Oct 2, 2016

PLAYING OTHELLO BY DEEP LEARNING NEURAL NETWORK

1. INTRODUCTION

Go has always been considered the pinnacle of artificial intelligence in gaming [1]. With over 300 possible moves at each turn, it was believed to be a game of instinct, a proof that we humans possess something unique to us, a proof we creators bear true intelligence while the machines do not, a proof that held true until AlphaGo defeated Korean professional Lee Sedol 4 to 1 in a 5-game series [2]. This remarkable feat unravels the endless possibilities of deep learning neural network, where work previously thought to be only achievable by humans may finally be handed to our machinery friend.

2. OBJECTIVE

In this project, I will attempt to apply technologies used in AlphaGo, in particular deep learning, on the game of Othello. My aim is to create a program which can play the game of Othello more successfully and efficiently than current programs using this technology. If time allows, I would attempt to beat 5 out of top 10 commercially available Othello software with this software by either (1) using significantly less computational power, or (2) straight up winning.

3. BACKGROUND

- Artificial Intelligence

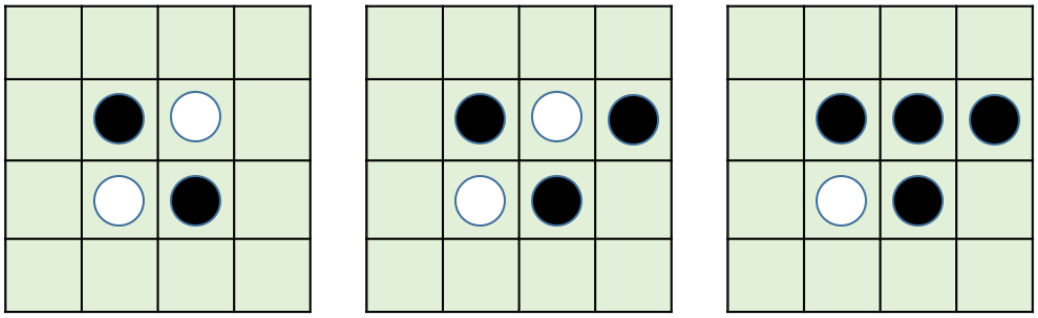
  Traditionally, artificial intelligence works around a professional teaching (coding) a computer to focus (evaluate) on significant parts. This requires a lot of coding and relies heavily on our design capabilities. To makes things even harder, we have to understand our way of thinking thoroughly to make the computer mimic what we are doing. Now, imagine how you would describe the process of recognizing numbers. How did you recognize a 7 from a 1? If the stoke in the middle helped you in it, why did you not recognize it as a 4? What angle must the top of 7 tilt at when it stops being a 7? If some ink failed to dye the paper, would your methods still work? This list of question goes on and on and is just one of the many reasons our traditional approach reached a bottleneck. Machines don’t have instincts.

- Deep Learning Neural Network

  Actually, machines might indeed have instincts. Before gaining spotlight in the field of gaming, deep learning has already achieved over 90% accuracy in speech and handwriting recognition [3]. By telling the machines to learn from their mistakes, speech recognition software actually discovered rules that even linguists are not aware of. This implies that instincts might just be extremely fast calculations of our subconscious brain, and all that was blocked by this barrier is now revealed to our trustworthy computers.

- Othello

  To apply our new founded techniques to Othello, we must first understand our game. In Othello, a game always starts with a cross pattern in the center of the board. At every turn, a player puts down a piece which has to wrap the two ends of an arbitrary number of opponent pieces in the middle. This converts those opponent pieces into friendly ones. This kinds of conversion can be horizontal, vertical and diagonal. The dark player starts first and the game continues until neither player can make a move.



  Similar to Go, Othello is a zero-sum, perfect-information, partisan, deterministic strategy game. This means that it is a game with no win-win situation, no concealed information (and thus no bluffing), practically equal position for both players and no randomness involved. This type of game is ideal for computers to play and is thus one of the reason it was chosen to be the topic.

  Unlike Go, Othello has very limited moves at each turn and connection between pieces are much less relevant. Hence, without the need of deep learning, the game was in fact already strongly solved at smaller sizes. For example, in 4x4 Othello games, white can always win under perfect play and the best black can do is lose by 3-11 [4]. This is similar in 6x6 games where black loses by 17-19 [5]. This is why our objective was not simply winning existing software, but playing at similar levels using less resources.

4. LITERATURE REVIEW

As the project was practically inspired by and revolved around AlphaGo, the research article of AlphaGo was read and critical technical terms are listed and explained below [6].

- Overall Structure

  The main structure of AlphaGo is the Monte Carlo Tree Search (MCTS). It is accompanied by two deep learning neural networks, namely the policy network and the value network [6]. One is developed for reducing branching factor, while the other for reducing depth of search when necessary. While the policy network is accurate, a fast rollout policy is developed for side-by-side comparison [6]. Supervised learning and reinforcement learning are used as standard machine learning approaches.

- Monte Carlo Tree Search

  MCTS is a general game tree search without branching at all. Instead, each probe is conducted all the way to the end by moving randomly at each state [7]. The rewards and visit count are then back propagated to better enhance the next probe [7]. This continues until time runs out, which makes it ideal for playing under time constraint. However, as we can easily tell, moving randomly does not match a rational player (opponent) and thus policy network is employed to better reflect reality.

- Policy Network

  For AlphaGo, 13-layer policy network was trained from 30 million positions from the KGS Go Server [6]. The network is provided game states and the next move by human expert and would gradually learn to predict this move [6]. The result was an accuracy of up to 57%, compared with a maximum of 44.4% for other research groups at that time [6]. However, it takes 3ms for computation, which hence limits the traversed depth of MCTS [6].

- Rollout Policy

  Rollout policy is a fast algorithm that analyzes known patterns on the board for the most probable moves. This is also trained with supervised learning of 8 million positions from human games on the Tygem server [6]. The rollout policy achieved a 24.2% accuracy using only 2 μs (0.067 % of Policy Network) [6]. In the end, both rollout policy and policy network are used in a parallel manner, and both results are taken into account [6].

- Value Network

  Value network is used to truncate tree search when deemed necessary. When looking at a game, professionals could understand if the dark or white side was at an advantage. Value network is trying to do exactly the same. At first, full game histories were provided to train the value network in a similar manner to policy network [6]. This led to overfitting and the machine actually "memorized" the training set [6]. The network was then trained with distinct states from different games played by itself and earlier versions [6]. The problem was then solved.

- Supervised Learning

  Supervised learning is used whenever a program is asked to provide answers to questions. It is supervised in the manner that both answers and questions are provided in the training examples. The program can then predict answers from similar questions later on. It is thus commonly used and as we can see, applied for all three of the policy network, rollout policy and value network [6].

- Reinforcement Learning

  Reinforcement learning basically means practicing. In the case of AlphaGo, the program played games again and again with its random former self, which can be the first or the 524th version [6]. The program can then identify its mistakes and update its value network and policy network accordingly.

- Evolution Algorithm

  This is the technique employed by Kurzweil Applied Intelligence described by Ray Kurzweil in his book "*How to create a mind*". This involves selective "breeding", or in other words mixing, of two well performing algorithm's weights. This is especially useful in deciding the hyper parameters (for example, how many layers should our neural network have) [3]. While this may not be much related to the success of gaming, I wish to attempt and observe this approach to understand the general approaches in machine learning.

5. PROJECT CHOICES

While Othello is a simpler game to Go, we can still introduce similar variables in the development of our project.

- Language

  Python would be the choice of language here. Among commonly used computer languages, it has the most support to machine learning and complex mathematical operation. While compared with MATLAB, which might be even more efficient in this manner, Python also has access to GUI programming, which is very favorable in later stages when playing against human opponents.

- Handicap

  If the development was smooth and if time allows, our program should be able to play against other existing software using handicap. This enables our program to develop and learn when no opponents of similar level could be found.

- Size

  Unlike Go, Othello can have a varying board size. We could increase it steadily while it is computationally feasible. This might be the most important variable in the end, as computational difficulty of Othello largely depends on size. If possible, we should develop a program which can play to any size.

6. METHODOLOGY

I will tackle the problem of Computer Othello in a similar manner as AlphaGo on Go. The main program will implement the Monte-Carlo Tree Search accompanied by Value Network and, if possible, Policy Network. The networks will be trained by a separate training program for modularity using existing play history. Finally, we will further enhance the program by playing with former versions of itself, already available software and human players.

7. DELIVERABLES

- Main Program

  At the end of the project, a main program written in Python will be delivered. It will play Othello at a superhuman level. It will also have an option to play at two different modes. In one mode, it will have a GUI (Graphical User Interface) to enable playing against human players. In the other mode, it wil read in and return moves in the most compact way while concealing GUI, minimizing overhead.

- Wrapper

  A wrapper is used to enable to competition between two computer Othello programs. This can be the competition between two versions of the above program, or between available programs on the market and our main program. We will provide support for as many existing programs as time allows. The wrapper will be in whichever language required to facilitate the communication between two programs. The burden of reinforcement learning will also lie on the wrapper.

- Training Program

  This program will be responsible for the supervised training part of the project. It will be mainly used to fine tune the value network using training samples provided. It will be written in Python as it only needs to deal with the main program.

- Records

  The records of training the program will be stored in an organized and readable manner. The idea of ranking and Elo may also be considered. This allows us to present our findings on how the program develops throughout the year. Quantizing the strength of program is also important for similar reasons.

8. CHALLENGES

- Software

  The timeframe might be too tight to implement every software tools used by AlphaGo. To counter this, the modules are listed in order of importance. By implement them in order, we could at least achieve the most when time was indeed a deciding factor.

  i. **Monte Carlo Tree Search**
  ii. **Value Network**
     This is the more important of the two network, as the width of Othello game search tree is much less of an issue compared with its depth. This means that there are very few possible moves. This is especially true when compared with Go.
  iii. **Policy Network**
  iv. **Rollout Policy**
  v. **Evolution Algorithm**

  In the above list, the first 3 parts are crucial to the execution of the program and will be completed before any fine-tuning. I will then work on Rollout Policy and Evolution Algorithm if time allows.

- Hardware

  As of current stage, there seems to be no hardware support from the Computer Science department. I would therefore use my own computer which has Intel 6700K as CPU and 16G RAM at the date of writing. This may limit our choice of algorithm and affects overhead in unforeseen manner. While we may also approach the department for better hardware support, it is more reasonable to not anticipate any and simply develop a program with minimal requirement.

9. SCHEDULE

- Sep – Oct 2016

  <u>Traditional AI Approach</u>

  By the end of October, a functional AI program written in Python should be able to compete at a 8x8 board. It should implement the Monte Carlo Tree Search and can adapt to different time limit as such. It would be anticipated to compete at an intermediate level and should not perform silly mistakes.

- Nov – Dec 2016

  <u>Value Network</u>

  By the end of 2016, there should be a value network which betters the performance of the program. Professional gameplay history would be used if access was available. If not, the 4 undergraduate working on similar topics would generate 200 games of gameplay to train the network.

- Jan – Mar 2017

  <u>Extended Research</u>

  At this stage, we would attempt to implement policy network, rollout policy and evolution algorithm. They would be provided as options in the final deliverable to better understand their impacts on the program. I believe that observation on results would be more valuable at this stage than the actual deliverable.

- April – May 2017

  <u>Final Product</u>

  I will wrap up my project at this stage. The program should be fully functional with lots of testing conducted to better its credibility. We should attempt to have an Elo ranking of the program with different options, as well as human players as comparison. We should also look into more literature to better understand the discrepancy of our expectation and outcome.

10. CONCLUSION

Computer has been around for decades, taking off tremendous workload off the parts we are proudest of – our brains. We unloaded our memory to the Internet, our computation to the mathematical software, but computer still doesn’t seem smart enough to considered more than a clever tool. This may be finally changed as deep learning enables computers to think more and more like us. There may be a time when our brains can finally take a rest. Until then, let us work this gluey grey matter to its greatest potential.

REFERENCE

1. Johnson G. To test a powerful computer, play an ancient game. The New York Times [Internet]. 1997 Jul 29. [cited 2016 Sep 17]; Technology: [about 5 screens]. Available from: http://www.nytimes.com/1997/07/29/science/to-test-a-powerful-computer-play-an-ancient-game.html?_r=0

2. Borowiec S. AlphaGo seals 4-1 victory over Go grandmaster Lee Sedol. The Guardian [Internet]. 2016 Mar 15. [cited 2016 Sep 17]; Technology: [about 2 screens]. Available from: https://www.theguardian.com/technology/2016/mar/15/googles-alphago-seals-4-1-victory-over-grandmaster-lee-sedol

3. Kurzweil R. How to create a mind: the secret of human thought revealed. New York City (US): Viking Penguin; 2012

4. AI Lab [Internet]. [updated 2008 Aug 22, cited 2016 Sep 19]. Available from: http://ailab.awardspace.com/othello4x4.html

5. Tothello [Internet]. [updated 2006 Jul, cited 2016 Sep 19]. Available from: http://www.tothello.com/html/solving_the_6x6_normal.html

6. Silver D, Huang A, Maddison CJ, Guez A, Sifre L, van den Driessche G, Schrittwieser J, Antonoglou J, Panneershelvam V, Lanctot M, Dieleman S, Grewe D, Nham J, Kalchbrenner N, Sutskever I, Lillibrap T, Leach M, Kavukcuoglu K, Graepel T, Hassabis D. Mastering the game of Go with deep neural networks and tree search. Nature [Internet]. 2016 [cited 2016 Sep 17]; 529. doi:10.1038/nature16961

7. Jeff Bradberry. [updated 2015 Sep 07, cited 2016 Sep 18]. Available from: https://jeffbradberry.com/posts/2015/09/intro-to-monte-carlo-tree-search/