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

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 produces 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.

Key words: computer games, computer Go, machine learning, evaluation function, recursive neural networks, knowledge transfer

1. Introduction

Go is an ancient board game–over 3,000 years old (Iwamoto, 1972; Cobb, 2002)–that poses unique opportunities and challenges for artificial intelligence and machine learning. The rules of Go are deceptively simple: two opponents alternatively 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 and 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 (Russell and Norvig, 2002; Plaat et al., 1996). 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 and Cazenave, 2001; Fotland, 1993; Chen, 1990; Graepel et al., 2001; Enzenberger, 1996, 2003).

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

* Corresponding author. Tel.: +1 949 8245809; fax: +1 824 9813.
Email address: pfbaldi@ics.uci.edu (Pierre Baldi).

so what would characterize them? Note that to many observers current computer solutions to chess appear to be "brute force", hence "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? Not withstanding 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 (Beal, 1990; Schrufer, 1989; Ryder, 1971; Abramson, 1990; Raiko, 2004). 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 where computer players are initialized randomly and play large numbers of games, such as evolutionary approaches or reinforcement learning (Sutton and Barto, 1998), have been tried (Schrauldolph et al., 1994). We have implemented these approaches and used them for small board sizes 5×5 and 7×7, where no training data was 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 9×9, 13×13, and 19×19. The data is briefly described below.

Data for 9×9 Boards: This dataset consists of 3,495 games. We randomly selected 3,166 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 4,175 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 or equal to 25–the equivalent of a 6 kyu player.

Data for 19×19 Boards: This high-quality dataset consists of 1,835 games played by professional players (at least 1 dan). A subset of 1,131 games (61.6%) are played by 9 dan players (the highest possible ranking). This is the dataset used in Stern et al. (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 (Chen and Chen, 1999; Benson, 1976), each intersection can be black, white, or neutral [1] . Black is represented as 1, white as 0, and neutral as 0.5. The same scheme with 0.5 for empty intersections, or more complicated schemes, can be used to represent ownership at various intermediate stages of the game. Let $O_{ij}(t)$ be the output of the learning system at intersection ij at time t in the game. Likewise, let $T_{ij}(t)$ be the corresponding training target. In the most simple case, we can use $T_{ij}(t) = A_{ij}(T)$, where T denotes the end of the game. In this case, the output $O_{ij}(t)$ can be interpreted as the probability $P_{ij}(t)$, estimated at time t, of owning the ij intersection at the end of the game. Likewise, $\sum_{ij} O_{ij}(t)$ is the estimate, computed at time t, of the total expected area at the end of the game.

Propagation of information provided by targets/rewards computed at the end of the game only, however, can be problematic. With a dataset of training examples, this problem can be addressed because intermediary area values $A_{ij}(t)$ are available for training for any t. In the simulations presented here, we use a simple scheme

$$T_{ij}(t) = (1 - w)A_{ij}(T) + wA_{ij}(t + k) \tag{1}$$

Here $w \geq 0$ is a parameter that controls the convex combination between the area at the end of the game and the area at some step $t + k$ in the nearer future. The special case $w = 0$ corresponds to the simple case described above where only the area at the end of the game is used in the target function.

To learn the local probabilities and hence the evaluation function from the targets, we propose to use a directed graphical model (Bayesian network) which in turn leads to a directed acyclic graph recursive neural network (DAG-RNN) architecture.

3.2. *DAG-RNN Architectures*

The basic architecture is closely related to an architecture originally proposed for a problem in a completely different area – the prediction of protein contact maps (Pollastri and Baldi, 2002; Baldi and Pollastri, 2003). As a Bayesian network, the basic architecture can also be viewed as a generalization of input-output bidirectional HMMs (Baldi et al., 1999) to two dimensional problems. The architecture is described in terms of the DAG (Directed Acyclic Graph) of Figure 1 where the nodes are arranged in 6 lattice planes reflecting the Go board spatial organization. Each plane contains $N \times N$ nodes arranged on the vertices of a square lattice. In addition to the input and output planes, there are four hidden planes for the lateral propagation and integration of information across the Go board in each cardinal direction. Thus each intersection ij in a $N \times N$ board is associated with six variables. These variables consist of the input I_{ij}, four hidden variables $H_{ij}^{NE}, H_{ij}^{NW}, H_{ij}^{SW}, H_{ij}^{SE}$, and the output O_{ij}. Within each hidden plane, the edges of the quadratic lattice are oriented towards one of the four cardinal directions (NE, NW, SE, and SW). Directed edges within a column of this architecture are given in Figure 1 and run from the input node to the hidden nodes, and from the input and hidden nodes to the output node.

To simplify belief propagation and learning, in a DAG-RNN the probabilistic relationship between a node variable and its parent node variables is replaced by a deterministic relationship parameterized by a neural network. Recursivity arises from weight sharing or stationarity, i.e. from the plate structure of the Bayesian network, when the same neural network is replicated at multiple locations. Thus the previous Bayesian network architecture, leads to a DAG-RNN architecture consisting of five neural networks in the form

$$\begin{cases} O_{ij} = \mathcal{N}_O(I_{ij}, H_{ij}^{NW}, H_{ij}^{NE}, H_{ij}^{SW}, H_{ij}^{SE}) \\ H_{ij}^{NE} = \mathcal{N}_{NE}(I_{ij}, H_{i-1j}^{NE}, H_{ij-1}^{NE}) \\ H_{ij}^{NW} = \mathcal{N}_{NW}(I_{ij}, H_{i+1j}^{NW}, H_{ij-1}^{NW}) \\ H_{ij}^{SW} = \mathcal{N}_{SW}(I_{i,j}, H_{i+1j}^{SW}, H_{ij+1}^{SW}) \\ H_{ij}^{SE} = \mathcal{N}_{SE}(I_{ij}, H_{i-1j}^{SE}, H_{ij+1}^{SE}) \end{cases} \tag{2}$$

where, for instance, $\mathcal{N}_O$ is a single neural network for the output that is shared across all spatial output locations. In addition, since Go is "isotropic" we use a single network shared across the four hidden planes. Thus $\mathcal{N}_{NE}$ is a single neural network for the lateral North-East propagation of information that is shared across all spatial location in the North-East plane. In addition, Go involves strong boundaries effects and therefore we add one neural network $\mathcal{N}_C$ for the corners, shared across all four corners, and one neural network $\mathcal{N}_S$ for each side position, shared across all side positions and across all four sides.

In short, the entire Go DAG-RNN architecture is described by four feedforward NNs (corner, side, lateral, output) that are shared at all corresponding locations. For each one of these feedforward neural networks, we have experimented with several architectures, but we typically use a single hidden layer. The DAG-RNN in the main simulation results uses 16 hidden nodes and 8 output nodes for the lateral propagation networks, and 16 hidden nodes and one output node for the output network. All transfer functions are logistic. The total number of free parameters is 5,642 for

[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 prisoner of the opponent.

Table 1
Input features. The first three features–stone type, influence, and propensity–are properties associated with the corresponding intersection and a fixed number of surrounding locations. The other properties are group properties involving variable numbers of neighboring stones.

Feature	Description
b,w,e	stone type: black, white or empty
influence	influence from the stones of the same color and the opposing color
propensity	local statistics computed from 3×3 patterns in the training data (section 3.3)
N_{eye}	number of true eyes
N_{1st}	number of liberties, which is the number of empty intersections connected to a group of stones. We also call it the 1st-order liberties
N_{2nd}	number of 2nd-order liberties, which is defined as the liberties of the 1st-order liberties
N_{3rd}	number of 3rd-order liberties, which is defined as the liberties of the 2nd-order liberties
N_{4th}	number of 4th-order liberties, which is defined as the liberties of the 3rd-order liberties
O_{1st}	features of the weakest connected opponent group (stone type, number of liberties, number of eyes)
O_{2nd}	features of the second weakest connected opponent group (stone type, number of liberties, number of eyes)

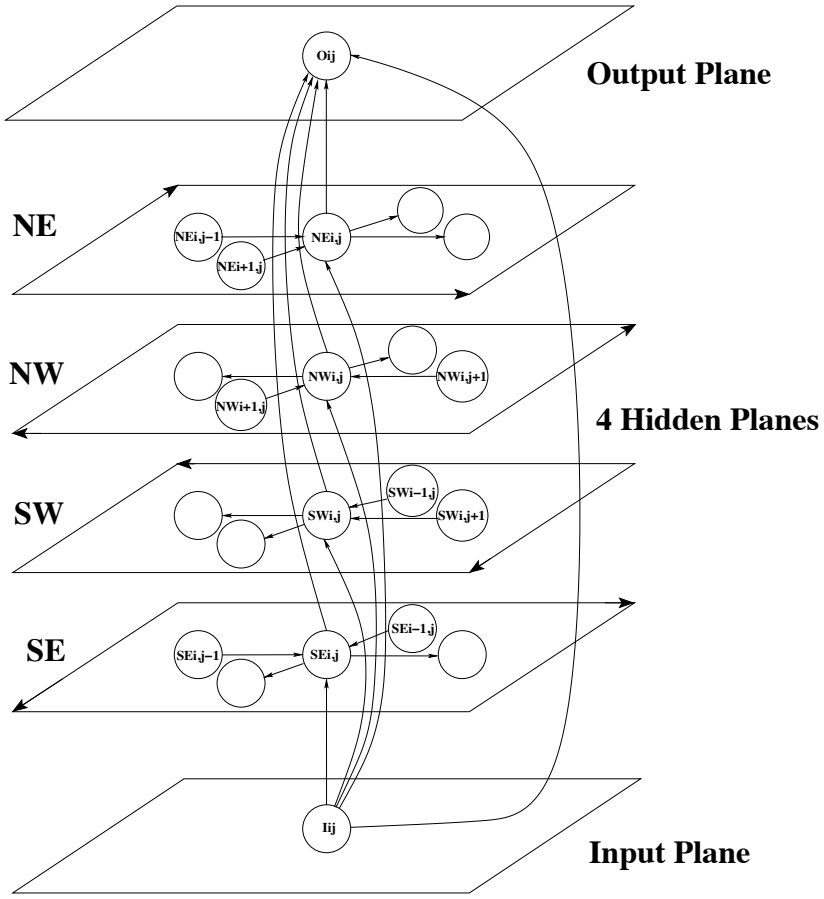


Fig. 1. The DAG underlying the recursive neural network architecture. Nodes are regularly arranged in one input plane, one output plane, and four hidden planes, one for each cardinal direction. In each plane, nodes are arranged on a square lattice. All the directed edges of the square lattice in each hidden plane are oriented in the direction of one of the four possible cardinal corners: NE, NW, SW, and SE. Node NE_{ij}, for instance, receives directed connections from its parent nodes $NE_{i,j-1}$ and $NEi+1,j$. Additional directed edges run vertically in each column ij. The input node I_{ij} is connected to four corresponding hidden nodes, one for each hidden plane. The input node and the hidden nodes are connected to the output node O_{ij}.

the architecture used in the Results. Note that in spite of the deterministic reparameterization of the Bayesian network, the overall model remains probabilistic in the sense that the outputs O_{ij} of the logistic functions at each spatial location vary between zero and one, and can be interpreted as territory ownership probabilities.

Because the underlying graph is acyclic, these networks can be unfolded in space and training can proceed by simple gradient descent (back-propagation) taking into account relevant symmetries and weight sharing. In terms of symmetries, for instance, all game positions can be used for training a 'black' player that is about to make a move, simply by switching the color of the stones on those training move where 'white' is about to play. Likewise, rotated board positions can be used for training. Because of the weight sharing, networks trained at one board size can immediately be applied to any other board size, thus providing a simple mechanism for reusing and extending acquired knowledge. For a board of size $N \times N$, the training procedure scales like $O(WMN^4)$ where W is the number of adjustable weights, and M is the number of training games. There are roughly N^2 board positions in a game and, for each position, N^2 outputs O_{ij} need to be trained, hence the $O(N^4)$ scaling. Both the game records and the positions within each selected game record are randomly selected, without replacement, during training. Weights are updated essentially on line, once every 10 game positions with a learning rate of 0.00001. Training a single player on the 9×9 dataset takes on the order of a week on a current desktop computer, corresponding roughly to 50 training epochs at 3 hours per epoch.

3.3. *Inputs*

At a given board intersection, the input vector I_{ij} has multiple components from the list given in Table 1. The first three components–stone type, influence, and propensity–are associated with the corresponding intersection and a *fixed* number of surrounding locations. Influence and propensity are described below in more detail. The remaining features correspond to *group* properties (Chen, 1989) involving variable numbers of neighboring stones and are self explanatory for those familiar with Go. The group G_{ij} associated with a given intersection is the maximal set of stones of the same color that are connected to it. Neighboring (or connected) opponent groups of G_{ij} are groups of the opposite color that are directly connected (adjacent) to G_{ij}. The idea of using higher order liberties is taken from Werf (Werf et al., 2003). O_{1st} and O_{2nd} provide the number of true eyes and the number of liberties of the weakest and the second weakest neighboring opponent groups. Weakness here is defined in alphabetical order with respect to the number of eyes first, followed by the number of liberties. Typical input vectors used in the experiments consist of a subset of these input features and typically have on the order of one hundred components.

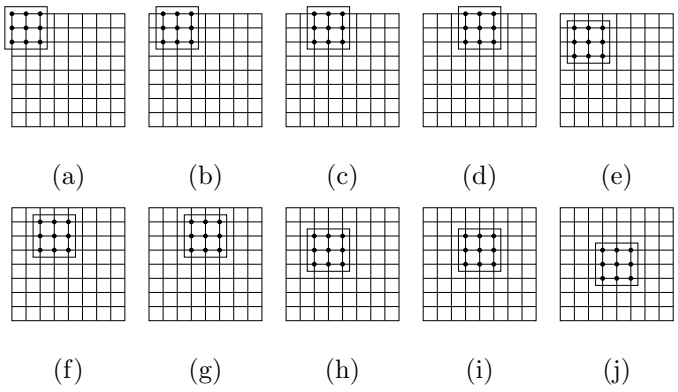


Fig. 2. Ten unique 3×3 grid locations on a 9×9 board under rotation and mirror symmetries.

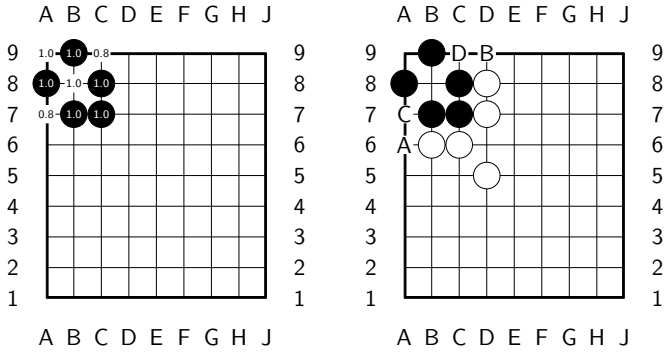
Influence: We use two types of influence calculation. Both algorithms are based on Chen's method (Chen, 2002). The first one is an exact implementation of Chen's method, which has a non-stringent influence propagation rule. The second one uses a stringent influence propagation rule. In Chen's method, any opponent stone can block the propagation of influence. In the stringent implementation, an opponent stone can block the propagation of influence if and only if it is stronger than the stone emitting the influence. Strength is again defined in alphabetical order with respect to the number of eyes first, followed by the number of liberties.

Propensity–Automated Learning and Scoring of a Pattern Library:

We develop a method to learn local patterns and their value automatically from a database of games. The basic method is illustrated in the case of 3×3 patterns used in some of the simulations. Considering rotation and mirror symmetries, there are 10 unique locations for a 3×3 window on a 9×9 board, as shown in Figure 2 (Ralaivola et al., 2005). For each such 3×3 window, there are 3^9 possible stone patterns. The 3×3 windows at different board locations have different symmetries. For example, the unique location at the center of the board has 8 symmetries (4 rotations and 2 mirror symmetries), while the corner location which can occur at 4 positions has only 2 mirror symmetries. It is easy to see that for each unique location, the product of the number of symmetries and the number of board positions where it can occur must equal 8. Given any 3×3 pattern of stones in a board window and a set of games, we then compute nine numbers, one for each intersection. These numbers are local indicators of strength or propensity. The propensity $S_{ij}(p)$ of each intersection ij associated with stone pattern p and a 3×3 window w is defined as:

$$S_{ij}^{w}(p) = \frac{NB_{ij}(p) - NW_{ij}(p)}{NB_{ij}(p) + NW_{ij}(p) + C} \tag{3}$$

where $NB_{ij}(p)$ is the number of times that pattern p ends with a black stone at intersection ij at the end of the games in the data, and $NW_{ij}(p)$ is the same for a white stone.



(a) (b)

Fig. 3. Example of a 3×3 pattern with high propensity values. The numbers on the left board represent the calculated propensities at each intersection. The right board represents a corresponding tactical battle that might occur in the course of a game when it is white's turn to place a stone. To attack the black pattern, white would need to take both C and D. However, none of these positions can be taken directly. Hence white places a stone in A, to which black can respond by placing a stone in C, or by placing a stone in a different region of the board. If black places a stone in a different region of the board, white has the option to take B, forcing black to take D.

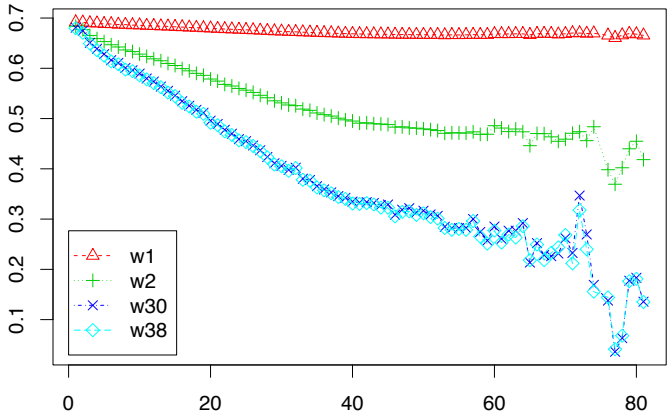


Fig. 4. Validation error versus game phase. Curves correspond to a DAG-RNN system trained and validated using 9×9 amateur game data. The phase of the game is represented on the x axis by the number of stones present on the board. The y axis correspond to the relative entropy between the predicted membership probabilities and the true probabilities, computed on the validation dataset. The four curves correspond to the validation error of the DAG-RNN after 1, 2, 30, and 38 epochs of training.

Both $NB_{ij}(p)$ and $NW_{ij}(p)$ are computed taking into account the location and the symmetries of the corresponding window w. C plays a regularizing role in the case of rare patterns and is set to 1 in the simulations. Thus $S^w_{ij}(p)$ is an empirical normalized estimate of the local differential propensity towards conquering the corresponding intersection in the local context provided by the corresponding pattern and window. An example of propensity calculations is given in Figure 3 showing one of the highest ranking patterns calculated from the 9×9 training data set.

In general, a given intersection ij on the board is covered by several 3×3 windows. For instance, a corner location is associated with only one 3×3 window, whereas the center location is associated with nine different windows. Thus, for a given intersection ij on a given board, we can compute a value $S^w_{ij}(p)$ for each different window that contains the intersection. In the following simulations, a single final value $S_{ij}(p)$ is computed by averaging over the different w's. However, more complex schemes that retain more information can easily be envisioned by, for instance: (1) computing also the standard deviation of the $S^w_{ij}(p)$ as a function of w; (2) using a weighted average, weighted by the importance of the window w; and (3) using the entire set of $S^w_{ij}(p)$ values, as w varies around ij, to augment the input vector.

3.4. *Move Selection and Search*

For a given position, the next move can be selected using one-level search by considering all possible legal moves and computing the estimate of the total expected area $E = \sum_{ij} O_{ij}(T)$ at the end of the game, or some intermediate position, or a combination of both, where $O_{ij}(t)$ are the outputs (predicted probabilities) of the DAG-RNNs. The next move can be chosen by maximizing this evaluation function (1-ply search). Alternatively, the next move can be chosen from all the legal moves with a probability proportional to $e^{E/Temp}$, where $Temp$ is a temperature parameter (Brugmann, 1993; Schrauldolph et al., 1994; Stern et al., 2005; Bouzy and Helmstetter, 2003). The value of $Temp$ plays an important role and is discussed in the Results. We have also experimented with a few other simple search schemes, such as 2-ply search (MinMax).

4. Results

We have trained a large number of players using the methods described above. In the absence of training data, we used pure bootstrap approaches (e.g. reinforcement learning) at sizes 5×5 and 7×7 with results that were encouraging but clearly insufficient. Not surprisingly, when used to play at larger board sizes, the RNNs trained at these small board sizes yield rather weak players. The quality of most 13×13 games available to us is too poor for proper training, although a small subset can be used for validation purposes. Furthermore, we do not have any data for sizes $N = 11, 15$, and 17. Thus the most interesting results we report are derived by training the DAG-RNNs using the 9×9 game data, and using these systems to play at 9×9 and, more importantly, at larger board sizes. Several 9×9 players achieve top comparable performance. For conciseness, here we report the results obtained with one of them, trained with target parameters $w = 0.25$ and $k = 2$ in Equation 1,

Figure 4 shows how the validation error changes as training progresses. Validation error here is defined as the relative entropy between the output probabilities produced by the RNN and the target probabilities, computed on

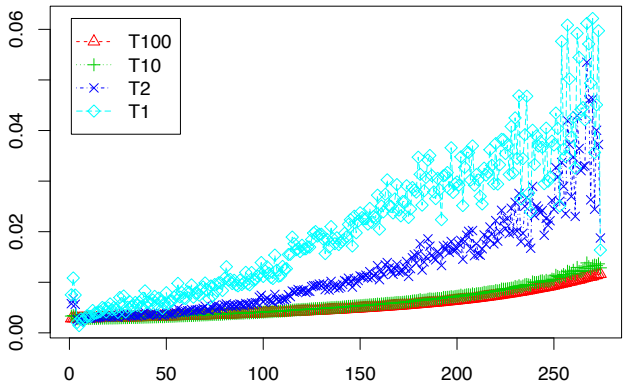


Fig. 5. Average move probabilities versus game phase when the temperature parameter $Temp$ is equal to 1, 2, 10 and 100 respectively. The moves are from the 19×19 human expert dataset. Probabilities are computed using a DAG-RNN system trained on the amateur 9×9 dataset.

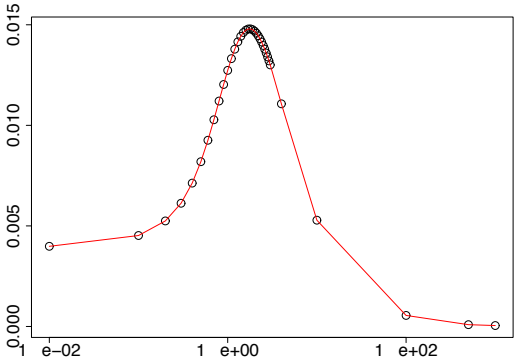


Fig. 6. The average probability of human expert moves after removing the 'matched' moves. The x axis represents temperature on a logarithmic scale. The average is computed using the 19×19 data with moves 80-83. The probabilities are computed from the output of the 9×9 system trained on human amateur data. The maximum has a value of $0.014796 \simeq 1/68$ and is reached for $Temp = 0.833$.

the validation data. The validation error decreases quickly during the first epochs, each epoch corresponding to one pass through the training data. In this case, no substantial decrease in validation error is observed after 30 training epochs. Note also how the error is smaller towards the end of the game due to the reduction in uncertainty as the game progresses.

For any input board position, the trained DAG-RNNs produce a set of local probabilities O_{ij}. To derive a corresponding player, or to assess the quality of the system by examining its ability to retrieve or rank human expert moves, it is necessary to convert these local probabilities into global probabilities of individual moves and to rank them. One important issue to convert local output probabilities into global move probabilities proportional to $\sum_{ij} O_{ij}/Temp$ is the choice of the temperature parameter $Temp$.

Figure 5 shows the average probability of the move played by human experts in the 19×19 board dataset, during different phases of the game. These probabilities are computed using the DAG-RNN trained using the 9×9 amateur dataset. The four curves correspond to four different values of the temperature parameter ($Temp = 1, 2, 10$ and 100). Overall, as expected, when the temperature decreases, the average probability increases. Furthermore, this probability increases also with the phase of the game. When the temperature is zero, the system assigns probability one to the move associated with the largest predicted area, and zero to all the other moves. Thus the maximum value of the average probability for the human moves is reached when $Temp = 0$ and is equal to the fraction of matched moves, i.e. the fraction of human moves that are ranked first according to the system. This fraction varies with the phase of the game. For instance, it is close to 10% for the 19×19 test dataset containing 2601 moves from games played by human experts, taken between move 80 and move 83.

A choice of $Temp = 0$, however, is not sound since it corresponds to a state of overconfidence and exclusive focus on a single move. Thus to determine the value of the temperature parameter, we maximize the average probability of the human moves after removing all the matched moves (Figure 6) which yields in this case an optimal temperature parameter $Temp = 0.833$ for moves 80-83.

With this choice of temperature we can now compute the probability of any move, rank all moves accordingly, and evaluate the performance of the system by looking at how it ranks and scores the moves played by human experts. Thus we can compute the average probability of moves played by human expert players according to the DAG-RNN or other probabilistic systems such as (Stern et al., 2005). In Table 2, we report such probabilities for several systems and at different board sizes. In this test, the 9×9 validation data set has 1,298 moves from amateur games between move 18 and 21. The 19×19 test data set has 2,601 moves from professional games between move 80 and 83. The value of temperature T is 1.25 for board size 9×9, 0.909 for 13×13, and 0.833 for 19×19. For size 19×19, we use the same test set used in (Stern et al., 2005). Boltzmann5 and BoltzmannLiberties are their results reported in the pre-published version of their NIPS 2004 paper. At this size, the probabilities in the table are computed using the 80-83 moves of each game. For boards of size 19×19, a random player that selects moves uniformly at random among all legal moves assigns a probability of $1/281 = 1/[(19 \times 19) - 80]$ to the moves played by professional players in the dataset. BoltzmannLiberties was able to improve this probability to 1/194. The best DAG-RNNs trained using amateur data at 9×9 are capable of bringing this probability further down by over one order of magnitude to 1/15. This probability corresponds to the fact that roughly 10% of human expert moves are ranked at the top by the DAG-RNN. Note that these information-retrieval metrics are only indicative of performance since, for instance, the move played by human experts may not always be the best possible move. But, to summarize, the DAG-RNN trained on amateur 9×9 data improves the probability of human expert moves on boards

Table 2
Probabilities assigned by different systems to moves played by human players. Random players choose one move uniformly at random over all legal moves. The 9×9 validation dataset has 1,298 moves from amateur games taken between move 18 and 21. The 19×19 dataset set has 2,601 moves from human expert games taken between move 80 and 83. For boards of size 19×19, a random player assigns probability 1/281 to the human expert moves. The DAG-RNN trained on the 9×9 amateur dataset, assigns an average probability of 1/15 to the human expert moves. About 1/10 human expert moves are ranked first among all possible legal moves by the DAG-RNN.

Board Size	System	Log Probability	Probability
9×9	Random player	-4.13	1/62
9×9	RNN(1-ply search)	-1.86	1/7
13×13	Random player	-4.88	1/132
13×13	RNN(1-ply search)	-2.27	1/10
19×19	Random player	-5.64	1/281
19×19	Boltzmann5	-5.55	1/254
19×19	BoltzmannLiberties	-5.27	1/194
19×19	RNN(1-ply search)	-2.70	**1/15[10% ranked 1st]**

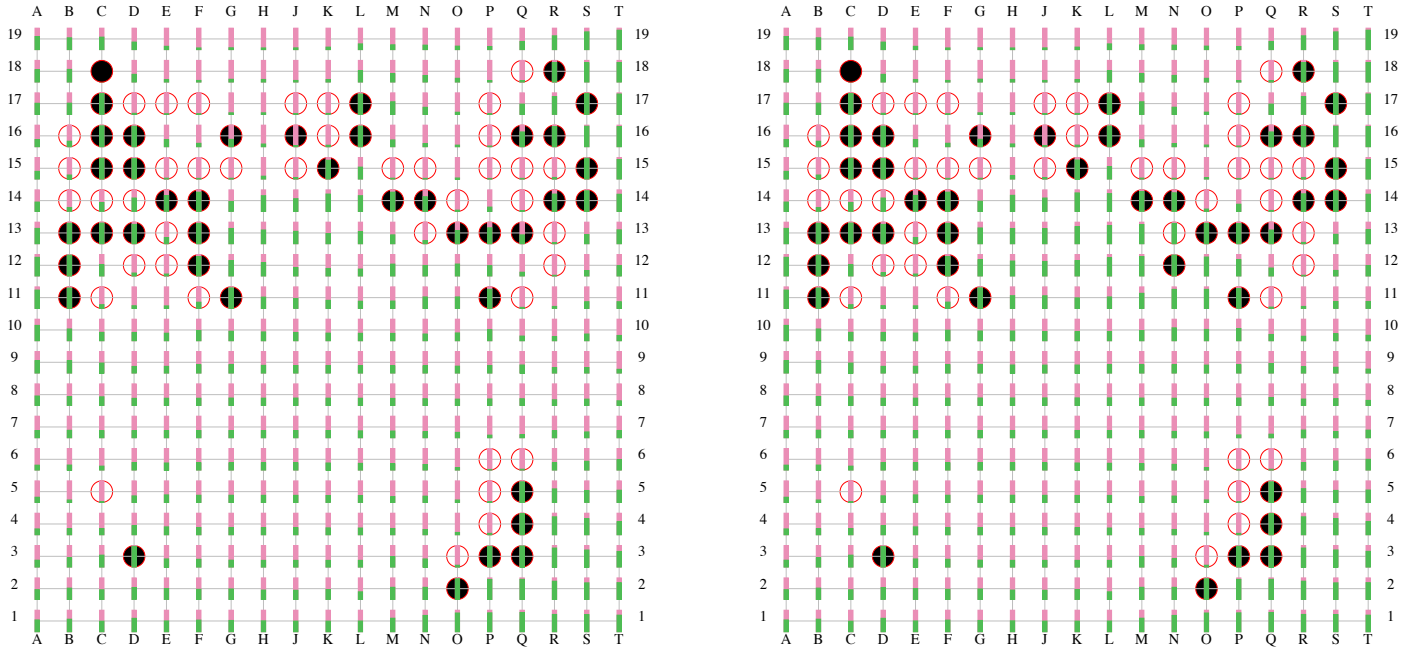


Fig. 7. Example of an outstanding move based on territory predictions made by the DAG-RNN. For each intersection, the height of the green bar represents the estimated probability that the intersection will be owned by black at the end of the game. The figure on the left shows the predicted probabilities if black passes. The figure on the right shows the predicted probabilities if black makes the move at N12. The overall predicted territory is 153.623 if black passes, and 164.007 if black plays N12. Move N12 causes the greatest increase in area and is the top-ranked move for the DAG-RNN. Indeed this is the move selected in the game played by Zhou, Heyang (black, 8 dan) and Chang, Hao (white, 9 dan) on 10/22/2000. Black won the game by 2.75 point.

of size 19×19 by more than one order of magnitude over random.

Figure 8 provides a ROC-like curve by displaying the percentage of moves made by professional human player on boards of size 19×19 that are contained in the m top-ranked moves according to the DAG-RNN trained on 9×9 amateur data, for various values of m across all phases of the game. For instance, when there are 80 stones on the board, and hence on the order of 300 legal moves available, there is a 50% chance that a move selected by a very highly ranked human player (dan 9) is found among the top 30 choices produced by the DAG-RNN.

A remarkable example where the top ranked move according to the DAG-RNN coincides with the move actually played in a game between two very highly-ranked players is given in Figure 7, illustrating also the underlying probabilistic territory calculations.

5. Discussion

In summary, we have designed a DAG-RNN approach for the game of Go and demonstrated that it can learn

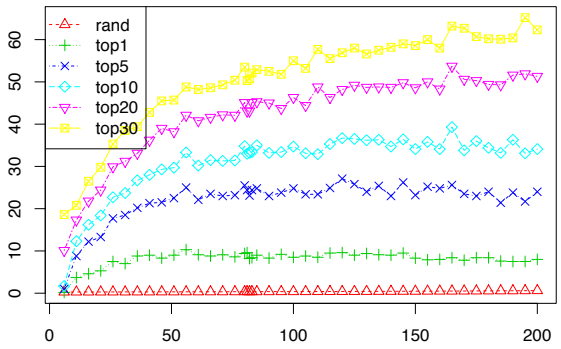


Fig. 8. Percentage of moves made by expert human players on boards of size 19×19 that are contained in the m top-ranked moves according to the DAG-RNN trained on 9×9 amateur data, for various values of m. The baseline associated with the red curve corresponds to a random uniform player.

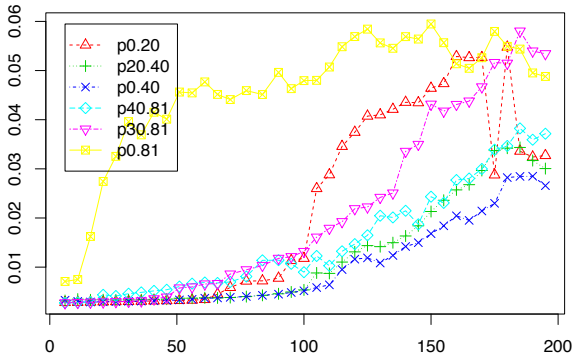


Fig. 9. Probability of moves played by human experts in the 19×19 dataset versus game phase. Probabilities are computed according to 6 DAG-RNN players trained using data from different phases of the game in the 9×9 dataset. Each player, represented as pn1.n2, is trained using data between move n1 and move n2. The target is the board configuration at n2. Probabilities are evaluated on the first 200 moves. Player p0.81 trained on all moves achieves the best results, particularly at early stages of the game.

territory predictions fairly well. Systems trained using only a set of 9×9 amateur games achieve surprisingly good performance on a 19×19 test set that contains 1,835 games played by human experts. While the system described here achieves over one order of magnitude of improvement in the probability of human expert moves over random, it is clear that one order of magnitude of improvement in this probability remains to be made in order to achieve human expert competence on boards of size 19×19. The methods and results presented clearly point also to several possible related directions of improvement.

A first obvious direction is to train systems of size greater than 9×9. In spite of the $O(N^4)$ scaling issue, training 19×19 systems is possible over a period of months. One issue, however, is the amount of training data. The data sets used are a good starting point but it ought to be possible to collect much larger datasets. Creating a large, freely available, repository of game records should be a key enabling step in order to develop better players and should have high priority. With current database and Web technologies, the creation of very large annotated game repositories is certainly possible.

A second direction is to further exploit patterns that are larger than 3×3, especially at the beginning of the game when the board is sparsely occupied and matching of large patterns is possible using, for instance, Zobrist hashing techniques (Zobrist, 1970). Techniques related to pattern matching are also likely to benefit from large databases and directly impinge on the fundamental question of the nature of intelligence raised in the Introduction, since a purely pattern matching approach is likely to be regarded as a brute force approach.

A third direction is to combine different players, such as players trained at different board sizes, or players trained on different phases of the game. Perhaps surprisingly, in preliminary results so far we have not been able to gain any significant improvements by training different players specialized on different phases of the game (Figure 9).

Finally, it is also clear that the exploration/exploitation tradeoff plays a key role in Go and better search strategies ought to be investigated, together with how one can effectively combine search and learning strategies together. Simple exhaustive schemes, such as 1-ply search are too weak both in terms of their shallowness and their time complexity (Table 3). With W adjustable weights, we have seen that the learning complexity scales like $O(WMN^4)$, with M training games. The 1-ply playing complexity on the other hand scales like $O(WN^6)$ ($O(N^2)$ moves in a game with $O(N^2)$ possible moves at each position and $O(N^2)$ sub-moves to explore).

Table 3
Time complexity for training and playing for different board sizes. Training column: approximate time complexity for training on one game ($O(WN^4)$). Playing column: approximate time complexity for playing one game using 1-ply search ($O(WN^6)$). The training time for Go boards of size 5×5 is used as the basic unit of time.

Board Size	Training	Playing
5×5	1	25
9×9	10.5	34
13×13	45.8	308.9
19×19	208.5	3010.9

Interestingly, random search methods (Abramson, 1990), have been used effectively to design a 9×9 Go program (Raiko, 2004), which can run on a palm computer. Random search methods, together with further analyses of the exploration/exploitation tradeoff in the armed bandit problem and in tree search (Auer et al., 2002; Kocsis and Szepesvari, 2006; Coulom, 2006), have also been used very recently to design an effective 9×9 player (Gelly et al., 2006). While random search methods can scale up to 19×19 in speed, they do not seem to scale up in player quality. Thus the problem of finding effective search methods, that preserve the speed and scalability of random search but achieve better results, possibly by leveraging an evaluation function, is likely to play a central role in the ongoing quest for intelligent Go programs.

Acknowledgments

The work of PB and LW has been supported by a Laurel Wilkening Faculty Innovation award and awards from NSF, BREP, and Sun Microsystems to PB. We would like to thank Jianlin Chen for developing a Go graphical user interface, Nicol Schraudolph for providing the 9×9 and 13×13 data, and David Stern for providing the 19×19 data.

References

Abramson, B., 1990. Expected-outcome: a general model of static evaluation. IEEE Transactions on Pattern Analysis and Machine Intelligence 12 (2), 182–193.

Auer, P., Cesa-Bianchi, N., Fischer, P., 2002. Finite-time analysis of the multiarmed bandit problem. Machine Learning 47 (2-3), 235–256.

Baldi, P., Brunak, S., Frasconi, P., Pollastri, G., Soda, G., 1999. Exploiting the past and the future in protein secondary structure prediction. Bioinformatics 15, 937–946.

Baldi, P., Pollastri, G., 2003. The principled design of large-scale recursive neural network architectures–DAG-RNNs and the protein structure prediction problem. Journal of Machine Learning Research 4, 575–602.

Beal, D. F., 1990. A generalised quiescence search algorithm. Artificial Intelligence 43 (1), 85–98.

Benson, D. B., 1976. Life in the game of Go. Information Science 10, 17–29.

Berlekamp, E., Wolfe, D., 1994. Mathematical Go–Chilling gets the last point. A K Peters, Wellesley, MA.

Bouzy, B., Cazenave, T., 2001. Computer Go: An AI-oriented survey. Artificial Intelligence 132 (1), 39–103.

Bouzy, B., Helmstetter, B., 2003. Monte Carlo Go developments. Advances in Computer Games conference 10.

Brugmann, B., 1993. Monte Carlo Go.

Chen, K., 1989. Group identification in computer Go. Heuristic Programming in Artificial Intelligence: the First Computer Olympiad 1, 195–210.

Chen, K., 1990. The move decision process of Go Intellect. Computer Go 14, 9–17.

Chen, K., 1992. Attack and defense. Heuristic Programming in Artificial Intelligence 3, 146–156.

Chen, K., Chen, Z., 1999. Static analysis of life and death in the game of Go. Information Sciences 121 (1-2), 113–134.

Chen, Z., 2002. Semi-empirical quantitative theory of Go part 1: Estimation of the influence of a wall. ICGA Journal 25 (4), 211–218.

Cobb, W. S., 2002. The Book of GO. Sterling Publishing Co., New York, NY.

Coulom, R., 2006. Efficient selectivity and backup operators in monte-carlo tree search. In: Proceedings of the 5th International Conference on Computers and Games, Turin, Italy.

Enzenberger, M., 1996. The integration of a priori knowledge into a Go playing neural network.

Enzenberger, M., 2003. Evaluation in Go by a neural network using soft segmentation. 10th Advances in Computer Games conference 10, 97–108.

Fotland, D., 1993. Knowledge representation in the Many Faces of Go.

Gelly, S., Wang, Y., Valizadegan, H., Teytaud, O., 2006. UCT and MoGo: exploration vs exploitation in computer-go. In: Demos of Advances in Neural Information Processing Systems.

Graepel, T., Goutrie, M., Kruger, M., Herbrich, R., 2001. Learning on graphs in the game of Go. In: Proceddings of the International Conference on Artificial Neural Networks, ICANN 2001. pp. 347–352.

Iwamoto, K., 1972. GO for Beginners. Pantheon Books, New York, NY.

Kocsis, L., Szepesvari, C., 2006. Bandit-based monte-carlo planning. In: ECML.

Plaat, A., Schaeffer, J., Pijls, W., de Bruin, A., 1996. Exploiting graph properties of game trees. In: 13th National Conference on Artificial Intelligence (AAAI'96). pp. 234–239.

Pollastri, G., Baldi, P., 2002. Prediction of contact maps by giohmms and recurrent neural networks using lateral propagation from all four cardinal corners. Bioinformatics 18, S62–S70.

Raiko, T., 2004. The Go-playing program called Go81. In: Proceedings of the Finnish Artificial Intelligence Conference, STeP 2004.

Ralaivola, L., Wu, L., Balid, P., 2005. SVM and Pattern-Enriched Common Fate Graphs for the game of Go. ESANN 2005 27-29, 485–490.

Russell, S. J., Norvig, P., 2002. Artificial Intelligence: A Modern Approach, 2nd Edition. Prentice Hall.

Ryder, J. L., 1971. Heuristic analysis of large trees as generated in the game of Go. Ph.D. thesis, Department of Computer Science, Standford University.

Schrauldolph, N. N., Dayan, P., Sejnowski, T. J., 1994. Temporal difference learning of position evaluation in the game of Go. In: Advances in Neural Information Processing Systems 6. pp. 817–824.

Schrufer, G., 1989. A strategic quiescence search. ICCA Journal 12 (1), 3–9.

Stern, D. H., Graepel, T., MacKay, D. J. C., 2005. Modelling uncertainty in the game of Go. In: Advances in Neural Information Processing Systems 17. pp. 1353–1360.

Sutton, S., Barto, A., 1998. Reinforcement Learning - An introduction. The MIT Press, Cambridge, Massachusetts, London, England.

Werf, E., Herik, H., Uiterwijk, J., 2003. Learning to score final positions in the game of Go. In: Advances in Computer Games: Many Games, Many Challenges. pp. 143–158.

Zobrist, A. L., 1970. A new hashing method with application for game playing. Technical report 88, University of Wisconsin, April 1970. Reprinted in ICCA Journal, 13(2), (1990), pp. 69-73.