Exploiting Sensor Symmetries in Example-based Training for Intelligent Agents

Bobby D. Bryant
Department of Computer Sciences
The University of Texas at Austin
bdbryant@cs.utexas.edu

Risto Miikkulainen
Department of Computer Sciences
The University of Texas at Austin
risto@cs.utexas.edu

Abstract— **Intelligent agents in games and simulators often operate in environments subject to symmetric transformations that produce new but equally legitimate environments, such as reflections or rotations of maps. That fact suggests two hypotheses of interest for machine-learning approaches to creating intelligent agents for use in such environments. First, that exploiting symmetric transformations can broaden the range of experience made available to the agents during training, and thus result in improved performance at the task for which they are trained. Second, that exploiting symmetric transformations during training can make the agents' response to environments not seen during training measurably more consistent. In this paper the two hypotheses are evaluated experimentally by exploiting sensor symmetries and potential symmetries of the environment while training intelligent agents for a strategy game. The experiments reveal that when a corpus of human-generated training examples is supplemented with artificial examples generated by means of reflections and rotations, improvement is obtained in both task performance and consistency of behavior.**

Keywords: Agents, Multi-Agent Systems, Adaptive Team of Agents, Games, Simulators, *Legion II*, Sensors, Symmetries, Human-generated Examples

I. Introduction

Intelligent agents in games and simulators often operate in geometric environments subject to reflections and rotations. For example, a two dimensional map can be reflected across an explorer agent or rotated about it, providing new and different but still reasonable maps. Similarly, the visible universe can be reflected or rotated on any of the three axes of a robotic construction worker in deep space. A well trained general purpose agent for deployment in such environments should be able to operate equally well in a given environment and its symmetric transformations. In general it is desirable for intelligent agents to exhibit symmetrical behavior as well. That is, if the optimal action in a given environment is to move to the left, then the optimal action in a mirror image of that environment would be to move to the right.

Symmetry of behavior is desirable for two reasons. First, if a correct or optimal move can be defined for a given context, failing to choose the symmetrical move in the symmetrical context will be sub-optimal behavior, and will degrade an agent's overall performance if it ever encounters such a context. Second, if the agent operates in an environment observable by humans, such as a game or a simulator, the humans will expect to see "visibly intelligent" behavior, i.e., they will expect the agent to always do the right thing because it is smart, rather than intermittently doing the right thing because it has been programmed or trained to manage only certain cases.

If an agent's controller operates by mapping sensory inputs onto behavioral responses, the desired symmetries can be identified by analyzing the structure of the agent and its sensors. For example, if the agent and its sensors are both bilaterally symmetrical then it will be desirable for the agent's responses to be bilaterally symmetrical as well. However, if they are not symmetrical – e.g. for a construction robot with a grip on one side and a tool on the other – then its optimal behavior is asymmetrical. Thus the desirable symmetry of behavior depends critically on the symmetry of the agent and its sensors.

When an agent's controller is directly programmed it is a straightforward task to ensure that its behavior observes the desired symmetries. However, when a controller is trained by machine learning there is no guarantee that it will learn symmetrical behavior. Therefore it will be useful to devise machine learning methods that encourage behavioral invariants across relevant symmetries in agents trained by those methods.

This paper reports initial results on solving the symmetry challenge with supervised learning. A controller is trained for agents that operate in an environment that supports multiple symmetries of reflection and rotation. Human-generated examples of appropriate contextual behavior are used for the training, and artificial examples are generated from the human examples to expose the learner to symmetrical contexts and the appropriate symmetrical moves. This training mechanism addresses both of the motivations for symmetrical behavioral invariants, i.e. it improves the agents' task performance and also provides measurable improvements in the symmetry of their behavior with respect to their environment, even in environments not seen during training.

The learning environment and the structure of the agents' sensors and controller are described in the following section, then the training methodology and experimental results are explained in section III. The experimental results are discussed in section IV, along with a look at future directions for the research.

II. The learning environment

The use of artificial examples generated by exploiting symmetries was tested in a game/simulator called *Legion II*, which is a slight modification of the *Legion I* game described in [1]. *Legion II* is a discrete-state strategy game designed

as a test bed for multi-agent learning problems, with legions controlled by artificial neural networks acting as the intelligent agents in the game.

A. The Legion II game/simulator

The *Legion II* game/simulator is played on a map that represents a province of the Roman empire, complete with several cities and a handful of legions for its garrison (figure 1). Gameplay requires the legions to minimize the pillage inflicted on the province by a steady stream of randomly appearing barbarian warbands. The barbarians collect a small amount of pillage each turn they spend in the open countryside, but a great deal each turn they spend in one of the cities.

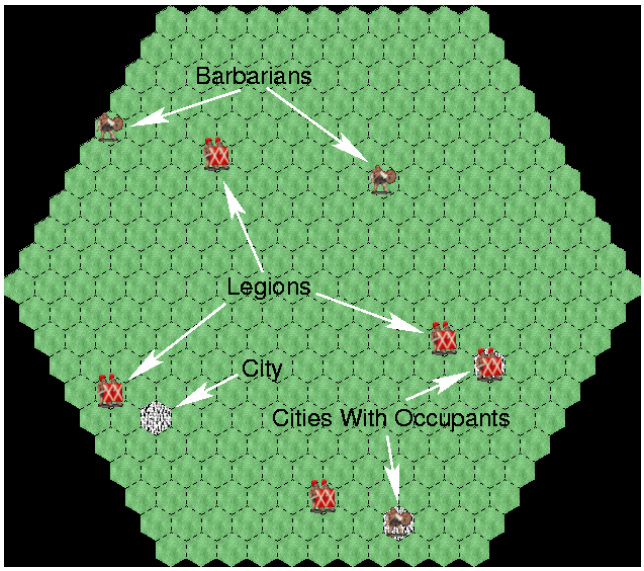

Fig. 1. **The *Legion II* game.** A large hexagonal playing area is tiled with smaller hexagons in order to quantize the positions of the game objects. Legions are shown iconically as close pairs of men ranked behind large rectangular shields, and barbarians as individuals bearing an axe and a smaller round shield. Each icon represents a large body of men, i.e. a legion or a warband. Cities are shown in white, with any occupant superimposed. All non-city hexes are farmland, shown with a mottled pattern. The game is a test bed for multi-agent learning methods, whereby the legions must learn to contest possession of the playing area with the barbarians. (An animation of the *Legion II* game can be viewed at `http://nn.cs.utexas.edu/keyword?ATA`.)

The game is parameterized to provide enough legions to garrison all the cities and have a few left over, which can be used to disperse any warbands they find prowling the countryside. The original purpose of this parameterization was to require the legions to learn an on-line division of labor between garrisoning the cities and patrolling the countryside, in a multi-agent cooperative architecture called an *Adaptive Team of Agents* [1]. The game is used here to test the use of training examples generated from symmetries, because it is a challenging learning task that offers multiple symmetries in its environment.

The *Legion II* map is in the shape of a large hexagon, divided into small hexagonal cells to discretize the placement of game objects such as legions and cities (figure 1). Moves are taken in sequential turns. During a turn each legion makes a move, and then each barbarian makes a move. All moves are atomic, i.e. during a game agent's move it can either elect to remain stationary for that turn or else move into one of the six hexagons of the map tiling adjacent to its current position.

Only one agent, whether legion or barbarian, can occupy any map cell at a time. A legion can bump off a barbarian by moving into its cell as if it were a chess piece; the barbarian is then removed from play. Barbarians cannot bump off legions: they can only hurt the legions by running up the pillage score. Neither legions nor barbarians can move into a cell occupied by one of their own kind, nor can they move off the edge of the map.

A game is started with the legions and cities placed at random positions on the map; the combinatorics allow a vast number of distinct game setups. The barbarians enter play at random unoccupied locations, one per turn. If the roving legions do not eliminate them they will accumulate over time until the map is almost entirely filled with barbarians, costing the province a fortune in goods lost to pillage.

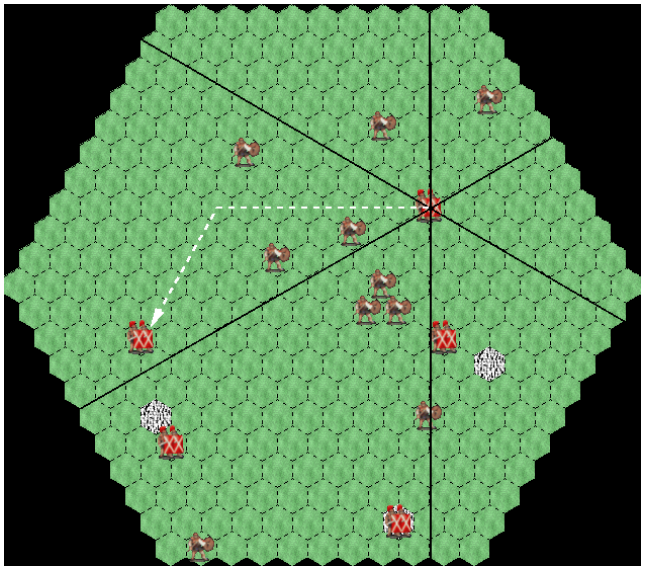

Fig. 2. **A legion's sensor fields.** A legion's sensor array divides the world into six symmetrical "pie slices", centered on the legion itself (black lines). The objects i falling within a slice are detected as the scalar aggregate $\sum_i 1/d_i$, where d is the hexagonal Manhattan distance to the object (white arrow). For any given sensory input the symmetries in the sensor architecture allow a set of six 60° rotations about the legion, plus a reflection of each rotation, for a total of twelve isomorphic sensory views of the world. If a legion makes the optimal move in all circumstances, then a reflection and/or rotation of its sensory inputs produces a corresponding reflection and/or rotation in its choice of moves. This behavioral invariant allows artificial training examples to be constructed from reflections and rotations of human-generated training examples.

Play continues for 200 turns, with the losses to pillage accumulated from turn to turn. At the end of the game the legions' score is the amount of pillage lost to the barbarians, rescaled to the range $[0, 100]$ so that the worst possible score is 100. Lower scores are better for the legions, because they represent less pillage. The learning methods described in this paper allow the legions to learn behaviors that reduce the score

to around 6 when tested on a random game setup never seen during training (i.e. to reduce pillage to about 6% of what the province would suffer if they had sat idle for the entire game).

The barbarians are programmed to follow a simple strategy of approaching cities and fleeing legions, with a slight preference for the approaching. The are not very bright, which suits the needs of the game and perhaps approximates the behavior of barbarians keen on pillage.

B. Agent sensors and controllers

The legions must be trained to acquire appropriate behaviors. They are provided with sensors that divide the map up into six pie slices centered on their own location (figure 2). All the relevant objects i in a pie slice are sensed as a single scalar value, calculated as $\sum_i 1/d_i$. This design provides only a fuzzy, alias-prone sense of what is in each sector of the legion's field of view, but it works well as a threat/opportunity indicator: a few barbarians nearby will be seen as a sensory signal similar to what would be seen of a large group of barbarians further away.

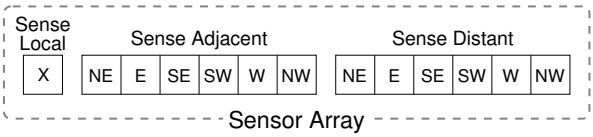

Fig. 3. **A legion's sensor architecture.** Each sensor array for a legion consists of three sub-arrays as shown here. A single-element sub-array (left) detects objects colocated in the map cell that the legion occupies. Two six-element sub-arrays detect objects in the six radial fields of view; one only detects adjacent objects, and the other only detects objects farther away. The legions are equipped with three complete sensor arrays with this structure, one each for detecting cities, barbarians, and other legions. The three 13-element arrays are concatenated to serve as a 39-element input layer for an artificial neural network that controls the legion's behavior (figure 4). Artificial reflections and rotations of a legion's view of the world can be generated on demand by appropriate permutations of the activation values of the sensors in the sub-arrays.

There is a separate sensor array for each type of object in play: cities, barbarians, and other legions. There are additional sensors in each array to provide more detail about what is in the map cells adjacent to the sensing legion, or colocated in the legion's own cell (figure 3). In practice only a city can be in the legion's own cell, but for simplicity the same sensor architecture is used for all three object types.

The scalar sensor values, 39 in all, are fed into a feed-forward neural network with a single hidden layer of ten neurons and an output layer of seven neurons (figure 4). The output neurons are associated with the seven possible actions a legion can take in its turn: remain stationary, or move into one of the six adjacent map cells. This localist *action unit coding* is decoded by selecting the action associated with the output neuron that has the highest activation level after the sensor signals have been propagated through the network.

The *Legion II* sensor architecture allows reflections and rotations of the world about a legion's egocentric viewpoint. The transformations can be represented by permutations of the values in the sensors. For example, a north-south reflection

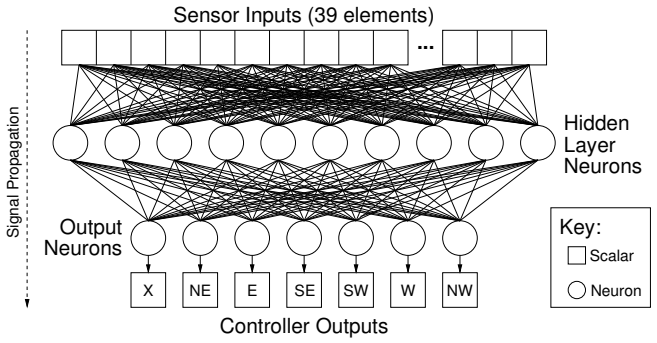

Fig. 4. **A legion's controller network.** During play the values obtained by a legion's sensors are propagated through an artificial neural network to create an activation pattern at the network's output. This pattern is then interpreted as a choice of one of the discrete actions available to the legion. When properly trained, the network serves as the controller for the legion as an intelligent agent.

can be implemented by swapping the northwest (NW) sensor values with the southwest (SW), and the NE with the SE. Similarly, a 60° clockwise rotation can be implemented by moving the sensor values for the eastern (E) sector to the southeastern (SE) sensor, for the SE to the SW, etc., all the way around the legion. The legions' choices of action for a reflected or rotated sensory input can be reflected or rotated by the same sort of swapping. For example, a 60° clockwise rotation would convert the choice of a NE move to an E move. The option to remain stationary is not affected by reflections or rotations: if a legion correctly chooses to remain stationary with a given sensory input, it should also remain stationary for any reflection or rotation of that input.

III. Experimental evaluation

Experiments were designed to test two hypotheses: first, that exploiting symmetric transformations can broaden the range of experience made available to the agents during training, and thus result in improved performance at their task; and second, that exploiting symmetric transformations during training can make the agents' response to environments not seen during training measurably more consistent. These hypotheses were tested by training sets of networks with human-generated examples, with or without supplementary examples created by reflecting and/or rotating them, and then applying appropriate metrics to the trained agents' performance and behavior during runs on a set of test games.

After a summary of the experimental methodology in section III-A, the first hypothesis is examined in section III-B and the second in section III-C.

A. Experimental methodology

Examples of human play were generated by providing *Legion II* with a user interface and playing 12 games, with the game engine recording the sensory input and associated choice of action for each of the 1,000 legion moves during a game. Each game was played from a different randomized starting setup in order to provide a greater diversity of examples.

In an Adaptive Team of Agents all of the agents have identical control policies [1]. This design is implemented in *Legion II* by using the same neural network to control each legion. Such uniform control means that all the examples recorded for the various legions during play can be pooled into a single set for training a controller network.

Artificial examples were created by the training program at run time, by permuting fields in the human-generated examples according to the patterns described in section II-B above. Since the legions in *Legion II* have no distinguished orientation, all the reflections were generated by flipping the sensory input and choice of move from north to south. When both reflections and rotations were used, the N-S reflection was applied to each rotation, to create a full set of twelve distinct training examples from each original.

The four possibilities of using vs. not using reflections and/or rotations define four sets of training examples. The choice between these sets defines four training methods for comparison. The four methods were used to train the standard *Legion II* controller network (figure 4) with backpropagation [2]. Training was repeated with from one to twelve games' worth of examples for each method. Due to the relatively large number of examples available, the learning rate η was set to the relatively low value of 0.001. On-line backpropagation was applied for 20,000 iterations over the training set, to ensure that none of the networks were undertrained, and the presentation order of the examples was reshuffled between each iteration.

After every tenth iteration of backpropagation across the training set the network in training was tested against a validation set, and saved if its performance was better than at any prior test on that set. At the end of the 20,000 iterations the most recently saved network was returned as the output of the training algorithm; this network provides better generalization than the network at the end of the training run, which may suffer from overtraining.

Validation was done by play on a set of actual games rather than by classifying a reserved set of test examples, so all the example moves were available for use in training. The validation set consisted of ten games with randomly generated setup positions and barbarian arrival points; they were reproduced as needed by saving the internal state of a random number generator at the start of training and restoring it each time it was necessary to re-create the validation set. Strict accounting on the number of random numbers consumed during play ensured that the same validation set was created each time.

Each differently parameterized training regime – method $\times$ number of games' examples used – was repeated 31 times with a different seed for the random number generator each time, producing a set of 31 networks trained by each parameterization. The seed controlled the randomization of the network's initial weights and generation of the validation set for that run. The 31 independent runs satisfy the requirement of a sample size of at least 30 when using parametric statistical significance tests [3], plus one extra so that there is always a clearly defined median performer if ever a single run needs to be singled out as "typical" for plotting or analysis.

After training, each network was tested by play on set of 31 test games, created randomly like the validation sets, but using a different seed to ensure independence from them. Unlike the validation games, the same 31 test games were used to evaluate every network. The test score for a training run was defined as the average score its network obtained on those 31 games. Thus there were 31 independent training runs for each parameterization, and the network produced by each training run was tested on a constant set of 31 games. The results of these tests are presented in the following sections.

B. Effect on performance

The first experiment illustrates the effect of adding artificially generated training examples on the performance of the controller networks. Networks were trained by each method on from one to twelve games' worth of examples. As described in section III-A, each game provided 1,000 human-generated examples, and the reflections and rotations greatly increased this number.

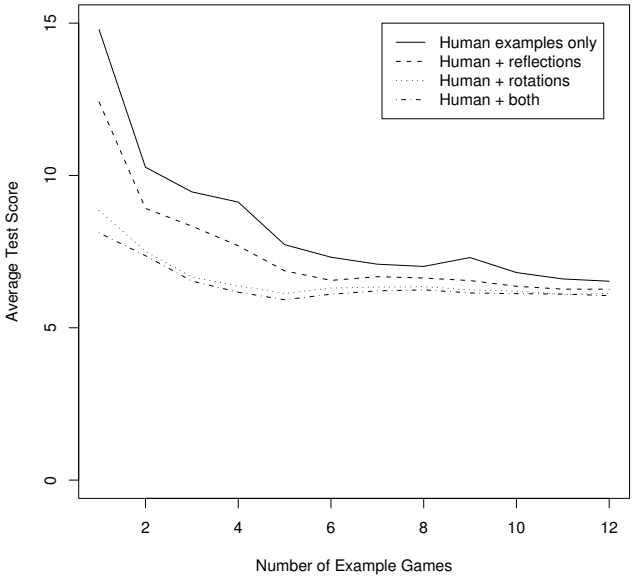

Fig. 5. **Effect of generated examples on performance.** Lines show the average test score for 31 runs of each method vs. the number of example games used for training. (Lower scores are better.) Each game provided 1,000 human-generated examples; reflections increased the number of examples to 2,000 per game, rotations to 6,000, and both together to 12,000. All three symmetry-exploiting methods provided significant improvement over the base method throughout the range of available examples, albeit by a diminishing amount as more human examples were made available.

The results of the experiment, summarized in figure 5, show that an increase in the number of example games generally improved learning when the human-generated examples alone were used for training, although with decreasing returns as more games were added. The three methods using the artificially generated examples improved learning over the

use of human examples alone, regardless of the number of games used; each of the three provided statistically significant improvement at the 95% confidence level everywhere. The improvement was very substantial when only a few example games were available, and the best performance obtained anywhere was when both reflections and rotations were used with only five games' worth of examples.

Rotations alone provided almost as much improvement as reflections and rotations together, and at only half the training time, since it only increased the number of exmples per game to 6,000 rather than 12,000. Thus in some circumstances using rotations alone may be an optimal trade-off between performance and training time. Reflections alone increased training only to 2000 examples per game, $1/3$ of what rotations alone provided, but with substantially less improvement in performance when fewer than six example games were available.

It is worthwhile to understand how much of the improved performance resulted from the increased number of training examples provided by the reflections and rotations, vs. how much resulted from the fact that the additional examples were reflections and rotations *per se*. A second experiment examined this distinction by normalizing the number of examples used by each method. For example, when a single example game was used in the first experiment, the human-example-only method had access to 1,000 examples, but the method using both reflections and rotations had access to $12 \times 1{,}000$ examples. For this second experiment the various methods were only allowed access to the same number of examples, regardless of how many could be created by reflections and rotations.

It was also necessary to control for the structural variety of the examples. Such variety arises from the fact that each training game is played with a different random set-up – most importantly, with randomized locations for the cities. In some games the cities are scattered, while in other games they are placed near one another. This sort of variety is very beneficial to generalization: the games in the test set may not be similar to any of the individual games in the human-generated training set, but agents exposed to a greater variety of set-ups during training learn to manage previously unseen situations better. Thus if the training example count is normalized by using the 12,000 human-generated examples from the twelve example games, to be compared against training with the 12,000 examples generated by applying reflections and rotations to the 1,000 human-generated examples from a single example game, the latter method will have less structural variety in its training examples, and its generalization will suffer.

So the second experiment controlled for both count and structural variety by selecting examples at random, without replacement, from the full set of 12,000 human-generated examples available for use. When the method of using human-generated examples alone selected n examples at random, the method using reflections selected $n/2$ examples and doubled the count by reflecting, the method using rotations selected $\lfloor n/6 \rfloor$ examples and multiplied the count by six by rotating, and the method using both reflections and rotations selected $\lfloor n/12 \rfloor$ examples and multiplied the count by twelve by reflecting and rotating. Since each method drew its randomly selected examples from the full set of the 12,000 available human-generated examples, each sampled the full structural variety of the examples. The divisions equalized the counts, to within rounding errors.

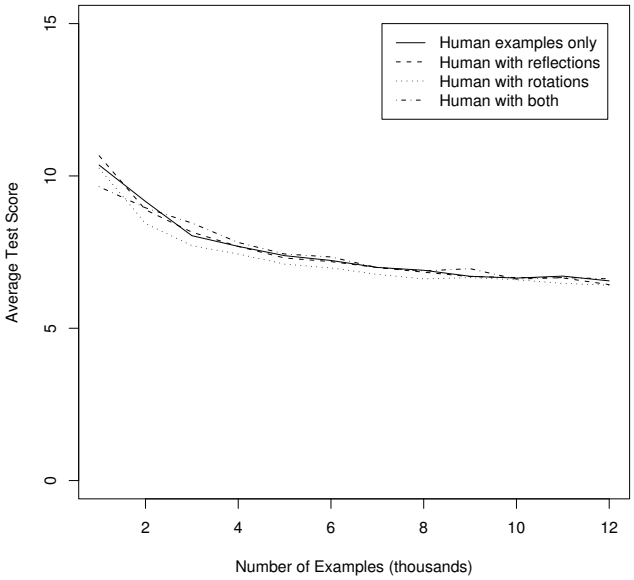

Fig. 6. **Effect of reflections and rotations on performance.** Lines show the average test score for 31 runs of each method vs. the number of examples used for training, when the examples used by the four methods were controlled for count and structural variety. The tight clustering of the performance curves shows that most of the improvements obtained by using reflections and rotations in the the first experiment (figure 5) were the result of the increased number of training examples they provided, rather than being the result of the use of reflections and rotations *per se*.

The results of the experiment, shown in figure 6, show little difference in the performance of the four methods when their training examples are controlled for count and structural variety. Thus most of the improvements obtained in the first experiment were the result of the increased number of training examples generated by the reflections and rotations, rather than by the fact that the additional examples were reflections or rotations *per se*.

C. Effect on behavioral consistency

Further experiments reveal the effect of artificially generated examples on the detailed behavior of the legions. As described in section I, a perfectly trained legion will show behavior that is invariant with respect to reflections and rotations. That is, if its sensory view of the world is reflected and/or rotated, then its response will be reflected and/or rotated the same way.

Although legion controllers trained by machine learning techniques are not guaranteed to provide perfect play, training them with reflected and/or rotated examples should make them behave more consistently with respect to reflections and rotations of their sensory input. This consistency can be measured

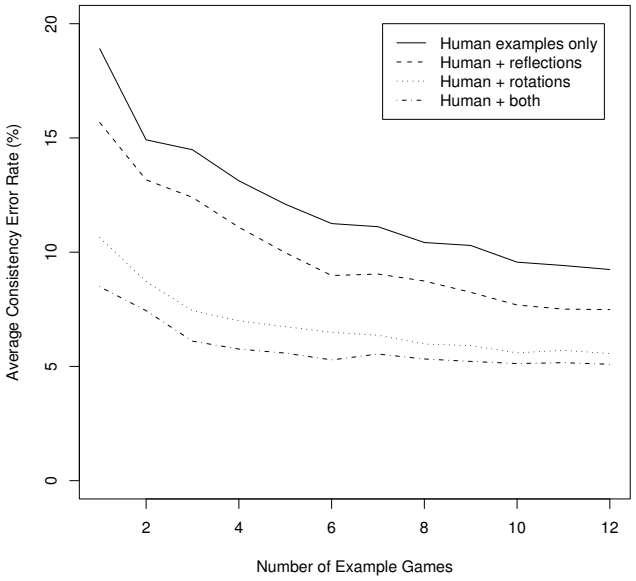

Fig. 7. **Effect of generated examples on consistency.** Lines show the average consistency error rates for 31 runs of each method, vs. the number of example games used for training. (Lower rates are better.) All three symmetry-exploiting methods provided a significant improvement in consistency throughout the range of available examples.

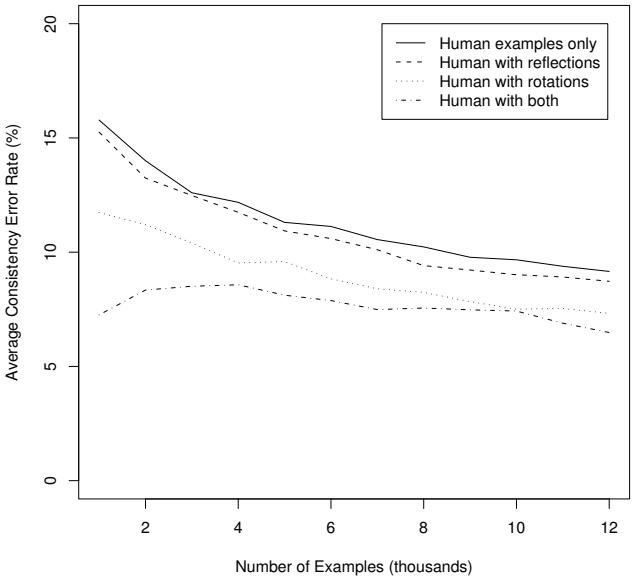

Fig. 8. **Effect of reflections and rotations on consistency.** Lines show the average consistency error rates for 31 runs of each method vs. the number of examples used for training, when the examples used by the four methods were controlled for count and structural variety. The substantial gaps between the lines show that much of the improvement obtained by using reflections and rotations in the the third experiment (figure 7) were the result of the use of reflections and rotations *per se*, rather than merely being a result of the increased number of training examples they provided.

when playing against the test set by generating reflections and rotations of the sensory patterns actually encountered during the test games, and making a side test of how the legions respond to those patterns. These responses are discarded after testing, so that they have no effect on play. For each move in a test game a count is made of how many of the twelve possible reflections and rotations result in a move that does not conform to the desired behavioral invariant. Each such failure is counted as a *consistency error*, and at the end of the test a consistency error rate can be calculated.

Since the perfect move for a legion is not actually known, the consistency errors are counted by deviation from a majority vote. That is, for each reflection and/or rotation of a sensory input, a move is obtained from the network and then un-reflected and/or un-rotated to produce an "absolute" move. The 12 absolute moves are counted as votes, and the winner of the vote is treated as the "correct" move for the current game state [4]. The reflections and rotations that do not produce a move that corresponds to the same reflection or rotation of the "correct" move are counted as consistency errors.

All of the networks produced by the performance experiments described in section III-B were tested to examine the effect of the various training regimes on behavioral consistency. The results, summarized in figure 7, show that the three methods using reflections and rotations reduce consistency errors substantially in comparison to the base method of using only the human-generated examples, regardless of how many example games are used. In every case the improvements were statistically significant at the 95% confidence level. The best consistency was obtained when both reflections and rotations were used, and as with the performance experiment (figure 5), a local minimum was obtained when relatively few training games were used (six in this case). The consistency error rate for this optimal method is approximately flat thereafter.

Again, it is worthwhile to understand how much of the reduced consistency error rate resulted from the increased number of training examples provided by the reflections and rotations, vs. how much resulted from the fact that the additional examples were reflections and rotations *per se*. Thus the networks from the normalization experiment were also tested for behavioral consistency. The results, shown in figure 8, show the familiar trend of improvement as the number of training examples increases, but also show very substantial differences between the four methods, even when their training examples are controlled for count and structural variety. Thus much of the improvement in the consistency error rate in the uncontrolled experiment (figure 7) can be attributed to the fact that the generated examples used reflections and/or rotations *per se*, rather than simply resulting from the increased number of training examples.

IV. Discussion and Future Work

The experiments show that training intelligent agents for games and simulators can benefit from the extra examples artificially generated from reflections and rotations of available human-generated examples. In accord with the hypotheses

stated in section III, the technique results in agents that score better in the game and behave more consistently.

The improved performance scores were shown to result primarily from the increased number of training examples provided by the reflections and rotations, but the improved behavioral consistency resulted largely from the fact that those examples were reflections and rotations *per se*. In principle, reflections and rotations *per se* could improve performance scores as well, by providing a more systematic coverage of the input space. However, in *Legion II*, the base method already provided over 80% consistency with respect to reflections and rotations when only a single game's examples were used for training (figure 7). The agents quickly learned the symmetries necessary for the situations that had the most impact on their game scores. Further improvements in behavioral consistency provided polish, but had no substantial impact on the scores. In other domains the base coverage may be more idiosyncratic, so that reflections and rotations *per se* would significantly improve performance.

The combination of both reflections and rotations provided the best results throughout. That method provided the best performance and consistency when relatively few example games were made available for training, five and six games respectively. This is a very promising result, because it suggests that good training can be obtained without excessive human effort at generating examples. Rapid learning from relatively few examples will be important for training agents in a Machine Learning Game [5], where a player trains game agents by example at run time, and for simulations where agents must be re-trained to adapt to changed environments, doctrines, or opponent strategies. Future work will thus investigate whether rapid learning from relatively few examples is seen in other applications, and also whether a greatly increased number of human-generated examples will ever converge to the same optimum.

The number of examples that can be generated from symmetries depends critically on the sensor geometry of the agent being trained. The number of radial sensors may vary with the application and implementation, providing a greater or lesser number of rotations. However, if radial sensors do not all encompass equal arcs then rotations may not be possible at all. For example, the agents in the NERO video game [5] also use "pie slice" sensors, but with narrower arcs to the front than to the rear, in order to improve their frontal resolution. There is therefore no suitable invariant for the rotation of the NERO agents' sensors.

However, the NERO agents, and probably most mechanical robots as well, have a bilateral symmetry that allows applying the behavioral invariant for reflections across their longitudinal axis. The results presented in this paper show that artificially generated examples provide significant training benefits even when only reflections are used, especially when relatively few human-generated examples are available (figures 5 and 7). Thus the methods examined here should prove useful even in situations with far more restrictive symmetries than in the *Legion II* game. On the other hand, agents operating in three-dimensional environments, such as under water or in outer space, may have a greater number of symmetries to be exploited, offering even greater advantage for these methods. Future work will also investigate the effect of exploiting symmetries in boardgames with symmetrical boards, such as *Go*, where an external player-agent manipulates passive playing pieces on the board.

Supervised learning is not always the best way to train agents for environments such as *Legion II*. Work in progress shows that training with neuroevolution ([6], [7], [8], [5]), using on-line gameplay for fitness evaluations, can produce controller networks that perform somewhat better than those produced by backpropagation in the experiments reported here. However, evolutionary results can sometimes be improved by combining evolution with supervised learning. Thus an obvious avenue of future work is to use examples artificially generated from sensor symmetries with methods such as Baldwinian or Lamarckian evolution ([9], [10]) in order to improve performance and behavioral consistency, the way they benefited ordinary backpropagation in the experiments reported in this paper.

V. Conclusions

Intelligent agents with sense-response controllers often have symmetries in their sensor architecture, and it is then possible to define behavioral invariants that would be observed across those symmetries by perfectly trained agents. This observation suggests that symmetrical invariants of sense-response behavior can be exploited for training the agents, making their responses more symmetrical and effective. This paper shows that both types of improvement are obtained in a game-like test environment, and suggests that further attempts to exploit sensor symmetries may provide similar benefits in other environments and with other learning methods.

Acknowledgments

This research was supported in part by the Digital Media Collaboratory at the IC^2 Institute at the University of Texas at Austin. The images used in *Legion II*'s animated display are derived from graphics supplied with the game *Freeciv*, `http://www.freeciv.org/`.

References

[1] B. D. Bryant and R. Miikkulainen, "Neuroevolution for adaptive teams," in *Proceeedings of the 2003 Congress on Evolutionary Computation (CEC 2003)*, vol. 3. Piscataway, NJ: IEEE, 2003, pp. 2194–2201. [Online]. Available: http://www.cse.unr.edu/~bdbryant/papers/bryant-2003-cec.pdf

[2] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning internal representations by error propagation," in *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Volume 1: Foundations*, D. E. Rumelhart and J. L. McClelland, Eds. Cambridge, MA: MIT Press, 1986, pp. 318–362.

[3] R. R. Pagano, *Understanding Statistics in the Behavioral Sciences*, 2nd ed. St. Paul, MN: West Publishing, 1986.

[4] B. D. Bryant, "Virtual bagging for an evolved agent controller," 2006, manuscript in preparation.

[5] K. O. Stanley, B. D. Bryant, and R. Miikkulainen, "Real-time neuroevolution in the NERO video game," *IEEE Transactions on Evolutionary Computation*, vol. 9, no. 6, pp. 653–668, 2005. [Online]. Available: http://www.cse.unr.edu/~bdbryant/papers/stanley-2005-tec.pdf

[6] J. Branke, "Evolutionary algorithms for neural network design and training," in *Proceedings 1st Nordic Workshop on Genetic Algorithms and Its Applications*, J. T. Alander, Ed. Vaasa, Finland: University of Vaasa Press, 1995, pp. 145 – 163. [Online]. Available: http://citeseer.nj.nec.com/branke95evolutionary.html

[7] X. Yao, "Evolving artificial neural networks," *Proceedings of the IEEE*, vol. 87, no. 9, pp. 1423–1447, 1999. [Online]. Available: ftp://www.cs.adfa.edu.au/pub/xin/yao_ie3proc_online.ps.gz

[8] F. Gomez, "Robust non-linear control through neuroevolution," Ph.D. dissertation, Department of Computer Sciences, The University of Texas at Austin, 2003.

[9] R. K. Belew and M. Mitchell, Eds., *Adaptive Individuals in Evolving Populations: Models and Algorithms*. Reading, MA: Addison-Wesley, 1996. [Online]. Available: http://www.santafe.edu/sfi/publications/Bookinforev/ipep.html

[10] D. Whitley, V. S. Gordon, and K. Mathias, "Lamarckian evolution, the Baldwin effect and function optimization," in *Proceedings of the International Conference on Evolutionary Computation*, Y. Davidor, H.-P. Schwefel, and R. Maenner, Eds., vol. 866, Jerusalem, Israel, October 1994.