Assisting Machine Learning Through Shaping, Advice and Examples

Igor V. Karpov, Vinod K. Valsalam and Risto Miikkulainen
Department of Computer Sciences
The University of Texas at Austin
Austin, TX 78712, USA
{ikarpov,vkv,risto}@cs.utexas.edu

Abstract

Many different methods for combining human expertise with machine learning in general, and evolutionary computation in particular, are possible. Which of these methods work best, and do they outperform human design and machine design alone? In order to answer this question, a human-subject experiment for comparing human-assisted machine learning methods was conducted. Three different approaches, i.e. advice, shaping, and demonstration, were employed to assist a powerful machine learning technique (neuroevolution) on a collection of agent training tasks, and contrasted with both a completely manual approach (scripting) and a completely hands-off one (neuroevolution alone). The results show that, (1) human-assisted evolution outperforms a manual scripting approach, (2) unassisted evolution performs consistently well across domains, and (3) different methods of assisting neuroevolution outperform unassisted evolution on different tasks. If done right, human-assisted neuroevolution can therefore be a powerful technique for constructing intelligent agents.

1 Introduction

When building autonomous agents with complex behavior, such as non-player characters for a video game, or a robotic soccer team, machine learning techniques can be highly useful. For example, evolution of artificial neural networks with augmenting topologies (NEAT) has been demonstrated to be effective in the NERO machine-learning video game, where human players compete to design the most effective policy for agents in a complex virtual environment [Stanley *et al.*, 2005].

However, human domain experts often lack sufficient experience in machine learning techniques to translate their knowledge into model parameters and training processes. Effective and natural ways to combine their domain expertise with machine learning are needed; such techniques would enable experts and non-experts alike to direct machine learning with their knowledge, thus speeding up the process of creating effective and interesting behaviors for autonomous agents.

Many candidate approaches to this problem of *human-assisted machine learning* have been proposed [Knox and Stone, 2009; Nicolescu and Mataric, 2002; Abbeel and Ng, 2004; Kuhlmann *et al.*, 2004; Torrey *et al.*, 2006]. In order to compare and contrast their advantages and disadvantages, this article presents a human subject experiment evaluating the relative merit of three possible approaches: giving advice in terms of rules, demonstrating the desired behavior through examples, and shaping machine learning through a sequence of gradually more challenging tasks.

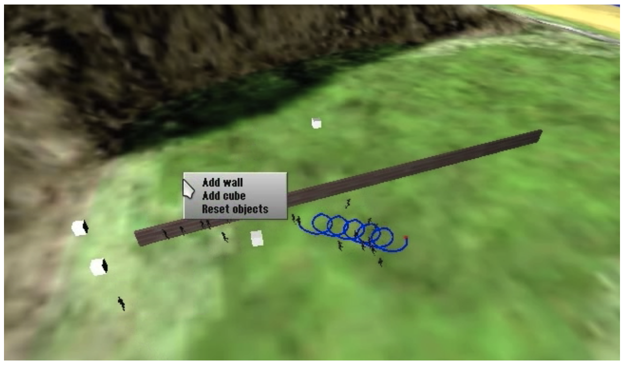

Figure 1: A screenshot of the 3D environment with a human user attempting to solve one of the test tasks using the shaping-assisted neuroevolution method.

In the experiment, the subjects took on the role of agent behavior engineers. Three different behavior design tasks commonly solved as part of the NERO machine-learning game were implemented in OpenNERO[1], an open-source platform for AI research and education (Figure 1). These tasks were a simple navigation around an obstacle (basic navigation), catching a fast moving obstacle (dynamic navigation), and going to a number of targets in sequence (sequential behavior). The experiment held the machine learning algorithm constant—it used the NeuroEvolution of Augmenting Topologies (NEAT) method [Stanley and Miikkulainen, 2002]—and varied the method that the human subjects used to assist neuroevolution in developing the desired policy. In addition to advice, examples, and shaping, neuroevolution

[1]opennero.googlecode.com

alone and a fully manual approach, scripting, were tested as controls.

Each subject was assigned one of the three methods of assisting machine learning. They were then asked to solve each of the three tasks twice, once using the scripting method, and once using their assigned human-assisted neuroevolution method. All the interactions of the subjects with the system were logged via screen capture video and logs created by OpenNERO.

In all tasks, human-assisted and unassisted neuroevolution were more effective than manual scripting. Unassisted evolution was consistently good across all tasks; However, a different human-assisted method outperformed it in different tasks, revealing interesting differences and opportunities for them:

Overall, human-assisted machine learning turned out to be a powerful approach, given that the right method can be chosen for the task. The rest of this article is organized as follows. Section 2 describes related work in combining human design and machine learning. Section 3 provides the details of the behavior design tasks created for the experiments, the architecture of the agents' sensors and actuators within the virtual environment, and the three alternative approaches to assisting neuroevolution with human knowledge. Section 4 describes how the experiments were conducted, and section 5 presents the results. Section 6 draws general conclusions about the results and outlines future work in leveraging human expertise with machine learning.

2 Related Work

Almost any machine learning technique applied to autonomous agent behavior can be considered human-assisted due to the time spent by the designers picking features and representations, tuning parameters, selecting a termination condition, or otherwise affecting the search bias. However, methods have been proposed that are specifically designed to be useful to domain experts without detailed access to the underlying machine learning algorithm.

One possibility is shaping the learning process via human evaluative feedback. For example, the TAMER system combines human reinforcement with MDP learning in order to speed up the learning process and make it more sample-efficient [Knox and Stone, 2010]. Similarly, fitness shaping methods can be used with evolutionary computation as demonstrated with neuroevolution and the NERO machine learning game [Stanley *et al.*, 2005].

Another approach involves formulating knowledge about the domain as rules, and providing these rules as a useful addition to the learning algorithm. An early example of this type of approach includes the work by Maclin and Shavlik, where rules were used to modify a value function approximated with an artificial neural network [Maclin and Shavlik, 1996]. More recently, modern natural language processing techniques have been recruited in order to free the human user from the need to use a specially crafted formal language [Kuhlmann *et al.*, 2004].

A third approach to assisting learning agents involves demonstration of expert performance in the domain. For example, imitation learning has been applied to learning driving policies for virtual race cars [Cardamone *et al.*, 2009]. A variant of the demonstration-based approaches is apprenticeship learning through inverse reinforcement, where the expert behavior is used to estimate the reward function which is then used for learning in conjunction with a model of the environment [Abbeel and Ng, 2004; Ramachandran and Amir, 2007].

The power of such approaches has been demonstrated many times. For example, apprenticeship learning was used to successfully learn autonomous control for aerobatic helicopters [Coates *et al.*, 2009]. Some attempts to concentrate the study on the human component of human-assisted machine learning systems are being made [Zang *et al.*, 2010b; Zang *et al.*, 2010a], however, much more work needs to be done in order to find out how these different methods of providing assistance to machine learning methods compare against each other, and against manual policy design methods.

3 System Description

The experiments described in this article were performed using three *behavior design tasks* implemented within the machine learning video game OpenNERO (Section 3.1). The tasks involved training a team of virtual agents (Section 3.2) using one of three approaches to human-assisted neuroevolution (Section 3.4) and a manual scripting approach.

3.1 Behavior Design Tasks

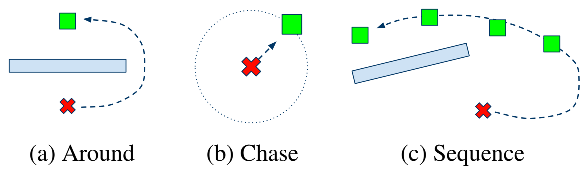

Figure 2: The three tasks used during the experiments. The blue bars represent walls, the red crosses represent the spawning location of the agents, and the green boxes represent the targets that the agents were required to reach in order to complete the task.

In order to explore the breadth of possible tasks that might face a designer of game bot behavior, three tasks were created for the experiment (Figure 2).

One task commonly encountered by game bots is the "Around" task, where the bots have to navigate to a target around some level geometry. In order to test this scenario, the subjects were presented with an obstacle avoidance task (Figure 2a). The fitness on the task is defined as 1.0 or greater if the agent starting at the origin reaches the goal and collects the cube positioned there.

The second task tested in the experiment is the "Chase" task (Figure 2b). The task consists of a target moving along a predefined trajectory (in this case, a circle). The goal of the agent is to reach the moving target and to collect the 1.0 fitness by doing so. Another part of the fitness is normalized to range between 0.0 (agent ending far away from the target) and 0.1 (agent ending close to the target).

The third task tested in the experiment is the "Sequence" task (Figure 2c). The task consists of four resources positioned behind two walls. It is possible for an agent to move around the walls and collect all four, but not with a lot of time to spare. The agent receives a $+1$ towards its fitness for reaching a resource, and a fraction of 0.1 for getting close to one.

3.2 Agent Configuration

Each agent can sense the virtual environment through a set of egocentric sensors and act in the environment using a set of actuators. The sensors consist of eight target cube sensors and four wall sensors. All observations were normalized to lie within the range $[0, 1]$.

The target sensors' value increases linearly with relative proximity of the cube and with angular alignment of the sensor axis with the ray cast towards the location of the cube. The value is cumulative for all cubes in the environment. The cubes disappear from agent's view after they are visited once. The wall sensors return binary observations that report whether or not a ray cast in the direction of the sensor intersects a wall in the environment.

The agents have two continuous action outputs. The actuator a_0 sets the speed of forward motion: it stops the agent when its value is set to 0, and moves the agent at the maximum allowed speed as the value approaches -1 or 1. The actuator a_1 sets the turn rate, ranging from -1 (maximum left turn) to 1 (maximum right turn).

The fitness is defined as $+1.0$ for each target reached, and a fraction of 0.1 for approaching the target faster, for getting closer to a target, or for closely following the example trace.

3.3 Manual Design Through Scripting

One possible approach to designing a complex behavior (and the one usually used in game industry) is to simply write a deterministic program to execute it. Thus each subject in the experiment was asked to solve the tasks using a custom scripting language that consists of a collection of if-statements that determine the actions of the agent given its current inputs.

A simple procedural language based on KBANN (Knowledge-Based Artificial Neural Networks; [Maclin and Shavlik, 1996]) and in earlier work on human-assisted neuroevolution [Yong *et al.*, 2006] was implemented in OpenNERO. The language consists of simple if-then-else constructs, where the if-statement conditions are based on binary comparisons of sensor values and then- and else-clauses can assign constant values to output actuators or to temporary register values. The script is processed using a lexical analyzer and parsed using a parser generated from a set of context-free grammar rules.

Using KBANN, the scripts written in this language are compiled into neural networks that implement the desired behavior. In the fully manual scripting case, these networks are used as the sole controllers. However, these networks can also be used as pieces of advice to the neuroevolution, as described next.

3.4 Human-Assisted Neuroevolution

Neuroevolution provides a suitable testbed to compare different ways of integrating human knowledge because it is capable of supporting many of them. For the experiment presented in this article, the subjects were asked to train populations of NEAT agents to complete a number of tasks using one of four approaches defined below.

Advice

The scripting language used in manual design can also be used to formulate pieces of advice, instead of the entire behavior at once. Such advice can then be converted into small neural networks, and spliced into the networks of the actively evolving population. It is easy for NEAT to incorporate such advice because it adds nodes and connections to evolving networks already as part of evolution. If the advice network provides a useful modification to the individual's behavior, it will be incorporated into subsequent generations, speeding up learning. If the piece of advice is not beneficial, it will be selected against or even reused for something else later in evolution.

Examples

A second way to assist evolution of agents is to provide them with a trace of an example behavior. The subjects were asked to provide such a trace by controlling an agent from a third-person perspective via keyboard commands.

In order to match the agent behavior with examples of different length and complexity, a segment-based approach was developed. A trace is recorded and stored as a sequence of agent positions. It is then broken up into segments 10 steps long. Each neural network is first evaluated on how well it matches the example, and then it is trained to match it better using backpropagation.

During evaluation, the network is allowed to move the agent for 10 steps starting at the beginning point of the first segment. If its path deviates too much from the example, the network is trained with backpropagation on this segment. If the network's path follows the example segment to within a small deviation, the next segment of the path is evaluated, until all segments have been processed. The network's fitness is a fraction of segments successfully matched.

During backpropagation training, the output actions that would have generated the example path are used as targets, and the input activations resulting from the agent's location as the input. Each step through the segment generates one backpropagation input/output pair; the network is trained on each pair once, backpropagating the error through the network topology until all weights leading from the inputs are updated. The backpropagation changes are then encoded back into the genetic representation of the network, and are thus inherited by its offspring.

Shaping via Environment

A third way to assist evolution of agents is to shape their behavior by modifying the task gradually. The tasks were instrumented with a graphical user interface with which the subjects could modify the environment and task parameters, thereby guiding the process of evolution. The idea is that the engineer will be able to find the appropriate *task decomposition* using such a method: After a simpler version of the task is solved, a more complicated version of the task may become easier for neuroevolution to master.

Several different kinds of shaping approaches were possible. In free environment modification, the user adds or removes targets and walls at any location on the field. The user can also scale and rotate these objects, as well as move them from one location to another. In restricted environment modification, specific keys on the keyboard are used to add, reset or remove the objects that were part of the original task. In parametrized environment modification, the subjects are able to modify the speed and trajectory of the moving target in the dynamic target domain. The subjects did indeed use each of these in the process of attempting to discover a useful shaping strategy.

4 Experiments

In order to compare the different methods of assisting evolution in constructing agent behavior, a human subject study was conducted. Participants in the study took on the roles of engineers attempting to create agents with a particular behavior in order to solve three machine-learning game tasks defined below.

The interactions with the system were recorded and analyzed to extract the success rate in solving the tasks using the different methods, how long it took to solve them, and how many times they had to restart before reaching a solution.

Additionally, in order to compare the performance of the human-assisted evolution and human manual scripting to evolution alone, 30 runs of evolution were performed on each of the three tasks.

Sixteen subjects were drawn from a freshman research course. The subjects had experience using neuroevolution in the context of a machine-learning game, and had a lecture describing how the method works. Each subject was introduced to the tasks and approaches they were going to use by means of a tutorial and a demonstration of the user interface. Each subject was randomly assigned a sequence of tasks and a pair of methods. They were then asked to use the pair of methods to solve the tasks in order. Every subject was asked to solve their task with the scripting method as one of the two. The other method was selected from one of shaping, example or advice. Each subject would thus use scripting and another method to solve all three tasks they were assigned.

Each subject was allowed two hours total to work on the tasks. Subjects were not allowed to work longer than 30 minutes on each task and method combination. Each subject could attain at most one successful solution for each of the six task/method combinations they were assigned.

All subjects were briefed before the experiment, describing the timeline of the experiment, the methods they were allowed to use, the user interface they were using for the experiment, the tasks they were solving, and the goal they had to achieve within each task.

All subjects filled out a survey immediately after the experiment rating the quality of their solutions, the quality of the approaches they used, how many times they had to restart on each task/method combination, and whether or not they understood the task.

When running neuroevolution, subjects had the choice of running it in "visible" mode that shows every action of every agent, or in "headless" mode, that runs evolution in the background and can make progress four to five times faster. Running neuroevolution alone was done using the "headless" mode.

5 Results

All results presented in this section exclude the first task/method combination for each subject. This was done because leave-one-out analysis shows an acclimatization effect of the subjects where they improve their performance after having gotten used to the interface during the first trial.

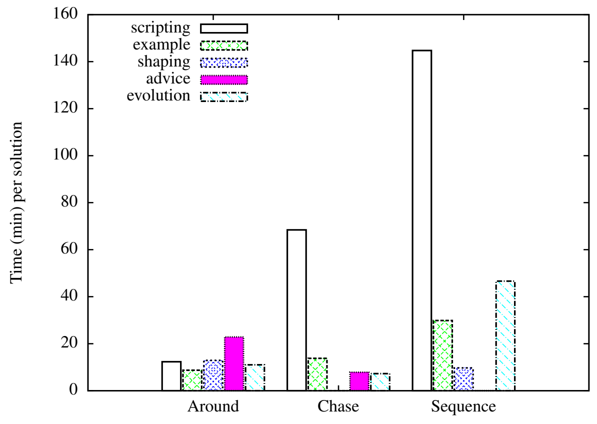

Figure 3: Average time spent by subjects before successfully solving each task. Bars not shown indicate an unknown value because no successful solutions were seen in the data. Evolution alone does consistently well and outperforms scripting, but is always outperformed by one of the human-assisted methods.

In order to collect objective measures of the performance of these methods, the interactions of the users with the OpenNERO interface were logged and recorded via screen capture. Three main measures were collected - the rate of success (i.e. whether or not the subject was able to solve a particular task using a particular method within the allotted time), the time to solution (how long the subject took to solve the task), and the number of restarts (how many times the subject restarted the interface before reaching a solution).

The average time per solution (the sum of the total time taken by all subjects divided by the number of successful solutions) is shown in Figure 3. An equivalent measure is also shown for neuroevolution without human assistance. This measure represents the expected amount of time one would have to wait before a solution is obtained, for each task and method. In all three cases, evolution performs well and outperforms scripting. Additionally, at least one human-assisted method outperforms neuroevolution alone on each task.

Another interesting result of the experiments becomes apparent when considering the detailed timelines of the success

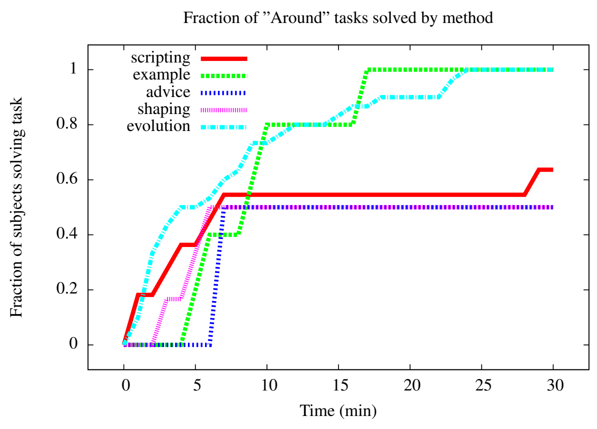

Figure 4: Fraction of "Around" tasks solved by each method over time. Here, scripting, shaping, and advice do not work as well as evolution alone, but example allows all the subjects to solve their task before evolution does, using fewer samples of the simulation in the process.

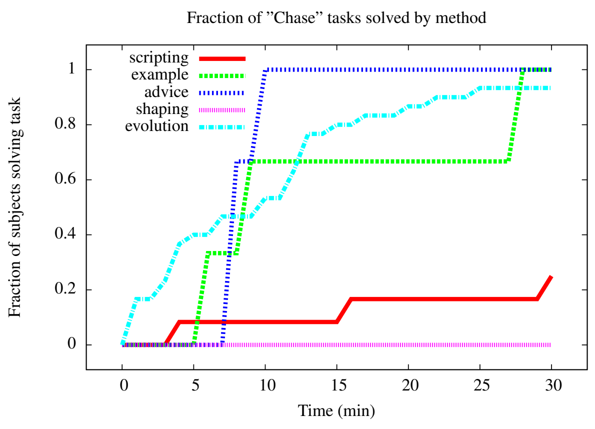

Figure 5: Fraction of "Chase" tasks solved by each method over time. In this task, shaping and scripting do not work at all, but advice allows the subjects to solve all the tasks before evolution alone.

rates of the methods (Figures 4, 5, and 6). In each case, unassisted and human-assisted evolution perform much better than scripting. While neuroevolution performs consistently well across the domains, it is outperformed by one of the human-assisted methods in each task – a different one in each case. While example traces work best in the Around task, advice works best in the Chase task and shaping works best in the Sequence task.

These results confirm that both neuroevolution and human-assisted neuroevolution outperform the manual scripting approach and that human-assisted neuroevolution is most effective, given the right approach to the task.

6 Discussion and Future Work

The result that different human assisted neuroevolution methods are best in different tasks makes sense when one considers the inherent characteristics of the tasks.

First, in the simple obstacle avoidance task, all methods work relatively well and the intuitive nature and ease of use of the example method allow users to complete the task in a short amount of time.

In the target chase task, in contrast, while it remains possible to solve the task with examples, the examples that are first attempted by the subjects are not as helpful, and the process therefore takes a longer time. Also, the task does not lend itself well to shaping because it contains a conceptual discontinuity: the behavior required to reach slow moving targets or targets that move within a smaller radius is drastically different from that of the complete solution. Thus shaping by modifying the environment is not a good fit for this task. However, formulating what needs to be done as either advice or script is relatively easy in this task, and neuroevolution with advice has an advantage because it can recover from mistakes made during rule design and learn some of the task autonomously.

In the Sequence task depicted in Figure 2c, the complicated nature of the environment and the possibility to decompose the task into logical subtasks allow shaping to outperform advice and example. In particular, an effective and simple shaping strategy is to simply remove all the targets except for the one that should come first in the sequence, and allow neuroevolution to master that task first, after adding the next target in the sequence and repeating until the complete task is learned.

In addition to performing well compared to unassisted neuroevolution in real time, it is important to note that human assisted neuroevolution is much more sample-efficient in that it requires many fewer interactions with the environment per unit time. In particular, while the entire time during the neuroevolution runs was spent evaluating neural network controllers in the environment at the highest possible speed, the human-assisted methods dedicate a fraction of time to other activities, such as writing scripts and advice pieces, providing example behavior traces, running neuroevolution in the slower real-time mode in order to verify the agents' performance, or simply thinking about the problem. Therefore it is a particularly good approach when testing candidate solutions is expensive.

The results of such experiments can be used to select and customize human-assisted evolution methods for building autonomous agents in virtual environments, as well as autonomous robots that interact with and learn from humans. They may also be used to train machine learning agents that adapt to a changing task on the fly. Most importantly, they give human engineers a degree of control that may make it easier for them to incorporate advanced machine learning techniques into the design process.

7 Conclusions

This article described the design and results of a human subject experiment evaluating three different approaches to combining human domain knowledge with neuroevolution, comparing them with both completely manual design and completely automated discovery through neuroevolution. The re-

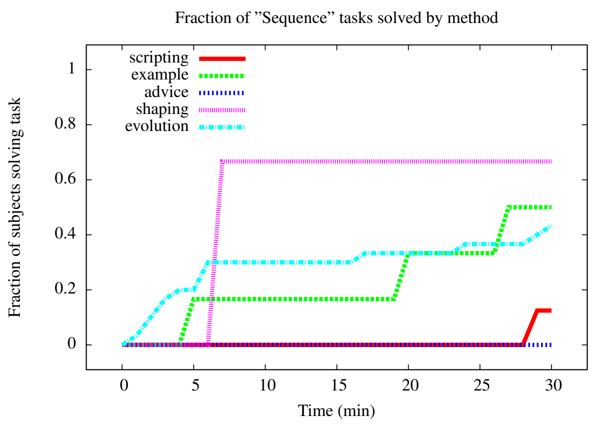

Figure 6: Fraction of "Sequence" tasks solved by each method over time. Here, the task is more challenging for all methods, but the best method turns out to be shaping because the task can be easily decomposed by modifying the environment.

sults demonstrate that (1) human-assisted evolution outperforms a manual scripting approach, (2) unassisted evolution performs consistently well across domains, and (3) different methods of human-assisted neuroevolution outperform unassisted evolution on different tasks. Therefore, if done right, human-assisted neuroevolution can be powerful technique for constructing intelligent agents in the future.

8 Acknowledgments

This work was supported in part by the National Science Foundation under grants IIS-0757479, IIS-0915038, and DBI-0939454. The authors would also like to acknowledge the Freshman Research Initiative at the University of Texas at Austin and the students of the Computational Intelligence in Game Design stream for participating in the study.

References

[Abbeel and Ng, 2004] P. Abbeel and A.Y. Ng. Apprenticeship Learning via Inverse Reinforcement Learning. In *Proceedings of the twenty-first International Conference on Machine Learning ICML'04*, page 1. ACM, 2004.

[Cardamone *et al.*, 2009] Luigi Cardamone, Daniele Loiacono, and Pier Luca Lanzi. Learning Drivers for TORCS through Imitation Using Supervised Methods. In *Proceedings of the IEEE 2009 Symposium on Computational Intelligence and Games (CIG'09)*, 2009.

[Coates *et al.*, 2009] Adam Coates, Pieter Abbeel, and Andrew Y. Ng. Apprenticeship Learning for Helicopter Control. *Commun. ACM*, 52:97–105, July 2009.

[Knox and Stone, 2009] W.B. Knox and P. Stone. Interactively Shaping Agents via Human Reinforcement: the TAMER Framework. In *Proceedings of the 5th International Conference on Knowledge Capture*, pages 9–16. ACM, 2009.

[Knox and Stone, 2010] W.B. Knox and P. Stone. Combining Manual Feedback with Subsequent MDP Reward Signals for Reinforcement Learning. In van der Hoek, Kaminka, Lespérance, Luck, and Sen, editors, *Proceedings of the 9th Int'l Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2010)*, pages 5–12, May 2010.

[Kuhlmann *et al.*, 2004] Gregory Kuhlmann, Peter Stone, Raymond Mooney, and Jude Shavlik. Guiding a reinforcement learner with natural language advice: Initial results in RoboCup soccer. In *The AAAI-2004 Workshop on Supervisory Control of Learning and Adaptive Systems*, July 2004.

[Maclin and Shavlik, 1996] R. Maclin and J. Shavlik. Creating advice-taking reinforcement learners. *Machine Learning*, 22:251–281, 1996.

[Nicolescu and Mataric, 2002] Monica N. Nicolescu and Maja J. Mataric. Learning and Interacting in Human-Robot Domains. *IEEE Transactions on Systems, Man and Cybernetics, Part A: Systems and Humans*, 31(5):419–430, 2002.

[Ramachandran and Amir, 2007] Deepak Ramachandran and Eyal Amir. Bayesian Inverse Reinforcement Learning. In *Twentieth International Joint Conference on Artificial Intelligence*, 2007.

[Stanley and Miikkulainen, 2002] Kenneth O. Stanley and Risto Miikkulainen. Evolving neural networks through augmenting topologies. *Evolutionary Computation*, 10:99–127, 2002.

[Stanley *et al.*, 2005] Kenneth O. Stanley, Bobby D. Bryant, and Risto Miikkulainen. Real-time neuroevolution in the NERO video game. *IEEE Transactions on Evolutionary Computation*, 9(6):653–668, 2005.

[Torrey *et al.*, 2006] L. Torrey, J. Shavlik, T. Walker, and R. Maclin. Skill acquisition via transfer learning and advice taking. In *Proceedings of the Seventeenth European Conference on Machine Learning (ECML'06)*, pages 425–436, Berlin, Germany, 2006.

[Yong *et al.*, 2006] Chern Han Yong, Kenneth O. Stanley, Igor V. Karpov, and Risto Miikkulainen. Incorporating Advice into Neuroevolution of Adaptive Agents. In *Proceedings of the Artificial Intelligence and Interactive Digital Entertainment Conference (AIIDE'06)*, pages 98–104, 2006.

[Zang *et al.*, 2010a] Peng Zang, Arya Irani, Peng Zhou, Charles L. Isbell, and Andrea Thomaz. Using Training Regimens to Teach Expanding Function Approximators. In *Proceedings of the Ninth International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 2010.

[Zang *et al.*, 2010b] Peng Zang, Runhe Tian, Charles L. Isbell, and Andrea Thomaz. Batch versus Interactive Learning by Demonstration. In *Proceedings of the Ninth International Conference on Development and Learning (ICDL)*, 2010.