Genetic Programming Applied to Othello: Introducing Students to Machine Learning Research

Eleazar Eskin and Eric Siegel
{eeskin,evs}@cs.columbia.edu
Department of Computer Science
Columbia University
New York, NY 10027

Abstract

In this paper we describe and analyze a three week assignment that was given in a Machine Learning course at Columbia University. The assignment presented students with an introduction to machine learning research. The assignment required students to apply Genetic Programming to evolve algorithms that play the board game Othello. The students were provided with an implemented experimental approach as a starting point. The students were required to perform their own experimental modifications corresponding to research issues in machine learning. The results of student experiments were good both in terms of research and in terms of student learning. All relevant code, documentation and information about GPOthello is available at the following url: `http://www.cs.columbia.edu/~evs/ml/othello.html` .

1 Introduction

Teaching a research topic has several inherent difficulties. Research material often requires a strong background in a given subject in computer science. Because computer science is such a broad area, most undergraduate curriculums tend not to focus in a particular area enough to provide the depth necessary to present the research topics. Furthermore, research topics are usually only reached at the very end of the semester. By the time they are covered, there is little time to assign a project on that material.

In addition, even a small software research project often involves an inordinate amount of implementation time before any interesting results are seen. Usually, the scope of a research project is beyond the scope of a typical assignment for a course. Sometimes, students are asked to reimplement a research project, but the majority of time a student spends on the project is programming as opposed to thinking about the research topics at hand.

In this paper we describe and analyze a three week assignment that was given in an Undergraduate/Masters level Machine Learning course at Columbia University (Professor: Eric Siegel, TA: Eleazar Eskin). The project was called GPOthello. The assignment was designed to avoid these problems and give students the experience of "doing" research.

For GPOthello, students applied the machine learning method genetic programming (GP) to the board game Othello. Othello is a checkers type game also known as Reversi. Each student or student team selected a unique experimental project investigating variations on the learning method, hypothesis representation, or both. To evaluate results, students had to deal with many machine learning issues, among them overlearning, temporal credit assignment, local optima, hypothesis representation bias, learning speed and noisy fitness measurements. The projects were mutually complementary.

Our assignment provided the students with an implementation of a non-trivial, complete experiment which they used as a starting point. The assignment required students to modify the experiment in several ways. A hierarchy of research-motivated options were presented in the assignment, although students were also encouraged to pursue their own original ideas.

GPOthello was designed to avoid the problems of teaching research. Since the first experiment was a complete implementation, the students need to make only minor modifications in the code which was given in order to run their own experiments. In addition, the fundamental algorithm of genetic programming is relatively simple, especially in the context of a machine learning class, compared to for example, Neural Network backpropagation. This allowed students to get up to speed with regards to the research issues very quickly.

2 Genetic Programming

Genetic programming is a supervised machine learning paradigm inspired by biology. Genetic programming evolves algorithms in a method modeled after natural selection. These algorithms are represented as function trees [1,4]. The function trees are intended to perform a given "task". The process of genetic programming attempts to create a function tree that adequately performs the task.

Essentially, natural selection and reproduction are simulated over a randomly generated collection or "population" of these algorithms or "hypotheses". Each member or hypothesis of the population is "evaluated" by measuring the performance of its function tree on a given task. Every hypothesis is then ranked based on it's evaluation. Each hypothesis has a certain chance of being eliminated from the population based on its ranking. Those with higher ranking are less likely to be eliminated, hence survival of the fittest.

In order to replace the hypotheses that are eliminated, some of the hypotheses that do not get eliminated reproduce in order to create new hypothesis in the population. There are traditionally two ways to reproduce. The first is called "mutation" where a hypothesis is modified randomly. The second is called "crossover" where two hypotheses swap subtrees. Crossover intuitively simulates sexual reproduction. Once these new hypotheses are created, we have a new population. This population is referred to as the next generation. For every generation this process is repeated until we obtain a function tree that performs the task adequately.

The intuitive idea behind genetic programming is that as the generations progress, the hypotheses of the population will tend to get better at their task because the ones that perform poorly are eliminated and the ones that perform well are kept in the population. Furthermore the ones that perform well are likely to reproduce and their subtrees which may contain beneficial features to the task are replicated.

3 GPOthello

In GPOthello, students had to apply genetic programming to develop an algorithm that can play the board game Othello [10]. The GP function tree is a heuristic function to evaluate a board configuration. The function tree "plays" the game Othello by selecting the move leading to the best position every turn. In this approach, there is no search beyond the next immediate move in the game.

The students were also provided with another previously developed Othello player, Edgar. Edgar, who was designed by Astro Teller [11], was trained with a different learning paradigm, so students could perform cross-paradigm comparisons.

Teller devised a clever learning method that is specifically designed for Othello [9]. It takes a unique approach to the temporal credit assignment problem of machine learning. Temporal credit assignment is how a system deals with delayed reinforcement, where you don't know how bad or good each move was, only what the final outcome was. Teller's approach contributes by representing temporal credit explicitly, and without imposing a manually-designed method to judges intermediate game board states. Usually, implicit measurements to handle temporal credit assignment are employed, e.g., the approach to Checkers in Chapter 1 of Tom Mitchell's "Machine Learning" text [6].

The experiments by Teller that lead to Edgar's strategy are preliminary, and we only describe the approach here to a limited degree of detail. More information about Edgar is available on the GPOthello project web site [9].

We approached the difficulty of introducing the machine learning research in several ways. Since the first experiment was completely implemented for the students, there was little overhead for the students' experiments. The second is that the GPOthello project is a full scale implementation of a genetic programming system, which raises various research issues in machine learning. This allowed students to be creative in choosing their experiments and made the online compendium of all the students' experiments interesting because of the diversity in the experiments. Finally, genetic programming is an exciting topic to many students and the basic algorithm can be grasped relatively easily. This allowed the students to pick up the concepts and implicit heuristics behind genetic programming very quickly and readily come up with their own ideas for relevant experiments.

3.1 The Given Experiment

An initial approach to the problem was implemented by the instructional staff and provided to the students.

In this approach, the structure of the function tree used various board statistics as the terminals (or leaves of the tree) and arithmetic operators as the nodes. The board statistics were computed based on the board configuration which would be an outcome of a prospective game move.

The internal nodes of the tree contained the following standard arithmetic functions: plus, minus, multiply and divide. The terminals (or leaves) of the tree are integer values derived from the board:

1. black - the number of black pieces on the board.
2. black_corners - the number of black pieces in the corners.
3. black_near_corners - the number of black pieces near a corner.
4. black_edges - the number of black pieces on the edge of the board.
5. white - the number of white pieces on the board.
6. white_corners - the number of white pieces in the corners.

7. white_near_corners - the number of white pieces near a corner.

8. white_edges - the number of white pieces on the edge of the board.

9. 10 - the numerical value 10 which was used as a constant.

For example, the sample GP function tree represented by (+ white (* 10 white_corners)) evaluates a board configuration by computing the sum of the number of white pieces and 10 times the number of white corners.

The fitness measure was computer from playing 5 games against a random player and summing the number of opponents pieces at the end of each game. At the end of the experiment, the best players in a population would have the lowest scores because they would tend to beat the random players by larger margins and lose very rarely.

In the final generation of the experiment, there were players developed that could beat the random players 47 out of 50 times. However, these players were not good enough to beat Edgar.

3.2 GP Research Issues

In the assignment, students were presented with several research issues in genetic programming. The specific details of how each aspect of the genetic programming system is implemented has significant impact in overall performance. Several important issues in genetic programming are the structure of the function trees, and the fitness measure.

Variations in the structure of the hypothesis can involve changing the terminals and the operators and also limiting the structure of the function trees.

The fitness measure refers to how a score is obtained for each hypothesis in the population. Variations on the fitness measure involve changing how this evaluation occurs. This usually involves evaluating the function in some kind of simulation. In the initial experiment the score was obtained by playing 5 games with a computer opponent. Variations include playing players against each other or against various levels of computer (or even human) opponents, or varying the number of games played.

Students were required to perform further experiments by making modifications to the given experiment. Students were to make modifications to try to improve the results, possibly developing good enough players to beat Edgar, or to investigate variations to learning, fitness or hypothesis representation— beating Edgar was not the primary goal. We asked students to make one modification to the experiment in each of three different areas corresponding to these research issues in genetic programming. Students had to vary the structure of the hypothesis, the learning method, and the fitness measure. We presented several directions that could be followed in each case:

- Variations on the hypothesis
  1. New primitives (i.e., terminals) for GP trees.
     - distance-from-corner, parity thereof.
     - number of choices this move gives opponent (e.g., 0 is good).
     - number of black/white pieces that can never be switched the remainder of this game.
     - statistics, e.g., "scatterness/clumpiness of white distribution"
     - average number of flips black/white has made per move this particular game
     - random constants
     - more fundamental, simple primitives, e.g., x/y coordinates of the piece just placed to get to this new board
     - configuration – could GP use this and automatically build the concepts of "edge", "corner",
     - "one-away-from-corner"
  2. Improve Edgar, e.g., provide a terminal which is Edgar's score for a given move.
  3. Perform a shallow search through the space of possible moves, e.g., minimax or Alpha-Beta, and apply the tree at the end-points of this search.
  4. Help the hypothesis behave differently in the end-game versus other points in the game. (credit: Chris Porcelli's idea).
  5. Use ADFs (Automatically Defined Functions). An ADF is separately evolving tree which the function tree can invoke. [5]
- Variations on the learning method
  1. Parameters: population size, number of generations, proportion of crossover, etc.
  2. Number of games played during fitness measure
  3. Vary opponents for fitness measure (e.g., play against different kinds of random players, Edgar, a previously-evolved player, a hand-made player, or combination thereof)
  4. Competitive fitness measure – population plays itself (single elimination tournament), or competing populations co-evolve.
  5. With competition, how do you select the best-of-generation individuals? How do you track the changes in "absolute, objective" fitness?
  6. Have a set of 32 individuals each make one move of a game and distribute the resulting scores amongst them equally. Each individual in the population can make one move in several games. This is a noisy fitness measure, but much faster (more generations and/or larger population size possible). It's so crazy it just might work! (applies whether or not competition is happening)

7. Have a population vote on each move.
8. Explicit Credit assignment. Compare each move's score to Edgar's score for that move.

- Methods to evaluate results
    1. Vary Opponents. Try to play against: human players, Edgar, other on-line Othello players, random players, previously evolved players, hand-written baselines of comparison.
    2. Head-to-head competition between results of various experiments including experiments performed by other classmates.
    3. Compute statistics of tendencies of how it plays, include visualizations of this.
    4. Analyze resulting function trees to "understand" their "strategy".

The goal of each student project was not necessarily to beat Edgar. Rather the goals included:

- Compare, contrast and discover methods to approach Othello with GP.
- Investigate methods to evaluate resulting Othello players.
- Compare results to another learned Othello player, Edgar.
- Use another Othello expert, Edgar, during training.

To avoid redundancy, we asked students to send us a description of their project to be screened by the instructors. We asked students to select a project that they could do in two weeks. We allowed the students work in teams. Based on student proposals, we asked several groups to communicate between each other and come to an agreement on specific differences between experiments.

Students handed in an HTML write-up of their project, about 3-6 pages in length (available on our web page). The write-up was to contain a written description of the experiments that were conducted, numerical results of those experiments with an analysis of the results, and a conclusion.

3.3 Time breakdown for assignment

In general, students spent about 4-10 hours on implementation of the project and about 10-20 hours on running and analyzing the experiments and working on the write up. Many of these experiments were left overnight to run. This caused students to have to start early on their assignments in order to finish them.

We allowed students who wanted to continue working on this assignment after the project was over to work on an extension of their GPOthello project for their final project in the class.

4 Results

4.1 Experimental Results of Student Projects

The write ups of students presented their results. Every project improved over the initial experiment. A complete compendium of the variations and student experiments are found on the GPOthello project web page. There were 18 total GPOthello projects. Several of the students came up with their own original primitives, variations on the learning method, and variations on the evaluation.

In general, the performance of the student projects were better than the first experiments performance. In a few cases, evolved players could even beat Edgar. The collection of students results gives an excellent comparison of various training methods and hypothesis representations.

4.2 How Much Students Learned about Machine Learning

The GPOthello assignment allowed students to experience the "feel" of research. Many of the students pursued their original ideas which can be shown by the fact that many of the modifications that students included in their experiments were not included in our list of sample modifications.

The quality of the write ups for the student projects were very high as can be seen on the web site. Students were able to effectively organize their write ups. The write up of process and evaluation improved the students ability to organize a research project and research paper leading to a collection of very good final project write ups, especially considering the limited time of three weeks for entire inception, planning, implementing, evaluation and write up of the work. In addition, every project effectively cited machine learning work which was a requirement in the assignment. Students learned how to cite correctly and effectively in their write ups. The students also included in their write ups a discussion of relevant research issues to their experiments.

The online compendium of the results of all of the projects gives a survey of various possible modifications and their relative performance.

Their writing up of their process and evaluation was a great precedent, improving their ability to organize a research project and research paper, and ultimately leading to high quality write ups of final semester projects on other machine learning work. The GPOthello project web page also provides access to several such final project write ups.

Several students in the course also decided to continue working on this project for their final project. In addition, 5 of the 15 students in the class opted to enroll for independent research projects in machine learning for the following semester.

The Machine Learning course which was taught in Fall 1997 received among the highest student evaluation for a computer science course. The course received

14 “5”’s and 1 “4” out of 15 students with “5” being the maximum score. Four out of five of the assignments in the course had a research flavor similar to the GPOthello project which built through the semester and piqued student interest in the topic and the course.

The project effectively fostered collaboration between students. Students word on this project in teams. In addition, the class as a whole did mutually complimentary work. After the assignment was given in Fall 97, the students in the following semester, Spring 98, built on what was done the previous semester using the online compendium of projects. For example, one student combined primitives which were proven to be effective from previous students experiments to obtain better results.

5 Conclusion

The GPOthello project was a manageable research project for students. Because the given experiment contained most of the implementation necessary for the students experiments, the students were able to perform their experiments without spending too much time on implementation overhead. In addition, because the basic algorithm of genetic programming is easy to grasp, the students were able to quickly generate their own original ideas for their experiments. Furthermore, the genetic programming implementation for GPOthello is a full scale implementation which raises many general issues in machine learning.

The results of the projects themselves were interesting. Students were able to improve the performance of their players by making modifications to the initial given approach. Many of these modifications were original student ideas, not provided by the instructional staff. The assignment also provided students with an opportunity to create research style evaluations and write ups.

The assignment and course in general also received an extremely positive response from the students. Many of these students continued pursuing research projects the following semester.

6 Obtain GPOthello for your Class

GPOthello is available via the World Wide Web. The homework assignment and all needed code are available for interested researchers or machine learning instructors. This includes ideas for GPOthello projects from which students could select. It also includes Java implementations of GP (jpgpp) and of Othello, which are hooked up; to perform many non-trivial experiments, this code can be modified easily by students who are not fluent with Java. In addition, each student’s Othello report on their project is available on the web. If you are interested in using it in your class or obtaining more information about the project, you can find the project and all relevant code at `http://www.cs.columbia.edu/~evs/ml/othello.html` .

7 Acknowledgements

Thanks to the excellent students for their projects. The students in the course were: (Spring 98) Truta, Juno Suk; (Fall 97) Yi-Min Chee, Chun Chao, Olga Merport, Federico Kattan, Chris Porcelli, Patrick Pan, Daby Mousse Sow, William Bauder, Mohamed F. Abdelsadek, Monty Kahan, Janak J Parekh, Scott Susser, David Evans, Barry Schiffman, Kayuri Mitsui, Xin Jin, Matt Bogosian, Alberto Goldberger, LianShu Huang, Jesse Schechter.

The idea to use Othello as a machine learning homework project was Astro Teller’s, and he provided some of the code used in the project, as well as Edgar. In addition, thanks to Judith Klavans for useful discussions concerning the writing of this paper.

References

[1] Cramer, N. *A Representation for the Adaptive Generation of Simple Sequential Programs.* Proceedings of the [First] International Conference on Genetic Algorithms. Lawrence Erlbaum. 1985.

[2] Huss, J. *Laboratory Projects for Promoting Hands-On Learning in a Computer Security Course.* SIGCSE Bulletin 27:2 June 1995.

[3] Klemetti, H., I. Lapinleimu and M. Sieranta. *A Programming Project: Trimming the Spring Algorithm for Drawing Hypergraphs.* SIGCSE Bulletin 27:3 September 1995.

[4] Koza, J.R. *Genetic Programming: On the programming of computers by means of natural selection.* MIT Press, 1992.

[5] Koza, J.R. *Genetic Programming II: Automatic Discovery of Reusable Programs* MIT Press. 1994.

[6] Mitchell, T. *Machine Learning.* McGraw Hill, 1997.

[7] Russel, S., P. Norvig. *Artificial Intelligence: A Modern Approach* Prentice Hall 1995.

[8] Siegel, E. V., Koza, J.R. *Genetic Programming: Papers from the 1995 Fall Symposium”.* Proceedings of AAAI-95.

[9] Siegel, E. V., Teller, A. *Edgar Learns to Play Othello* Available at: `http://www.cs.columbia.edu/~evs/ml/hw4EDGAR.html`

[10] Smith R., Gray, B. *Co-Adaptive Genetic Algorithms: An Example in Othello Strategy* Proceedings of the Florida Artificial Intelligence Research Symposium. 1994

[11] Teller, A. *Exegesis* Random House, 1997.