Available online at www.sciencedirect.com
ScienceDirect
Neural Networks 21 (2008) 1392–1400
Neural Networks
www.elsevier.com/locate/neunet

Learning to play Go using recursive neural networks

Lin Wu, Pierre Baldi*

School of Information and Computer Sciences, Institute for Genomics and Bioinformatics, University of California Irvine, Irvine, CA 92697, USA

Received 11 August 2007; accepted 21 February 2008

Abstract

Go is an ancient board game that poses unique opportunities and challenges for artificial intelligence. Currently, there are no computer Go programs that can play at the level of a good human player. However, the emergence of large repositories of games is opening the door for new machine learning approaches to address this challenge. Here we develop a machine learning approach to Go, and related board games, focusing primarily on the problem of learning a good evaluation function in a scalable way. Scalability is essential at multiple levels, from the library of local tactical patterns, to the integration of patterns across the board, to the size of the board itself. The system we propose is capable of automatically learning the propensity of local patterns from a library of games. Propensity and other local tactical information are fed into recursive neural networks, derived from a probabilistic Bayesian network architecture. The recursive neural networks in turn integrate local information across the board in all four cardinal directions and produce local outputs that represent local territory ownership probabilities. The aggregation of these probabilities provides an effective strategic evaluation function that is an estimate of the expected area at the end, or at various other stages, of the game. Local area targets for training can be derived from datasets of games played by human players. In this approach, while requiring a learning time proportional to N^4, skills learned on a board of size N^2 can easily be transferred to boards of other sizes. A system trained using only 9×9 amateur game data performs surprisingly well on a test set derived from 19×19 professional game data. Possible directions for further improvements are briefly discussed.

Keywords: Computer games; Computer Go; Machine learning; Evaluation function; Recursive neural networks; Knowledge transfer

1. Introduction

Go is an ancient board game – over 3000 years old (Cobb, 2002; Iwamoto, 1972) – that poses unique opportunities and challenges for artificial intelligence and machine learning. The rules of Go are deceptively simple: two opponents alternately place black and white stones on the empty intersections of an odd-sized square board, traditionally of size 19×19. The goal of the game, in simple terms, is for each player to capture as much territory as possible across the board by encircling the opponent's stones. This disarming simplicity of the rules of Go, however, conceals a formidable combinatorial complexity (Berlekamp & Wolfe, 1994). On a 19×19 board, there are approximately $3^{19\times19} = 10^{172.24}$ possible board configurations and, on average, on the order of 200–300 possible moves at each step of the game, preventing any form of semi-exhaustive search. For comparison purposes, the game of chess has a much smaller branching factor, on the order of 35–40 (Plaat, Schaeffer, Pijls, & de Bruin, 1996; Russell & Norvig, 2002). Today, computer chess programs, built essentially on search techniques and running on a simple desktop, can rival or even surpass the best human players. In contrast, and in spite of several decades of research efforts and progress in hardware speed, the best Go programs of today are easily defeated by an average human amateur player (Bouzy & Cazenave, 2001; Chen, 1990; Enzenberger, 1996, 2003; Fotland, 1993; Graepel, Goutrie, Kruger, & Herbrich, 2001).

Besides the intrinsic challenge of the game, and the non-trivial market created by over 100 million players worldwide, Go raises other important questions for our understanding of natural or artificial intelligence in the distilled setting created by the simple rules of the game, uncluttered by the endless complexities of the "real world". For instance, are there any "intelligent" solutions to Go and if so what would characterize them? Note that, to many observers, current computer solutions to chess appear to be "brute force", hence

* Corresponding author. Tel.: +1 949 8245809; fax: +1 949 8249813.
E-mail address: pfbaldi@ics.uci.edu (P. Baldi).

0893-6080/$ - see front matter © 2008 Elsevier Ltd. All rights reserved.
doi:10.1016/j.neunet.2008.02.002

"unintelligent". But is this perception correct, or an illusion—is there something like true intelligence beyond "brute force" and computational power? Where is Go situated in the apparent tug-of-war between intelligence and sheer computational power? To understand such questions, the methods used to develop computer Go programs are as important as the performance of the resulting players. This is particularly true in the context of certain current efforts aimed at solving Go by pattern matching within very large databases of Go games. How "intelligent" is pattern matching?

A second set of issues has to do with whether our attempts at solving Go ought to mimic how human experts play? Notwithstanding the fact that we do not know how human brains play Go, here the experience gained again with chess, as well as many other problems, clearly shows that mimicking the human brain is not necessary in order to develop good computer Go programs. However, we may still find useful inspiration in the way humans seem to play, for instance in trying to balance tactical and strategic goals.

Finally, and in connection with human playing, we note that humans in general do not learn to play Go directly on boards of size 19×19, but rather start by learning on smaller board sizes, typically 9×9. How can we develop algorithms capable of transferring knowledge from one board size to another?

The emergence of large datasets of Go game records, and the ability to record even larger datasets through Web and peer-to-peer technologies, opens the door for new pattern-matching and statistical machine learning approaches to Go. Here we take modest steps towards addressing these challenges by developing a scalable machine learning approach to Go capable of knowledge transfer across board sizes. Clearly good evaluation functions and search algorithms are essential ingredients of computer board-game systems (Abramson, 1990; Beal, 1990; Raiko, 2004; Ryder, 1971; Schrufer, 1989). Here we focus primarily on the problem of learning a good evaluation function for Go in a scalable way. We do include simple search algorithms in our system, as many other programs do, but these algorithms are not the primary focus. By scalability we imply that a main goal is to develop a system more or less automatically, using machine learning approaches, with minimal human intervention and handcrafting. The system ought to be able to transfer information across board sizes, for instance from 9×9 to 19×19.

We take inspiration in three ingredients that seem to be essential to the Go human evaluation process: the understanding of local patterns (Chen, 1992), the ability to combine patterns, and the ability to relate tactical and strategic goals. Our system is built to learn these three capabilities automatically and attempts to combine the strengths of existing systems while avoiding some of their weaknesses. The system is capable of automatically learning the propensity of local patterns from a library of games. Propensity and other local tactical information are fed into a set of recursive neural networks, derived from a probabilistic Bayesian network architecture. The networks in turn integrate local information across the board and produce local outputs that represent local territory ownership probabilities. The aggregation of these probabilities provides an effective strategic evaluation function that is an estimate of the expected area at the end (or at other stages) of the game. Local area targets for training can be derived from datasets of human games. The main results we present here are derived on a 19×19 board using a player trained using only 9×9 game data.

2. Data

Because the approach to be described emphasizes scalability and learning, we are able to train our systems at a given board size and use it to play at different sizes, both larger and smaller. Pure bootstrap approaches to Go have been tried where computer players are initialized randomly and play large numbers of games, such as evolutionary approaches or reinforcement learning (Sutton & Barto, 1998; Schrauldolph, Dayan, & Sejnowski, 1994). We have implemented these approaches and used them for small board sizes 5×5 and 7×7, where no training data were available to us. However, in our experience, these approaches do not scale up well to larger board sizes. For larger board sizes, better results are obtained using training data derived from records of games played by humans. We have used available data for training and validation at board sizes of 9×9, 13×13, and 19×19. The data are briefly described below.

Data for 9×9 **Boards**: This dataset consists of 3495 games. We randomly selected 3166 games (90.6%) for training, and the remaining 328 games (9.4%) for validation. Most of the games in this data set are played by amateurs. A subset of 424 games (12.13%) have at least one player with an olf ranking of 29, corresponding to a 3 kyu player.

Data for 13×13 **Boards**: This dataset consists of 4175 games. Most of the games, however, are played by rather weak players and therefore cannot be used effectively for training. For validation purposes, however, we retained a subset of 91 games where both players have an olf ranking greater than or equal to 25—the equivalent of a 6 kyu player.

Data for 19×19 **Boards**: This high-quality dataset consists of 1835 games played by professional players (at least 1 dan). A subset of 1131 games (61.6%) are played by 9 dan players (the highest possible ranking). This is the dataset used in Stern, Graepel, and MacKay (2005).

3. System architecture

3.1. Evaluation function, outputs, and targets

Because Go is a game about territory, it is sensible to try to compute "expected territory" as the evaluation function, and to decompose this expectation as a sum of local probabilities. More specifically, let $A_{ij}(t)$ denote the ownership of intersection ij on the board at time t during the game. At the end of a game (Benson, 1976; Chen & Chen, 1999), each intersection can be black, white, or neutral.[1]

[1] This is called "seki". Seki is a stalling situation where two live groups share liberties and neither group can fill these liberties without becoming a prisoner of the opponent.