This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TG.2017.2785042, IEEE Transactions on Games

Move Prediction using Deep Convolutional Neural Networks in Hex

Chao Gao, Ryan Hayward and Martin Müller

***Abstract*— Using deep convolutional neural networks for move prediction has led to massive progress in Computer Go. Like Go, Hex has a large branching factor that limits the success of shallow and selective search. We show that deep convolutional neural networks can be used to produce reliable move evaluation in the game of Hex. We begin by collecting self-play games of MoHex 2.0. We then train the neural networks by canonical maximum likelihood. The trained model was evaluated by playing against top programs Wolve and MoHex 2.0. Without any search, the resulting neural network produces similar playing strength as the highly optimized Resistance evaluation function used in Wolve. Finally, using the neural networks as prior knowledge, the reigning Monte Carlo tree search based world champion player MoHex 2.0 can be enhanced.**

I. Introduction

The game of Hex is a two-player zero sum game that is invented independently by Piet Hein [1] and by John Nash in 1948 [2]. It was presented to the general public by Martin Gardner in 1959 [3]. Since the seminal work by Shannon in 1950s [4], it has been an active domain of Artificial Intelligence research.

Hex is a perfect information game where two players could play alternatively. Typically, black starts first. There is no draw in Hex — this can be proven using the Brouwer fixed-point theorem in two dimensions [5]. It has also been shown by "strategy stealing" [6] that a winning strategy exists for the first player. However, an explicit winning strategy remains obscure. Indeed, solving arbitrary Hex positions has been proven to be PSPACE-complete [7, 8]. In practical playing, to reduce the paramount advantage of the first player, a swap rule can be applied, which means the second player can either steal the first player's opening move or play a norm move.

Like Go before ago, Hex has a large branching factor, which makes fixed-depth exhaustive search difficult. Unlike Go, Hex has a reasonably strong evaluation function that is based on electric resistance [9, 10, 11]. Hence, alpha-beta search programs that utilize knowledge and connection strategy computation have achieved modest success [12, 13]. Nevertheless, it is known that the resistance evaluation function is often pathological [11]. Following the trend in computer Go, Monte Carlo Tree Search [14, 15, 16] was applied to Hex, leading to MoHex [17], which has better playing performance than alpha-beta search [18]. Further developments by Huang et al. [19] added learnt prior preferences to MoHex. The result program MoHex 2.0 is better than MoHex. Indeed, MoHex 2.0 has been the reigning champion in the Computer Olympiad since 2013. The improvement in performance of MoHex 2.0 is mainly due to its successful grasp of prior knowledge, which is expressed by a linear function trained on expert game records [20].

The recent breakthrough [21, 22, 23, 24] in computer Go has shown that using deep Convolutional Neural Networks (CNN) [25] for representation and learning game knowledge is far superior to earlier approaches. Motivated by the success in computer Go, we investigate how to use deep convolutional neural nets to represent and learn knowledge in Hex. The goal of this paper is to pave the road towards an AlphaGo style playing system for computer Hex – a task that poses the following challenges:

- Representation. Compact representation is essential to any learning model. Typically, feeding more input features to the neural nets yields better prediction accuracy. However, since trained neural nets will eventually be used in search, it needs to be sufficient fast to evaluate. The common wisdom in deep learning is "deeper models will produce better results", but the development of a successful architecture often requires many trials, and again, the cost of forward inference should also be considered.
- Sophistication. Training data obtained from either human or computer game records is essentially imperfect. Appropriate dealing with the imperfectness of the training data should be beneficial.
- Search. Correct and effective combination of the neural net and search is non-trivial. The major challenge is that the speed of neural nets on regular hardware is too slow to be intensively used in the search.

The above challenges ought to be solved one by one. In this paper, we show that with a compact representation that utilizes the most commonly seen *bridge pattern*, reasonable move predication accuracy can be achieved. Also, results from several other architectures are presented. We compare our trained model with a highly optimized evaluation function Resistance. Similar playing strength is observed. We integrate our policy net with MoHex 2.0, enhanced performance is observed when using neural net as a prior probability for node expansion.

The rest of this paper is organized as follows: In Section II, we give a brief description of the rules and tactics of Hex. In Section III we discuss previous work on Computer Hex. Section IV presents our move prediction neural network. Evaluation results are presented in Section V. Finally, we

Chao Gao, Ryan Hayward and Martin Mueller are with the Department of Computing Science, University of Alberta, Edmonton, Canada (email: {cgao3, hayward, mmueller}@ualberta.ca).

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TG.2017.2785042, IEEE Transactions on Games

summarize this paper and point out future work in Section VI.

II. Rules of Hex

Hex is played on an $n \times n$ hexagonal board, where players black and white take turns to place a stone in an empty hexagonal cell. The goal of Hex is to form a chain that connects the player's two sides of the board. The *swap rule* is usually applied to mitigate the first player's advantage. 11x11 and 13x13 board sizes are most popular and are adopted in computer tournaments. 19x19 board size is also played by human players. Hex cannot end in a draw, i.e., no chain can be completely blocked except a winning chain of the opposite. Figure 1 shows a Hex game on a 11x11 board.

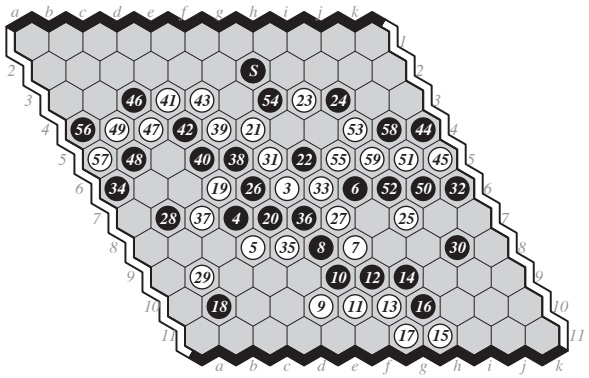

Fig. 1. A Hex game played on 11x11 board. White wins by forming a path connecting its two sides.

Playing Hex well requires complex tactics. The ability to see and form virtually connected components on the board is most critical. The bridge pattern is the most commonly seen virtual connection in Hex. Figure 2 shows a board position where black and white each have a bridge. For the black bridge in Figure 2, $D1$ and $C3$ are called *bridge endpoints*, $C2$ and $D2$ are *bridge carriers*. It is obvious that if black plays at $D4$, a new bridge can be formed, so $D4$ can be called as a black *form bridge* cell. If black plays at $B2$ to threaten white's bridge, white can respond at $A3$, to *save bridge*.

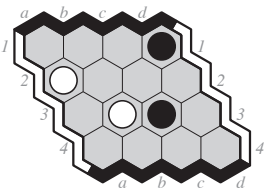

Fig. 2. An example that illustrates the bridge virtual connection pattern. Black and white each have a bridge.

III. Previous work on Computer Hex

Deep selective minimax search directed by an evaluation function has led to grandmaster play in chess [26]. However, due to the lack of a reliable evaluation function, such search techniques turned out to be less useful in Hex. Inspired by the seminal work of Shannon [4], using electronic circuit resistance as an evaluation function was proposed by Anshelevich [9, 10], which leads to the first modern Hex playing programs Hexy [27], and later Six [12]. Due to the characteristics of the electric model, in both programs, the information provided by Virtual Connection (VC) computation is vital to the quality of the evaluation. Virtual connection is a point to point connection strategy, where each point could be an empty cell, a group of connected stones or a board side. For a given board state, usually there is an exponentially large number of VCs. Anshelevich [10] proposed a hierarchical algebra that is able to compute a subset of them. The algorithm is named as H-Search. It starts with trivial VC patterns, and then works by repeatedly applying two combination rules to form larger VCs. H-Search was shown to be incomplete [10]. A number of improvements of H-Search are described in [11]. Most recently, Pawlewicz et al. [28] presented many heuristic techniques to improve H-Search, making it is able to compute a critical subset of VCs.

Inferior Cell Analysis (ICA) [29, 11] uses identified dead (or dominated, inferior) local patterns to automatically derive cells that can be removed from consideration. Combining some ICA and VC computation, Hayward et al. [29] developed an algorithm that is strong enough to solve arbitrary 7x7 Hex states in reasonable time. Later, after extending the inferior cell analysis and enhancing VC computation by adding inferior cell patterns, all 8x8 openings were solved by Henderson et al. [30]. Based on ICA and VC augmented Resistance, the alpha-beta search player Wolve was developed, which won the golden medal in 2008 Computer Olympiad [13]. Other strong computer players that use those VC and ICA computation include DeepHex [31] and Ezo [32].

However, the Resistance evaluation, though, has been improved by adding VC and ICA information, is still frequently pathological, as it tends to favour fillin, dominated cells [11]. Such a problem is then sidestepped by the development of Monte-carlo search, which evaluates a game position by simulating random playouts. Convergence properties of Monte-carlo tree search are discussed in [14, 33].

The success of Monte Carlo tree search (MCTS) [14, 15, 16] is largely due to its superior ability to bias the search to promising nodes in the tree, by dynamically backing up the evaluations from random sequences of self-play. Therefore a major effort for enhancements of MCTS is to increase the quality of such biases by incorporating learnt off-line prior knowledge [34]. In this spirit, MoHex 2.0 by Huang et al. [19] biases preferences learnt from expert game records. MoHex 2.0 is 300 Elo stronger than MoHex [19] on the 13x13 board size.

Recently, the breakthrough in Image Classification [25] has demonstrated the superior representation ability of deep convolutional networks over other learning models. This success has spread to the domain of computer Go [21, 22,

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TG.2017.2785042, IEEE Transactions on Games

23, 24], leading to super human strength Go playing system AlphaGo [24]. Inspired by this progress, we study how to bring the advantage of deep convolutional networks into the domain of Hex. As the first step, we focus on supervised learning in this paper.

IV. Move prediction with CNN

The availability of big data is essential for any deep learning models. In this section, we first introduce our training data, then design input planes for CNN that utilize the bridge pattern. After that our architecture and training method will be presented.

A. Data

According to [19], pattern weights in MoHex 2.0 were trained on a dataset consisting of 19760 13x13 games from Little Golem [1] and 15116 games by MoHex played against Wolve. Since those data are no longer available, we generated a new dataset from MoHex 2.0 self-play by setting a time limit per move varying from 28s to 40s, iterating over all opening moves, with the parallel solver turned off. In this way, we collected 15520 games in total. All those self-play games were produced on *Cybera*[2] cloud instances with Intel(R) Xeon(R) CPU E5-2650 v3 2.30GHz and 4GB RAM.

We extract 1098952 distinct state-action pairs (s, a) from those games. The data was split into 90% as training and 10% as test set. To show detailed characteristics of our data, we plot the distribution state-action pairs with different move numbers in Figure 3. As expected, the majority of those positions is from the middle game.

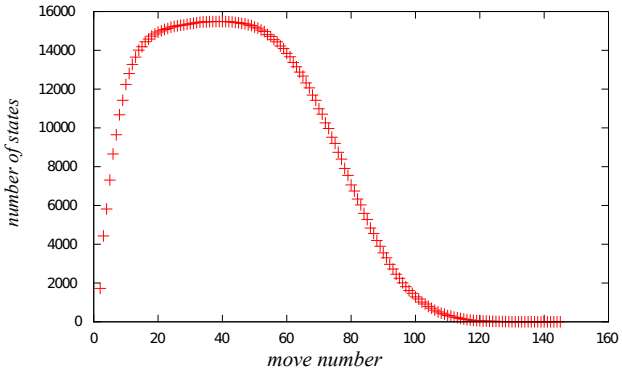

Fig. 3. Distribution of (s, a) with different move numbers.

We did not include human games from Little Golem in our data, primarily because we found that most human players at this site are much weaker than MoHex 2.0.

[1] http://www.littlegolem.net
[2] http://www.cybera.ca

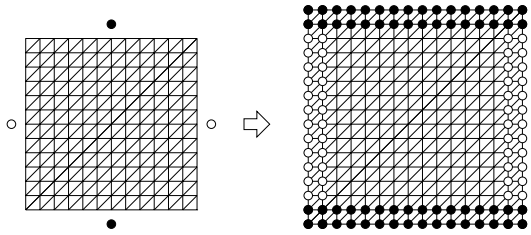

Fig. 4. Hexagonal board mapped as a square grid with moves played at intersections (left). Two extra rows of padding at each side used as input to neural net (right).

TABLE I
INPUT FEATURES

Feature	plane index	description
black/white/empty	0/1/2	black/white/empty points
black bridge	3	black bridge endpoints
white bridge	4	white bridge endpoints
to play	5/6	black/white to play
to play bridge	7	to play save bridge
to play bridge	8	to play form bridge

B. Input Planes

Each position s was preprocessed into feature planes before feeding to the neural net. Since the goal of Hex is to connect two sides of the board, to represent such an objective, we use extra borders to pad the Hex board. Figure 4 shows our padding for 13x13 Hex. In addition to the basic board state representation (black, white, empty points), we utilize the most commonly seen *bridge pattern* to enrich our input features. Table I lists feature representation in each plane. The input features consist of 9 planes with only binary values. We tried to add history information as [23], but experimental results show that it does not seem to help in Hex. This input feature extends the bridge pattern used in [35].

C. Architecture and Training

Inspired by the previous work on Go [21, 22, 23], the neural network we designed for Hex stacks multiple convolution and non-linearity layers. It contains d ($d > 1$) 5×5 or 3×3 convolution layers. Each convolution layer has w filters. Input to the first convolution layer is of dimension $17 \times 17 \times 9$. The first convolution layer convolves using filter size 5×5 with stride of 1, and then applies rectifier function ReLU to its output. Convolution layers 2 to $d - 1$ zero pad the input to an image of 15×15, then convolves with filter size 3×3 and stride of 1, followed by ReLU.

As previous work in Go [21, 22, 23], to avoid losing information, no down-sampling layers such as pooling are applied. The final convolution layer convolves using 1 filter of kernel size 1×1 with stride 1. A position bias is added before applying the final softmax function. The output of this neural net is a probability distribution over each move on the board. The architecture is shown as Figure 5.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TG.2017.2785042, IEEE Transactions on Games

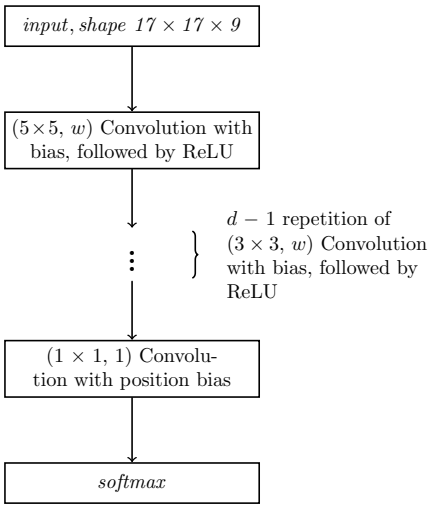

Fig. 5. Architecture, d repetition of 5×5 or 3×3 convolution layers, each with w filters.

The Hex board is symmetric under 180 degree rotation. Therefore, for each training example, with probability 0.5, a rotated symmetric position is randomly selected to train. The target is to maximize the likelihood of parameterized neural net prediction with moves in the training examples. Let $\mathcal{D}$ be the training set, the loss function can be expressed as:

$$\mathcal{L}(\theta; \mathcal{D}) = -\sum_{(s,a)\in\mathcal{D}} \log p_\theta(a|s) \quad (1)$$

For the training, we use adaptive stochastic gradient descent *Adam* optimizer [36]. Default parameters were used, as it has been shown that *Adam* optimizer's default hyperparameters values are widely applicable [36]. We found that, compared to vanilla stochastic gradient descent, faster convergence rate is observed with Adam. We train the neural net with batch size 128. Every 500 steps, accuracy on test data is evaluated. We stop the training after 150,000 steps, the model that achieves best test accuracy is saved. This takes less than 48 hours for a network with $d = 8, w = 128$. The neural net was implemented with Tensorflow [37], and trained on an Intel i7-6700 CPU machine with 32GB RAM and a NVIDIA GTX 1080 GPU.

V. Results

In this section, we report detailed evaluation results of our neural networks. We first present the prediction accuracy of several architectures. Playing strength evaluations are presented subsequently.

A. Prediction Accuracy

It is not very clear which choices of d and w is the best for Hex. We experimentally investigate different architectures by varying d and w. Table II presents the top 1 move prediction for 5 different architectures.

TABLE II
PREDICTION ACCURACY OF CNN MODELS WITH VARYING d AND w

d, w	Best accuracy on test data
5, 64 filters	49.5%
5, 128 filers	53.4%
7, 128 filters	54.7%
8, 128 filters	54.8%
9, 128 filters	54.5%

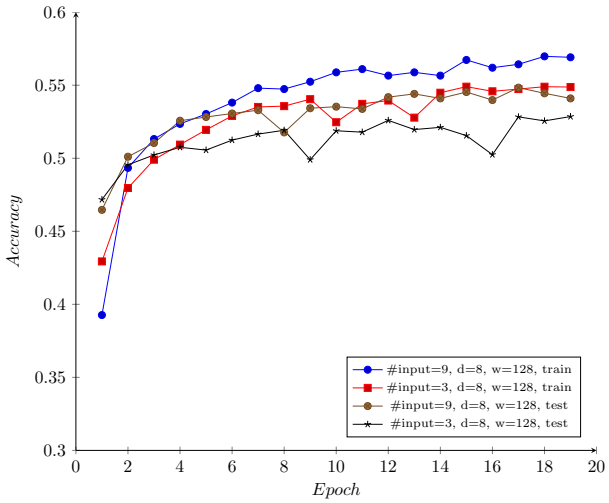

Fig. 6. Comparison of 3 input planes and 9 input planes on training and testing. The training accuracies is the average over an epoch, after each epoch, performance of the model is measured on the test data.

As shown in Table II, neural nets with depth $7-9$ generally produces better results than shallow ones. We obtain the best accuracy with $d = 8, w = 128$. For comparison, the prediction accuracy of $d = 8, w = 128$ architecture on *training* data is 57.6%.

To show the advantage of utilizing the bridge pattern in the input layer, fixing the architecture as $d = 8, w = 128$, we plot the training accuracy over epochs for 3 input planes (i.e., black, white, and empty points) vs 9 input planes in Figure 6. The neural net with 9 input planes performs consistently better.

It is also of interest to know the top $k > 1$ accuracy, since if k is small, the accuracy is high, this confidence might be used to effectively reduce search space. Figure 7 illustrates the top k accuracy for $d = 8, w = 128$ with 9 input planes. The prediction accuracy is above 90% for $k = 8$, a very small number compared to the average branching factor of the 13x13 Hex. When $k = 12$, the predication accuracy exceeds 95%. In the next section, we report the playing strength of the $d = 8, w = 128$ policy neural network.

B. Playing Strength of CNN without Search

We first investigate the quality of the neural net $CNN_{d=8,w=128}$ by testing its playing strength without any search. The highly optimized Resistance evaluation function of Wolve is used as a benchmark to measure the relative

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TG.2017.2785042, IEEE Transactions on Games

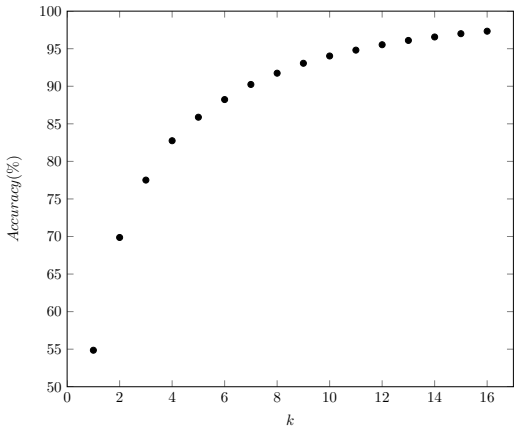

Fig. 7. Top k prediction accuracies of the $d = 8,\ w = 128$ neural network.

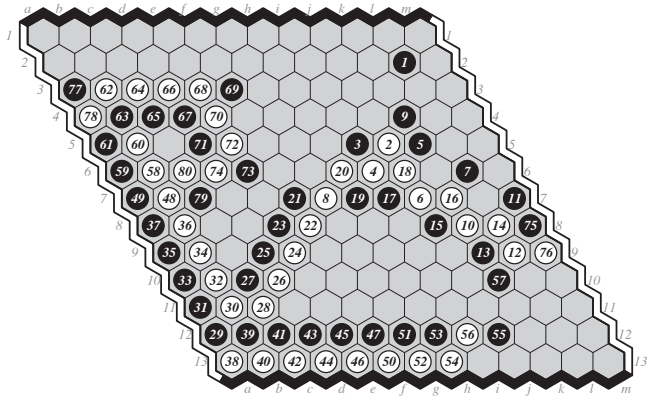

Fig. 8. A game played by 1-ply Wolve (Black) against policy net (White) $CNN_{d=8,w=128}$; neural network won as white.

playing strength. In experiments, this is emulated by Wolve's 1-ply search. It should be noted that, in computer tournament, 2-ply Wolve was already pretty strong and 4-ply is usually adopted since 2009 [13, 17].

Table III shows the winrate of $CNN_{d=8,w=128}$ against the 1-ply Wolve, 6000 games were played from opening board position, in which 3000 neural net plays as black, 3000 as white. No swap rule was used. Summing the winrate in Table III as black and as white, we obtain 97.8%, which implies that the policy neural net model p_θ by $CNN_{d=8,w=128}$ has similar playing strength as the heavy VC and ICA augmented resistance evaluation [3], even though with VC computation, Wolve tends to play perfectly when the game is very close to the end. For speed, we note that the pure policy net player is orders of magnitudes faster than 1-ply Wolve.

TABLE III
WINRATE OF $CNN_{d=8,w=128}$ AGAINST OPTIMIZED RESISTANCE.

Opponent	$CNN_{d=8,w=128}$ as black	$CNN_{d=8,w=128}$ as white
1-ply wolve	64.8%	33.0%

Figure 8 demonstrates a typical game played by the neural net against resistance evaluation. The first move of Wolve is $l2$, presumably because there is a easily computable virtual connection to the top, which greatly influences the resistance evaluation. Move 11 of Wolve is really bad, as it is rendered inferior by White's move $k8$. There is no way Black can connect to the bottom by $m7$ unless White misses $l9$. The game reflects that due to the limit of the virtual connection computation, Resistance tends to favor fill-in dominated cells as they often greatly increase the resistance in the circuit network.

The result is remarkable in the sense that, without any search, the neural network can still answer most black moves accurately, implying that the neural net has already grasped some sophisticated aspects of the game.

[3]Self-play of 1-ply Wolve always ends with a first player win, since the program is deterministic.

It is also interesting to see how well the neural net plays against tournament 4-ply tournament Wolve. Since this version of Wolve is slow, we only played 800 games in total. Table IV shows the result. We also include the result against 1000 simulation limited MoHex 2.0 in Table IV. Even against strong Hex bots, the neural net could achieve about 10% and 20% winrates respectively.

As a comparison, we note that in [38], the authors trained a neural net with deep Q learning. After two weeks of training, their model achieved a playing strength of about 20.4% winrate against 1 second limited MoHex 2.0 as black, and 2.1% as white. While it is not exactly clear how many simulations were achieved within 1 second, in our Intel i7-6700 CPU machine, MoHex 2.0 with 1000 simulations almost always takes a time around 1 second. We suspect the strength difference is due to the following reasons: 1) The training data are different: in [38], the neural net is first trained on a set of games generated from noised Wolve, and then trained on games of neural net self-play. We believe those games are inferior to the games accumulated from MoHex 2.0 self-play. 2) The training method is different. While Q-learning has seen a grand success in single agent Atari games [39], two-player games are more challenging by the fact that the simulated "environment" is non-stationary. 3) The input planes in [38] contain only black/white/empty points plus black/white group information. Perhaps our simple connection pattern augmented representation is better at representing the key tactics of playing Hex.

Our neural net model and training data are publicly available [4].

TABLE IV
WINS OF $CNN_{d=8,w=128}$ AGAINST 4PLY-WOLVE, AND 1000 SIMULATIONS MOHEX 2.0.

Opponent	$CNN_{d=8,w=128}$ as black	$CNN_{d=8,w=128}$ as white
4-ply Wolve	17%	3.25%
MoHex2.0 with 10^3 simulations	33%	8.5%

[4]http://webdocs.cs.ualberta.ca/~cgao3/TCIAIG

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TG.2017.2785042, IEEE Transactions on Games

While this comparison shows that the direct playing strength of our neural network is similar to Resistance, it has some advantages over Resistance: it does not have the pathological behavior as Resistance, also it does not require heavy computation of inferior cells and virtual connections.

C. Combined with Search

The ultimate goal of this work is to develop a stronger Hex player by combining the neural net with search. Unlike a global pattern matching using a neural network, what search tries to build is a local non-parametric model that has the advantage of being flexible and precise. Such is particular important when exact responses are required.

Combining neural net with search is challenging [22, 23, 24], primarily because move evaluation by deep neural networks is much slower than traditional methods. Previous work in computer Go either employ non-blocking asynchronous batch evaluation [24], or simple synchronous evaluation [23]. In both approaches, neural network evaluation is used as prior knowledge before expanding a node, giving preferred nodes higher prior probability.

We combine our neural network with MoHex 2.0, the reigning world champion player since 2013. MoHex 2.0 is an enhanced version of MoHex 2011 [17]. It is built upon benzene [5], adopting the *smartgame* and *gtpengine* libraries from Fuego [40]. The major improvement of MoHex 2.0 is its prior-knowledge-augmented probabilistic playout.

More precisely, the MCTS phases in MoHex 2.0 work as follows:

- *In tree phase*: In this phase, starting from the root node, a child node is selected from its parent until a leaf is reached. At tree node s, a move is selected according to this formula:

$$score(s,a) = (1-w) \times (Q(s,a) + c_b \times \sqrt{\frac{\ln N(s)}{N(s,a)}}) + w \times R(s,a) + c_{pb} \times \frac{\rho(s,a)}{\sqrt{N(s,a)+1}}$$

  where $N(s)$ is the visit count of s, $N(s,a)$ is the visit count of move a at s, $Q(s,a)$ is the Q-value of (s,a), $R(s,a)$ is the RAVE value [41], $\rho(s,a)$ is the prior probability calculated from move pattern weights. w is dynamically adjusted during the search, c_b and c_{pb} are turned constants.
- *Expansion*: A leaf node is expanded only when its visit count exceeds a *expansion threshold*, which is set to 10 in MoHex 2.0.
- *Pattern based playout*: Pattern weights have been trained offline. In each playout, a softmax policy selects moves according to move pattern weights.
- *Back-propagation update*. After the playout, game result is recorded, then MCTS updates values in visited nodes according to the playout result.

[5] `http://benzene.sourceforge.net/`

The best performance of MoHex 2.0 is achieved after tuning its parameters by CLOP [19, 20]. Other optimizations implemented in MoHex 2.0 include: 1) A pre-search: Inferior cells and connection strategies are computed from the root node before MCTS. If a winning connection is discovered, a winning move will be selected, search withdraws. 2) Knowledge computation: Whenever the visit count of a node exceeds a *knowledge threshold*, H-search to compute virtual connection and inferior cell analysis are applied. This often leads to effective move pruning. 3) Time management. A search is said to be unstable if by the end, the move with the largest visit count disagrees with the move with highest Q-value. MoHex 2.0 extends the search by half of its original time in this case.

After the search terminates, MoHex 2.0 finally selects the move with the largest visit count.

In MoHex 2.0, the prior knowledge $\rho(s,a)$ is computed by a rough estimate from relative move pattern weights. To see the effectiveness of our policy neural network, the straightforward modification is to replace $\rho(s,a)$ by $p_\theta(s,a)$ — the move probability computed by our policy neural network. All other tuning parameters are left unchanged.

The new program after adding neural net $CNN_{d=8,w=128}$ is named as MoHex-CNN. It is modified directly upon MoHex 2.0 code in benzene, and then compiled with the Tensorflow C++ libraries. On the same i7-6700 CPU, 32GB RAM, GTX-1080 GPU machine [6], we run several tournaments to compare MoHex 2.0 and MoHex-CNN. Similar to AlphaGo [24], we also prepare another program MoHex-CNN$_{puct}$ that uses a variant of PUCT [42]. It selects moves according to $score(s,a) = Q(s,a) + c_{pb} \times p_\theta(s,a) \times \frac{\sqrt{N(s)}}{N(s,a)+1}$.

The first tournament uses the same number of simulations for each program. From empty board, 400 games were played by MoHex-CNN and MoHex-CNN$_{puct}$ against MoHex 2.0, in which MoHex 2.0 plays 200 games as black and 200 as white.

Table V contains the playing results. It is clear that, under those settings, the new programs MoHex-CNN and MoHex-CNN$_{puct}$ are stronger than MoHex 2.0, since they consistently won more than 62% over all played games. Since the difference between MoHex-CNN and MoHex 2.0 lies only in the prior probability initialization, the results indicate that the more accurate prior knowledge due to CNN can indeed improve MoHex, given the same number of simulations.

In practice, evaluation speed of the neural network is a concern. In MoHex, we find that whenever expanding a node, *prior pruning* (use ICA and VC to remove proven inferior cells) is applied. Since those computations are costly and are independent from neural net evaluation, we implement an asynchronous evaluation that runs in parallel with this prior pruning. As a result of this implementation, the computation

[6] For single thread tasks, this computer is about 3.5 times faster than those used for producing the training data.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TG.2017.2785042, IEEE Transactions on Games

TABLE V
WINRATES AGAINST MOHEX 2.0 OF MOHEX-CNN AND MOHEX-CNN$_{puct}$ WITH SAME NUMBER OF SIMULATIONS.

Opponent	#Simulations	MoHex2.0 as white	MoHex2.0 as black	Overall winrate
MoHex-CNN	10^3	94.5%	44.5%	69.5%
	10^4	90%	56%	73%
MoHex-CNN$_{puct}$	10^3	86%	39%	62.5%
	10^4	83.5%	53%	68.3%

overhead becomes very small: in our experiments on the i7-6700 CPU machine with a GTX 1080 GPU, MoHex with CNNs took about 0.19 ms per simulation, while MoHex 2.0 took about 0.17 ms.

The next experiment compares MoHex-CNN and MoHex-CNN$_{puct}$ to MoHex 2.0 with equal time budget: 10 seconds per move. Since when the swap rule is applied, the second player has the option to steal the opening move, we run the tournament by iterating over all opening cells in the board. For 13x13 Hex, there are 85 distinct openings after removing symmetries. For each opening, we run 5 independent games for each color between MoHex-CNN$_{puct}$ or MoHex-CNN against MoHex 2.0.

Table VI summarizes the result of those 850 games played by MoHex with CNNs and MoHex 2.0. It is clear that, both MoHex-CNN$_{puct}$ and MoHex-CNN have better performance against MoHex 2.0 even with same computation time. Consistent with the results in Table V, MoHex-CNN plays better than MoHex-CNN$_{puct}$, presumably because MoHex-CNN$_{puct}$ is too aggressive on nodes suggested by the neural net while MoHex-CNN retains the good exploration due to RAVE.

MoHex-CNN won the 2017 computer Hex Olympiad against Ezo-CNN [43], an updated version of Ezo using neural networks as its move ordering function.

TABLE VI
WINRATES AGAINST MOHEX 2.0 OF MOHEX-CNN$_{puct}$ AND MOHEX-CNN WITH SAME TIME LIMIT PER MOVE, 95% CONFIDENCE.

Opponent	MoHex2.0 as white (%)	MoHex2.0 as black (%)
MoHex-CNN$_{puct}$	71.7 ± 0.1	53.2 ± 0.1
MoHex-CNN	78.6 ± 0.4	61.2 ± 0.4

VI. CONCLUSIONS AND FUTURE WORK

In this work, we have presented neural network that predicts the next move played by expert computer player. We achieved a prediction accuracy near 55%, and showed that such a high prediction accuracy can be used in a strong Hex player. Like the previous work in computer Go, we find that even without any search, the neural network plays Hex reasonably well: it has a similar playing strength as a highly optimized Resistance evaluation function. Furthermore, by using a neural net as a learnt prior in MCTS, we were able to improve the state-of-the-art MCTS player MoHex 2.0.

There is still a great deal to be explored for the game of Hex. We believe that future work can be conducted in the following directions:

- For the neural network design, we have only explored a relatively simple architecture. Perhaps better neural nets exist, such as Residual neural nets [44, 45, 46]. Introducing more regularization techniques like dropout [47, 48], batch normalization [49], stochastic depth [50], swapout [51] might further improve the prediction accuracy.
- For the training, the neural nets were trained to maximize the likelihood of a single next move, i.e., to minimize the KL divergence between p_θ and a Dirac delta distribution. Recent studies have shown that predicting a more smoothed distribution helps to reduce over-fitting [52, 53], and using task reward to define such a distribution establishes a link between maximum likelihood and goal-driven reinforcement learning [54]. Since the data we have trained upon is inherently imperfect, incorporating rewards to the training target may be beneficial.
- Since evaluation of CNN is typically slow, more sophisticated asynchronous evaluation is worth further studying. However, with AI accelerators such as TPU [55] and better neural net designs, perhaps simple synchronous evaluation will become adequate.
- Training a value estimation network [24] could also be useful in search.

VII. ACKNOWLEDGMENTS

We appreciate the anonymous reviewers' valuable comments, which have been helpful in improving the quality of this paper. We gratefully acknowledge NSERC for funding this research. We thank Cybera for providing computing resources to produce the training data.

REFERENCES

[1] P. Hein, "Vil de laere polygon," *Article in Politiken newspaper*, vol. 26, 1942.
[2] J. F. Nash, "Some games and machines for playing them," 1952.
[3] M. Gardner, "The Scientific American book of mathematical puzzles and diversions," 1959.
[4] C. E. Shannon, "Computers and automata," *Proceedings of the IRE*, vol. 41, no. 10, pp. 1234–1241, 1953.
[5] D. Gale, "The game of Hex and the Brouwer fixed-point theorem," *The American Mathematical Monthly*, vol. 86, no. 10, pp. 818–827, 1979.
[6] E. R. Berlekamp, J. H. Conway, and R. K. Guy, *Winning ways for your mathematical plays*, vol. 3. AK Peters Natick, 2003.
[7] S. Reisch, "Hex ist PSPACE-vollständig," *Acta Informatica*, vol. 15, no. 2, pp. 167–191, 1981.
[8] Édouard Bonnet, F. Jamain, and A. Saffidine, "On the complexity of connection games," *Theoretical Computer Science*, vol. 644, pp. 2 – 28, 2016. Recent Advances in Computer Games.
[9] V. V. Anshelevich, "The game of Hex: An automatic theorem proving approach to game programming," in *AAAI/IAAI*, pp. 189–194, 2000.

[10] V. V. Anshelevich, "A hierarchical approach to computer Hex," *Artificial Intelligence*, vol. 134, no. 1, pp. 101–120, 2002.

[11] P. T. Henderson, *Playing and solving the game of Hex*. PhD thesis, University of Alberta, 2010.

[12] R. Hayward, "Six wins hex tournament," *ICGA Journal*, vol. 29, no. 3, pp. 163–165, 2006.

[13] B. Arneson, R. Hayward, and P. Henderson, "Wolve wins Hex tournament," *ICGA Journal*, vol. 32, no. 1, pp. 49–53, 2008.

[14] L. Kocsis and C. Szepesvári, "Bandit based Monte-Carlo planning," in *European conference on machine learning*, pp. 282–293, Springer, 2006.

[15] R. Coulom, "Efficient selectivity and backup operators in Monte-Carlo tree search," in *International Conference on Computers and Games*, pp. 72–83, Springer, 2006.

[16] C. B. Browne, E. Powley, D. Whitehouse, S. M. Lucas, P. I. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis, and S. Colton, "A survey of Monte Carlo tree search methods," *IEEE Transactions on Computational Intelligence and AI in games*, vol. 4, no. 1, pp. 1–43, 2012.

[17] B. Arneson, R. B. Hayward, and P. Henderson, "Monte Carlo tree search in Hex," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 2, no. 4, pp. 251–258, 2010.

[18] B. Arneson, R. Hayward, and P. Henderson, "Mohex wins Hex tournament," *ICGA journal*, vol. 32, no. 2, p. 114, 2009.

[19] S.-C. Huang, B. Arneson, R. B. Hayward, M. Müller, and J. Pawlewicz, "Mohex 2.0: a pattern-based MCTS hex player," in *International Conference on Computers and Games*, pp. 60–71, Springer, 2013.

[20] R. Coulom, "Computing Elo ratings of move patterns in the game of Go," in *Computer Games Workshop*, 2007.

[21] C. Clark and A. Storkey, "Training deep convolutional neural networks to play Go," in *International Conference on Machine Learning*, pp. 1766–1774, 2015.

[22] C. J. Maddison, A. Huang, I. Sutskever, and D. Silver, "Move evaluation in Go using deep convolutional neural networks," in *International Conference on Learning Representations*, 2015.

[23] Y. Tian and Y. Zhu, "Better computer Go player with neural network and long-term prediction," in *International Conference on Learning Representations*, 2015.

[24] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, *et al.*, "Mastering the game of Go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, pp. 484–489, 2016.

[25] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, pp. 1097–1105, 2012.

[26] M. Campbell, A. J. Hoane, and F.-h. Hsu, "Deep blue," *Artificial intelligence*, vol. 134, no. 1-2, pp. 57–83, 2002.

[27] V. V. Anshelevich, "Hexy wins hex tournament," *ICGA Journal*, vol. 23, no. 3, pp. 181–184, 2000.

[28] J. Pawlewicz, R. Hayward, P. Henderson, and B. Arneson, "Stronger Virtual Connections in Hex," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 7, no. 2, pp. 156–166, 2015.

[29] R. Hayward, Y. Björnsson, M. Johanson, M. Kan, N. Po, and J. van Rijswijck, "Solving 7×7 hex: Virtual connections and game-state reduction," in *Advances in Computer Games*, pp. 261–278, Springer, 2004.

[30] P. Henderson, B. Arneson, and R. B. Hayward, "Solving 8×8 Hex," in *Proc. IJCAI*, vol. 9, pp. 505–510, Citeseer, 2009.

[31] J. Pawlewicz and R. B. Hayward, "Sibling conspiracy number search," in *Eighth Annual Symposium on Combinatorial Search*, 2015.

[32] K. Takada, M. Honjo, H. Iizuka, and M. Yamamoto, "Developing evaluation function of Hex using board network characteristics and SVM," *Transactions of the Japanese Society for Artificial Intelligence*, vol. 30, pp. 729–736, 2015.

[33] R. Munos *et al.*, "From bandits to Monte-Carlo Tree Search: The optimistic principle applied to optimization and planning," *Foundations and Trends® in Machine Learning*, vol. 7, no. 1, pp. 1–129, 2014.

[34] S. Gelly and D. Silver, "Combining online and offline knowledge in UCT," in *Proceedings of the 24th international conference on Machine learning*, pp. 273–280, ACM, 2007.

[35] C. Gao, M. Müller, and R. Hayward, "Focused depth-first proof number search using convolutional neural networks for the game of hex," in *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI-17*, pp. 3668–3674, 2017.

[36] D. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *International Conference on Learning Representations*, 2014.

[37] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015. Software available from tensorflow.org.

[38] K. Young, G. Vasan, and R. Hayward, "Neurohex: A deep q-learning hex agent," in *Computer Games*, pp. 3–18, Springer, 2016.

[39] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, *et al.*, "Human-level

control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[40] M. Enzenberger, M. Müller, B. Arneson, and R. Segal, "Fuego—an open-source framework for board games and go engine based on monte carlo tree search," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 2, no. 4, pp. 259–270, 2010.

[41] S. Gelly and D. Silver, "Monte-Carlo tree search and rapid action value estimation in computer Go," *Artificial Intelligence*, vol. 175, no. 11, pp. 1856–1875, 2011.

[42] C. D. Rosin, "Multi-armed bandits with episode context," *Annals of Mathematics and Artificial Intelligence*, vol. 61, no. 3, pp. 203–230, 2011.

[43] K. Takata, H. Iizuka, and M. Yamaoto, "Computer Hex using move evaluation method based on convolutional neural network," in *IJCAI 2017 Computer Games Workshop*, 2017, in-press.

[44] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

[45] K. He, X. Zhang, S. Ren, and J. Sun, "Identity mappings in deep residual networks," in *European Conference on Computer Vision*, pp. 630–645, Springer, 2016.

[46] S. Zagoruyko and N. Komodakis, "Wide residual networks," *arXiv preprint arXiv:1605.07146*, 2016.

[47] N. Srivastava, G. E. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting," *Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[48] J. Tompson, R. Goroshin, A. Jain, Y. LeCun, and C. Bregler, "Efficient object localization using convolutional networks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 648–656, 2015.

[49] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *International Conference on Machine Learning*, pp. 448–456, 2015.

[50] G. Huang, Y. Sun, Z. Liu, D. Sedra, and K. Q. Weinberger, "Deep networks with stochastic depth," in *European Conference on Computer Vision*, pp. 646–661, Springer, 2016.

[51] S. Singh, D. Hoiem, and D. Forsyth, "Swapout: Learning an ensemble of deep architectures," in *Advances in Neural Information Processing Systems*, pp. 28–36, 2016.

[52] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2818–2826, 2016.

[53] G. Pereyra, G. Tucker, J. Chorowski, Ł. Kaiser, and G. Hinton, "Regularizing neural networks by penalizing confident output distributions," *ICLR 2017 workshop*, 2017.

[54] M. Norouzi, S. Bengio, N. Jaitly, M. Schuster, Y. Wu, D. Schuurmans, *et al.*, "Reward augmented maximum likelihood for neural structured prediction," in *Advances In Neural Information Processing Systems*, pp. 1723–1731, 2016.

[55] N. P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borchers, R. Boyle, P.-l. Cantin, C. Chao, C. Clark, J. Coriell, M. Daley, M. Dau, J. Dean, B. Gelb, T. V. Ghaemmaghami, R. Gottipati, W. Gulland, R. Hagmann, C. R. Ho, D. Hogberg, J. Hu, R. Hundt, D. Hurt, J. Ibarz, A. Jaffey, A. Jaworski, A. Kaplan, H. Khaitan, D. Killebrew, A. Koch, N. Kumar, S. Lacy, J. Laudon, J. Law, D. Le, C. Leary, Z. Liu, K. Lucke, A. Lundin, G. MacKean, A. Maggiore, M. Mahony, K. Miller, R. Nagarajan, R. Narayanaswami, R. Ni, K. Nix, T. Norrie, M. Omernick, N. Penukonda, A. Phelps, J. Ross, M. Ross, A. Salek, E. Samadiani, C. Severn, G. Sizikov, M. Snelham, J. Souter, D. Steinberg, A. Swing, M. Tan, G. Thorson, B. Tian, H. Toma, E. Tuttle, V. Vasudevan, R. Walter, W. Wang, E. Wilcox, and D. H. Yoon, "In-datacenter performance analysis of a tensor processing unit," in *Proceedings of the 44th Annual International Symposium on Computer Architecture*, ISCA '17, pp. 1–12, ACM, 2017.