Beyond Monte Carlo Tree Search: Playing Go with Deep Alternative Neural Network and Long-Term Evaluation

Jinzhuo Wang, Wenmin Wang, Ronggang Wang, Wen Gao[†]
School of Electronics and Computer Engineering, Peking University
[†]School of Electronics Engineering and Computer Science, Peking University
jzwang@pku.edu.cn, wangwm@ece.pku.edu.cn, rgwang@ece.pku.edu.cn, wgao@pku.edu.cn

Abstract

Monte Carlo tree search (MCTS) is extremely popular in computer Go which determines each action by enormous simulations in a broad and deep search tree. However, human experts select most actions by pattern analysis and careful evaluation rather than brute search of millions of future interactions. In this paper, we propose a computer Go system that follows experts way of thinking and playing. Our system consists of two parts. The first part is a novel deep alternative neural network (DANN) used to generate candidates of next move. Compared with existing deep convolutional neural network (DCNN), DANN inserts recurrent layer after each convolutional layer and stacks them in an alternative manner. We show such setting can preserve more contexts of local features and its evolutions which are beneficial for move prediction. The second part is a long-term evaluation (LTE) module used to provide a reliable evaluation of candidates rather than a single probability from move predictor. This is consistent with human experts nature of playing since they can foresee tens of steps to give an accurate estimation of candidates. In our system, for each candidate, LTE calculates a cumulative reward after several future interactions when local variations are settled. Combining criteria from the two parts, our system determines the optimal choice of next move. For more comprehensive experiments, we introduce a new professional Go dataset (PGD), consisting of $253,233$ professional records. Experiments on GoGoD and PGD datasets show the DANN can substantially improve performance of move prediction over pure DCNN. When combining LTE, our system outperforms most relevant approaches and open engines based on MCTS.

Introduction

Go is a game of profound complexity and draws a lot attention. Although its rules are very simple (Müller 2002), it is difficult to construct a suitable value function of actions in most of situations mainly due to its high branching factors and subtle board situations that are sensitive to small changes.

Previous solutions focus on simulating future possible interactions to evaluate candidates. In such methods, Monte Carlo tree search (MCTS) (Gelly and Silver 2011) is the most popular one, which constructs a broad and deep search tree to simulate and evaluate each action. However, the playing strength of MCTS-based Go programs is still far from human-level due to its major limitation of uneven performance. Well known weaknesses include *capturing race* or *semeai*, positions with multiple cumulative evaluation errors, *ko fights*, and close endgames (Rimmel et al. 2010; Huang and Müller 2013). We attribute it to the following reasons. First, the effectivity of truncating search tree is based on prior knowledge and far away from perfect play (Müller 2002). Second, when the board is spacious especially at *opening*, simulation is expensive and useless. Besides, the outputs of leaves in Monte Carlo tree are difficult to be precisely evaluated (Browne et al. 2012). Last but most important, MCTS does not follow professionals' way of playing since professionals hardly make brute simulation of every possible future positions. Instead, in most situations, they first obtain some candidates using pattern analysis and determine the optimal one by evaluating these candidates.

Recently as deep learning revolutionizes and gradually dominate many tasks in computer vision community, researcher start to borrow deep learning techniques for move prediction and develop computer Go systems (Clark and Storkey 2015; Maddison et al. 2015; Tian and Zhu 2016; Silver et al. 2016). However, compared with visual signals (e.g. 224×224 in image domain), Go board has a much smaller size (19×19), which poses the importance of relative position. This is consistent with playing Go as situation can dramatically alter with a minor change in position. On the other hand, existing DCNNs often mine such contexts by stacking more convolutional layers (e.g. up to 13 layers in (Silver et al. 2016)) to exploit high-order encodings of low-level features. Simply increasing layers not only suffers parameter burden but also does not embed contexts of local features and its evolutions.

Based on the above discussions, this paper introduces a computer Go system consisting of two major parts. The first part is a novel deep architecture used to provide a probability distribution of legal candidates learned from professionals' records. These candidates are further sent to a long-term evaluation part by considering local future impact instead of immediate reward. We expect the model focus on several suggested important regions rather than blind simulation of every corner in the board. The primary contributions of this work are summarized as follows.

- We propose a novel deep alternative neural network (DANN) to learn a pattern recognizer for move prediction. The proposed DANN enjoys the advantages of both CNN and recurrent neural network (RNN), preserving contexts of local features and its evolutions which we show are essential for playing Go. Compared with existing DCNN-based models, DANN can substantially improve the move prediction performance using less layers and parameters.
- To further enhance the candidates generated from DANN, we present a novel recurrent model to make a long-term evaluation around each candidate for the final choice. We formulate the process as a partially observe Markov decision process (POMDP) problem and propose a reinforcement learning solution with careful control of variance.
- We introduce a new professional Go dataset (PGD) for comprehensive evaluation. PGD consists of 329.4k modern professional records and considered useful for computer Go community. Thorough experiments on GoGoD and PGD demonstrates the advantages of our system over DCNNs and MCTS on both move prediction and win rate against open source engines.

Related Work

Monte Carlo tree search (MCTS). It is a best-first search method based on randomized explorations of search space, which does not require a positional evaluation function (Browne et al. 2012). Using the results of previous explorations, the algorithm gradually grows a game tree, and successively becomes better at accurately estimating the values of the optimal moves (Bouzy and Helmstetter 2004) (Coulom 2006). Such programs have led to strong amateur level performance, but a considerable gap still remains between top professionals and the strongest computer programs. The majority of recent progress is due to increased quantity and quality of prior knowledge, which is used to bias the search towards more promising states, and it is widely believed that this knowledge is the major bottleneck towards further progress. The first successful current Go program (Kocsis and Szepesvári 2006) was based on MCTS. Their basic algorithm was augmented in MoGo (Gelly and Silver 2007) to leverage prior knowledge to bootstrap value estimates in the search tree. Training with professionals' moves was enhanced in Fuego (Enzenberger et al. 2010) and Pachi (Baudiš and Gailly 2011) and achieved strong amateur level.

Supervised pattern-matching policy learning. Go professionals rely heavily on pattern analysis rather than brute simulation in most cases (Clark and Storkey 2015; Xiao and Müller 2016). They can gain strong intuitions about what are the best moves to consider at a glance. This is in contrast to MCTS which simulates enormous possible future positions. The prediction functions are expected to be non-smooth and highly complex, since it is fair to assume professionals think in complex, non-linear ways when they choose moves. To this end, neural networks especially CNNs are widely used (Schraudolph, Dayan, and Sejnowski 1994; Enzenberger 1996; Richards, Moriarty, and Miikkulainen 1998; Sutskever and Nair 2008). Recent works in image recognition have demonstrated considerable advantages of DCNNs (Krizhevsky, Sutskever, and Hinton 2012; Simonyan and Zisserman 2014; Szegedy et al. 2015) and showed substantial improvement over shallow networks based on manually designed features or simple patterns extracted from previous games (Silver 2009). DCNNs have yielded several state-of-the-art Go playing system (Clark and Storkey 2015; Maddison et al. 2015; Tian and Zhu 2016; Silver et al. 2016) with $8, 12$ and 13 convolutional layers. Besides, these works have also indicated combining DCNNs and MCTS can improve the overall playing strength. Similar conclusions are validated in state-of-the-art Go programs (Enzenberger et al. 2010; Baudiš and Gailly 2011). The major difference between this paper and existing combinations comes from two perspectives. The first one is that we use a novel architecture DANN to generate candidates with more consideration of local contexts and its evolutions. The proposed architecture shows substantial improvement with pure DCNN with fewer layers and parameters. The second is that we use a long-term evaluation to analyze previous candidates instead of MCTS to assist the final choice. This strategy is faster than MCTS because the former needs to consider a large search space.

The Proposed Computer Go System

The proposed system is illustrated in Figure 1). Given a situation, we first obtain a probability distribution of legal points that are learned from a pattern-aware prediction instrument based on supervised policy learning from existing professional records. We then further analyze those candidates with high-confidence by a long-term evaluation to pursue a expectation reward after several future steps when the local situation is settled. The action with the highest score of criteria combination of two criteria is our final choice.

Deep Alternative Neural Network

We train a novel deep alternative neural network (DANN) to generate a probability distribution of legal points given current board situation as an input. We treat the 19×19 board as a 19×19 image with multiple channels. Each channel encodes a different aspect of board information (see details in Table 1). In the following we describe the structure of DANN including its key component (alternative layer) and overall architecture. Afterwards we discuss its relations and advantages over popular DCNNs.

Alternative layer. The key component of DANN is the alternative layer (AL), which consists of a standard convolutional layer followed by a designed recurrent layer. Specifically, convolution operation is first performed to extract features from local neighborhoods on feature maps in the previous layers. Then a recurrent layer is applied to the output and iteratively proceeds for T times. This procedure makes each unit evolve over discrete time steps and aggregate larger receptive fields (RFs). More formally, the input of a unit at position (x, y, z) in the jth feature map of the ith AL at time t, denoted as $u_{ij}^{xyz}(t)$, is given by

$$\begin{aligned} u_{ij}^{xyz}(t) &= u_{ij}^{xyz}(0) + f(\mathbf{w}_{ij}^{r} u_{ij}^{xyz}(t-1)) + b_{ij} \\ u_{ij}^{xyz}(0) &= f(\mathbf{w}_{(i-1)j}^{c} u_{(i-1)j}^{xyz}) \end{aligned} \tag{1}$$

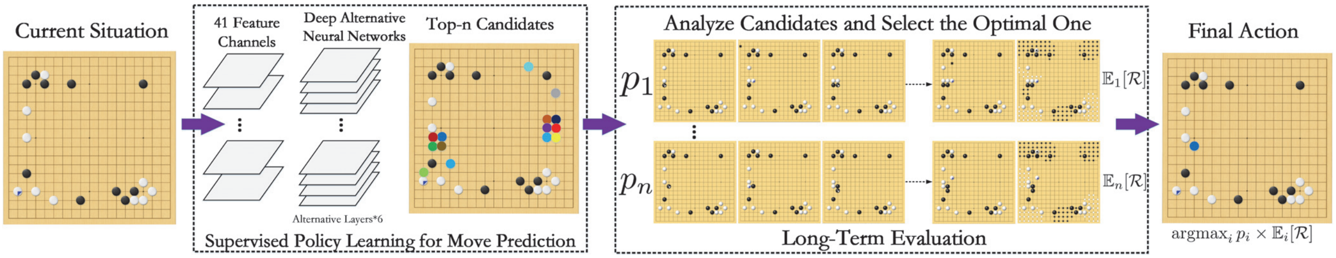
Current Situation
41 Feature Channels
Deep Alternative Neural Networks
Top-n Candidates
Alternative Layers*6
Supervised Policy Learning for Move Prediction
Analyze Candidates and Select the Optimal One
p_1
p_n
$\mathbb{E}_1[\mathcal{R}]$
$\mathbb{E}_n[\mathcal{R}]$
Long-Term Evaluation
Final Action
$\text{argmax}_i\, p_i \times \mathbb{E}_i[\mathcal{R}]$

Figure 1: The proposed computer Go system with deep alternative neural network (DANN) and long-term evaluation. Given a situation the system generate several candidates by DANN that are learned from professional records. These candidates are further analyzed using a long-term evaluation with consideration of future rewards to determine a final action.

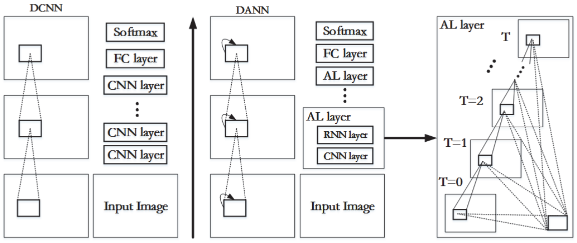

Figure 2: Comparison of DANN (right) and DCNN (left).

where $u_{ij}^{xyz}(0)$ denotes the feed-forward output of convolutional layer, $u_{ij}^{xyz}(t-1)$ is the recurrent input of previous time, $\mathbf{w}_k^c$ and $\mathbf{w}_k^r$ are the vectorized feed-forward kernels and recurrent kernels, b_{ij} is the bias for jth feature map in ith layer, $u_{ij}^{xyz}(0)$ is the output of convolutional output of previous layer and $f(\mathbf{w}_{ij}^r u_{ij}^{xyz}(t-1)))$ is induced by the recurrent connections. f is defined as popular rectified linear unit (ReLU) function $f(x) = \max(0, x)$, followed by a local response normalization (LRN)

$$\text{LRN}(u_{ij}^{xyz}) = \frac{u_{ij}^{xyz}}{\left(1 + \frac{\alpha}{\mathcal{L}} \sum_{k'=\max(0,k-\mathcal{L}/2)}^{\min(\mathcal{K},k+\mathcal{L}/2)} (u_{ij}^{xyz})^2\right)^{\beta}} \quad (2)$$

where $\mathcal{K}$ is the number of feature maps, α and β are constants controlling the amplitude of normalization. The LRN forces the units in the same location to compete for high activities, which mimics the lateral inhibition in the cortex. In our experiments, LRN is found to consistently improve the accuracy, though slightly. Following (Krizhevsky, Sutskever, and Hinton 2012), α and β are set to 0.001 and 0.75, respectively. $\mathcal{L}$ is set to $\mathcal{K}/8 + 1$.

Equation 1 describes the dynamic behavior of AL where contexts are involved after local features are extracted. Unfolding this layer for T time steps results in a feed-forward subnetwork of depth $\text{T} + 1$ as shown in the right of Figure 2. While the recurrent input evolves over iterations, the feed-forward input remains the same in all iterations. When $\text{T} = 0$ only the feed-forward input is present. The subnetwork has several paths from the input layer to the output layer. The longest path goes through all unfolded recurrent connections, while the shortest path goes through the feed-forward connection only. The effective RF of an AL unit in the feature maps of the previous layer expands when the iteration number increases. If both input and recurrent kernels in equation have square shapes in each feature map of size L_{feed} and L_{rec}, then the effective RF of an AL unit is also square, whose side length is $\text{L}_{\text{feed}} + \text{L}_{\text{rec}} \times (\text{T} + 1)$.

Advantages over DCNNs. The recurrent connections in DANN provide three major advantages compared with popular DCNNs used for move prediction (Clark and Storkey 2015; Maddison et al. 2015; Tian and Zhu 2016; Silver et al. 2016). First, they enable every unit to incorporate contexts in an arbitrarily large region in the current layer, which is particular suitable in the game of Go as the input signal is very small where the contexts are essential. As the time steps increase, the state of every unit is influenced by other units in a larger and larger neighborhood in the current layer. In consequence, the size of regions that each unit can watch in the input space also increases. In standard convolutional layers, the size of effective RFs of the units in the current layer is fixed, and watching a larger region is only possible for units in higher layers. But unfortunately the context seen by higher-level units cannot influence the states of the units in the current layer without top-down connections. Second, the recurrent connections increase the network depth while keeping the number of adjustable parameters constant by weight sharing. Specially, stacking higher layers consume more parameters while AL uses only additional constant parameters compared to standard convolutional layer. This is consistent with the trend of modern deep architectures, i.e., going deeper with relatively small number of parameters (Simonyan and Zisserman 2014; Szegedy et al. 2015). Note that simply increasing the depth of CNN by sharing weights between layers can result in the same depth and the same number parameters as DANN. We have tried such a model which leads to a lower performance. The third advantage is the time-unfolded manner in Figure 2, which is actually a CNN with multiple paths between the input layer to the output layer to facilitate the learning procedure. On one hand, the existence of longer paths makes it possible for the model to learn highly complex features. On the other hand, the existence of shorter paths may help gradient of backpropagation during training. Multi-path is also used in (Lee et al. 2015; Szegedy et al. 2015), but extra objective functions are used in hidden layers to alleviate the difficulty in training deep networks, which are not used in DANN.

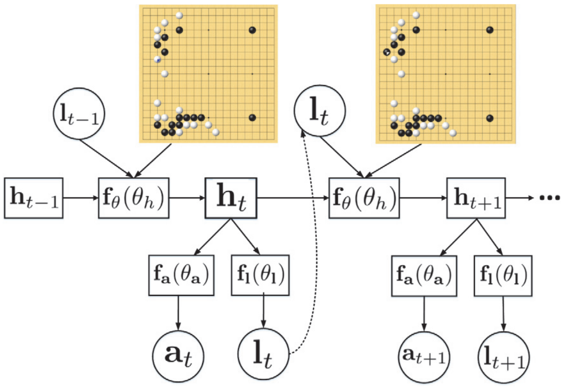
$\mathbf{l}_{t-1}$
$\mathbf{h}_{t-1}$
$\mathbf{f}_\theta(\theta_h)$
$\mathbf{h}_t$
$\mathbf{l}_t$
$\mathbf{f}_\theta(\theta_h)$
$\mathbf{h}_{t+1}$
...
$\mathbf{f}_\mathbf{a}(\theta_\mathbf{a})$
$\mathbf{f}_\mathbf{l}(\theta_\mathbf{l})$
$\mathbf{f}_\mathbf{a}(\theta_\mathbf{a})$
$\mathbf{f}_\mathbf{l}(\theta_\mathbf{l})$
$\mathbf{a}_t$
$\mathbf{l}_t$
$\mathbf{a}_{t+1}$
$\mathbf{l}_{t+1}$

Figure 3: A recurrent model for long-term evaluation.

Overall architecture. The overall architecture of our DANN has 6 ALs with 64, 128, 256, 256, 512 and 512 kernels, followed by 2 fully connected (FC) layers of size 1024 each. We use 3×3 kernel for convolutional layer and recurrent layers of all 6 ALs. After each AL, the network includes a ReLU activation. We use max pooling kernels of 2×2 size. All of these convolutional layers and recurrent layers are applied with appropriate padding and stride. FC layers are followed by a ReLU and a softmax, which outputs the probabilities of Go board and illegal points are set 0.

Long-Term Evaluation of Candidates

DANN provides a probability distribution of next move candidates give a situation. We further enhance this model by evaluating these candidates in a long-term consideration since predicting only the immediate next move limits the information received by lower layers (Tian and Zhu 2016). Besides, many situations in intensive battle or *capture chase* is far beyond fair evaluation and need to be accurately judged when local variation is settled. We aim to avoiding shortsighted moves. There are some works such as (Littman 1994) that consider playing games as a sequential decision process of a goal-directed agent interacting with visual environment. We extend this idea to evaluate candidates in a similar manner. We calculate the cumulative rewards of each candidate with several future interactions. Combining previous probabilities criterion, we obtain a final score and determine the optimal action.

Recurrent model and internal state. Figure 3 shows our model structure, which is build a agent around a RNN. To avoiding blind search space like MCTS, the agent observes the environment only via a bandwidth-limited sensor, i.e. it never senses the full board. It may extract information only in a local region around candidates. The goal of our model is to provide a reliable evaluation of each candidate and assist the final choice. The agent maintains an internal state which summarizes information extracted from past observations. It encodes the agent's knowledge of the environment and is instrumental to deciding how to act and where to deploy the next action. This internal state is formed by the hidden units $\mathbf{h}_t$ of the recurrent neural network and updated over time by the core network $\mathbf{h}_t = \mathbf{f}_\mathbf{h}(\mathbf{h}_{t-1}, \mathbf{l}_{t-1}; \theta_\mathbf{h})$.

Action and reward. At each step, the agent performs two actions. It decides how to deploy its sensor via the sensor control $\mathbf{l}_t$, and an action $\mathbf{a}_t$ which might affect the state of the environment. The location are chosen stochastically from a distribution parameterized by the location network $\mathbf{l}_t \sim p(\cdot|\mathbf{f}_\mathbf{l}(\mathbf{h}_t; \theta_\mathbf{l})$. The action is similarly drawn from a distribution conditioned on a second network output at $\mathbf{a}_t \sim p(\cdot|\mathbf{f}_\mathbf{a}(\mathbf{h}_t; \theta_\mathbf{a})$. Finally, our model can also be augmented with an additional action that decides when it will stop when local fights are settled. After executing an action the agent receives a new visual observation and a reward signal r. The goal of the agent is to maximize the sum of the reward signal $\mathcal{R} = \sum_{t=1}^{T} r_t$. The above setup is a special instance of partially observable Markov decision process (POMDP), where the true state of whole board is unobserved.

Training. The policy of the agent, possibly in combination with the dynamics of interactions, induces a distribution over possible interaction sequences and we aim to maximize the reward under the distribution of

$$\mathcal{J}(\theta) = \mathbb{E}_{p(s_{1:T};\theta)}[\sum_{t=1}^{T} r_t] = \mathbb{E}_{p(s_{1:T};\theta)}[\mathcal{R}] \tag{3}$$

Maximizing $\mathcal{J}$ exactly is difficult because it involves an expectation over interaction sequences which may in turn involve unknown environment dynamics. Viewing the problem as a POMDP problem, however, allows us to bring techniques from the RL literature to bear. As shown in (Williams 1992) a sample approximation to the gradient is given by

$$\begin{aligned} \nabla_\theta \mathcal{J} &= \sum_{t=1}^{T} \mathbb{E}_p(s_{1:T}; \theta)[\nabla_\theta \log \pi(u_t|s_{1:t}; \theta)\mathcal{R}] \\ &\approx \frac{1}{M} \sum_{i=1}^{M} \sum_{t=1}^{T} \nabla_\theta \log \pi(u_t^i|s_{1:t}^i; \theta)\mathcal{R}^i \end{aligned} \tag{4}$$

where s^i's are the interaction sequences obtained by running the current agent π_θ for $i = 1 \cdots M$ episodes. The learning rule of Equation 4 is also known as the REINFORCE rule, and it involves running the agent with its current policy to obtain samples of interaction sequences $s_{1:T}$ and then adjusting the parameters θ such that the log-probability of chosen actions that have led to high cumulative reward is increased, while that of actions having produced low reward is decreased. $\nabla_\theta \log \pi(u_t^i|s_{1:t}^i; \theta)$ in Equation 4 is just the gradient of the RNN and can be computed by standard backpropagation (Wierstra et al. 2007).

Variance reduction. Equation 4 provides us with an unbiased estimate of the gradient but it may have high variance. It is common to consider a gradient estimate of the form

$$\frac{1}{M} \sum_{i=1}^{M} \sum_{t=1}^{T} \nabla_\theta \log \pi(u_t^i|s_{1:t}^i; \theta)(\mathcal{R}_t^i - b_t) \tag{5}$$

where $\mathcal{R}_t^i = \sum_{t'=1}^{T} r_{t'}^i$ is the cumulative reward following the execution of action u_t^i, and b_t is a baseline that depends on $s_{1:t}^i$ but not on the action u_t^i itself. This estimate is equal

Table 1: Input feature channels for DANN.

Feature	#	Description
Ladder capture	1	Whether point is a successful ladder capture
Ladder escape	1	Whether point is a successful ladder escape
Sensibleness	1	Whether point is legal and does not fill eyes
Legality	1	Whether point is legal for current player
Player color	1	Whether current player is black
Zeros	1	A constant plane filled with 0
Stone color	3	Player stone/opponent stone/empty
Liberties	4	Number of liberties (empty adjacent points)
Liberties∗	6	Number of liberties (after this move)
Turn since	6	Number of liberties (after this move)
Capture size	8	How many opponent stones would be captured
Self-atari size	8	How many own stones would be captured

to Equation 4 in expectation but may have lower variance. We select the value function of baseline following (Sutton et al. 1999) in the form of $b_t = \mathbb{E}_\pi[R_t]$. We use this type of baseline and learn it by reducing the squared error between $\mathcal{R}_t^{i,}$ and b_t.

Final score. We define the final score for each candidate using the criteria of both DANN and long-term evaluation. We select the action of the highest score of $\mathcal{S} = p \times \mathbb{E}[\mathcal{R}]$ as the final choice of our system, where p is the probability produced by the softmax layer of DANN.

Experiments

Setup

Datasets. The first dataset we used is GoGoD (2015 winter version). The dataset consists of $82,609$ historical and modern games. We limited our experiments to a subset of games that satisfied the following criteria: 19×19 board, modern (played after 1950), "standard" komi (komi $\in \{5.5, 6.5, 7.5\}$), and no handicap stones. We did not distinguish between rulesets (most games followed Chinese or Japanese rules). Our criteria produced a training set of around $70,000$ games. We did not use popular KGS dataset because it consists of more games by lower *dan* players. The average level is approximately 5 *dan*.

Besides, we have collected a new professional Go dataset (PGD) consisting of $253,233$ professional records, which exceeds GoGoD and KGS in both quantity and playing strength. PGD was parsed from non-profit web sites. All records are saved as widely used smart go file (SGF), named as `DT_EV_PW_WR_PB_BR_RE.sgf`, where `DT`, `EV`, `PW`, `WR`, `PB`, `BR` and `RE` represent date, tournament type, black player name, black playing strength, white player name, white playing strength and result.

Feature channels. The features that we used come directly from the raw representation of the game rules (stones, liberties, captures, legality, turns since) as in Table 1. Many of the features are split into multiple planes of binary values, for example in the case of liberties there are separate binary features representing whether each intersection has 1 liberty, 2 liberties, 3 liberties, $>= 4$ liberties.

Implementation details. The major implementations of DANN including convolutional layers, recurrent layers and optimizations are derived from Torch7 toolbox (Collobert, Kavukcuoglu, and Farabet 2011). We use SGD applied to mini-batches with negative log likelihood criterion. The size of mini-batch is set 200. Training is performed by minimizing the cross-entropy loss function using the backpropagation through time (BPTT) algorithm (Werbos 1990). This is equivalent to using the standard BP algorithm on the time-unfolded network. The final gradient of a shared weight is the sum of its gradients over all time steps. The initial learning rate for networks learned from scratch is 3×10^{-3} and it is 3×10^{-4} for networks fine-tuned from pre-trained models. The momentum is set to 0.9 and weight decay is initialized with 5×10^{-3} and reduced by 10^{-1} factor at every decrease of the learning rate.

Move Prediction

We first evaluate different configurations of DANN. Then we compare our best DANN model with DCNN-based methods. Finally, we study the impact of long-term evaluation and report the overall performance on move prediction.

Model investigation of DANN. There are two crucial configurations for DANN model. The first one is the AL setting including its order and number. The other one is the unfolding time T in recurrent layers. Comparison details are reported in Table 2, where B_6C_2FC is a baseline composed of similar configuration with DANN but using standard convolutional layers instead of ALs. The first column of left table in Table 2 has only one AL layer and the accuracy comparison demonstrates the benefits of inserting AL in advance. We attribute it to the context mining of lower features. The fourth column of left table in Table 2 shows the performance increases as the number of AL increases, which verifies the effectiveness of inserting recurrent layer. Specifically, the order of AL can contribute a performance gain up to 11% which indicates that mining contexts of lower layer is beneficial for playing Go. Right table in Table 2 uses 5AL_2FC and 6AL_2FC to study the impact of T and the results prove larger T leads to better performance in most cases. Given such results we use our best DANN model 6AL_2FC in the following experiments.

Comparison with DCNN-based methods. Figure 4 reports the performance comparison of our best DANN model and related approaches using pure DCNN. Following (Maddison et al. 2015) we evaluate the accuracy that the correct move is within the network's n most confident predictions. As Figure 4 shows, our model consistently outperform two recent approaches (Maddison et al. 2015; Tian and Zhu 2016) using pure DCNN on two datasets. Also, note that our architecture consume less layers where the parameters are also saved .

Combining long-term evaluation. Next we examine the influence of our long-term evaluation (LTE). We focus mainly on the future step that is used to achieve the expectation of reward, and the episode number when solving Equation 4. We combine our best DANN model with LTE on Top-1 move prediction accuracy. Table 5 demonstrates the details. As can be seen, the best performance is achieved around 15 to 21 steps. As for the episode, LTE often converges after around 200 episodes. Using the optimal setting of both part, the overall top-1 accuracy can be obtained at

Table 2: Performance comparison (top-1) with different configurations of DANN on GoGoD and PGD datasets.

Architecture	GoGoD	PGD	Architecture	GoGoD	PGD
B_6C_2FC	32.2%	37.2%	2AL_4C_2FC	35.0%	39.3%
AL_5C_2FC	40.5%	42.5%	3AL_3C_2FC	42.7%	43.1%
C_AL_4C_2FC	41.1%	41.1%	4AL_2C_2FC	47.4%	42.2%
2C_AL_3C_2FC	45.6%	44.8%	5AL_C_2FC	47.2%	46.4%
3C_AL_2C_2FC	47.4%	43.0%	6AL_2FC	53.5%	51.8%
4C_AL_C_2FC	49.4%	46.6%			
5C_AL_2FC	51.9%	49.3%			

Architecture	GoGoD	PGD
5AL_2FC, $T = 2$	46.9%	42.3%
5AL_2FC, $T = 3$	48.2%	47.1%
5AL_2FC, $T = 4$	52.3%	45.8%
5AL_2FC, $T = 5$	55.0%	51.6%
6AL_2FC, $T = 2$	46.4%	47.2%
6AL_2FC, $T = 3$	51.1%	51.6%
6AL_2FC, $T = 4$	55.3%	49.4%
6AL_2FC, $T = 5$	57.7%	53.8%

Table 3: Win rate comparison against open source engines between our system and previous work.

	GnuGo	MoGo 10k	Pachi 10k	Pachi 100k	Fuego 10k	Fuego 100k
8-layer-DCNN + MCTS (Clark and Storkey 2015)	91.0%	-	-	-	14.0%	14.0%
12-layer-DCNN + MCTS (Maddison et al. 2015)	97.2%	45.9%	47.4%	11.0%	23.3%	12.5%
12-layer-DCNN + MCTS (Tian and Zhu 2016)	100±0.0%	-	94.3±1.7%	72.69±1.9%	93.2±1.5%	89.7±2.1%
6-layer-DANN + LTE (Ours)	100±0.0%	72.5±1.8%	83.1±1.4%	65.3±1.6%	82.6±1.2%	76.5±1.6%

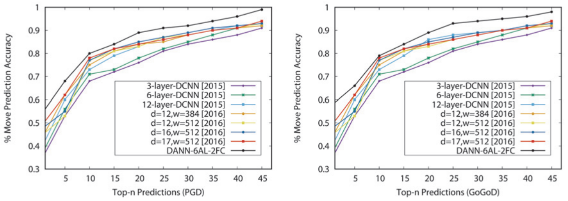

Figure 4: Top-n comparison of our best DANN model and DCNN-based methods on GoGoD and PGD datasets.

61% and 56% on GoGod and PGD datasets, respectively.

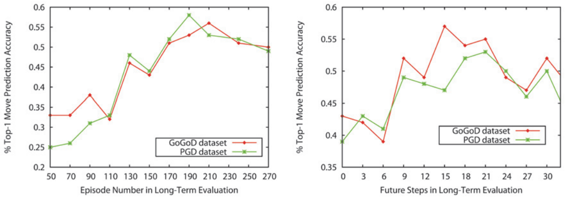

Figure 5: Impact of future steps and episode number in long-term evaluation on GoGoD and PGD datasets.

Playing Strength

Finally, we evaluate the overall playing strength of our system by playing against several publicly available benchmark programs. All programs were played at the strongest available settings, and a fixed number of rollouts per move. We used GnuGo 3.8 level 10, MoGo (Gelly and Silver 2007), Pachi 11.99 (Genjo-devel) with the pattern files, and Fuego 1.1 throughout our experiments. For each setting, 3 groups of 100 games were played. We report the average win rate and standard deviation computed from group averages. All the game experiments mentioned in this paper used komi 7.5 and Chinese rules. Pondering (keep searching when the opponent is thinking) in Pachi and Fuego are on. As Table 3 shows, our system outperform most MCTS-based Go programs. Also, the win rate of our approach is higher than that of previous works except (Tian and Zhu 2016).

Conclusion

In this work, we have proposed a computer Go system based on a novel deep alternative neural networks (DANN) and long-term evaluation (LTE). We also public a new dataset consisting of around 25k professional records. On two datasets, we showed that DANN can predict the next move made by Go professionals with an accuracy that substantially exceeds previous deep convolutional neural network (DCNN) methods. LTE strategy can further enhance the quality of candidates selection, by combining the influence of future interaction instead of immediate reward. Without brute simulation of possible interaction in a large and deep search space, our system is able to outperform most MCTS-based open source Go programs.

Future work mainly includes the improvement of DANN structure for move prediction and more reliable LTE implementation. Advance techniques in computer vision community such as residual networks may help DANN obtain further improvement. As for LTE, domain knowledge of Go will be attempted to provide a more reliable estimation of next move candidates.

Acknowledgement

This work was supported by Shenzhen Peacock Plan (20130408-183003656).

References

Baudiš, P., and Gailly, J.-l. 2011. Pachi: State of the art open source go program. In *Advances in Computer Games*. 24–38.

Bouzy, B., and Helmstetter, B. 2004. Monte-carlo go developments. In *Advances in computer games*. 159–174.

Browne, C. B.; Powley, E.; Whitehouse, D.; Lucas, S. M.; Cowling, P. I.; Rohlfshagen, P.; Tavener, S.; Perez, D.;

Samothrakis, S.; and Colton, S. 2012. A survey of monte carlo tree search methods. *Computational Intelligence and AI in Games, IEEE Transactions on* 4(1):1–43.

Clark, C., and Storkey, A. 2015. Training deep convolutional neural networks to play go. In *Proceedings of the 32nd International Conference on Machine Learning*, 1766–1774.

Collobert, R.; Kavukcuoglu, K.; and Farabet, C. 2011. Torch7: A matlab-like environment for machine learning. In *BigLearn, NIPS Workshop*, number EPFL-CONF-192376.

Coulom, R. 2006. Efficient selectivity and backup operators in monte-carlo tree search. In *Computers and games*. 72–83.

Enzenberger, M.; Müller, M.; Arneson, B.; and Segal, R. 2010. Fuego- an open-source framework for board games and go engine based on monte carlo tree search. *Computational Intelligence and AI in Games, IEEE Transactions on* 2(4):259–270.

Enzenberger, M. 1996. The integration of a priori knowledge into a go playing neural network. *URL: http://www. markus-enzenberger. de/neurogo. html*.

Gelly, S., and Silver, D. 2007. Combining online and offline knowledge in uct. In *International Conference on Machine Learning*, 273–280. ACM.

Gelly, S., and Silver, D. 2011. Monte-carlo tree search and rapid action value estimation in computer go. *Artificial Intelligence* 175(11):1856–1875.

Huang, S.-C., and Müller, M. 2013. Investigating the limits of monte-carlo tree search methods in computer go. In *Computers and Games*. 39–48.

Kocsis, L., and Szepesvári, C. 2006. Bandit based monte-carlo planning. In *European Conference Machine Learning*. 282–293.

Krizhevsky, A.; Sutskever, I.; and Hinton, G. E. 2012. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, 1097–1105.

Lee, C.-Y.; Xie, S.; Gallagher, P.; Zhang, Z.; and Tu, Z. 2015. Deeply-supervised nets. In *International Conference on Artificial Intelligence and Statistics*, volume 2, 6.

Littman, M. L. 1994. Markov games as a framework for multi-agent reinforcement learning. In *Proceedings of the eleventh international conference on machine learning*, volume 157, 157–163.

Maddison, C. J.; Huang, A.; Sutskever, I.; and Silver, D. 2015. Move evaluation in go using deep convolutional neural networks. In *International Conference on Learning Representation*.

Müller, M. 2002. Computer go. *Artificial Intelligence* 134(1):145–179.

Richards, N.; Moriarty, D. E.; and Miikkulainen, R. 1998. Evolving neural networks to play go. *Applied Intelligence* 8(1):85–96.

Rimmel, A.; Teytaud, O.; Lee, C.-S.; Yen, S.-J.; Wang, M.-H.; and Tsai, S.-R. 2010. Current frontiers in computer go. *Computational Intelligence and AI in Games, IEEE Transactions on* 2(4):229–238.

Schraudolph, N. N.; Dayan, P.; and Sejnowski, T. J. 1994. Temporal difference learning of position evaluation in the game of go. In *Advances in Neural Information Processing Systems*, 817–817.

Silver, D.; Huang, A.; Maddison, C. J.; Guez, A.; Sifre, L.; Van Den Driessche, G.; Schrittwieser, J.; Antonoglou, I.; Panneershelvam, V.; Lanctot, M.; et al. 2016. Mastering the game of go with deep neural networks and tree search. *Nature* 529(7587):484–489.

Silver, D. 2009. Reinforcement learning and simulation-based search. *Doctor of philosophy, University of Alberta*.

Simonyan, K., and Zisserman, A. 2014. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*.

Sutskever, I., and Nair, V. 2008. Mimicking go experts with convolutional neural networks. In *International Conference on Artificial Neural Networks*. 101–110.

Sutton, R. S.; McAllester, D. A.; Singh, S. P.; Mansour, Y.; et al. 1999. Policy gradient methods for reinforcement learning with function approximation. In *Advances in neural information processing systems*, volume 99, 1057–1063.

Szegedy, C.; Liu, W.; Jia, Y.; Sermanet, P.; Reed, S.; Anguelov, D.; Erhan, D.; Vanhoucke, V.; and Rabinovich, A. 2015. Going deeper with convolutions. In *Computer Vision and Pattern Recognition*, 1–9.

Tian, Y., and Zhu, Y. 2016. Better computer go player with neural network and long-term prediction. In *International Conference on Learning Representation*.

Werbos, P. J. 1990. Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE* 78(10):1550–1560.

Wierstra, D.; Foerster, A.; Peters, J.; and Schmidhuber, J. 2007. Solving deep memory pomdps with recurrent policy gradients. In *International Conference on Artificial Neural Networks*, 697–706.

Williams, R. J. 1992. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning* 8(3-4):229–256.

Xiao, C., and Müller, M. 2016. Factorization ranking model for move prediction in the game of go. In *AAAI Conference on Artificial Intelligence*.