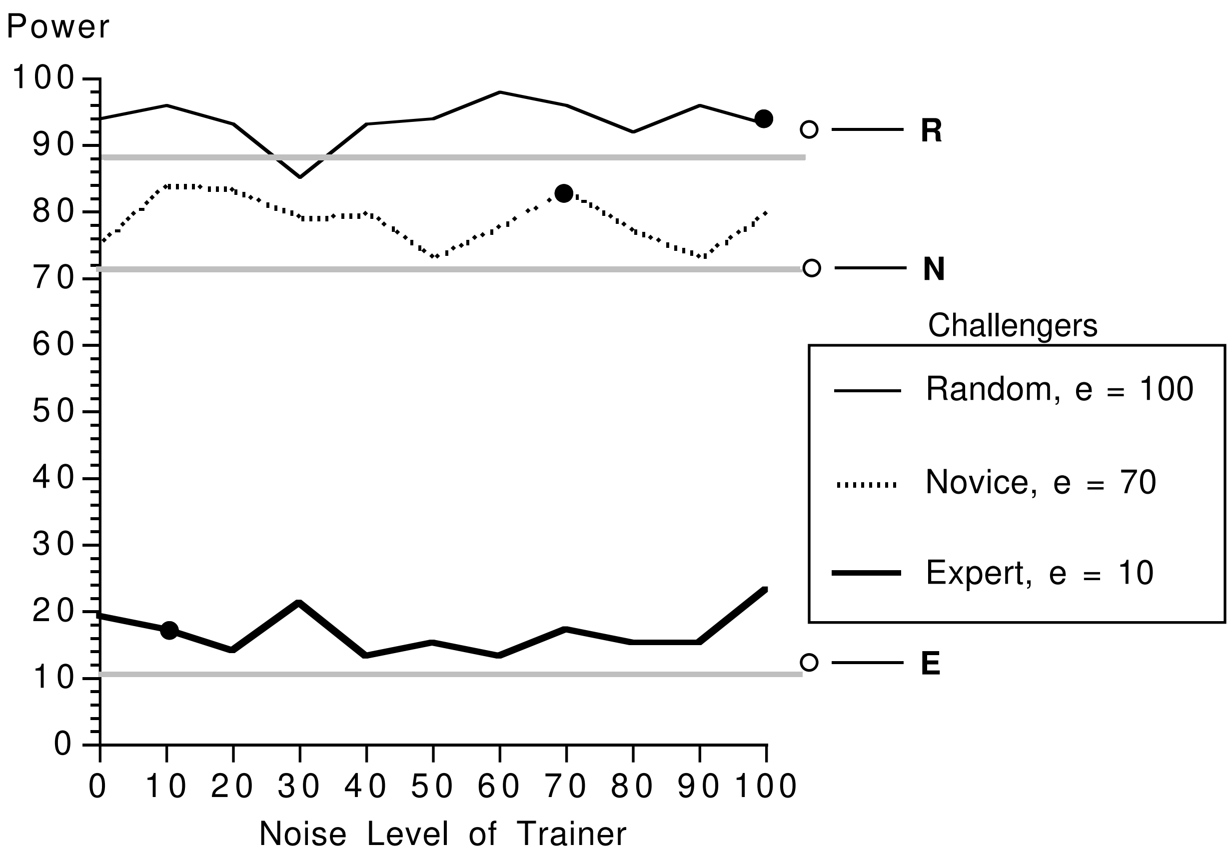


Figure 5: Power at tic-tac-toe measured against three different challengers after training. Training experiences have a noise level ranging from 0 to 100%. Points emphasized by black circles measure performance against a challenger exactly like the trainer. White circles labeled with letters denote values for self-training. Horizontal gray lines indicate the 95% confidence lower bound for maximal power.

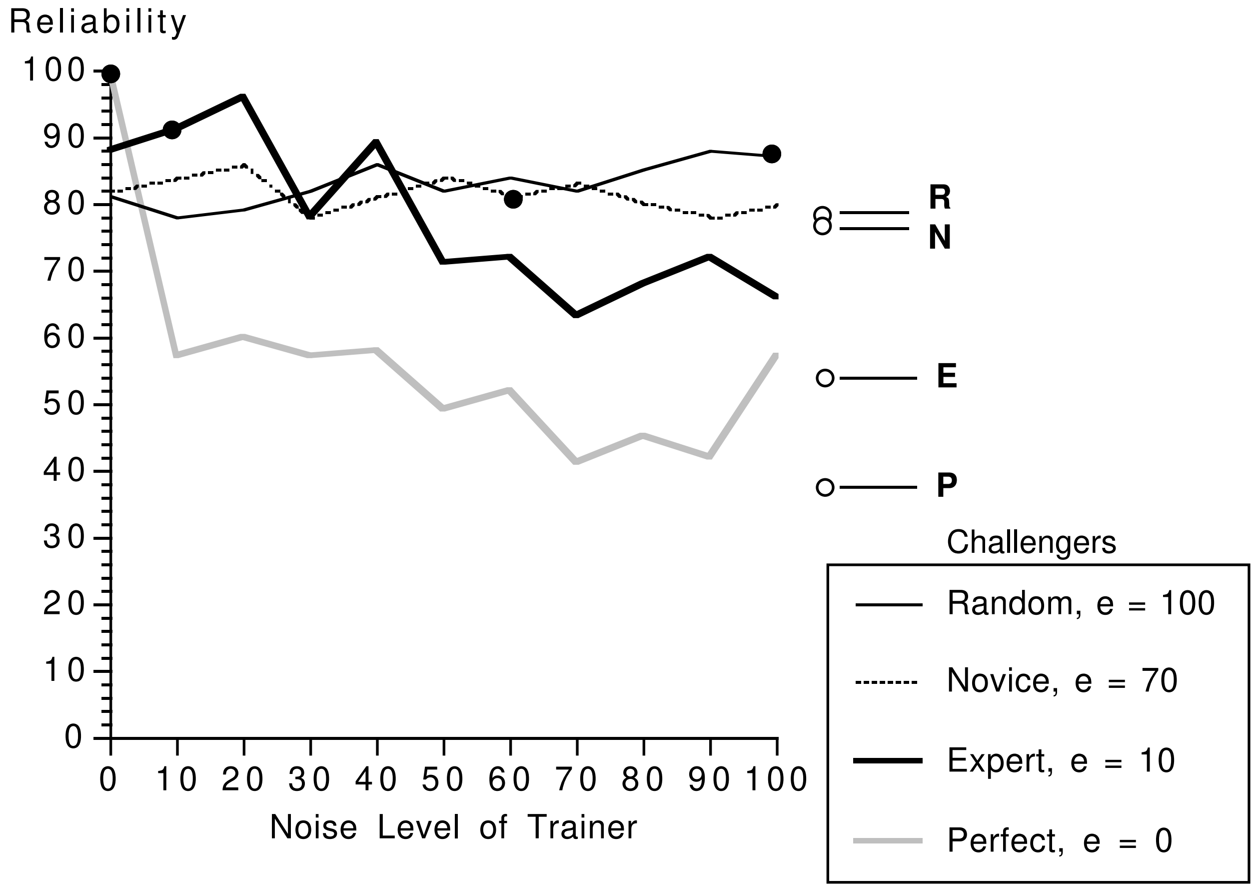


Figure 6: Reliability at lose tic-tac-toe measured against four different challengers after training. Training experiences have a noise level ranging from 0 to 100%. Points emphasized by black circles measure performance against a challenger exactly like the trainer. White circles labeled with letters denote values for self-training.

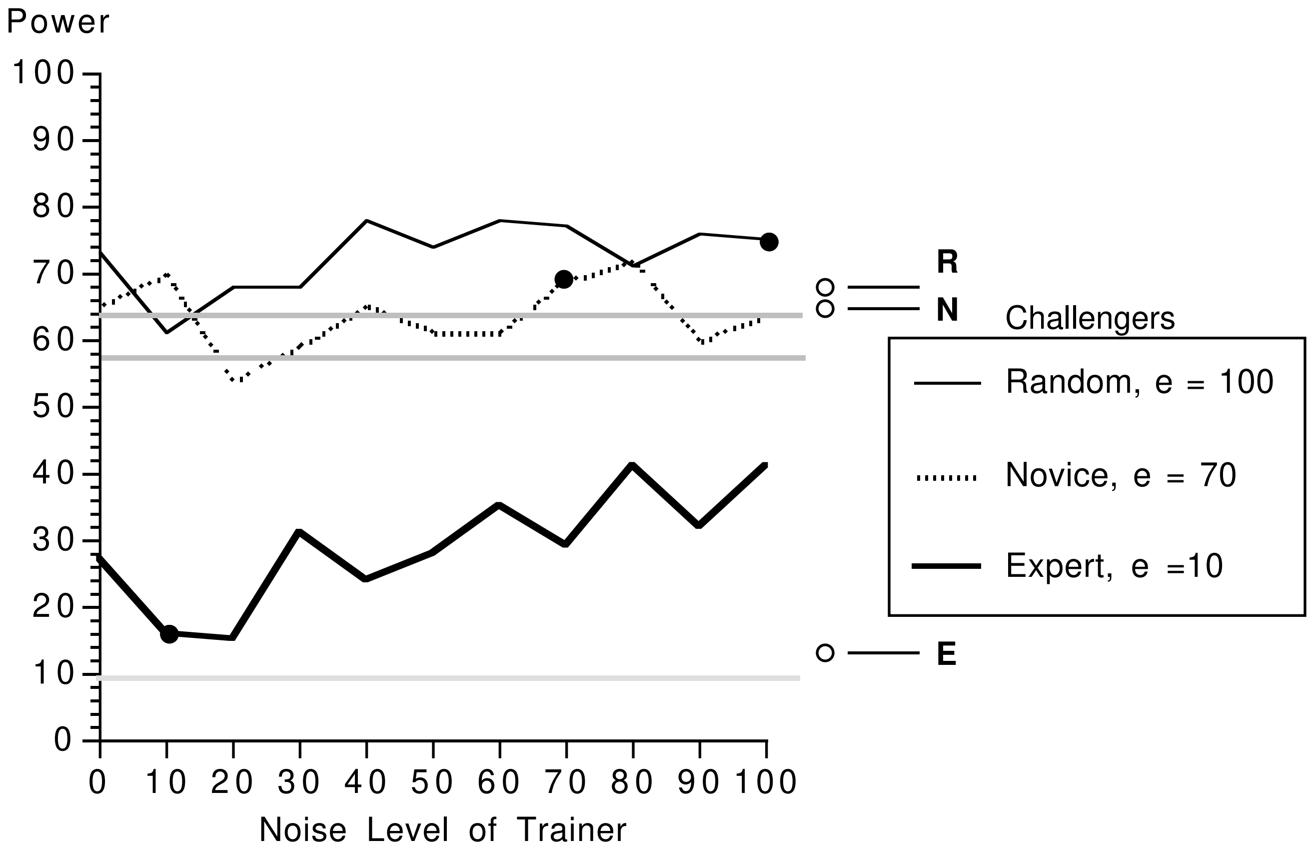


Figure 7: Power at lose tic-tac-toe measured against three different challengers after training. Training experiences have a noise level ranging from 0 to 100%. Points emphasized by black circles measure performance against a challenger exactly like the trainer. White circles labeled with letters denote values for self-training. Horizontal gray lines indicate the 95% confidence lower bound for maximal power.

These graphs are dense with information. Consider the elements of Figure 4 one at a time. There are four curves, one to describe how well Hoyle played during its testing experiences against each of its four challengers. The horizontal axis measures the value of *e*, the noise level of the trainer during the learning experience, and the vertical axis measures reliability during the testing experience. (Recall that maximum reliability is 100%.) Each point emphasized by a black circle measures Hoyle's post-learning performance against a challenger exactly like its trainer. The rightmost black circle, for example, indicates that Hoyle was 100% reliable against the random challenger after it trained with $e = 0$. As you move from left to right along any challenger's curve, you see the impact of a higher noise level during training on post-training performance against that challenger. Reliability after self-training is summarized by four white circles, each labeled with an appropriate letter. For example, after self-training Hoyle is 70% reliable in tic-tac-toe against the perfect (P) challenger.

Figures 5 and 7 are similar to Figures 4 and 6, except that the data describe power against three of the challengers instead of reliability. (Recall that power against a perfect player in a draw game must be 0%, by definition.) The horizontal gray lines indicate the 95% confidence lower bound for maximal power, based on Table 1.

An important idea in the analysis of these graphs is that learning during training is often *incomplete,* that is, that a program can meet the behavioral standard (appear to have learned to draw consistently) without knowing how to play perfectly, or even very well. For example, even after it had drawn in each of the last 10 consecutive contests of lose tic-tac-toe against a perfect player as trainer, Hoyle went on to *lose* 12% of its testing contests against the expert challenger. Hoyle lost these contests because it encountered situations during testing in which it did not know how to make the correct move, i.e., the behavioral standard had been an overly optimistic gauge of the program's skill. Thus learning had been incomplete; training against a perfect player was inadequate preparation for competition against a strong player that made occasional mistakes.

3.1 Training against a perfect player

Training against a perfect player proves an imperfect choice, as indicated by the leftmost point on each curve in Figures 4 through 7, where $e = 0$. (Points are plotted with x values from 0 to 100, at intervals of 10.) Reliability at tic-tac-toe is only 94% against the perfect challenger, worse than after some other training. Reliability at lose tic-tac-toe is the best after training against a perfect player, but fails the game players' Turing test against the other challengers. Even at achi, where many trainers achieved 100% reliability against all the challengers, the perfect player did not. The narrowness of the perfect play skeleton is further demonstrated in Section 5.

3.2 Self-training

During self-training the program plays both sides in every contest. As the data points marked by white circles in Figures 4 through 7 indicate, self-training is a uniformly poor choice. The speed with which self-training meets the behavioral standard, the relatively small amount learned (discussed in Section 5), and the resultant poor testing performance support the theory that self-training resembles the diagram in Figure 1(c), where some paths are thoroughly investigated and repeatedly broadened, but overlap very little with the perfect play skeleton. Self-training in tic-tac-toe was significantly less reliable against the perfect challenger than $e = 10$ training. Self-training at lose tic-tac-toe was significantly less reliable against the perfect and expert challengers, and less powerful against the expert challenger, than $e = 0$ training. Even at achi, the easy game for Hoyle, self-training was inferior.

3.3 Fallible training

The *fallible trainer* introduces noise into the training experience. There is a spectrum of fallible trainers in this experiment with e values in multiples of 10, from 0 to 100. A fallible trainer with $e = 0$ plays perfectly, and has been reported upon above; a fallible trainer with $e = 100$ is a random player. The value of e measures not the frequency of error but frequency of opportunity for error.

The game players' Turing test, with its demands for both reliability and power, proves an elusive standard. For tic-tac-toe, Hoyle passed the game players' Turing test after $e = 10$ training only. It achieved 100% reliability that way and maximal power. For lose tic-tac-toe, however, no training even came close to passing the test. Noise generally decreased reliability against the perfect and expert challengers, and did little to improve power. Reliability did not improve with the fallibility of the challenger, although power often did. For achi, Hoyle passed the game players' Turing test after training with all the fallible trainers except $e = 30$ and 70. (Training with $e = 10$, 20, 80, and 90 even produced 100% reliability.)

Noise both broadens experience and provides a bad example. A higher noise level in the trainer makes Hoyle less reliable at tic-tac-toe and lose tic-tac-toe against the perfect, expert, and novice challengers, as if the trainer's lack of skill taught the program to make certain mistakes. In lose tic-tac-toe, the most difficult of the three games for Hoyle to learn, the noise level is correlated with the number of wins against the expert challenger, i.e., the more fallible the trainer, the more powerful the program. No statistically significant impact of noise on power or reliability can be detected in achi.

3.4 Summary

The intuitions of Figures 1(a), 1(b), and 1(c) are well supported by these preliminary experiments.

When the program learns with a perfect player as its trainer, it repeatedly fails the game players' Turing test, losing to the imperfect challengers while it withstands the competition of the perfect challenger. (Note the values for $e = 0$ on the curves in Figures 4 and 6.) The fallible trainers for $e > 0$ are something of an improvement; they provide experience beyond the perfect play skeleton, but not in the most effective and/or efficient manner. With enough low-level noise the program can learn to pass the game players' Turing test, but only for the two easier games.

The effects of noise on training are several. First, as a model of behavior for a learner that imitates its opposition, noise sets a poor example. One Advisor encourages Hoyle to imitate the trainer. The program, however, learns to disregard such an error if it eventually deduces that the error leads to a significant loss state, or learns to discount the error somewhat if it induces that the error leads to a dangerous state. Second, as a challenge in competition, noise is a highly variable factor. As implemented here, noise is simply a randomly chosen legal move. The move may not be an error, or may be an error from which recovery is possible, i.e., one that does not forfeit the contest. The impact of noise is thus in part a function of the nature of the game tree. There are many possibilities: a game may have many equally good moves and only a single disastrous one from most states, or it may have one excellent move and several mediocre ones, and so on. Game trees with isolated or clustered disastrous possibilities lead to overconfidence in the face of noise; game trees with a very few winning strategies may produce a learner that draws reliably but never learns to win because its noisy trainer never shows the way. In lose tic-tac-toe, noise can be so harmful that it costs the trainer the contest (note, for example, the lowest curve in Figure 7), even though the trainer's errors may be misleading. Finally, as a distraction into unknown parts of the game tree, the impact of noise varies with the skill that the learner has already acquired. From observation in the laboratory, noise early in training appears to be far more likely to cost Hoyle a contest than noise later on, once Hoyle has acquired some useful knowledge about the game.

4. Lesson and Practice Training: The Central Experiments

This section discusses the implementation of lesson and practice training, and its superiority over the trainers in the preliminary experiments with self-training and fallible training. Lesson and practice training, the kind Hoyle normally receives during its discovery mode, is intended to look like Figure 1(d), a knowledge-based, non-random broadening of the perfect play skeleton. Hoyle's training models the way many human novices learn to play a game, as an interleaved sequence of high-quality contests against strong opposition (the *lesson*) with knowledge-based, experience-broadening contests against itself (the *practice*). For some number of lesson contests l and some number of practice contests p, *l&p training* denotes a tournament where Hoyle alternately plays l consecutive contests against a perfect player followed by p consecutive, knowledge-based practice contests where Hoyle plays against itself.

To explain the benefit of Hoyle's practice contests, it is necessary to consider the move selection process in greater detail. Recall that Hoyle chooses a move based upon its Advisors' comments. The Advisors have some heuristic accuracy, if not authentic prescience. The program's architecture partitions them into priority classes called *tiers.* Once a decision is reached, subsequent tiers are not consulted. There are four Advisors in the first tier and three more in the second. Within these first two tiers, the Advisors are consulted in a fixed order. Some have absolute authority to make a decision; others have the authority to eliminate a move from further consideration. For example, if Victory, in the first tier, detects a move that will win the contest, then Panic, in the second tier, will never have the opportunity to recommend a move that will block a potential winning move for the other contestant.

After consultation with these early Advisors, if no decision has been made and there is still more than one move under consideration, all 15 Advisors in the third tier generate their comments. Then a simple arithmetic calculation, called a *move selection paradigm*, chooses a move based on the heuristic advice of these later Advisors. There are four basic move selection paradigms: one tallies the comment strengths for each move and selects the move with the highest score; another effectively smoothes the strengths of the comments to "yes," "no," or "don't care;" another limits each Advisor to its single strongest comment; and another both smoothes and limits.

Given a set of advice, different paradigms may well choose different moves. This variation is due not only to the way the advice is combined but also to the fact that each move selection paradigm breaks a tie among equally attractive alternatives by random selection. When Hoyle plays with different paradigms against the same trainer, it has been observed to try different kinds of moves, as if the paradigm defined a strategy or a style of play. This propensity to variation, while not well-defined, is observable in the laboratory. During practice, Hoyle plays one paradigm against another.

Thus *the practice in lesson and practice training is not pure self-training, but a thoughtful, knowledge-based exploration of the search space.* Under every move selection paradigm, the skill and commonsense embodied by the rationales behind the first seven Advisors, including vetoes of particularly disastrous choices, is always brought to bear before the other Advisors have the opportunity to comment. There are four sources of knowledge that drive exploration during practice: the early Advisors that prevent serious errors and oversights, the later Advisors with their heuristic wisdom, the model of correctness epitomized by the perfect player and often imitated by Hoyle, and the useful knowledge that some of the Advisors reference.

In lesson and practice training, l should be a small even number, small to treat lessons as a scarce resource, and even to guarantee that the learner and the lesson giver move first equally often. In all three games, Hoyle was tested with 2&7-training, i.e., cycles of two contests against the perfect player followed by seven contests against itself. To keep l small with respect to both p and to the behavioral standard $n = 10$, l was set to 2. To force more than one lesson and practice cycle before $n = 10$ could terminate the learning phase, and to make p relatively large with respect to l (thereby treating lessons as a scarce resource), p was set to 7. Before each practice contest, Hoyle chose two of four input paradigms at random with replacement, i.e., Hoyle practiced against itself using a fixed paradigm for the version that moved first and a second, possibly identical, fixed paradigm for the version that moved second. Varying the move selection paradigm produces a broader variety of reasonable moves than would be experienced by a single move selection paradigm training against a perfect player, or even by a variety of paradigms training against a perfect player, but moves that are still influenced by knowledge extracted from lessons with the perfect player.

Table 2: Results for learning lose tic-tac-toe with 2&7 lesson and practice training, compared with training against a perfect player for n = 10 and 20. Space is the additional memory allocated for acquired knowledge. The two most important lines to compare appear in boldface.

	Contests			Power			Reliability			
	Played	Lessons	Space	Expert	Novice	Random	Perfect	Expert	Novice	Random
2&7 training										
n = 10	122.7	27	323.0	17	63	77	95	92	88	90
***n* = 20**	**127.7**	**28**	**299.0**	**18**	**63**	**85**	**100**	**98**	**97**	**100**
Perfect player										
***n* = 10**	**35.6**	**35.6**	**89.2**	**27**	**65**	**73**	**100**	**88**	**82**	**81**
n = 20	49.3	49.3	101.0	12	59	80	100	93	80	88

With lesson and practice training Hoyle passes the game players' Turing test for lose tic-tac-toe, the game of the three that it had found hardest to learn with the other trainers. Compared to the data of the preliminary experiments, there are noteworthy and statistically significant improvements with 2&7 lesson and practice training for $n = 20$. Table 2 summarizes the results for $n = 10$ and 20. (Contests, lessons, and space are discussed further in Section 5.) Among the lesson and practice runs was one perfectly reliable run for $n = 20$, a marked improvement over the best results in the preliminary experiments.

The reader is cautioned not to misread the data of Table 2. Training against a perfect player for lose tic-tac-toe with $n = 20$ certainly results in stronger play than with $n = 10$, but the program still fails the game players' Turing test with such training. Performance does not simply improve, or continue to improve, with increased learning time. (Indeed, additional work referenced in Section 5 indicates that this is not the case.) Also distinguish carefully here between learning time (number of contests required to meet the behavioral standard) and the number of lessons that occur during that learning time (all the contests against a perfect player but only 2/9 of those in 2&7 training).

The most important comparisons are between the second and third lines of Table 2 (in boldface), since 2&7 training with $n = 20$ averages 28 lessons, somewhat less than the average exposure to the perfect player (35.6 contests) during $e = 0$ training with $n = 10$. *With lesson and practice training Hoyle develops much greater reliability against the fallible challengers and much greater power against the random challenger.* Compared with self-training and with the full spectrum of fallible trainers ($e = 0, 10, 20, \ldots, 100$), lesson and practice training for lose tic-tac-toe is consistently:

- more reliable against the perfect challenger than all training except that against a perfect player, and less reliable than no other training
- more reliable against the expert challenger than all but two and less reliable than no other training
- more reliable against the novice challenger and the random challenger than all other training
- more powerful against the expert challenger than five others, and less powerful than no other training
- more powerful against the novice challenger than eight others, and less powerful than no other training
- within maximal power against all the challengers[9]

Table 3: A comparison of 2&7 lesson and practice training for tic-tac-toe with other training. The most important lines to compare appear in boldface. Space is the additional memory allocated for acquired knowledge.

	Contests			Power			Reliability			
	Played	Lessons	Space	Expert	Novice	Random	Perfect	Expert	Novice	Random
Fallible Training $e = 0$										
***n*=10**	**12.0**	**12.0**	**7.4**	**19**	**75**	**94**	**94**	**95**	**98**	**100**
n= 20	22.9	22.9	9.0	15	84	94	100	100	100	100
Fallible Training $e = 10$										
n= 10	12.6	12.6	13.6	17	84	96	100	100	100	100
Self-Training										
n= 10	10.0	0.0	1.0	14	72	93	70	100	98	100
2&7 Lesson and Practice Training										
n= 10	11.6	2.6	5.2	25	83	93	82	91	97	100
***n*=20**	**50.0**	**11.1**	**22.6**	**18**	**84**	**97**	**94**	**97**	**100**	**100**

Lesson and practice training also held up in the simpler games. With lesson and practice training Hoyle passed the game players' Turing test for achi. On every lesson and practice run for achi the program achieved perfect reliability, never lost to the novice or random challenger, and achieved maximal power.[10] After lesson and practice training the program also passed the game players' Turing test for tic-tac-toe. Table 3 compares 2&7 lesson and practice training with $n = 10$ and 20 against the best of the preliminary trials ($e = 10$), and against the $e = 0$ training and self-training

[9] The apparent decline in power against the expert and novice challengers is not statistically significant.

[10] Lesson and practice training was as effective as any except $e = 20$, which was slightly more powerful against the expert challenger.

lesson and practice training purports to blend. Once again, the appropriate comparison for $e = 0$, $n = 10$ training is 2&7 training with $n = 20$, which includes 11 lessons, slightly less than the average exposure to the perfect player. 2&7 training clearly produces results superior to self-training and on a par with $e = 0$, $n = 10$ training. After about four times as many contests, 2&7 training stores about three times as much data. This is an indication that, as in Figure 1(d), the perfect play skeleton is indeed being broadened by lesson and practice training. The less than perfect reliability of 2&7 training against the perfect and expert challengers is due to the stochastic nature of the perfect trainer. As long as Hoyle was presented with the fork of Figure 3 during lesson and practice training, it always achieved perfect reliability against every challenger with no statistically significant decrease in power over the best of the preliminary training. The paucity of lessons, however, occasionally gave Hoyle insufficient exposure to the perfect player, so that it learned incompletely and lost several contests to the perfect and expert challengers.

5. Probing the Experimental Design

Once begun, training experiments such as these spawn many alternatives. This section briefly summarizes some of them; the thoughtful reader by now has likely conjectured many others. Each paragraph is introduced by the question it addresses. Full data and comparative figures for the work summarized here are available in (Epstein, 1993).

Does more training (i.e., a higher value of n than 10) result in better play? Although a higher value of the behavioral standard n forces Hoyle to play more contests and to perform expertly over a longer continuous period, it has little impact on the adequacy of a trainer. There is no statistically significant change in power or reliability for 2&7 lesson and practice training after $n = 20$. More training against a perfect player does not improve power or reliability with $n = 20$, 50, 75, or 100. More self-training improves only reliability against the perfect and expert challengers, and those figures remain substandard for the game players' Turing test. Additional fallible training does result in stronger play only against the imperfect challengers, but the resultant improvement is usually a singular one and rarely suffices for the game players' Turing test. A higher behavioral standard rarely improves power, and power against the expert player only shows significant *decreases* with a higher n for $e \geq 50$, as if more practice in an error-ridden environment makes Hoyle play more conservatively. Even with larger values of n, Hoyle continues to lose to its lesser challengers and to fail the game players' Turing test for lose tic-tac-toe.

How does the nature of the trainer affect the time required to learn to play expertly? Mid-range noise (40% to 70%) prolongs learning time the most, roughly doubling it. Too little noise in the trainer produces a learning experience that is relatively fast because it is so narrow; too much noise produces a learning experience that is relatively fast because it is deceptively weak against the somewhat knowledgeable Advisors.

How does the nature of the trainer affect the amount of permanent memory allocated for useful knowledge? It varies with the game. In Tables 2 and 3, "space" measures the amount of useful knowledge accumulated during each trial. Useful knowledge is heuristically acquired, so that noise in the trainer may result in the retention of irrelevant data. Lesson and practice training typically produces larger quantities of useful knowledge than training against a perfect player. In both varieties of tic-tac-toe, lesson and practice training stores about three times as much useful knowledge as is stored after roughly the same number of lessons (not contests) against a perfect player. In achi, lesson and practice training retains far more useful knowledge than when training against a perfect player (where only the single basic unit is acquired), but between 58% and 75% less than the useful knowledge collected when training against any of the fallible trainers. In lose tic-tac-toe and achi, the longer the program spends learning, the more useful knowledge it acquires.[11]

Is there a different impact when training error is due to lack of foresight or lack of knowledge,

---

[11]For achi, this additional useful knowledge from fallible training is accurate, sophisticated, and in no way improves the program's already excellent performance, i.e., it is irrelevant.

rather than to random selection? No. A trainer whose errors are attributable to misconceptions or lack of depth, rather than random noise, is no better than a fallible trainer. This was explored with a series of *informed trainers* that apply an input, game-dependent evaluation function in an alpha-beta search to a fixed depth (Nilsson, 1980). (Details are again available in (Epstein, 1993).) With a depth of d, the informed trainer uses the evaluation function to examine all the relevant nodes d moves after the current state, and minimaxes the result back up to the current state to select its next move. When an informed trainer makes a mistake, it is because the evaluation function is only an approximation of the knowledge inherent in the game tree, not because the trainer has made a randomly chosen move. Learned behavior after informed training appears strongly related to both the construction of the evaluation function and to the game. The evaluation functions for an informed trainer tested here are completely accurate only at the end of a contest; elsewhere they are deliberately reasonable but imperfect. In all three games, the nature of the informed trainer makes the learning experience somewhat deceptive because crucial experience (like the fork in Figure 3 for tic-tac-toe and achi and the single correct opening for lose tic-tac-toe) is less likely to arise. For example, all informed training at lose tic-tac-toe produces a program that is less reliable against the perfect challenger than training against a perfect player.

Are the intuitions of Figure 1 supported by these experiments? Yes. Intuitions from Figure 1 about training against a perfect player and self-training are confirmed when the behavioral standard is increased. The narrowness of the perfect play skeleton and the repetitive nature of self-training are confirmed by the failure of useful knowledge to increase with n during such training.

Why not just train for some fixed number of contests and then compare performance, without a behavioral standard? The behavioral standard is a more reliable termination condition for learning than training for a fixed number of contests. Whereas a behavioral standard terminates learning based on consistent ability to perform, fixed learning time trials effectively turn the program loose in the space for some predetermined amount of exploration and then stop learning regardless of the program's performance thus far. For lose tic-tac-toe, the only game with a broad variation in learning time, trials were run in which the learning experience lasted exactly 64 contests, the average learning time observed across all the trials in Section 3. Trials of three 64-contest runs were performed for self-training and for fallible training with $e = 0$, 10, 20, 50, 70, and 100. Unlike training with the behavioral standard, fixed learning time training produced submaximal power in several instances, even though 64 was on average *more* contests than that training had originally run. In many cases, reliability after fixed learning time was significantly worse than with the behavioral standard, particularly against the weaker challengers. There were some improvements in reliability against the perfect and expert challengers; they were directly attributable to the fact that the runs were longer, with greater, randomized opportunity to encounter additional important nodes in the search space. (An ideal trainer, recall, would introduce those nodes deliberately.)

Does a trainer set an upper bound on the learning program's skill? No. Hoyle does not always play best against its trainer (as opposed to the other challengers), and can actually learn to outplay its trainer or to outperform its trainer against a specified challenger. A program may be said to play as well as its trainer when it is 100% reliable against the trainer, and reaches power and reliability statistically equivalent to what the trainer would achieve against the same challengers within a 95% confidence level. For tic-tac-toe, with $e = 10$ training Hoyle learns to play more reliably against the expert and the novice challengers, as reliably against the random challenger, and more powerfully against all the challengers than its own trainer would. For lose tic-tac-toe, with $e = 10$ training Hoyle learns to play more reliably against the expert challenger and more powerfully against the novice and random challengers, but less reliably against the novice and random challengers than its own trainer would. For achi, Hoyle learns to outperform its imperfect trainers in all these measures. With $e = 70$ training and with $e = 100$ training Hoyle substantially outperforms its trainer in both power and reliability at all three games.

There are many other training experiments worthy of investigation.[12] One might use a trainer that

[12]Several of the following were suggested by the reviewers.

always makes the worst possible move, or one that was designed to expose as much of the search space as possible. One might design an "active" learner that deliberately chose an ill-understood (neither clearly good nor clearly bad) move, to observe the trainer's reaction. (Such a deliberately digressing learner would take longer to meet a behavioral standard, and might require a different metric.) One might design a cyclic experiment where training and testing tournaments of fixed length were interspersed until some new performance standard was met. One might train against varying levels of skilled opposition, perhaps increasing the challenge as the learner developed reliability. One might explore values for *l&p* training other than 2&7, say 14&1, or 4&100, or let the ratio of *l* to *p* decrease as the learner knows more, forcing additional unsupervised practice time as expertise improves. To gauge learning, one might compare the program's performance on states that it has seen during training with those it has not. If the reader has postulated these or other variations by now, one intention of this work has been justified: to direct careful attention to the training environment.

6. Related Work

Traditional AI game playing programs, like Deep Thought and HiTech, use fast, deep search to identify relevant future positions and evaluate their strength (Anantharaman, Campbell, & Hsu, 1990; Berliner & Ebeling, 1989). These programs rely on special-purpose hardware, clever storage and retrieval tactics, a few well-known search heuristics, and raw computing power to search deeply and quickly. There is a growing consensus in the AI game-playing community, however, that a game like Go will never be played as well as chess is with such techniques, because Go's search space is so much larger than that of chess, and because Go offers so many more possibilities at each choice point.

As a result, there has been substantial recent interest in programs that *learn* to play games. Among the best known of these newer artifacts are TD-gammon, Morph, and Hoyle. TD-gammon learns to play backgammon with a neural net that, after much practice, holds its own against a world master (Tesauro, 1992). Morph learns to play chess with a pattern cache that is gradually improving against a strong commercial chess program (Levinson, et al., 1991). Hoyle learns to play many simpler, two-person, perfect information, finite board games extremely well against a variety of experts (Epstein, 1992). Although Hoyle's learning strategies and rationales appear simplistic, they are far more effective than rote learning. Experiments with MENACE, a matchbox tic-tac-toe machine, trained a rote learner against a human expert player (Michie, 1986). MENACE represented each of 300-odd states as a matchbox whose contents represented the learned move to be made in that state. MENACE learned these expert moves in about 150 contests to simulate perfect play. After an average of 12.6 contests of $e = 10$ training, Hoyle simulates perfect play at the same game while retaining only 13.6 expert moves.

The best known game-learning programs are those that are competitive with human champions in popular games. Samuel's Checker Player was an early effort that learned an evaluation function based on input features of the checkerboard [Samuel, 1963; Samuel, 1967]. More recently, two neural net programs have learned to play backgammon extremely well, Neurogammon and TD-gammon (Tesauro, 1992; Tesauro & Sejnowski, 1989). Neurogammon was trained to select the moves made in 15,000 game states. Some of these states were drawn from textbooks on how to play backgammon; most were drawn from 400 contests where Tesauro, a strong but not world-class player, had played both sides. At the First Computer Olympiad in London in 1989, Neurogammon was clearly the strongest non-human competitor. When Neurogammon played TD-gammon, however, it lost 60% of the time. There are several explanations for TD-gammon's greater strength: TD-gammon had much more extensive training, on approximately 200,000 contests; TD-gammon used the TD(λ) algorithm instead of Neurogammon's standard back-propagation; and TD-gammon trained against itself (Rumelhart, Hinton, & Williams, 1986; Sutton, 1988).

Tesauro actually trained TD-gammon several times, but only wrote about the best performance, the one due to self-training (Tesauro, 1991). After training against Neurogammon as an external expert, TD-gammon again played slightly worse than after self-training. *Priming* is another kind of training, where the learner first observes two excellent players in a tournament against each other before competing itself. After priming on Neurogammon's 15,000 game states, TD-gammon played slightly worse than after self-training. Play after priming and after external expert training showed no significant difference in quality, although, during both learning periods, the program was able to progress away from random moves much more quickly than it had when it learned against itself (Tesauro, 1991). Self-training was a substantially slower way than priming to acquire broad expertise; TD-gammon spent the first 25% of its training playing "long, looping contests that seemed to go nowhere" (Tesauro, 1991).

Tesauro attributes TD-gammon's ability to learn to play so well in part to the variety of training situations that were forced upon it by the non-determinism of the dice during learning; Gelfand found $e > 0$ essential (Flax, Gelfand, Lane, & Handelman, 1992; Tesauro, 1992). The results described here, in conjunction with theirs, argue that the conclusion about the need for variety in training is independent of the particular learning strategies used.

Experiments with a simple pattern-learner and reinforcement training for several games on a three-by-three grid have indicated that priming may *slow* learning (Painter, 1993). One possible explanation for this is that the initial bias so developed is irrelevant, or even wrong, for many of the game states that the novice learning program soon faces. The head start must thus be partially unlearned before useful learning can take place.[13] N-N/Tree, a hybrid learning program with a neural net, was also found to suffer from priming (Flax, et al., 1992). For these reasons, priming was not addressed here.

Recent work on reinforcement learning has addressed the impact of infrequent guidance by an instructor in non-adversarial domains. Lin's instructor gave a simulated learning robot an example of a perfect solution (Lin, 1991). Clouse and Utgoff showed that controllers for two simulated physical domains with large search spaces are also learned more quickly when a trainer intervened with the correct action (Clouse & Utgoff, 1992). They used a behavioral standard of 10,000 correct actions for cart-pole and the equivalent of 1&1485 lesson and practice training on the ACE/ASE algorithm. For their race track domain, the behavioral standard was five successful driving demonstrations with the equivalent of 1&12 lesson and practice training. Their improvement over standard reinforcement learning was more dramatic than those reported here because the original learner had no domain knowledge and required extensive training.

Finally, this research differs from prior work by educators and psychologists because it is able to start each learning experience with a machine that offers a tabula rasa, a clean slate. Because people cannot clear their minds of all prior experience, and because they may learn differently, the results may not be analogous to people. Hoyle's learning speed and output results, however, do simulate an experienced game player encountering an unfamiliar game (Epstein, 1992).

7. Conclusions

The nature of a game-playing trainer has until now been a matter of convenience. Input book games require the tedious assembly of databases. People of any caliber play too slowly, tire too quickly, and may be fairly rigid in their approach. That leaves only opposition from a machine.

Self-training in these experiments proves to be a poor choice: unreliable and not particularly powerful. Self-training with an effective learning algorithm does not prepare the program for strong opposition because it deprives the program of an important knowledge source that people learn from, the expert model. We argue that it is difficult enough to learn at the important nodes in the search space, without having to discover those important nodes as well. For more difficult games without inherent variation and without a perfect player, this research offers no reason to believe that self-training will ever result in expert play.

If the trainer always plays perfectly, the learner may not have the skill to deal with errors, or even with suboptimal moves, let alone exploit them to its advantage. The data presented here

[13]Painter also found that the random trainer produced a less reliable player.

confirm that such training offers no experience with large portions of the problem space.[14] If the learner is to function robustly in a large and dynamic environment, there should be some breadth involved in its training, beyond the places that the best of all possible decisions might lead.

When a learner cannot visit every node in its domain's search space, it ought to visit those nodes from which it can learn the most. The function of a trainer is to drive the exploration of portions of a very large search space in preparation for performance anywhere in it. The popularity of puzzle problems constructed for difficult games shows that examples of knowledge-rich nodes are intrinsically interesting to people who solve problems in the search space. Such a puzzle is usually designed around a subtle error made by one participant. Even though it may entail an oversight, this error is rarely a random decision; it reflects some conscious rationale on the part of the mover. Good training incorporates this kind of error.

For games without an element of chance, variety in the opposition can broaden training, but not well enough to simulate human expertise. A fallible trainer with $e > 0$ introduces some variety into the paths traced by the contests through the game tree. For Hoyle, however, training against weak opposition is inadequate preparation for a stronger opponent, while training against strong opposition also turns out to be less adequate for a weaker opponent.

Lesson-and-practice training is shown in these experiments to be a particularly effective way to introduce high-quality variety. It alternates a few contests of strong opposition from a perfect player (the lesson) with many contests of weak opposition from the developing learning program itself (the practice). Under this regimen, the expert is treated as a scarce resource, and Hoyle learns to play more reliably and more powerfully, albeit with concomitant demands on learning time and memory. Unlike fallible training and self-training, *lesson and practice training appears to elaborate on the key portions of the game tree: those that the expert would highlight and those that reflect incomplete understanding of the task at hand.* This was achieved with a relatively low ($n = 20$) behavioral standard. To date Hoyle has learned to be extremely reliable and very powerful against all the challengers in 15 additional games, most of which are far more difficult than those described here. Although there have not been careful comparative studies of different trainers for these other games, lesson and practice training has become the environment of choice for this work.

Lesson and practice training for more difficult games requires three modifications on the implementation described here. First, for games like chess or Go the lesson giver must be approximated by a knowledgeable, imperfect player intended to guide the learner to knowledge-rich portions of the search space. Assuming that people find interesting exactly those situations that offer an opportunity to learn, such a trainer requires a measure of interestingness still best defined by people. If, therefore, one attempts to apply these results to more difficult games, a human should administer the lessons, preferably from a historical database of important contests or cases. Second, the deliberate variation required during practice should have a knowledge-based, strategic or tactical rationale behind it. A learning program like Hoyle, that makes the reasons for its decisions explicit and can determine ways to vary them without undermining its fundamentally correct principles, has some advantage here. Finally, a good trainer should not leave the introduction of essential material, like the fork in Figure 3, to chance; the trainer should make certain that the learner encounters it.

Much of this work validates the intuitions of Figure 1. The failure of useful knowledge to grow after longer training against a perfect player confirms the existence of a perfect play skeleton like that of Figure 1(a). The failure of fallible training to meet the game players' Turing test confirms the ineffectuality of the randomized additions in Figure 1(b).The failure of useful knowledge to grow after more self-training confirms the repetitive nature of Figure 1(c), while the particularly poor reliability and power after self-training confirm the lack of overlap between Figure 1(c) and the perfect play skeleton. The power and reliability achieved after lesson and practice training

---

[14]The "no experience" claim is based on the lack of growth of useful knowledge as the behavioral standard is extended. It might be possible to learn to play perfectly from very limited experience if the learner were able to draw on a vast store of knowledge, perhaps by analogy or further search for which the training was a catalyst.

confirm the non-random extension of the perfect play skeleton in Figure 1(d).

Additional experiments demonstrate that the classic "corrections" suggested for traditional training techniques do not match the performance achieved with lesson and practice training. Longer training, via a higher behavioral standard, improves Hoyle's reliability but not its power, while it substantially increases the program's learning time and memory requirements. Evaluation-function-based error in an alpha-beta player of fixed depth is no more instructive than random error.[15] Training for a fixed number of contests masks the appropriate moment to stop learning. Mid-range noise levels induce greater memory requirements and protract learning time without providing greater power. Only lesson and practice training keeps reliability and power directly proportional to the fallibility of the challenger.

The lessons of this paper have implications for learning in any search space so large that it may never all be seen during or after learning:

- Training drives the exploration of portions of a very large search space in preparation for performance anywhere in it.
- Fast training does not necessarily lead to good learning.
- Experience can be an effective indicator of when to terminate training.
- Some correct knowledge is irrelevant to good performance.
- Self-training is so narrow that it may never access important knowledge.
- Some key knowledge is best taught by an expert, rather than rediscovered, but thoughtful variation is essential to successful training.
- Any kind of error during training develops a stronger expert, one that behaves as if it had more extensive knowledge of the search space.

Lesson and practice training incorporates all these principles in its design, to support the discovery of key knowledge while carefully managing scarce resources.

Once game-learning programs surpass people, they can continue to train against each other. To the extent that such programs develop different styles of play, competition among them should strengthen their abilities. In so imperfect an environment, however, there can be little guarantee that they will ever play perfectly, or know everything, about the game at which they display expertise.

Acknowledgments

The author thanks Jack Gelfand, Cullen Schaffer, and Gerry Tesauro for insightful discussions. The thoughtful and detailed comments of three anonymous referees and the diligence and care of the editor greatly improved the final product. Kouros Esfahany and Joanna Lesniak provided expert-level data generation and programming support.

References

Berliner, H. & Ebeling, C. (1989). Pattern Knowledge and Search: The SUPREM Architecture. *Artificial Intelligence*, *38*(2), 161-198.

Clouse, J. A. & Utgoff, P. E. (1992). A Teaching Method for Reinforcement Learning. In *Proceedings of the Ninth International Conference on Machine Learning*. Ed. D. Sleeman, & P. Edwards. Morgan Kaufmann, 92-101.

Cohen, D. I. A. (1972). The Solution of a Simple Game. *Mathematics Magazine*, *45*(4), 213-216.

Conover, W. J. (1980). *Practical Non-Parametric Statistics, second edition* . New York, NY: John Wiley and Sons.

Epstein, S. L. (1990). Learning Plans for Competitive Domains. In *Proceedings of the Seventh*

[15] One might expect, however, that in some games with a larger branch factor, the probability of making a suboptimal choice, rather than a terrible one, would decrease, so that lack of foresight and knowledge in a trainer would be less damaging to learning than random noise would be. The potential tradeoff between partial knowledge and the ways it might lead the learner astray seems worth some additional exploration.

International Conference on Machine Learning. Ed. B. W. Porter, & R. J. Mooney. Morgan Kaufmann, 190-197.

Epstein, S. L. (1992). Prior Knowledge Strengthens Learning to Control Search in Weak Theory Domains. *International Journal of Intelligent Systems*, *7*, 547-586.

Epstein, S. L. (1993). *The Hoyle Training Experiments* (CS-TR-93-01). Department of Computer Science, Hunter College of The City University of New York.

Epstein, S. L. (To appear). For the Right Reasons: The FORR Architecture for Learning in a Skill Domain. *Cognitive Science*.

Ericsson, K. A. & Smith, J. (1991). Prospects and Limits of the Empirical Study of Expertise: An Introduction. In K. A. Ericsson, & J. Smith (Ed.), *Toward a General Theory of Expertise - Prospects and Limits* (pp. 1-38). Cambridge: Cambridge University Press.

Flax, M. G., Gelfand, J. J., Lane, S. H. & Handelman, D. A. (1992). Integrating Neural Network and Tree Search Approaches to Produce an Auto-Supervised System that Learns to Play Games. In *Proceedings of the 1992 International Joint Conference on Neural Networks*.

Levinson, R. & Snyder, R. (1991). Adaptive Pattern-Oriented Chess. In *Proceedings of the Eighth International Machine Learning Workshop*. Morgan Kaufmann, 85-89.

Lin, L.-J. (1991). Programming Robots Using Reinforcement Learning and Teaching. In *Proceedings of the Ninth National Conference on Artificial Intelligence*. MIT Press, 781-786.

Michie, D. (1986). Trial and Error. In *On Machine Intelligence* (pp. 11-23). Ellis Horwood Ltd.

Nilsson, N. J. (1980). *Principles of Artificial Intelligence* . Palo Alto: Tioga Publishing.

Painter, J. (1993). *Pattern Recognition for Decision Making in a Competitive Environment*. Master's, Hunter College of the City University of New York.

Rumelhart, D. E., Hinton, G. E. & Williams, R. J. (1986). Learning Internal Representation by Error Propagation. In D. E. Rumelhart, & J. McClelland (Ed.), *Parallel Distributed Processing, Vol. 1* Cambridge, MA: MIT Press.

Sutton, R. S. (1988). Learning to Predict by the Methods of Temporal Differences. *Machine Learning*, *3*, 9-44.

Tesauro, G. (1991). Personal communication.

Tesauro, G. (1992). Practical Issues in Temporal Difference Learning. *Machine Learning*, *8*(3/4), 257-277.

Tesauro, G. & Sejnowski, T. J. (1989). A Parallel Network that Learns to Play Backgammon. *Artificial Intelligence*, *39*(3), 357 - 390.

Turing, A. (1963). Computing Machinery and Intelligence. In E. A. Feigenbaum (Ed.), *Computers and Thought* New York: McGraw-Hill.