# Learning Expertise from the Opposition*

**Susan L. Epstein**
Department of Computer Science
Hunter College and The Graduate School of The City University of New York
695 Park Avenue, New York, NY 10021
USA

## Abstract

For very difficult games, like Go, it is increasingly clear that the most competitive programs will be those whose expertise is developed through learning during competition. This paper explores how the nature of the opposition during training affects the quality of learned behavior in 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. Variations in the playing skill learned from many kinds of opposition are described here for three different games. The results argue for a broad variety of training experience with play at many levels. This variety can either be driven by inherent elements of chance in the game or be introduced deliberately into the training. A case is made for extensive, thoughtful training of systems that learn, and for cautious reliance upon them.

## 1. Introduction

Educators promulgate many philosophies about what makes a good learning situation for humans, but it is difficult to compare how the same individual learns the same skill in more than one environment; prior learning experiences are to some extent ineradicable. With a computer program, however, it is possible to learn from the beginning as often as one likes, to compare and contrast learning environments in a variety of situations, and to test the resultant skill extensively without permitting further learning. In particular, with machines instead of people, one can ask how the nature of the opposition determines what, and how quickly, a game player learns.

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 than traditionally anticipated. The contribution of this paper is its analysis of the impact of the training environment on learning to play games. It includes the formulation o appropriate performance metrics and recommendations fo training a program to play games.

*This work was supported in part by NSF 9001936 and PSC-CUNY 668287.

## 2. Competitive Learning and Expertise

Traditional AI game playing programs, like Deep Though and HiTech, use fast, deep search to identify relevant futur positions and evaluate their strength [Anantharaman *et al.* 1990; Berliner & Ebeling, 1989]. These programs rely o special-purpose hardware, clever storage and retrieval tactic a few well-known search heuristics, and raw computin power to search deeply and quickly. There is a growin consensus in the AI game-playing community, howeve that a game like Go cannot be played as well as chess i with such techniques, because Go's search space is so muc larger than that of chess, and because Go offers so man more possibilities at each choice point.

As a result, there has been substantial recent interest i programs that learn to play games. Samuel's Checker Playe was an early effort that learned an evaluation function base on input features of the checkers board [Samuel, 196 1967]. TD-gammon learns to play backgammon with neural net that, after much practice, holds its own against world master [Tesauro, 1992]. Morph learns to play ches with a pattern cache that is gradually improving against strong commercial chess program [Levinson and Snyde 1991]. Hoyle learns to play many simpler two-perso perfect information board games extremely well against variety of experts [Epstein, 1992]. TD-gammon learns th weights for its neural net, Morph learns patterns, and Hoyl learns useful knowledge about each game, knowledge that i probably correct and possibly applicable in a variety c contexts.

Do game-learning programs learn to play perfectly, o only as well as people? Against what kind of opposition d they learn to play best? Does the nature of the oppositio affect their learning speed or long-term memor requirements? How does learning differ when th opposition's errors are due to lack of foresight, to lack o knowledge, or to random decisions? This paper describes recent experiment with Hoyle to address these issue Because Hoyle learns a broad variety of games against an specified opposition, it can be used to explore whether th answers to these questions are game-dependent.

## Experimental Design

h *trial* for this experiment has Hoyle learn a game while ing against another program, called a *trainer*, and then Hoyle's post-learning playing skill against four kinds pposition.

### The learning program

le is based upon FORR, a general architecture for a ning and problem solving expert, one that postulates and talizes upon regularities (Epstein, 1991). Hoyle's ain is two-person, perfect information board games. en the definition of a new game, Hoyle begins as a rule- ing novice that plays against an external, presumably ert, model. This model is only observed, never queried. Hoyle plays it gradually improves, often becoming ert or even perfect at a game.

ach *game* Hoyle can play is an instantiation, a pre- ified, input instance, of a *game frame*. The only specific wledge Hoyle has about a new game before playing is values associated with these slots. Some slots hold stants: the name of the game, the markers assigned to participant, the initial state of the board before play, ther the board is two-dimensional or three-dimensional, often to scroll the screen during play, which places on oard are considered adjacent in games where pieces may , which lines on the board are considered wins if the e is won that way. Other slots hold the names of LISP tions: they display the current game on the screen, read filter input moves, generate and effect legal moves, ct the end of a contest and who has won, and transform board back and forth between a list and a visual esentation. These functions are very brief, typically a of less than 100 lines of code per game.

oyle's *game-playing algorithm* is a script that provides defined, uniform, procedural direction to the program. game-playing algorithm enables Hoyle to perform as if ere experienced in game playing, without expertise at particular game. This script detects when it is the gram's turn to move, ensures that the participants rnately make legal moves, and announces the end of each test, along with any winner. Given a valid game nition and the game-playing algorithm, Hoyle simulates le-abiding novice, one that makes legal, if not astute, es.

he game-playing algorithm also triggers Hoyle's rner. The *Learner* is a set of algorithms for the discovery *useful knowledge*, knowledge that is expected to be vant and may be correct. Based on its playing experience, le computes and stores game-dependent useful wledge. The Learner has a uniform, heuristic, game- pendent learning procedure for each item of useful wledge. If the Learner were to retain everything Hoyle eriences, useful knowledge for an interesting game could kly become unmanageably large. Therefore the learning rithms generalize and are highly selective about what retain. There are useful knowledge slots to record rage contest length, applicable two-dimensional metries, good openings, moves that expert opposition ears to have found valuable, relevant forks, important test histories, whether going first or second is an advantage, and *significant states,* situations that will inevitably be won or lost when both participants play expertly.

The application of learned useful knowledge is the task of Hoyle's Advisors. An *Advisor* is a heuristic that makes comments about legal moves when it is the program's turn to make one. A *comment* is the Advisor's name, a move, and a *weight*, an integer from 0 to 10, indicating an opinion somewhere in the spectrum from strong aversion (0) to enthusiastic support (10). Each Advisor constructs its comments based upon the current state and the useful knowledge for the current game. For example, Victory compares useful knowledge with the current legal moves, and recommends with a weight 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.) From the Advisors' comments, a simple arithmetic calculation selects a move that is forwarded to the game-playing algorithm for execution.

### 3.2 The trainers

There are five kinds of trainers in this experiment: self, random, perfect, fallible, and informed. The *self trainer* is the learning program itself, which takes both sides in every contest. The *random trainer* makes randomly-chosen legal moves, has no knowledge, and treats every possibility as equally likely. The *perfect trainer* plays as if it had exhaustively searched the tree and minimaxed the result back up to the current state to select its next move [Nilsson, 1980]. If there is more than one equally good move, the perfect trainer will make a random choice from among them. A perfect trainer for tic-tac-toe, for example, opens half the time in the center and half the time in a randomly-chosen corner. A perfect trainer is designed to provide a variety of high-quality expert play without mistakes. The *fallible trainer* is a variation on the perfect trainer, a variation that periodically has the opportunity to make a mistake. The fallible trainer with an error rate of $e\%$ makes the perfect trainer's move $100 - e\%$ of the time, and makes a randomly-chosen, legal, not necessarily imperfect move $e\%$ of the time. There is a spectrum of fallible trainers in this experiment with $e$ values in multiples of 10, from 10 to 90. Although a 10% chance of error may seem high, in many games the *branch factor* (number of legal alternative moves) decreases as play progresses, so that the likelihood of an error diminishes towards the end of a contest. For example, if there are three legal moves left 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 at that point even though $e = 10$. The *informed trainer* applies an input, game-dependent evaluation function in an alpha-beta search to a fixed depth [Nilsson, 1980]. 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 an approximation of the knowledge inherent in the game tree, not because the trainer has made a randomly chosen move. Informed training is flawed by lack of foresight and lack of knowledge; fallible training is flawed by random error. Each evaluation function is absolutely correct at the end of a contest and reasonable but imperfect elsewhere, as if the trainer had good but incomplete understanding of the game. The informed trainers in this experiment, each with its own trial, have $d$ values from 2 through 8.

### 3.3 The challengers

Post-learning playing skill is thoroughly tested against varied opposition called *challengers:* 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* uses the same algorithm as the perfect trainer. The *expert challenger* simulates an expert equivalent to a fallible trainer with a 10% error rate; 90% of the time it plays flawlessly, 10% of the time it may err. The *novice challenger* simulates an expert equivalent to a fallible trainer with a 70% error rate; only 30% of the time is it guaranteed to play perfectly. Error rates for both the expert and the novice were selected from laboratory observation of their resultant play quality. Finally, the *random challenger* makes random legal moves.

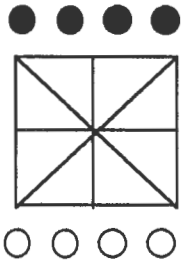

*Figure 1. The initial state for achi.*

### 3.4 The games

The three games in this experiment 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 of your markers in any empty square. 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 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 1. The first participant has four black markers; the second has four white ones. Initially the board is empty, black moves first, and a turn is placing one of your markers on the intersection of two or more lines; there are nine such positions. Once all four of your markers are on the board, a turn is moving one of your markers to the single empty position. The first one to place three of the same markers in a row, vertically, horizontally, o diagonally, wins. There are eight such winning lines. Pla ends in a draw when it cycles through the same state for th fourth time.

The construction of a perfect trainer and fallible trainer requires a complete and correct *perfect play theory* for game, a real-time algorithm that identifies the best possibl move from every possible game state. The construction of thoughtful but partially flawed evaluation function require good human understanding of the features of a game. Th games for which people have both a perfect play theory an such understanding are fairly simple. One way to make th learning task more difficult is to select games where th perfect play theory known to humans is not naturall expressible in the learning program's representation. Los tic-tac-toe was chosen because Hoyle cannot represent it perfect play theory explicitly, although it can eventuall acquire enough useful knowledge to play it perfectly. Los tic-tac-toe was also selected because a randomly chose move is likely to be a fatal error, and because the object i to *avoid* achieving a pattern, unlike the other two games Achi was chosen because of the contrasts it offers to th other two: it has two stages with different move types, it i cyclic, and its branch factor remains four throughout th second stage of every contest. All these games have certai commonalities that make comparison appropriate: thei boards are isomorphic and each is known to be a *draw gam* (when played by perfect participants a contest always ends i a draw).

### 3.5 The learning and testing cycle

A *trial* in this experiment consists of a learning experienc followed by a testing experience. At the beginning of a tria Hoyle has no specific knowledge about any of the games. *learning experience* is determined by the choice of a gam and a trainer. Since there are three possible games and, wit the values for $e$ and $d$, 19 trainers, there are 57 trials. Onc the game and the trainer for a trial are specified, Hoyle play a tournament of contests at the specified game with it designated trainer. After the program is judged to hav learned to play, learning is turned off, and Hoyle's skill i evaluated in a *testing experience,* a 20-contest tournamen against each of the challengers. In both training and testin tournaments, Hoyle and its opposition alternate playin first.

This design makes several assumptions. The learner is nc required to discover any role advantage inherent in the tree the program is told that these are draw games. The progra is instructed to stop learning when it meets a *behavior standard,* i.e., when it draws or wins 10 consecutiv contests; it evaluates its playing performance based on thi externally specified standard. Finally, the program i expected to perform well against any competition. It shoul be able to exploit its opposition's errors and to deal we with foolish moves.

Every learning experience is non-deterministic because th trainer or Hoyle can make random legal move choices fro time to time. A single run for a trial may therefore not b representative of the trainer's impact on learnin performance. To compensate for this uncertainty, each tri is run five times, and the results averaged.

### The evaluation criteria

ıasuring whether or not a program plays perfectly requires er exhaustive testing (the play of all possible contests) perfect play theory for the game. For most interesting ıes neither of these is an option. An alternative standard ɔ require that the program achieve the best possible *:ome* (win, loss, or draw) the game tree offers, from any ıe state, against any opposition. Evaluation of learning ɔd on contest outcome seeks effective play, rather than ɛct play. It still demands, however, that the participant full advantage of the opposition's mistakes, and ıtification of those mistakes is, once again, difficult to ısure.

more workable standard is *role performance*. In a draw ıe, role performance requires a draw whether one moves or second in the contest, even against a perfect player. ɔe losses in a draw game are always avoidable, a loss by learner indicates imperfect performance. A win by the ner in a draw game, however, only indicates the ɔessful exploitation of a fatal error made by the osition. This experiment uses role performance as the ning criterion: for a draw game and a challenger, *ıbility* (the ability to withstand the competition) is ısured by the percentage of wins and draws, and *power* ability to exploit the opposition's errors) is measured he percentage of wins. A perfect player would be 100% ıble against all challengers and maximize its power as umstances permitted. In a draw game, power against the ert challenger is always 0%.

ɔ compare training behaviors across trainer type, con- ct a similarity metric for post-learning playing behaviors ollows. Let a trial of the experiment be represented as | $b_{ij}$ |, a 4×3 matrix where $b_{ij}$ is the average of the ɔutcome (number of wins, losses, or draws) against the challenger (perfect, expert, novice, or random). Define *challenger difference measure* of two trials **B** and **B'** for *i*th challenger to be the sum of the squares of the corre- nding differences in their relevant rows

$$\sum_{j=1}^{3} (b_{ij} - b'_{ij})^2$$

l define the *difference of two testing behaviors* as the sum ıeir four challenger differences,

$$\sum_{i=1}^{4} \left( \sum_{j=1}^{3} (b_{ij} - b'_{ij})^2 \right).$$

ıtical testing behaviors have a zero difference in each row a zero difference overall. The testing behavior among , $\mathbf{B_2}$, ..., $\mathbf{B_k}$} *most similar* to behavior **B** is the one ɔse difference from **B** is a minimum.

inally, a learning program with an imperfect trainer may that learning takes longer, or that it is burdened by ıy unimportant recollections. For each trial *learning time* neasured by the number of contests to meet the avioral standard, in this experiment a minimum of 10. *rning space* is measured by the additional memory cated to Hoyle's associated useful knowledge cache, a ristically restricted set of game states, moves, and contest ɔries recorded from playing experience.

## 4. Results and Discussion

The results of the 57 trials are summarized here. An important idea 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, after drawing 10 consecutive contests of lose tic-tac-toe against a perfect trainer, Hoyle then lost 12% of its contests against the expert challenger. This indicates that learning was incomplete, i.e., that training against a perfect player inadequately prepared the program for competition against a strong player that makes occasional mistakes. In the discussion that follows, Cox and Stuart's non-parametric binomial test for trend is used to calculate whether or not there is a correlation between two sequences of numbers [Conover, 1980]. All correlations cited are 93.75% or better unless otherwise indicated. When one of the sequences is $e$ values, data from the perfect trainer ($e = 0$) and the random trainer ($e = 100$) are also included.

### 4.1 Individual games

#### 4.1.1 Tic-tac-toe

Hoyle's Advisors are immediately able to support a fairly high level of play at tic-tac-toe. Against any informed trainer, for example, they never lose a 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. The perfect trainer periodically presents this game state; Hoyle fails, learns from its loss, and never makes a mistake in that state again. During training, however, there is no guarantee when or if that state will arise, particularly with a less than perfect trainer. Hoyle can learn many other things from less high-quality mistakes during training, but this particular piece of useful knowledge is essential for perfect reliability.

Only when $e = 10$ was perfect reliability at tic-tac-toe achieved consistently, i.e., against all the challengers in every run. In every other trial, even in perfect training, Hoyle lost a contest to some challenger in at least one run. These losses ranged from 2% to 5.5% in trials where some other run achieved perfect reliability. This suggests that at best $e = 10$ can be trusted to develop reliability. Reliability against the perfect challenger, the expert challenger, and the novice challenger were negatively correlated with $e$ value, i.e., *the more fallible the tic-tac-toe trainer, the less reliably Hoyle performed.*

Hoyle's power in tic-tac-toe ranged from 12% to 23% against the expert, from 72% to 85% against the novice, and 85% to 98% against the random challenger. Maximal power against the random challenger was developed from training with $e = 60$, against the novice with $d = 6$, and against the expert with random training.

Learning time averaged roughly 12 contests for $e \leq 70$ and for perfect training; in all the other trials it was an overconfident 10. Learning time was negatively correlated with $e$ value; the more fallible the trainer, the faster the behavioral standard was met. Learning space ranged from 1 unit, for self training and all informed training, to 24.6 for $e = 70$. Learning space was positively correlated with $e$ value;

the more fallible the trainer the more, presumably useful, knowledge was acquired. The self-trained program lost 30% of its contests to the perfect challenger and 2% to the novice challenger, but also managed to win 14% against the expert, 72% against the novice, and 93% against the random challenger.

### 4.1.2 Lose tic-tac-toe

Lose tic-tac-toe is a game where relatively few moves are optimal, and even a single suboptimal move usually costs one the contest [Cohen, 1972]. For X there is typically exactly one correct move, including the single correct opening. As a result, the perfect play algorithm must be quite rigid and offers little opportunity to acquire power during training.

Only after perfect training was Hoyle always perfectly reliable, and then only against the perfect challenger. After perfect training Hoyle still lost 12% of its contests to the expert, 18% to the novice, and 19% to the random challenger. Clearly, learning had been incomplete. (Inspection revealed that during training Hoyle usually met the behavioral standard by the endgame skill it had acquired.) For any other training, reliability was dramatically worse; losses to the perfect challenger averaged from 40% to 61%, to the expert from 4% to 46%, to the novice from 14% to 29%, and to the random challenger from 7% to 23%. The program sporadically achieved perfect reliability against the perfect or the expert challenger on a single run for several low $e$ values, but still went on to lose at least four testing contests against the other challengers in the same run. The most consistently reliable performance against the expert was achieved by $e = 20$, against the novice by $e = 20$ and $d = 2$, and against the random challenger by $d = 5$. Reliability against the perfect, expert, and novice challengers decreased with the error rate, i.e., the trainer's lack of skill appears to have misguided the learner. Reliability against the random challenger, however, *increased* with the error rate.

Hoyle's power at lose tic-tac-toe ranged from 14% to 41% against the expert, from 54% to 72% against the novice, and from 60% to 80% against the random challenger. Maximal power against the random challenger was developed from training with $d = 5$, against the novice with $e = 80$, and against the expert in three trials, with $e = 80$, with random training, and with $d = 4$. Power against the expert challenger and against the random challenger *increases* with the error rate of the trainer. Presumably *the lack of errors during training made the program less able to maneuver in the search space when the testers erred.*

Learning time ranged from 10 contests, during two runs for $e = 80$, to 156 for a run when $e = 70$. Learning space ranged from 56.8 units for $d = 7$ to 340.8 for $d = 3$. Learning space increased with learning time for fallible training. The self-trained program lost 61% of its contests to the perfect challenger, 46% to the expert, 23% to the novice, and 22% to the random challenger, but also managed to win 14% against the expert, 66% against the novice, and 69% against the random challenger.

### 4.1.3 Achi

Achi is a game where most serious errors occur early, in the stage when markers are first placed on the board. Contests average 68 moves, offering ample opportunity for careless error while play cycles to a draw.

When $e = 20$, 30, 80, 90 and when $d = 2$, 4, 5, 6, 8 perfect reliability was achieved consistently, i.e., against all the challengers in every run. For other training, losses were rare (about .03%) and never to the expert challenger. Hoyle's power in achi ranged from 58% to 77% against the expert from 97% to 100% against the novice, and 99% to 100% against the random challenger. Maximal power against the expert was achieved with $e = 20$.

Learning time ranged from 10 to 13 contests, but was greater than 10 in only 3 runs. Learning space ranged from 1 unit, for self training, all informed training, and perfect training, to 52.4 for $e = 90$. Learning space is positively correlated with learning time for fallible training. Inspection of the useful knowledge cache revealed that, against a fallible trainer, the program always learns some accurate and quite sophisticated achi strategy that it does not acquire during perfect training. The resultant increase in its learning space from fallible training does not, however, improve the program's reliability or make a statistically significant change in its power. Although the knowledge was correct and clever, it had no visible impact on the evaluation criteria posited here, i.e., *the fallible achi trainer induced learning that was a waste of resources.* The self-trained program loses 1% of its contests to the random challenger, but also manages to win 72% against the expert, 100% against the novice, and 99% against the random challenger.

## 4.2 The impact of trainer error

Although it is possible to train a program to be at least fairly reliable for each of the three games, imperfect training offers better preparation for the occasional opportunities that arise across a broad range of competition. In lose tic-tac-toe the most difficult of the three games for Hoyle to learn, the $e$ value is correlated with the number of wins against the expert challenger; the *more* fallible the trainer, the more powerful the program. *When learning is incomplete Hoyle's best preparation for imperfect play is a fallible trainer.* After perfect training the program was 100% reliable against the perfect challenger, but only 88% against the expert challenger, 82% against the novice challenger, and 81% against the random challenger.

When self training is compared to the entire range of $e$ values for fallible training under the similarity metric of Section 3, there is a dramatic and distinctive similarity between self testing and $e = 60\%$ for tic-tac-toe, $e = 50\%$ and 80% for achi (where the difference from self training is almost 0), and $e = 70\%$ for lose tic-tac-toe, i.e., *self training is like learning against a fallible player many of whose moves may be errors.* At tic-tac-toe, self training produced the lowest power against the novice and near the lowest against the expert; it was also only 70% reliable against the perfect challenger. At lose tic-tac-toe, self training produced the lowest power against the expert and the lowest or near the lowest reliability against every challenger. Only at achi the game where useful knowledge had the least impact, was self training moderately reliable and powerful.

The difference between learned behavior after informed trainer error and after random trainer error appears strongly related to both the nature of the evaluation function and the

ıe. For tic-tac-toe and achi, informed training relied upon number of potential winning and losing lines (up to 8) the board. This greedy approach always opens in the ter, regardless of depth. As a result, no informed training ırs Hoyle the opportunity to learn the simple fork from a ıer opening that made the learner so reliable after perfect ning at tic-tac-toe. Hoyle learns a minimum during ırmed training at any depth in these two games, so its ıltant testing behavior is simply the luck of the draw inst its challengers.

'or lose tic-tac-toe, however, unless the trainer makes a ɔf mistakes, a program that meets the behavioral standard ɔing to have to learn to open in the center. This is a ve most people, and therefore the evaluation function we d, find highly counterintuitive. An informed trainer, ardless of depth, will never open in the center: either it not search deeply enough or it will exercise its left-to-ıt bias. For Hoyle to learn the correct opening against an ırmed trainer, it must observe both a corner and a side ning, and then prove that those openings, rather than any r move, were responsible for the subsequent losses. Even ı, the program must learn how to play *after* the correct ning. Against a fallible trainer, there will be more ortunity for this to happen; against an informed trainer / half the contests (where Hoyle goes first) even offer the ortunity to learn to play X perfectly. As a result, ırmed training in lose tic-tac-toe is often slower than ble training, and the resultant quality of play can be ker.

## Related Work

s research differs from prior work by educators and chologists because it is able to start each learning erience with a machine that offers a *tabula rasa,* a clean e. Because people cannot clear their minds of all prior erience, and because they may learn differently, the ılts may not be analogous to people. Hoyle's learning ed and output results, however, do simulate an erienced game player encountering an unfamiliar game stein, 1992].

Neurogammon, an earlier version of TD-gammon, was a ral net program that learned to play backgammon ıauro and Sejnowski, 1989]. The program was trained to ct the moves made in 400 contests where Tesauro, a ng but not world-class player, had played both sides. At First Computer Olympiad in London in 1989, ırogammon was clearly the strongest non-human ıpetitor. When Neurogammon plays TD-gammon, vever, it only wins 40% of the time. There are several ious explanations for TD-gammon's improved strength. t, TD-gammon had much more extensive training; it ned on approximately 200,000 contests. Second, TD-ımon uses the TD(λ) algorithm instead of ırogammon's standard back-propagation [Sutton, 1988; nelhart *et al.*, 1986]. Finally, TD-gammon trains against lf, probably with a more varied set of experiences.

ın alternative kind of training attempts to prime the ner, to provide it with a head start by first observing two fect players in competition before making any moves lf. Experiments with a simple pattern-learner and reinforcement training for several games on a three-by-three grid have indicated that priming *slows* learning [Painter, 1992]. 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. Painter also found that a random trainer produced a less reliable player. N-N/Tree, a hybrid learning program with a neural net, was also found to suffer from priming [Flax *et al.*, 1990]. N-N/Tree also learned to play far better against a fallible player with $e = 5$ than against a perfect player.

## 6. Conclusions

The role of the trainer in a competitive machine learning experience has usually been a matter of convenience. Input book games require the tedious assembly of databases. Human opposition of any caliber plays too slowly, tires too quickly, and may be fairly rigid in approach. That leaves only opposition from another machine.

One way to see training for a competitive domain is as a set of paths through a game tree. If the trainer always plays perfectly, the learner will have no experience with large portions of the problem space. After such an overly narrow learning experience, there is no reason to believe that a program will have the skill to deal with errors, or even with suboptimal moves, let alone exploit them to its advantage. The data presented here confirm this, particularly in a game where relatively few moves are good choices. (Go is reputed to be such a game.) A competitive learning experience against a perfect player is flawed, and the resultant performance is disappointing. No single trainer, in any of the games, achieved maximum power against all the challengers. *Training against weak opposition is inadequate preparation for a stronger opponent, but training against strong opposition also turns out to be less adequate for a weaker opponent when learning is incomplete.*

A somewhat less than perfect training experience introduces some variety into the paths through the game tree. Whether this variety is engendered by random noise or by lack of foresight and knowledge was not significant in the three games considered here. This experiment suggests that a trainer informed by an evaluation function but hampered by lack of exhaustive search not only has the narrowness of the perfect trainer, but is also no more valuable as its depth increases. One might expect, however, that in a game with a larger branching 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.

It is possible that the results described here are a function of the learning program, i.e., Hoyle, rather than of the trainer. For example, one facet of Hoyle is its ability (but by no means proclivity) to imitate expertise it has observed in the opposition. TD-gammon, N/N-Tree, and Morph all imitate the opposition too, but with different learning methods from Hoyle's. A program that ignored the behavior

of the opposition might be less susceptible to the influence of its trainer, although one could argue that it was not particularly intelligent either. 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 [Tesauro, 1991; Flax *et al.*, 1990]. The results described here, in conjunction with theirs, suggest that the conclusion about the need for variety in training is a function of the learning task, not of the particular learning methods used here.

When learning time was relatively constant, learning space was observed (87.5% correlation for achi, 93.75% for tic-tac-toe) to increase with the fallibility of the trainer. When learning time in fallible training varied widely (in lose tic-tac-toe), learning space varied with it. In both cases, this is because the heuristics pick up a good deal of data that probably does not strengthen play, simply because the program was exposed to so many mistakes. A neural net has no such difficulty. It would be interesting to see whether other methods also have larger long-term memory requirements when training against weaker opposition.

Learning time was dependent on many factors. Hoyle simulates an expert game player; it has general heuristics for playing all games even before it learns about any specific game. This makes its initial level of play higher than, say, a neural net that randomized its initial weights. As a result, Hoyle could meet the behavioral standard relatively quickly in tic-tac-toe and achi. A fallible trainer often introduces peculiar situations that the program would fail on, but learn not to repeat, with a resultant increase in learning time. The more fallible the trainer, the more often this can happen. On the other hand, a fallible trainer also makes mistakes that the program can exploit. The more fallible the trainer, the more often this happens too, so that a highly fallible trainer may allow the program to meet the behavioral standard too quickly. All these factors visibly interacted, masking any correlation.

Self training offers a natural, gradual progression from weaker to stronger. It should therefore prepare the program for any opposition as good as itself. Of course, using a program as its own trainer deprives it of an important knowledge source that people learn from, the expert model. Self training is also a substantially slower way to acquire broad expertise; TD-gammon spent the first 25% of its training playing "long, looping contests that seemed to go nowhere" [Tesauro, 1991].

One solution to the high estimated $e$ values similar to self training might be to raise the behavioral standard above 10 to lengthen the training time. In a game with a small search space this should augment Hoyle's useful knowledge and improve its performance, but there can be no guarantee of perfection. This was demonstrated in single runs with $e$ from 20 to 60 for lose tic-tac-toe, with the behavioral standard set at 100 instead of 10. Learning time with this higher behavioral standard ranged from 297 to 505 contests, instead of 10 to 156, and learning space from 336 units to 809, instead of 56.8 to 340.8. In every run with the higher behavioral standard, despite the fact that Hoyle had seen much more of the search space, the program still lost from 1% to 2% of its contests and showed no significant change in power. It is important to note that those losses were only to the novice or the random challenger. *A higher behavioral standard improved reliability but not power.* A higher behavioral standard also substantially increased learning time and memory requirements. In 505 contests averaging 9 states with markers on the board, Hoyle encounters 4545 (not necessarily distinct) states out of 5478 possible distinct states in the entire search space. Even after the opportunity to encounter as much as 83% of the entire search space, Hoyle is not perfectly reliable. *What is required is not only more training but broader based training, i.e., novel experiences.*

For games without an element of chance, variety in training can be introduced with a broad spectrum of opposition. As a result of this experiment, we recommend a hybrid training experience for any program in a competitive domain, one that interleaves sessions against a perfect trainer with practice against itself. That is the method Hoyle now uses in discovery mode, to develop its own expertise without human guidance. When Hoyle trains this way, it learns to be perfectly reliable and very powerful against all the challengers.

For more difficult games where no perfect trainer is possible, this research offers no reason to believe that self-training can ever result in perfect play. For such games, these results would advocate *lesson-and-practice training:* alternating sets of a few contests against the best player available (the lesson) and many contests against the learning program itself (the practice). Whether or not a perfect player is available, training should be periodically spiced with particularly fallible opposition and a random move or two, to keep a novice from taking the learner by surprise. Under this regimen, 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. A program can meet a behavioral standard and still have much to learn.

## Acknowledgments

The author thanks Jack Gelfand and Gerry Tesauro for insightful discussions. Kouros Esfahany and Joanna Lesniak provided expert-level data generation and programming support.

## References

[Anantharaman *et al.*, 1990] T. Anantharaman, M. S Campbell, and F.-h. Hsu. Singular Extensions: Adding Selectivity to Brute-Force Searching. *Artificial Intelligence* 43 (1): 99-110, 1990.

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

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

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

stein, 1991] S. L. Epstein. *Learning under a Weak heory*, Technical Report 91-01, Department of Computer cience, Hunter College, 1991.

stein, 1992] S. L. Epstein. Prior Knowledge Strengthens earning to Control Search in Weak Theory Domains. ıternational Journal of Intelligent Systems, to appear.

ıx *et al.*, 1990] M. G. Flax, J. J. Gelfand, S. H. Lane, ıd D. A. Handelman. Integrating Neural Network and ree Search Approaches to Produce an Auto-Supervised ystem that Learns to Play Games. In *Proceedings of The erospace Applications of Artificial Intelligence onference,* Dayton, OH, 1990.

vinson and Snyder, 1991] R. Levinson and R. Snyder. daptive Pattern-Oriented Chess. In *Proceedings of the ighth International Machine Learning Workshop,* 85-89. an Mateo, CA, August, 1991. Morgan Kaufmann.

sson, 1980] N. J. Nilsson. *Principles of Artificial ıtelligence.* Tioga Publishing, Palo Alto, CA, 1980.

nter, 1992] J. Painter. Pattern Recognition for Decision laking in a Competitive Environment. Master's diss., unter College of the City University of New York, in reparation.

melhart *et al.,* 1986] D. E. Rumelhart, G. E. Hinton, ıd R. J. Williams. Learning Internal Representation by rror Propagation. In *Parallel Distributed Processing,* ol. 1, ed. D. E. Rumelhart and J. McClelland. MIT ress, Cambridge, MA, 1986.

nuel, 1963] A. L. Samuel. Some Studies in Machine earning Using the Game of Checkers. In *Computers and hought,* ed. E. A. Feigenbaum and J. Feldman. McGraw-ill, New York, NY, 1963.

nuel, 1967] A. L. Samuel. Some Studies in Machine earning Using the Game of Checkers. II - Recent rogress. *IBM Journal of Research and Development* 11 5): 601-617, 1967.

:ton, 1988] R. S. Sutton. Learning to Predict by the lethods of Temporal Differences. *Machine Learning* 3 (9-4, 1988.

auro and Sejnowski, 1989] G. Tesauro and T. J. ejnowski. A Parallel Network that Learns to Play ackgammon. *Artificial Intelligence* 39 (3): 357 - 390, 989.

auro, 1991] G. Tesauro. Personal communication.

auro, 1992] G. Tesauro. Practical Issues in Temporal ifference Learning. *Machine Learning,* to appear.