NEATMinesweeper

by Michael Wheatman, Brian Charous, David Pickart, and Oliver Heywood

Evolutionary Computation and Artificial Life Final Project

Neural Network based Minesweeper player.

Goal

Our goal in this project is to create a neural network based minesweeper player. The player should be able to outperform a random player.

Description of Evolutionary Algorithm

We decided to use the NEAT algorithm. NEAT stands for Neuroevolution of Augmenting Topologies. The algorithm works by evolving artificial neural networks by evolving both the structure and weighting parameters of the network. The process of evolving the networks strikes a balance between finding individuals with the best fitness and maintaining diversity.

Representation and Initialization

We represent our evolved player as a neural network with 25 inputs and 25 outputs. The minesweeper board is a 5 x 5 grid, so each input and output of the neural network corresponds to a position on the board. The network accepts the current board state and returns a list of values for board positions. The player than clicks the unclicked cell with the highest value.

Parameters

Be chose to represent the board as a 5 x 5 grid. We store the best neural network as an xml file called minesweeper_champion.xml. We configure the evolver to favor complexity in the neural network over simplicity by entering a complexity threshold of 2000.

How We Chose Parameters

We chose the grid size for two reasons. For one, five by five is the smallest grid that remains at least somewhat interesting. Additionally, a five by five grid is required to analyze with certainty if a square is able to be clicked. We settled on the complexity threshold via trial and error.

A Good Individual

View this page at github.com/michaelwheatman/NEATMinesweeper to see the xml. Or generate an individual and view minesweeper_champion.xml in any text editor. <!-- language: lang-xml -->

Original Hypothesis

We know from playing Minesweeper ourselves that it is possible to build a Neural Network that can play Minesweeper. After all, human brains are just very large neural networks. Additionally, we know that it is possible to build a AI solver for Minesweeper that works with a high degree of success. We thus originally hypothesized that our evolved solver would be better than a random player. We hoped that it could perform as well as a deterministic approach.

Compared To Random

To test our evolved player, we had it and a random player attempt 100 randomly generated boards with a board size of 25 cells. Using a fitness evaluation based on the number of squares revealed, the evolved player attained an average fitness of 18.31, which means that it found 13 unmined squares before losing on average. The random player had an average fitness of 13.99, which means that it found 9 squares on average before losing on average. This means that our evolved player performed 30.9% better than random.

We also attempted a fitness evaluation based on the number of clicks the player makes. Using this evaluation with a board size of 25 cells, the evolved player averaged 6.28 clicks over 100 trials while the random player averaged 3.71 clicks. This means that our evolved player performed 69.3% better than random. Using click count evaluation with a board size of 100 cells, the evolved player averaged 4.94 clicks over 100 trials while the random player averaged 3.08 clicks. This means that out evolved player performed 60.4% better than random.

Board Size	Fitness Eval Type	Random	Evolved	Evolved Better Than Random By
25 Cells	Squares Revealed	13.99	18.31	30.9%
25 Cells	Clicks	3.71	6.28	69.3%
100 Cells	Clicks	3.08	4.94	60.4%

Thoughts on the Project

How well did we succeed

We were successful at evolving players that were better than random. However, from research and experience, we know that there are deterministic algorithms that are able to solve minesweeper boards to a greater degree.

Problems Encountered

Our NEAT implementation seems to avoid increasing the number of hidden nodes. We attempted to address this by raising the complexity threshold but this didn't seem to have any effect.

Sources

We used this tutorial: http://www.nashcoding.com/2010/07/17/tutorial-evolving-neural-networks-with-sharpneat-2-part-1/

We used the SharpNEAT C# Library: http://sharpneat.sourceforge.net/

We also used the following sources:

Randall, T., Cowling, P., Baker, R., & Jiang, P. (2009). Using neural networks for strategy selection in real-time strategy games. In Symposium on AI &amp.

NEAT Wikipedia Page: http://en.wikipedia.org/wiki/Neuroevolution*of*augmenting_topologies

How to Run

On OS X, Linux:

- Get Mono: http://www.mono-project.com/download/
- Compile: `make`
- Run the neural network `mono Evolver.exe <board size>`
- Press `Enter` to end the evolver when the fitness has stabilized.
- To run Random Player `mono Random.exe <board size> <optional flag 't' to print boards>`
- To run Evolved Player `mono AI.exe minesweeper_champion.xml <board size> <optional flag 't' to print boards>`
- For more complicated analysis, you can run multiple trials, average results, change board size, etc.
- Example of how to run multiple trials of Evolved Player on a 5x5 board and print average fitness `./aiTest.sh 5 && cat aiPlayer.txt | python3 average.py`