Machine Learning Engineer Nanodegree

Capstone Proposal

Chris Wilkins May, 2017

Proposal

Domain Background

The paper by R.S. Sutton and A.G Barto (1) defines Reinforcement Learning as 'learning what to do so as to maximize a numerical reward signal'. Reinforcement Learning has been used in order to train computers to play many different games, with increasing success. From TD-Gammon (2) playing Backgammon at a world-class level, to the work of Mnih et al (3) playing Atari games, to Google's DeepMind beating the world's leading Go player, reinforcement learning has drastically improved in solving ever-more complex games.

The idea for this project is to use reinforcement learning to play a different board game.

The game chosen for this project is called King's Table, or Hnefatafl (4), and is an ancient Viking board game played between two players. It has similarities to Chess, in that there is a King and the capture of the King piece signals the end of the game. But it differs since only one side has the King piece, the side with the King (the Defenders) has the goal of getting the King to one of the four corner squares in order to win the game. The opposing side (the Attackers) has the goal of stopping the King from getting to the corner squares, and can win the game by surrounding the King on all sides with Attacking pieces. The Defenders have fewer pieces, but an easier goal at first glance, while the Attackers have more pieces but the more difficult task. This disparity between Attackers and Defenders makes the game interesting from a Machine Learning perspective. It is fairly obvious how to implement a simple algorithm which finds the shortest path for the King to one of the corner squares as a strategy for the Defenders, this doesn't require any Machine Learning to achieve success. The strategy for the Attackers is more subtle, being able to block all the exit paths is something that inexperienced players struggle with. Can a computer learn this?

Problem Statement

The goal of this project is to use Reinforcement Learning to make an agent learn to play King's Table as the Attackers to a level where it can beat a beginner-level heuristic-based computer player. Since the initial agent will probably lose the vast majority of games against a beginner-level computer player, the agent will also be measured by the number of rounds it can keep playing before losing.

Datasets and Inputs

Since the agent will be learning from playing against a computer opponent, there are no external datasets. However, given the state space is large, the board is 11x11 and the 37 pieces (24 Attackers and 13 Defenders) have over 100 choices for valid moves to play at any given game state, it won't be possible to use a Q-Table for all states and actions. For this reason, it will be necessary to use Deep Reinforcement Learning to approximate the value function for each state.

No external source of data is available to evaluate the value of a particular state of the game board, so a large number of random games will be played to build up a source of data for experience replay purposes.

Solution Statement

The agent will use Deep Reinforcement Learning to automatically learn the optimal strategy by playing games against a computer opponent. The agent will try to learn the strategy from scratch, with no hand-crafted features added to the model.

Benchmark Model

The computer player will use a shortest-path algorithm to try to find the shortest path to the exit squares. If there are no paths to the exit due to the King being blocked by it's own pieces, the computer player will try to move pieces out of the way of the King until the King has a free path to the exit. If the Attacking pieces have managed to block all the exits, the computer player will look for opportunities to capture Attacking pieces. If no such capture opportunities exist, the computer player will select a random move from the valid available moves.

This rules-based agent easily defeats an opponent playing random moves, and can usually win against a beginner human opponent. This should make it a useful opponent for a learning agent.

Evaluation Metrics

Winning Percentage - The number of games won against the computer opponent will be the first, and most important metric. It will be calculated by running a set number of games against the computer and measuring the average number of games won by the agent.

Game Length - As mentioned above, the agent is expected to lose all the games against the computer while learning. In the event that the win percentage is very small, the average length of the games played will be used as a secondary metric for how well the agent is learning.

Project Design

The project will be implemented in Python3, using the Tensorflow library for the Deep Network. The first step to be implemented will be to design a simulator which will allow the agent to play games against the computer opponent. The simulator will generate the board and the two sets of pieces, and implement the rules of the game with regards to moving pieces and capturing. The logic for the computer opponent will be built into the simulator, and will use a shortest-path algorithm to find the best move, as described above.

The simulator should support two modes: a training mode where the agent plays a large number of games against the computer opponent in order to learn, and a testing mode where the agent plays a set number of games against the computer using the trained model. The training mode should be focused on exploring the state space, whereas the testing mode should focus on choosing the best moves based on the learning so far.

The agent will choose moves based on an epsilon-decay model, with an initial set of games played with random moves in order to build up a set of observations for experience replay. Once the observation stage is complete, the agent will play a set number of games as the epsilon value decays in order to train the model. Once this training stage is complete, the agent will run a smaller set of test games, choosing the move with the highest Q-value in order to evaluate the model.

Once the metrics around win/loss percentage and average game length are recorded for a particular model, the parameters of the model can be changed in order to test new models. In addition, techniques like Double Q-learning and Dueling Q-learning could be implemented to compare different types of model.

References

1. Reinforcement Learning: An Introduction - http://people.inf.elte.hu/lorincz/Files/RL_2006/SuttonBook.pdf
2. Temporal Difference Learning and TD-Gammon - http://www.bkgm.com/articles/tesauro/tdl.html
3. Human Level control through deep Reinforcement Learning - http://files.davidqiu.com/research/nature14236.pdf
4. Hnefatafl: The Game of the Vikings - http://tafl.cyningstan.com/