Teaching Computational Modeling to Non-Computer Scientists

Terrence C Stewart (tcstewar@connect.carleton.ca)
Institute of Cognitive Science, Carleton University
1125 Colonel By Drive, Ottawa, ON, K1S 5B6, Canada

Abstract

Most courses on computational modeling involve having students skilled in the computer science implement various computational models. Here, a course is presented which is aimed at students with minimal programming background. The focus is on using a large number of different models on various problems. Students learn how well various algorithms work, rather than how to implement them. They are provided with simple implementations and sample code that makes use of these implementations, and are then asked to adapt the given system to deal with other problems. This material is presented as part of the ongoing development of this course, and it is hoped that others may find it useful for their own courses and research.

Introduction

Courses in computational modeling tend to presume a significant amount of programming skill on the part of the student. Typical projects include implementing neural networks from scratch, or writing one's own genetic algorithm to solve a particular problem. These sorts of projects mean that students who are not specialists in computer science spend a disproportionate amount of time debugging. Furthermore, since the focus in these courses is on implementation, less time is spent discussing what sorts of problems they tend to work well or work poorly on.

In this paper, a course is presented which addresses these problems. The course ran at Carleton University in the winter term of 2004 as Cognitive Science 5001: Cognition and Artificial Systems. The only prerequisite was either graduate standing or a single previous programming course of any sort, and it was open to both graduate and fourth-year undergraduate students within the Institute of Cognitive Science.

Course Organization

Since the goal of the course was to give students practical experience using these various algorithms, the major content (worth 70%) of the course was weekly assignments. After each week's lecture, students were given a working implementation of a model (such as Kohonen Maps or Evolutionary Strategies). They were also provided with working code that used this implementation to perform a typical task. The students were then asked to modify this software to perform different tasks, and to modify the various parameter values of the models. The performance of these models in these conditions was then examined.

To allow the students to demonstrate a more in-depth mastery of one particular algorithm, the final paper for the course involved taking a published paper and replicating those results therein. For this paper, students were also expected to discuss the effects of varying various parameters of the model, and look at the effect of varying the problem representations used. In this way, the students perform a useful role within the cognitive modeling community, as this detailed replication is relatively rare.

The Computational Models

For all of these models, the students worked exclusively with the Python programming language, which was chosen for its ease of use and clear syntax. None of the students had any prior exposure to it. This standardization allowed for direct comparisons between models, and made it easy to combine models together. This author has also found Python to be highly suitable for modeling research outside of the classroom.

The programs were carefully designed to be as easy to read, understand, and modify as possible. Students commented that this significantly increased their confidence that their changes caused the programs to do what they wanted them to do. Furthermore, the success of this clarity was embarrassingly shown when one student (who had no previous programming experience) uncovered bugs in the implementations of two different models. These were quickly fixed, and demonstrate the code's readability.

Cellular Automata

To begin the course, students examined systems both with agents in the world (e.g. Langton's Vants) and without (e.g. Conway's Game of Life and Langton's Loops). Variations of different rules and different types of Vants were investigated. A collective sorting simulation (Deneubourg et al, 1991) was also replicated. This module was used throughout in the course as a grid-world environment.

Genetic Algorithms and Evolutionary Strategies

A wide variety of evolutionary options were investigated. This included various implementations of mutation and crossover, elitism, tournament and roulette selection, rank-based selection, steady-state models, and the recent development of neutral network theory, including extrema selection (Stewart, 2001). These techniques were applied to the traveling salesperson problem, genetic programming, and the evolution of agent behavior on a grid-world.

In a somewhat uncommon aspect of the course, Genetic Algorithms were directly compared to Evolutionary Strategies. Students were exposed to the significant advantage that ES algorithms have on parameter-optimizing problems (in this case, an ant pheromone model). They were also encouraged to investigate the effect of using ES-$(\mu/\rho+\lambda)$ selection within more typical GA problems.

Perceptrons and Multi-Layer Perceptrons

Here students used back-propagation learning to investigate the abilities of this common neural network architecture. Of interest was the effect of learning rate, momentum, varying numbers of hidden nodes, different approaches to representations, and the significant effect of normalizing input and output data. These systems were also compared to a memorization-based approach to supervised learning to indicate the practical speed advantages of these models.

Simple Recurrent Neural Networks

In this section, the core findings of Elman's (1990) paper introducing this model were replicated. The capabilities of the networks for maintaining context over longer and longer periods of time were investigated, as was the effect of using a non-obvious representation scheme in his syllable-identification problem.

Kohonen Self-Organizing Feature Maps

Kohonen Maps were compared to standard clustering algorithms, and demonstrated in terms of their ability to change the dimensionality of input data. Special attention was given to recent results (such as Touzet, 1997) showing that Kohonen maps can be used as an automatic way of adjusting representations before presenting them to standard supervised learning systems. To show this, the Kohonen map was used as a pre-processor to change the representation of the input data into a form that proved to be more suitable for learning by the backprop networks.

Fuzzy ART and Fuzzy ARTMAP

Both the unsupervised and the supervised versions of Adaptive Resonance Theory were investigated. The advantages in terms of learning rate were discussed, as was their ability to indicate that a particular input is unknown to the network. The models were also dissected to reveal the learned rules. The effect of the vigilance parameter and the order of presentation of the input was also studied.

Q-Learning and Sarsa

Both of these Temporal Difference algorithms were compared, highlighting the effects of the very slight difference between the two. Their behavior on the Cliff grid-world was discussed, and students were able to develop their own worlds to run the algorithms in. Students adjusted the various parameters of the learning algorithm, and saw the effects of changing the reinforcement regime.

ACT-R

In a potentially surprising move, a stripped-down implementation of ACT-R was also included as part of the course. This version only supported the rule-matching and chunk-activation learning aspects of ACT-R. This was sufficient for demonstrating the successful Rock-Paper-Scissors ACT-R model (Lebiere & West, 1999), allowing students to see how the model reacts to different patterns in its opponent's play. By watching the model adjust to different opponent strategies, students gained insight into how the ACT-R learning rule dynamically affects its behavior. Also, it provided some evidence that it is possible to develop an ACT-R model that is separate from the current LISP-based implementation.

Future Additions

In preparation for teaching this course again, a few small changes are being made to the course. The interfaces to the unsupervised and supervised learning algorithms are being standardized so that they can be easily swapped, allowing direct comparisons to be made. Some of the slower code (such as the back-propagation system) is being modified to interface with existing (significantly faster) C implementations, while still maintaining the same simple interface as used in the course. Also, the ACT-R library is being expanded with the missing learning mechanisms.

Conclusions

The course successfully exposed the students to a variety of widely used models, and allowed them more time to work with these models than would have been possible if they were to perform implementations themselves. The implementations given to the students were in a clear and readable language, allowing them to inspect the algorithms themselves to see exactly how they worked. These modules were also flexible enough to be easily combined and used by the students to replicate published research. Indeed, in a number of cases, the students' results revealed mistaken assumptions in the original research. The course was also found to be accessible to students with no previous programming experience.

Everyone is encouraged to make use of the materials developed for this course. The complete lecture notes and all software modules are available online at:

http://chat.carleton.ca/~tcstewar/cgsc5001/

References

Deneubourg, J.L., Goss, S., Franks, N., Sendova-Franks, A., Detrain, C., & Chretien, L. (1991). The Dynamics of Collective Sorting. Robot-like Ants and Ant-like Robots, *From Animals to Animats*, MIT Press.

Elman, J.L. (1990). Finding structure in time. *Cognitive Science, 14*, 179-211.

Langton, C.G. (1986). Studying Artificial Life with Cellular Automata, *Physica 22D*, North-Holland, 120-149.

Lebiere, C., & West, R. L. (1999). Using ACT-R to model the dynamic properties of simple games. *The Proceedings of Cognitive Science.*

Stewart, T.C. (2001). Extrema Selection: Accelerated Evolution on Neutral Networks. *IEEE Congress on Evolutionary Computation*, 25-29.

Touzet, C.F. (1997). Neural Reinforcement Learning for Behaviour Synthesis. *Robotics and Autonomous Systems*, *Special issue on Learning Robot: the New Wave*.