Imagine you are observing the decision-making of a software agent and notice some imperfections. What should you be able to say to it? What would be a good “advice language” that (a) allows one to provide, in a natural, flexible, and imprecise manner, new information to the agent, while (b) ensuring that the agent’s learning mechanism can use the advice? We discuss these questions about broadening the “information pipeline” between human teacher and machine learner, briefly describe an implemented system that provides initial answers, and describe preliminary work on an advice-taking, adaptive Internet wanderer that searches for pages of interest to the user.

Our general approach has the following design principles:

1. The user and the adaptive agent should be able to communicate using more than the “labeled feature vectors” that are the most common method for teaching a machine learner.

2. The user should provide advice using domain-specific terms and should not have to understand (or even be aware of) the agent’s underlying representations and learning algorithms.

3. The user should be allowed to use vague and imprecise terms when providing advice, and the advice should not be required to be perfectly correct or fully complete.

4. The user should be able to define new terms for use in subsequent advice, describe alterations to old advice, or even completely retract old advice.

5. The user should be able to provide advice at any time, based on its observations of the agent’s behavior. (This contrasts to the standard approach in theory refinement, where the advice – usually called a “domain theory” – is provided before learning begins.)

6. There should not be a requirement that every one of the agent’s decisions be evaluated (by the environment or the user), but the agent should be able to use effectively any evaluations provided (verbal advice or scalar reinforcements).

7. Any provided advice should be immediately assimilated by the agent, but should not be followed absolutely and instead should be revised based on subsequent experience using it.

Our solution that addresses the above design goals builds on three research fields: (i) backpropagation training of neural networks (Rumelhart, Hinton, & Williams 1986), (ii) learning from coarse and delayed reinforcements (Barto, Sutton, & Watkins 1990), and (iii) direct insertion of symbolic knowledge into neural networks (Towell & Shavlik 1994). Figure 1 illustrates our RATLE system (Maclin 1995).

Our agent uses standard techniques from connectionist reinforcement learning (Lin 1992) to (slowly) learn from the user’s or environment’s scalar feedback. When the user has verbal advice to provide, the learner first parses the advice. Next, it makes specific any imprecise terms in the advice (by giving them fuzzy-logic interpretations (Berenji & Khedkar 1992), and inserts this “operationalized” advice directly into the neural-network representation of its knowledge base, thereby rapidly assimilating the user’s advice. Figure 2 schematically shows how advice is inserted into an existing neural network. Importantly, the advice can subsequently be refined both by neural training based on experience and additional advice from the advisor.

Previously, we successfully evaluated this design on video-game-like worlds (Maclin & Shavlik 1995). These tasks mapped nicely into our approach: the agent sensed its world (using simulated sonar-like sen-
sors), chose a compass direction in which to move, and intermittently received reinforcements for capturing objects (positive) or being captured (negative); the human advisor occasionally provided context-specific plans that suggested the next few actions the agent should perform. Our advice language was an programming language — for ease of parsing — that accepted domain-specific terms (e.g., opponent, North), fuzzy terms (e.g., near, many) and simple control (e.g., while, if). Our experiments showed that (i) advice can significantly improve the performance (i.e., total score in the game) of an agent, as compared to the standard reinforcement learning approach; (ii) advice produces the same resulting level of performance no matter when it is added; (iii) a second piece of advice can produce further gains; and (iv) it is important for the agent to be able to adjust the advice it receives.

We are now extending this approach to World-Wide Web (WWW) searching, and have to address questions about the sensors, effectors, reward structure, and advice language. Our current plan is that the sensors are a "windowed" representation of the words on a WWW page; the effectors will be follow-this-link, show-this-page-to-the-user, and backup-the-history-list; rewards are received whenever the user chooses to "score" a page shown to him or her, and a small penalty is charged for each page visited and a larger one for each failed attempt to access a page; finally, the appropriate extensions to our advice language is an open question.

We have obtained the source code to NCSA’s Mosaic WWW browser, and have added to Mosaic an agent that chooses which links to follow and decides whether to show the current WWW page to the user. The user can score these pages, and based on these scores the agent refines its decision-making neural networks. Based on experience with this traditionally designed reinforcement-learning agent, we are developing a collection of possible pieces of useful advice, creating a language that can represent this advice, and altering as necessary our advice-insertion algorithms.

In conclusion, we are developing an appealing approach for learning from both instruction and experience. This work increases the expressiveness of the dialog between humans and machine learners, without requiring that the human provide absolutely correct information to the learner. Our approach appears to be a promising one for designing adaptive, instructable software, and this hypothesis is currently being investigated by developing a prototype instructable WWW searching agent.

References