# TRAINING INTELLIGENT AGENTS USING HUMAN DATA COLLECTED ON THE INTERNET

ELIZABETH SKLAR
*Dept. of Computer Science*
*Boston College*
*Chestnut Hill MA 02167 USA*
*E-mail: sklar@cs.mu.oz.au*

ALAN D. BLAIR
*Dept. of Computer Science*
*and Software Engineering*
*University of Melbourne*
*Victoria 3010 AUSTRALIA*
*E-mail: blair@cs.mu.oz.au*

JORDAN B. POLLACK
*DEMO Lab*
*Dept. of Computer Science*
*Brandeis University*
*Waltham, MA 02454-9110 USA*
*E-mail: pollack@cs.brandeis.edu*

To facilitate human learning in a game, it is desirable to select proper opponents for humans so that players will advance through a series of games and boredom and frustration will be avoided. In the work presented here, we use human input as the basis for constructing a graded population of artificial opponents — intelligent software agents — so that for future encounters, we could choose appropriate opponents from this population in order to provide continuous challenge to individual human learners. In this paper, we describe a method for training such a population of agents using human data collected at two Internet gaming sites. We compare two different approaches for using this data: individual and collective. We detail both approaches and present experimental results.

*Keywords:* software agents, human learning, neural networks, Internet

## 1 Introduction

Hidden inside every mouse click and every key stroke is valuable information that can be tapped, to reveal something of the human who entered each action. On the Internet, these inputs are called *clickstream* data, "derived from a user's navigational choices expressed during the course of visiting a World Wide Web site or other online area."[1]

Clickstream data can be analyzed in two ways: individually, as input from single users, or collectively, as input from groups of users. Individualized input may be utilized to create user profiles that can guide activities on a web site tailored to the needs of a particular person. Data mining the clickstream to customize to individual users is nothing new. Starting as early as 1969, Teitelman began working on the automatic error correction facility that grew into DWIM (Do What I Mean).[2] In 1991, Allen Cypher demonstrated "Eager", an agent that learned to recognise repetitive tasks in an email application and offered to jump in and take over for the user.[3] In 1994, Pattie Maes used machine learning techniques to train agents to help with email, to filter news messages and to recommend entertainment, gradually gaining confidence at predicting what the user wants to do next.[4]

Today, commercial products like MicrosoftWord provide context-sensitive "wizards" that observe their users and pop up to assist with current tasks. Internet sites like altavista[a] recognise keywords in search requests, offering alternate suggestions to help users hone in on desired information. At the amazon.com[b] book store, after finding one title, other books are recommended to users who might be interested in alternate or follow-up reading. On many sites, advertisements which at first seem benign, slowly adapt their content to the user's input, subtly wooing unsuspecting surfers.

Input from users may also be examined collectively and grouped to illuminate trends in human behavior. Users can be clustered, based on a feature like age or gender or win rate (of a game), and the behavioral data for all humans exhibiting the same feature value can be grouped and analyzed, in an attempt to recognize characteristics of different user groups.

An Internet system allows us to combine user profile knowledge with statistics on group behavior (from a potentially very large set of humans) in order to make more informed decisions about software adaptation than input from a single source would provide. These techniques may prove especially useful when applied to educational software. The work presented here examines these ideas in the context of an Internet learning community where humans and software agents play games against each other.

## 2 Motivation

Many believe that the secret to education is motivating the student. Researchers in human learning have been trying to identify the elements of electronic environments that work to captivate young learners. In 1991, Elliot

[a] http://www.altavista.com
[b] http://www.amazon.com

Soloway wrote "Oh, if kids were only as motivated in school as they are in playing Nintendo."[5] Two years later, Herb Brody wrote: "Children assimilate information and acquire skills with astonishing speed when playing video games. Although much of this gain is of dubious value, the phenomenon suggests a potent medium for learning more practical things."[6]

Thomas Malone is probably the most frequently referenced author on the topic of motivation in educational games. In the late 1970's and early 1980's, he conducted comprehensive experimental research to identify elements of educational games that made them intrinsically motivating.[7] He highlights three characteristics: *challenge*, *fantasy* and *curiosity*.

We are primarily interested in the first characteristic. Challenge involves games having an obvious goal and an uncertain outcome. Malone recommends that goals be "personally meaningful", reaching beyond simple demonstration of a certain skill; instead, goals should be intrinsically practical or creative. He emphasizes that achieving the goal should not be guaranteed and suggests several elements that can help provide this uncertainty: variable difficulty level, multiple goal levels, hidden information, randomness. He states that "involvement of other people, both cooperatively and competitively, can also be an important way of making computer-based learning more fun."[8]

We concentrate on multi-player games, particularly on the Internet because it is widely accessible. The Internet offers the additional advantage that participants can be anonymous. Indeed, participants do not even have to be human — they can be software agents.

We take a population-based approach to agency.[9] Rather than building one complex agent that can play a game using many different strategies, we create a population of simple software agents, each exhibiting single strategies. The notion of training agents to play games has been around since (at least) the 1950's, beginning with checkers[10] and chess[11,12], and branching out to include backgammon[13,14,15], tic-tac-toe[16], Prisoner's Dilemma[17,18] and the game of tag.[19] With these efforts, the goal was to build a champion agent capable of defeating all of its opponents.

Our work differs because our goal is to produce a population of agents exhibiting a range of behaviors that can challenge human learners at a variety of skill levels. Rather than trying to engineer sets of strategies associated with specific ability levels or to adapt to individual players, we observe the performance of humans interacting in our system and use these data to seed the population of agents.

This paper describes our efforts training agents in two domains: one is a video game and the other is an educational game. In both cases, the agents were trained using human data gathered on our web site. We use this data both

individually and collectively. With the individual, or one-to-one, method, we use input from one human to train a single agent. With the collective, or many-to-one, approach, we use input from a group of humans to train a single agent. The first major section details the video game domain, outlining the agent architecture, the specifics of the training algorithm and experimental results. The second major section provides similar discussion of the educational game and additionally compares the results obtained in the two domains. Finally, we summarize our conclusions and highlight future directions.

## 3 The first domain: Tron

Tron is a video game which became popular in the 1980's, after the release of the Disney film with the same name. In Tron, two futuristic motorcycles run at constant speed, making right angle turns and leaving solid wall trails behind them — until one crashes into a wall and dies. In earlier work led by Pablo Funes[20], we built a Java version of the Tron game and released it on the Internet[c] (illustrated in figure 1). Human visitors play against an evolving population of intelligent agents, controlled by genetic programs (GP).[21] During the first 30 months on-line (beginning in September 1997), the Tron system collected data on over 200,000 games played by over 4000 humans and 3000 agents.

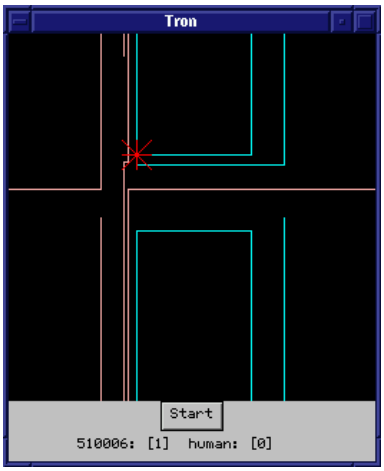



Figure 1: The game of Tron.

[c] http://www.demo.cs.brandeis.edu/tron

In our version of Tron, the motorcycles are abstracted and are represented only by their trails. Two players — one human and one software agent — each control a motorcycle, starting near the middle of the screen and heading in the same direction. The players may move past the edges of the screen and re-appear on the opposite side in a wrap-around, or *toroidal*, game arena. The size of the arena is $256 \times 256$ pixels. The agents are provided with 8 simple sensors with which to perceive their environment (see figure 2). The game runs in simulated real-time (i.e., play is regulated by synchronised time steps), where each player selects moves: *left*, *right* or *straight*.

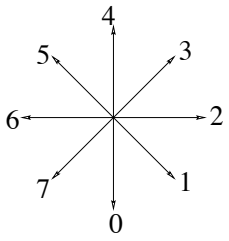


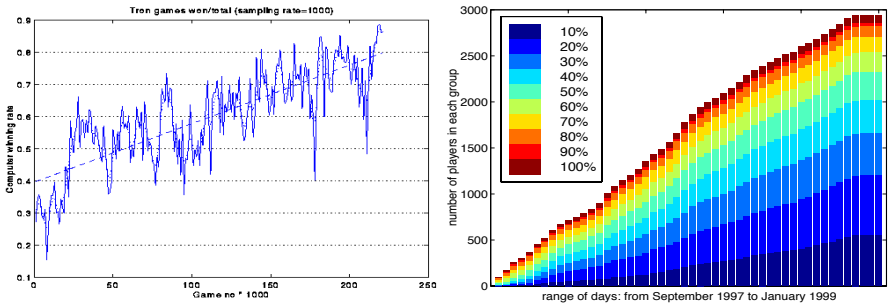

Figure 2: Agent sensors.
Each sensor evaluates the distance in pixels from the current position to the nearest obstacle in one direction, and returns a maximum value of 1.0 for an immediate obstacle (i.e., a wall in an adjacent pixel), a lower number for an obstacle further away, and 0.0 when there are no walls in sight.

Our general performance measure is the **win rate**, calculated as the number of games won divided by the number of games played. The overall win rate of the agent population has increased from 28% at the beginning of our experiment (September 1997) to nearly 80%, as shown in figure 3(a). During this time, the number of human participants has increased. Figure 3(b) illustrates the distribution of performances within the human population, grouped by (human) win rate. While some segments of the population grow a bit faster than others, overall the site has maintained a mix of human performances.

The data collected on the Internet site consists of these win rate results as well as the content of each game (referred to as the **moves string**). This includes the length of the game (i.e., number of time steps) and, for every turn made by either player, the global direction of the turn (i.e., north, south, east or west) and the time step in which the turn was made.

### *3.1 Agent Training and Control*

We trained agents to play Tron, with the goal of approximating the behaviour of the human population in the population of trained agents. The training procedure uses *supervised learning*[22,23], as follows. We designate a player to be the *trainer* and select a sequence of games (i.e., moves strings) that were played




(a) Agent win rate.

(b) Distribution of human population.

Figure 3: Results from the Internet experiment.

by that player, against a series of *opponents*, and we replay these games. After each time step, play is suspended and the sensors of the trainer are evaluated. These values are fed to a third player, the *trainee* (the agent being trained), who makes a prediction of which move the trainer will make next. The move predicted by the trainee is then compared to the move made by the trainer, and the trainee's control mechanism is adjusted accordingly.

The trained agents are controlled by a feed-forward neural network (see figure 4). We adjust the networks during training using the backpropagation algorithm[24] with Hinton's cross-entropy cost function.[25] The results presented here were obtained with $momentum = 0.9$ and $learningrate = 0.0002$.

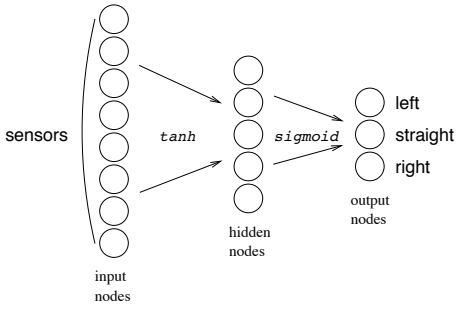
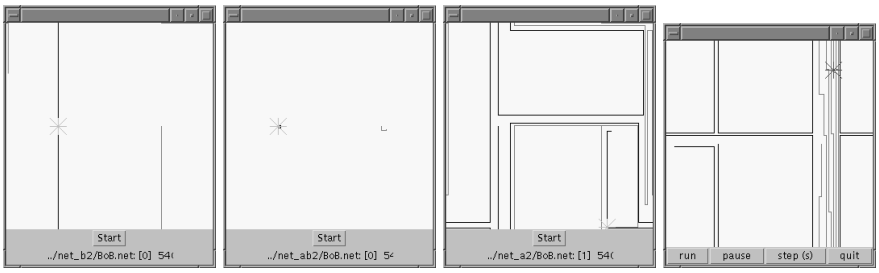



Figure 4: Agent control architecture. Each agent is controlled by a feed-forward neural network with 8 input units (one for each of the sensors in figure 2), 5 hidden units and 3 output units — representing each of the three possible actions (*left*, *right*, *straight*); the one with the largest value is selected as the action for the agent.

### *3.2 Challenges*

The supervised learning method described above is designed to minimize the classification error of each move (i.e., choosing $left$, *right* or *straight*). However, a player will typically go *straight* for 98% of time steps, so there is a danger that a trainee will minimize this error simply by choosing this option 100% of the time; and indeed, this behaviour is exactly what we observed in many of our experiments. Such a player will necessarily die after 256 time steps (see figure 5). Conversely, if turns are emphasized too heavily, a player will turn all the time and die even faster (figure 5b).




(a) a trainee that makes no turns (b) a trainee that only makes turns (c) a trainee that learns to turn (d) the trainer

Figure 5: A comparison of different trainees.
All had the same trainer; trainee variations include using 12-input network and different move evaluation strategies. All games are played against the same GP opponent. The player of interest is represented by the solid black line and starts on the left hand side of the arena.

The discrepancy between minimizing move classification error and playing a good game has been noted in other domains[14] and is particularly pronounced in Tron. Every left or right turn is generally preceded by a succession of straight moves and there is a natural tendency for the straight moves to drown out the turn, since they will typically occur close together in sensor space. In order to address this problem, we settled on an evaluation strategy based on the frequency of each type of move. During training, we construct a table (table 1) that tallies the number of times the trainer and trainee turn, and then emphasize turns proportionally, based on these values.

### *3.3 Experiments and Results*

We trained three populations of players: one with GP trainers and two with human trainers. Although our goal is to approximate the behaviour of the

Table 1: Frequency of moves, for the best human trainer.

| | | **trainee** | | |
|---|---|---|---|---|
| | | *left* | *straight* | *right* |
| **trainer** | *left* | 852 | 5360 | 161 |
| | *straight* | 5723 | 658290 | 5150 |
| | *right* | 123 | 4668 | 868 |

human population, we initially tuned our training algorithm by training agents to emulate the behaviour of the GP players from the Internet site. These GPs are deterministic players (so their behaviour is easier to predict than humans'), thus providing a natural first step toward our goal.

Separate training and evaluation sets were compiled for both training efforts, as detailed in figure 6. There were 69 GPs who had played more than 1000 games on the Internet site (agents1000); these were used as trainers. There were 135 GPs who had played between 100 and 1000 games (agents100); these were used for evaluation purposes. There were 58 humans who had played more than 500 games on the Internet site (humans500); these were used as human trainers.

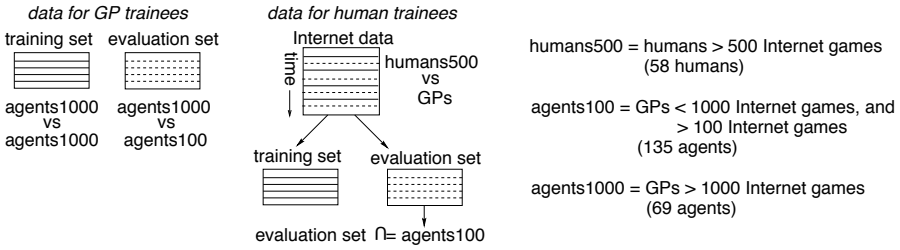



Figure 6: Data sets for training and evaluation.

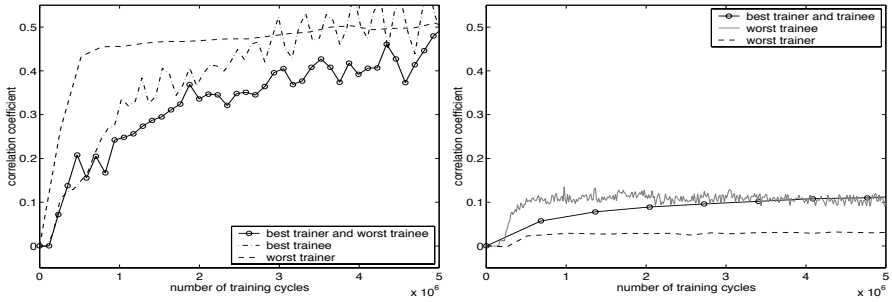
The humans500 data set was used both individually and collectively. First, 58 individual trainees were produced, based on a one-to-one correspondance between trainers and trainees. Second, 10 collective trainees were produced, based on a many-to-one correspondance between trainers and trainees, where the 58 individuals were sorted into 10 groups based on their win rates (e.g., group 1 had 0-10% win rate, group 2 had 10-20% win rate, etc.).

Each GP trainer played against agents1000 to produce a training set and against agents100 to produce an evaluation set. The games played by humans500 were alternately placed into training and evaluation sets, and then

the evaluation set was culled so that it consisted entirely of games played against members of the agents100 group.

We examine our training efforts in two ways. First, we look directly at the training runs and show the improvement of the networks during training. Second, we present the win rates of the two populations of trainees, obtained from playing them against a fixed set of opponents, and compare trainers with their trainees.

Our measure of improvement during training is based on the frequency of moves table and how it changes. Referring back to table 1, if the trainee were a perfect clone of its trainer, then all values outside the diagonal would be 0 and the *correlation coefficient* between the two players would be 1. In reality, the GP trainees reach a correlation of approximately 0.5, while the human trainees peak at around 0.14. For comparison, we computed correlation coefficients for 127 random players[d], resulting in a much smaller correlation of 0.003. Figure 7 shows the change in correlation coefficient during training for selected trainees.

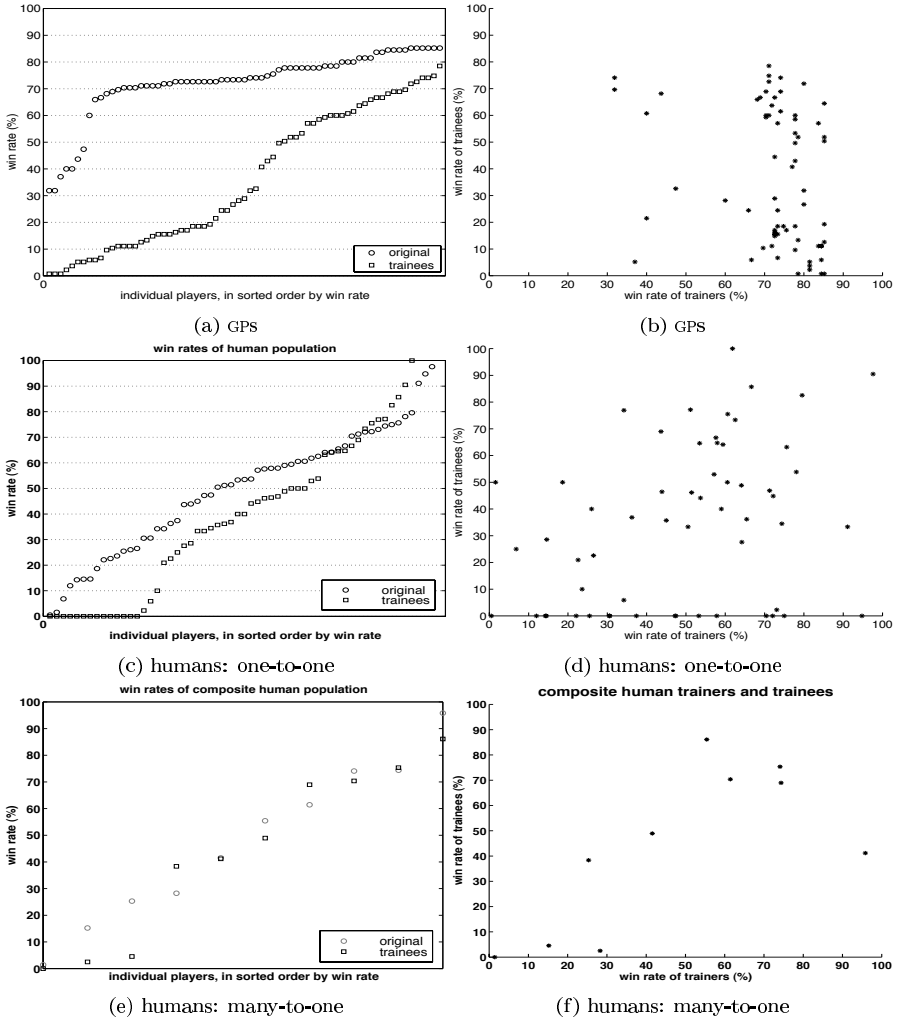



(a) GPs (b) humans

Figure 7: Change in correlation coefficient during training runs.

The win rates in the evaluation games for the trainers and trainees are shown in figure 8, for each of three training efforts: (1) GP training (figures 8a and 8b), (2) human one-to-one training (figures 8c and 8d), and (3) human many-to-one training (figures 8e and 8f). There are two types of plots shown. The first column contains the first type of plot (for each training group, figures 8a, 8c and 8e). Here, the players are sorted within each population according to their win rate, so the ordering of individuals is different within each trainer and trainee population. The plot demonstrates that the controllers have learned to play Tron at a variety of different levels.

[d] i.e., players that choose a move randomly at each time step.




(a) GPS (b) GPS

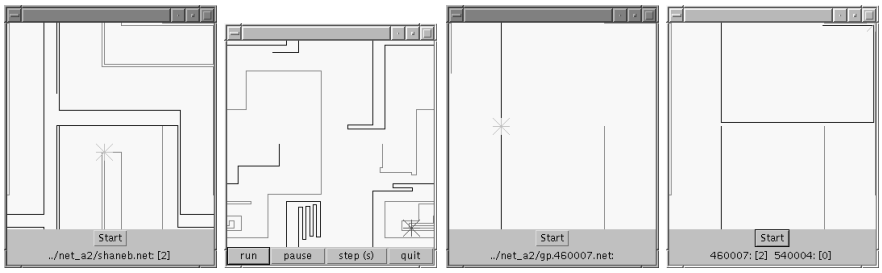
(c) humans: one-to-one (d) humans: one-to-one

(e) humans: many-to-one (f) humans: many-to-one

Figure 8: Win rates of trainer and trainee populations.
The horizontal lines denote boundaries for grouping players (according to win rate); the human trainers produce a population of trainees with a distribution across these groupings fairly similar to their own.

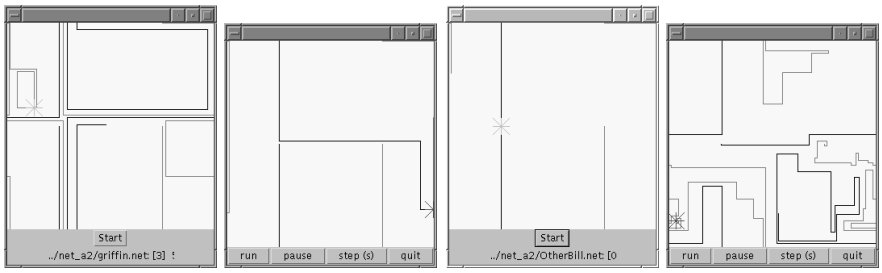
The second column contains the second type of plot (for each training group, figures 8b, 8d and 8f). Here, we plot the win rate of individual trainees against the win rate of their corresponding trainers. It is interesting to notice that the best human trainer (from figure 8d) has given rise to the best trainee (see figures 9a and 9b), while the best GP trainer (from figure 8b) has produced the worst trainee (see figures 9c and 9d). A few of the trainees play very poorly. These are cases where the network either fails to make any turns or makes turns at every move (in spite of the strategy described in section 3.2). Also, in a number of cases, the trainee outperforms its trainer.

Finally we step away from statistics and highlight some of the trainers and their trainees by showing selected games against the same opponent. Note two situations where a trainer that is a bad player produces a trainee that plays well (figures 9e and 9f), and a trainer that is a good player produces a trainee that plays poorly (figures 9g and 9h).




(a) trainee (b) trainer (c) trainee (d) trainer

*best human trainer is also best trainee* *best* GP *trainer is also worst trainee*




(e) trainee (f) trainer (g) trainee (h) trainer

*bad player can produce good trainee* *good player can produce bad trainee*

Figure 9: Sample games of individual trainers and trainees.
All games are played against the same GP opponent. The player of interest is represented by the solid black line and starts on the left hand side of the arena.

### *3.4 Discussion*

The overwhelming dominance of the *straight* move inherent in the Tron domain makes it difficult for most controllers to learn when to turn. Indeed, this characteristic proved to be extremely challenging, and initially we produced hundreds of networks that never learned to turn. The evaluation strategy that we settled on (based on the frequency of moves table) has allowed players to learn effectively. However, we believe that this method works to produce players that turn only when necessary, and cannot result in more varied behaviours such as those illustrated in figures 5d, 9b and 9h. While this precise evaluation strategy is highly domain dependent, the technique may be quite valuable for training in domains where one input tends to swamp others and for learning to generalize human behaviour in more complex domains.

We make several observations about the results we have obtained, speculating on the discrepancies between trainers and trainees and addressing the issues raised at the beginning of section 3.3. How can we explain a trainer that wins 2% of the time, yet produces a trainee that wins 50% of the time (see figure 8b)? The trainee is not being trained on whether it wins or not — in fact the trainee doesn't know if it wins at all. The trainee learns only from a sequence of moves. If the trainer makes nine good moves and then a bad one ends the game, the trainee has still gained from 90% of this experience.

Does our method produce controllers that can play a decent game of Tron? Yes — and one conclusion we can draw from our statistics is that a population of humans can act as effective trainers for a graded population of agents, because there is naturally more variation in behaviour both across an entire population of humans and within a single stochastic human player. It is important for artificially trained players to experience a wide variety of behaviours, otherwise they will not be robust and will only perform well against players with styles similar to those of their trainers.

Were we able to produce a population that approximates the behaviour of its trainers? This is a difficult question to answer. While the correlation between individual GP trainers and trainees based on choice of move is much higher than that for humans, the correlation between win rates of individual trainers and trainees[e] is better for humans. We speculate that the discrepancies may be due to artifacts of the domain and the nature of each type of controller. Features that contribute include: GPs are deterministic players (vs. non-deterministic humans), and GPs share a limited view of their environment, using the same sensors that are employed by the trainee networks. The human players, in contrast, have a global view of the playing arena which is not

[e] against the same opponent

practical for artificial controllers in this context.

Humans often produce different responses when presented with the same situation multiple times. Clearly then, it is not possible for a deterministic controller to model the behaviour of the humans exactly. Further work is exploring adding some measure of non-determinism to the controller. Nonetheless, we propose to take advantage of networks that are able to filter out mistakes that humans make and thus achieve performance superior to that of their trainers — as was the case for 19 of the 58 human trainees.

## 4 The second domain: CEL

CEL (Community of Evolving Learners)[26] is an Internet learning community created for children. It is located on the web[f] and is open to anyone with a Java-enabled browser. Inside CEL, participants engage in multi-player educational games. If not enough humans are logged into the site, then software agents act as artificial players, maintaining an active presence in the system at all times and thereby sustaining the community.

The CEL system was designed as a framework to host experiments focused on learning, in humans and in machines. CEL differs from other Internet learning communities — particularly because it is more accessible, it enforces user anonymity, it is designed for children, it supports real-time multi-user activities and it offers a shareable server that can act as host to others' activities. Its basis is in computer science, not education, so the purpose is not to put forward a new pedagogical example. On the contrary, the goal is to establish a platform that others with research interests in human learning, cognitive science or artificial intelligence can use to define and implement their own studies. The work presented here represents one such study, in which human data collected in the CEL system was used to train software agents to play a simple keyboarding (typing) game.

### *4.1 A brief tour of CEL*

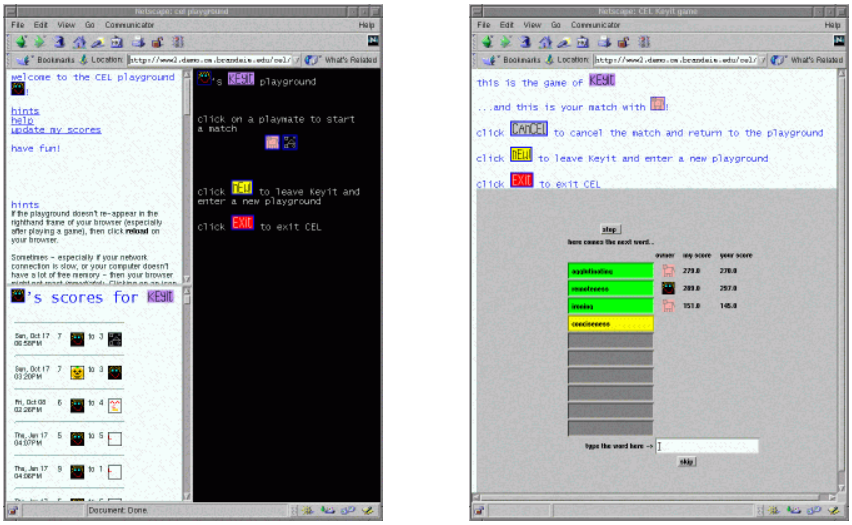
Students log into the CEL web site with an individual user name and password. In order to maintain privacy, the user name is never shown to others; instead, participants are represented inside the system by two-dimensional graphical icons called *IDsigns*, which users create using a pixel editing tool.

After logging in, students are shown a simple menu page containing a list of available activities. Clicking on a game icon selects that activity. Next, users are placed in an open *playground*, a page that contains a matrix filled

[f] http://www.demo.cs.brandeis.edu/cel

with IDsigns belonging to other users who are currently logged into CEL and are playing the same game (figure 10a). These are a user's *playmates*; together they comprise a user's *playgroup*.

By clicking on a playmate's IDsign, a student invites that playmate to join her in a match. The match begins when the browser displays a game page, containing a Java applet that facilitates play. Both players participate according to the particular format of the selected game. When the match is over, each player is returned to his playground and is then free to engage in another match with another (or the same) playmate.




(a) A sample playground. (b) The game of Keyit.

Figure 10: CEL screens.

The work presented here is based on a simple game called Keyit (figure 10b), a two-player activity in which participants are each given ten words to type and are scored based on speed and accuracy. The same set of words is presented to both players, selected from a database containing nearly 35,000 words. Every word in the database is characterized by a vector of seven feature values: word length, keyboarding level[g], Scrabble score[h], number of vowels,

[g]Several standards define an order for introducing keys to students learning typing. We assign a value based on the highest keyboarding level of any of the letters in each word, according to the ordering listed here: `http://www.absurd.org/jb/typodrome/`.

[h]Scrabble is a board game in which players take turns making interconnecting words by

number of consonants and number of 2 and 3-consonant clusters. For each player, a timer begins when she types the first letter of a word and stops when she presses the *Enter* key to terminate the word. Time is measured using the system clock on her computer.

In order to protect young players, there is no chat facility inside CEL. Participants communicate only through the moves of the games they are playing. Most multi-user environments involve some type of natural language communication, even MUD's[i], which generally use a restricted form of English (or other common spoken language). Deploying a believable software agent as a substitute human partner in the CEL environment is therefore a simpler task than in other settings.

### *4.2 Agent control*

Inside CEL, software agents need to exhibit three categories of behaviors:

1. system behavior,
2. playground behavior, and
3. game behavior.

System behavior refers to high-level actions like logging into and out of CEL at particular times of day and selecting different playgrounds. Playground behavior refers to entering and exiting playgrounds and inviting playmates to engage in matches. Game behavior refers to the play within a specific game. A top-level controller for the agent decides which behavior to follow, based on the agent's current state (e.g., residing in a playground or playing a game). Here, we limit our discussion to game behavior, specifically for the game of Keyit.

The basic task is as follows: given a word, characterized by its corresponding set of seven feature values, output the length of time to type the word. In addition to the feature values, we also consider the amount of time that has elapsed since the previous word was typed[j]. The agents are controlled by feed-forward neural networks. The network architecture is shown in figure 11. There are 8 input nodes, corresponding to each of the seven feature values

placing letter tiles on a grid in crossword puzzle fashion. Each letter is assigned a value, according to its frequency of use in American English. Players receive a score for each word they place — the sum of the values for each letter in the word.

[i] Multi-User Domain

[j] This only pertains to words within the same game. The first word in a game has an elapsed time value of 0.

(normalized) plus the elapsed time. The elapsed time is partially normalized to a value between 0 and (close to) 1. There are 3 hidden nodes and one output node, which contains the time to type the input word, in hundredths of a second.

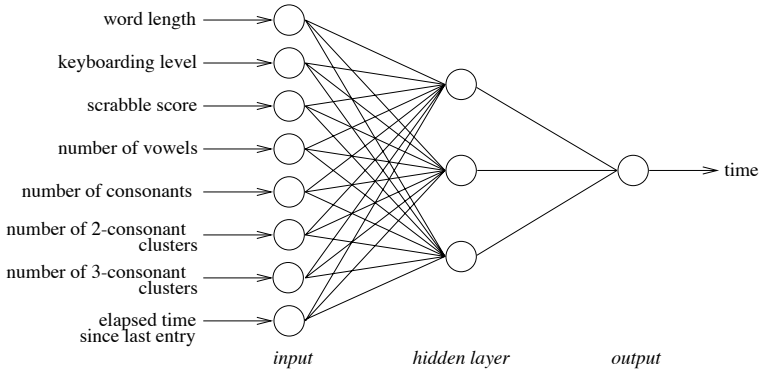



Figure 11: Neural network architecture.

### *4.3 Agent training*

In 1999, a pilot study was conducted in which CEL was used by 44 fourth and fifth grade children at a public primary school[k]. The primary objective of this study was to examine the effectiveness of the CEL mechanism, and so the activities were limited to simple games, one of which was Keyit. The last 19 days of data collection (spread over four months) were used as the basis for training the agents to play Keyit. Note that the humans were learning throughout this period, so the networks were trained to approximate the average performance of each human across the entire time period.

For each child involved in the pilot study, we gathered all the moves from all games of Keyit. A "move" includes a timestamp, the word being typed and the amount of time that the player took to type the word. Then we calculated the time that had elapsed between moves (based on consecutive timestamps) and, along with the seven feature values for each word, created two files — one for training and one for testing — placing moves from alternate games in each file.

We conducted two training experiments. First, we defined a one-to-one correspondence between human trainers and network trainees, with the goal of

[k]All participants had signed parental permission.

producing 44 agents whose behaviors emulate their individual trainers. Second, we employed a many-to-one correspondence between groups of human trainers and network trainees. For the second method, we clustered human trainers into eight groups, based on their similarity in average typing speed. The objective here was to generate a population of agents that could challenge human learners across a range of abilities.

Figure 12 contains data for all the humans involved in the pilot study, plotted in ascending order according to typing speed (in letters/second). We highlight four students: one fourth grade boy (id = 119), one fourth grade girl (id = 98), one fifth grade boy (id = 88) and one fifth grade girl (id = 89). The plot also indicates the groupings of players, used for the many-to-one training scenarios. The players are clustered according to typing speed, in increments of 0.5 letters/sec.

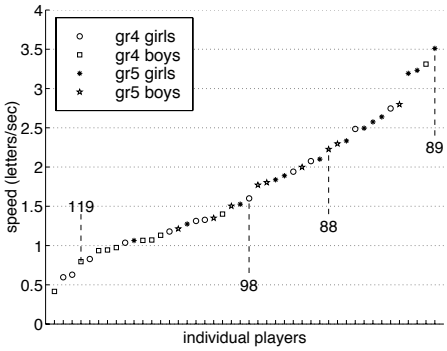



Figure 12: average typing speeds of players, with standard deviation.

As with the Tron networks described in the previous section, we trained the Keyit networks using supervised learning. The networks were presented with a series of moves, and they predicted the trainer's speed for those moves. Based on the accuracy of the trainees' predictions, the network weights were adjusted using backpropagation. The results presented here were obtained with a learning rate of 0.00001. All the networks were trained for 10,000 epochs, but progress generally leveled off after 2500 epochs.

Throughout the training sequence, we kept track of the prediction error for the network — the difference between the typing time predicted by the network and the actual typing time of the training set. We saved one "best" network for each training sequence, corresponding to the set of weights which resulted in the smallest prediction error. After the training sequences were

completed, we evaluated the best networks for each effort by comparing its prediction with the human's data, for both the training set and the (reserved) test set of data.

### *4.4 Results*

We look at the results of the training efforts in several ways. First, we look at the training period and show how the network improved its predictive ability during training. Figure 13 shows the performance of the networks trained for the four sample players (88, 89, 98 and 119). The plots in the top row illustrate the prediction error for the networks. The solid curve plots the error based on the test data set; the dashed curve plots the error based on the training data set. The plots in the bottom row show how the error in typing speed improves over time, when the networks are confronted with the test data set (solid curve) and the training data set (dashed curve).

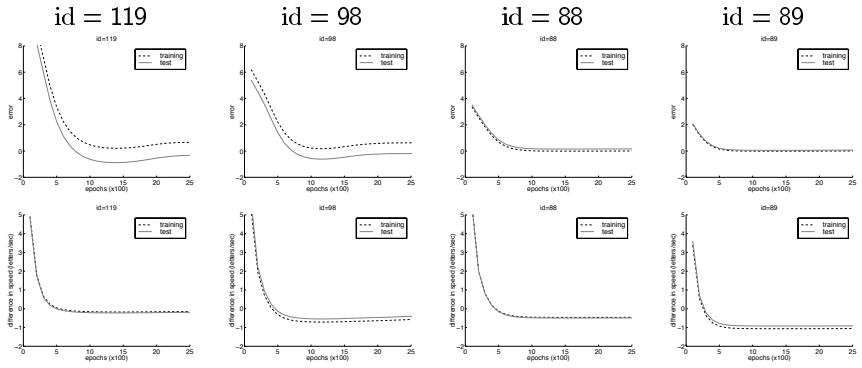
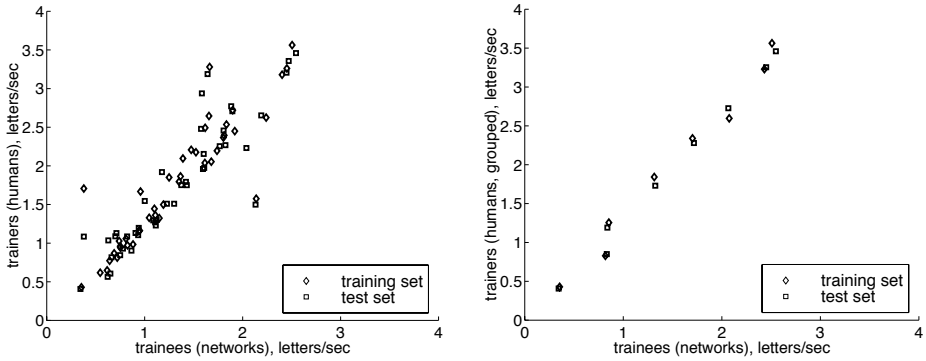



Figure 13: Improvement during training.

The networks learn quite quickly, sometimes within 500 epochs. It is interesting to note that in some cases, as with players 98 and 119, the difference in prediction error between the training and test data sets is relatively marked; however the difference in typing speeds is negligible.

Another way in which we examine the training effort is by studying the correlation between the trainers and the best trainees. Figure 14 plots the typing speed for the trainees (horizontal axes) versus their trainers (vertical axes), for both the test and training data sets, for the one-to-one and many-to-one training efforts. The correlation coefficients are listed in table 2, illustrating the average relationship between trainers and trainees across both populations.

The correlation is much higher for the many-to-one trainees than the one-to-one trainees.



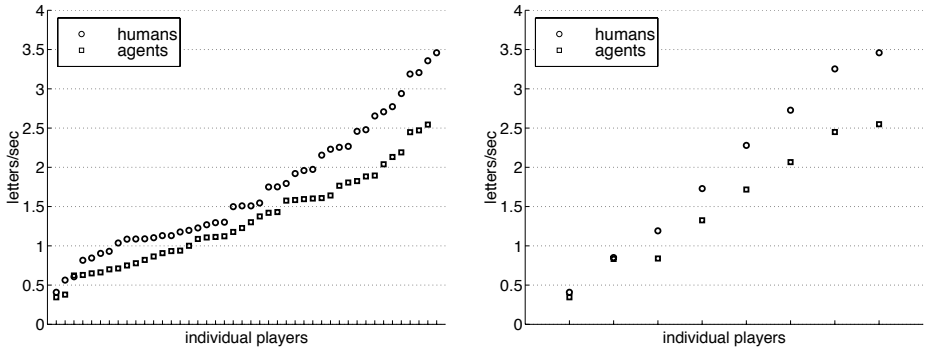

(a) one-to-one (b) many-to-one

Figure 14: Correlation between trainers and best trainees.

Table 2: Correlation coefficients: individual training.

| | training set | test set |
|---|---|---|
| one-to-one | 0.6364 | 0.4204 |
| many-to-one | 0.9910 | 0.9965 |

The final way in which we study the results takes a collective, or many-to-one, approach. Figure 15 compares the average speeds of the human population with those of the agent populations, for both training schemes. The comparison is made by first sorting both populations according to speed and then calculating the correlation coefficients. In the one-to-one case, sorting the trainees re-orders the comparisons that are made when computing the correlation coefficient, and so the correlation is higher. In the many-to-one case, the population-based correlation between trainers and trainees is precisely the same as in the individual case, because the training went so well that sorting the trainees does not change their order and so the two comparisons are equivalent. Note that the average speeds for the agent population were based on data collected during the testing runs only.




(a) one-to-one (b) many-to-one

Figure 15: Correlation between populations of trainers and best trainees.

Table 3: Correlation coefficients: collective training.

| | test set |
|---|---|
| one-to-one | 0.8002 |
| many-to-one | 0.9965 |

## 5 Conclusion

In both domains discussed here, our goal was to approximate the behavior of the human population in a population of software agents. We make several observations from these experiments, first in regard to measuring performance of training efforts and second in regard to organizing data to be used for training.

When measuring the performance of training efforts, it is extremely difficult to emulate exactly the behavior of individual humans, even in the limited domains we have studied here. Making direct comparisons after one-to-one training efforts, between trainee and trainer, does not give a reliable indication of how well the training runs have gone (e.g., figures 8b, 8d, 8f and 14). Instead, statistical evaluations should be made by comparing features of the trainee population with features of the trainer population (e.g, figures 8a, 8c, 8f and 15).

When organizing data to be used for training, the collective approach, rather than the individual approach, produces more robust trainees. Group human trainers who exhibit similar features and pool their input data, then use this collective data set to train a smaller number of trainees. The result

will be fewer, but more experienced, trainees — because they have trained on a wider data set that is the collective experience of a group of similar humans. (e.g., figures 8e, 8f, 14b and 15b).

Our goal is to produce a population of graded agents, using human behaviour as the basis for constructing the graded population, and then to select opponents from this population that are appropriate learning partners for humans at various stages of advancement. The next step with this work is to deploy the agent populations that have been described here and implement selection algorithms that will choose the appropriate learning partners.

Future work involves building agents that can adapt their performance on-line. One method for accomplishing this would be to train an agent using data from the first few games, deploy the agent and then continue to train it further, by incorporating moves from subsequent games of its human trainer.

## 6 Acknowledgements

Special thanks to Pablo Funes, for his lead on the Tron project. Additional thanks to Travis Gephardt, Matthew Hugger and Maccabee Levine for implementation help, and to Tom Banaszewski and Jackie Kagey for supporting the CEL pilot study. This research was funded by the Office of Naval Research under grant N00014-98-1-0435 and by a University of Queensland Postdoctoral Fellowship.

## 7 References

1. CAIP. Privacy code, 1996.
2. W. Teitelman. A display oriented programmer's assistant. *International Journal of Man-Machine Studies*, 11:157–187, 1979.
3. A. Cypher. Eager: Programming repetitive tasks by example. In *Proceedings of CHI'91*, 1991.
4. P. Maes. Agents that reduce work and information overload. *Communications of the ACM*, 37(7):31–40,146, 1994.
5. E. Soloway. How the nintendo generation learns. *Communications of the ACM*, 34(9), 1991.
6. H. Brody. Video games that teach? *Technology Review*, November/December, 1993.
7. T. Malone. Toward a theory of intrinsically motivating instruction. *Cognitive Science*, 4:333–369, 1981.
8. T. Malone. What makes computer games fun? *Byte*, December 1981.
9. M. Minsky. *Society of Mind*. Picador, London, 1987.

10. A. L. Samuel. Some studies in machine learning using the game of checkers. *IBM Journal of Research and Development*, 3:210–229, 1959.
11. C. E. Shannon. Programming a computer for playing chess. *Philosophical Magazine [Series 7]*, 41, 1950.
12. H. J. Berliner and C. Ebeling. Pattern knowledge and search: The suprem architecture. *Artificial Intelligence*, 38(2), 1989.
13. H. J. Berliner. Backgammon computer program beats world champion. *Artificial Intelligence*, 14, 1980.
14. G. Tesauro. Practical issues in temporal difference learning. *Machine Learning*, 8, 1992.
15. J. B. Pollack and A. D. Blair. Co-evolution in the successful learning of backgammon strategy. *Machine Learning*, 32:225–240, 1998.
16. P. J. Angeline and J. B. Pollack. Competitive environments evolve better solutions for complex tasks. In S. Forrest, editor, *Genetic Algorithms: Proceedings of the Fifth International Conference (GA93)*, 1993.
17. R. Axelrod. *The Evolution of Cooperation*. Basic Books, 1984.
18. T. Haynes, R. Wainwright, S. Sen, and D. Schoenefeld. Strongly typed genetic programming in evolving cooperative strategies. In L. Eshelman, editor, *Genetic Algorithms: Proceedings of the Sixth International Conference (ICGA95)*, 1995.
19. C. W. Reynolds. Competition, coevolution and the game of tag. In R. A. Brooks and P. Maes, editors, *Proceedings of the Fourth International Workshop on the Synthesis and Simulation of Living Systems*. MIT Press, 1994.
20. P. Funes, E. Sklar, H. Juillé, and J. B. Pollack. Animal-animat coevolution: Using the animal population as fitness function. In *From Animals to Animats 5: Proceedings of the Fifth International Conference on Simulation of Adaptive Behavior*, 1998.
21. J. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, Cambridge, MA, 1992.
22. D. Pomerleau. *Neural Network Perception for Mobile Robot Guidance*. Kluwer Academic, 1993.
23. G. Wyeth. Training a vision guided robot. *Machine Learning*, 31, 1998.
24. D. Rumelhart, G. Hinton, and R. Williams. Learning representations by back-propagating errors. *Nature*, 323, 1986.
25. G. Hinton. Connectionist learning procedures. *Artificial Intelligence*, 40, 1989.
26. E. Sklar. *CEL: A Framework for Enabling an Internet Learning Community*. PhD thesis, Brandeis University, 2000.