Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence (IJCAI-18)

Three-Head Neural Network Architecture for Monte Carlo Tree Search

Chao Gao, Martin Müller, Ryan Hayward
University of Alberta
{cgao3, mmueller, hayward}@ualberta.ca

Abstract

AlphaGo Zero pioneered the concept of two-head neural networks in Monte Carlo Tree Search (MCTS), where the policy output is used for prior action probability and the state-value estimate is used for leaf node evaluation.

We propose a three-head neural net architecture with policy, state- and action-value outputs, which could lead to more efficient MCTS since neural leaf estimate can still be back-propagated in tree with delayed node expansion and evaluation. To effectively train the newly introduced action-value head on the same game dataset as for two-head nets, we exploit the optimal relations between parent and children nodes for data augmentation and regularization. In our experiments for the game of Hex, the action-value head learning achieves similar error as the state-value prediction of a two-head architecture. The resulting neural net models are then combined with the same Policy Value MCTS (PV-MCTS) implementation. We show that, due to more efficient use of neural net evaluations, PV-MCTS with three-head neural nets consistently performs better than the two-head ones, significantly outplaying the state-of-the-art player MoHex-CNN.

1 Introduction

Monte Carlo Tree Search (MCTS) [Coulom, 2006; Kocsis and Szepesvári, 2006; Browne *et al.*, 2012] is a modern heuristic search paradigm that has been applied to a large number of sequential decision making problems. It has led to spectacular progress in two-player games with large branching factors such as Go [Enzenberger *et al.*, 2010; Silver *et al.*, 2016] and Hex [Arneson *et al.*, 2010; Huang *et al.*, 2013; Gao *et al.*, 2017].

MCTS grows a selective tree by concentrating on nodes with better averaged estimations. Each search iteration consists of four distinct phases: 1) The *in-tree* phase traverses the current tree from the root until a leaf is reached. Child nodes are selected by a function such as UCT [Kocsis and Szepesvári, 2006], which balances *exploration* and *exploitation*; 2) The *expansion* phase expands a leaf node, typically after its visit count has reached an expansion threshold; 3) Leaf nodes are evaluated, for example by randomized rollouts; 4) Results are back-propagated in the tree. The leaf evaluation result can be viewed as a "critic" which influences later growth of the tree. While MCTS works without any game-specific evaluation, its performance is often drastically improved by incorporating domain knowledge [Gelly and Silver, 2007].

Deep neural networks [LeCun *et al.*, 2015] have been applied to a variety of sequential decision problems, including single agent Atari games [Mnih *et al.*, 2015], or two-player games such as Go [Clark and Storkey, 2015; Tian and Zhu, 2015; Maddison *et al.*, 2015; Silver *et al.*, 2016; 2017] or Hex [Gao *et al.*, 2017; Anthony *et al.*, 2017]. In those works, the neural nets are served as an expressive function approximator that can provide high quality policy or value outputs after well-training.

AlpahGo Zero [Silver *et al.*, 2017] introduced a PV-MCTS framework with a two-head neural net that gives policy and state-value outputs, used respectively for prior probability initialization and leaf node evaluation. However, one limitation of two-head net is that it requires to always expand the leaf nodes to obtain neural leaf evaluation. In contrast, many MCTS programs expand a node only when its visit count exceeds a positive threshold. Evaluating each node by a neural network is computationally expensive. In AlphaGo Zero, despite running on special purpose TPU hardware, asynchronous evaluation is still needed [Silver *et al.*, 2017].

Contributions: In this paper, we introduce a Three Head Neural Network (3HNN) architecture for Monte Carlo Tree Search. The main advantage brought by this architecture is that the action-value can be immediately back-propagated to the tree without expanding a node. To train 3HNN, we utilize optimal minimax consistency between a parent state and its successor states. Specifically, the observation that a losing game state implies all after-states be winning (for the opponent) is used for augmenting training data for the action-value head. The inconsistency between state- and action-value predictions is added to the loss function as a penalty.

We apply 3HNN to the game of Hex. We show that, using the proposed techniques, when trained on the same set of expert games, the obtained three-head neural nets attain similar policy and value prediction accuracies as the two-head

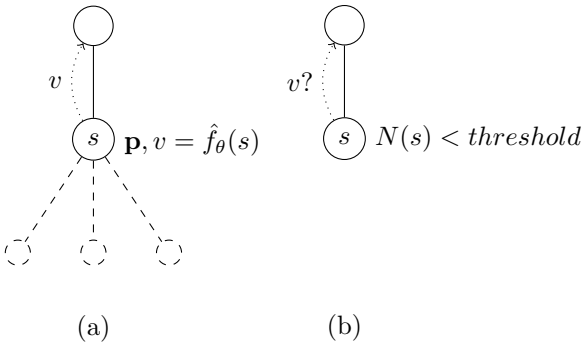

Figure 1: Two-head architecture in PV-MCTS: The leaf node must be expanded (a) with threshold 0, otherwise no neural net value estimation can be backed up (b). $\hat{f}_\theta$ represents a two-head neural net that each evaluation of state s yields a vector of move probabilities $\mathbf{p}$ and state-value v. $N(s)$ is the visit count of s.

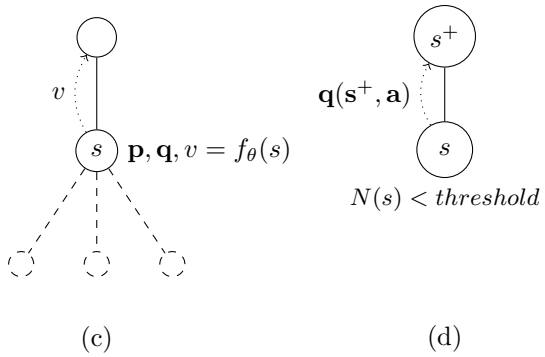

Figure 2: PV-MCTS with three-head neural net f_θ: The leaf node be expanded with any threshold. The action-value estimate can be backed up even if the leaf has not been expanded, because it was stored there upon node creation.

ones. However, when combined with PV-MCTS, due to the usage of action-value head in delayed node expansion, our new programs with 3HNN consistently achieve better performance than those using two-head (policy and state-value) neural nets, significantly outperforming MoHex-CNN [Gao *et al.*, 2017] — the 2017 computer Olympiad champion on 13×13 Hex — without training on new game datasets.

The rest of the paper is organized as follows. In Section 2, we describe our three-head neural net architecture for MCTS. Section 3 discusses related work. We present experimental results in Section 4, and conclude the paper in Section 5.

2 Three-Head Neural Net for Monte Carlo Tree Search

In this section, we review the limitation of PV-MCTS with two-head neural net, and then discuss how to effectively train the three-head neural net on existing training data. The aim of this paper to learn better neural nets that would lead to more efficient MCTS, given fixed training games.

2.1 PV-MCTS with Delayed Node Expansion

In Policy Value Monte Carlo Tree Search (PV-MCTS), typically, each neural net evaluation is computed along with node expansion. The move probabilities are saved to children nodes, then used as prior probability in the *selection* phase of MCTS. The state-value estimate is used to replace Monte Carlo leaf evaluation. However, with a Two Head Neural Net (2HNN), PV-MCTS is restricted to use expansion threshold of 0, because otherwise there would be no neural value estimation backed up. See Figure 1.

To address such an issue, we propose a three-head neural net that outputs also a vector of action-values, illustrated in Figure 2. When expanding a node s, s is evaluated by the parameterized neural net, yielding three outputs: policy $\mathbf{p}$, a vector of next action-values $\mathbf{q}$, and state-value v. The policy and action-value information are stored by newly created children nodes. When the visit count $N(s)$ is below expansion threshold, the stored action-value can be backed up to the tree.

It is apparent that, to have an efficient usage of neural net evaluations, PV-MCTS with 2HNN can either 1) restrict the expansion threshold to be 0, or 2) still use playout result when the leaf node is below an expansion threshold $\zeta \geq 1$.

For case 1), suppose PV-MCTS-2HNN and PV-MCTS-3HNN are allocated with the same amount of computation time $\tilde{T}$ on the same hardware. Let t and t' be respectively the one simulation time cost for PV-MCTS-2HNN and PV-MCTS-3HNN. Assuming negligible time overhead for the extra action-head in 3HNN, then $t' \leq t$ since PV-MCTS-2HNN calls the neural net every simulation while PV-MCTS-3HNN does not. Thus, the number of neural leaf estimates received by PV-MCTS-3HNN is $\frac{\tilde{T}}{t'} - \frac{\tilde{T}}{t}$ more than that of PV-MCTS-2HNN.

For case 2), a reasonable assumption is that PV-MCTS-3HNN and PV-MCTS-2HNN will have equal computation time for the same number of simulations, then we have the following observation:

Proposition 1. *Suppose the total number of simulations for MCTS is T, expansion threshold is $\zeta \geq 1$, after the search terminates, PV-MCTS-3HNN receives at least $T - \frac{T}{\zeta}$ more neural leaf estimates than PV-MCTS-2HNN.*

Proof. It is clear that the number of neural leaf estimates received by PV-MCTS-3HNN is T, since every simulation will back up a leaf estimate of either state-value or action value. For PV-MCTS-2HNN, let $\hat{n}$ be the number of internal nodes after T simulations, L be the set of leaf nodes. The number of neural leaf estimates used by PV-MCTS-2HNN is thus $\hat{n}$, since neural net is called whenever there is an expansion. Another observation is that $\hat{n}\zeta \leq T$, because $T = \hat{n}\zeta + \sum_{\check{s}\in L} N(\check{s})$, so $\hat{n} \leq \frac{T}{\zeta}$. Therefore, PV-MCTS-3HNN uses at least $T - \frac{T}{\zeta}$ more neural leaf estimates than PV-MCTS-2HNN. □

2.2 Training 3HNN

In the loss function described in [Silver *et al.*, 2017], the state-value head is trained by replaying each game backward and

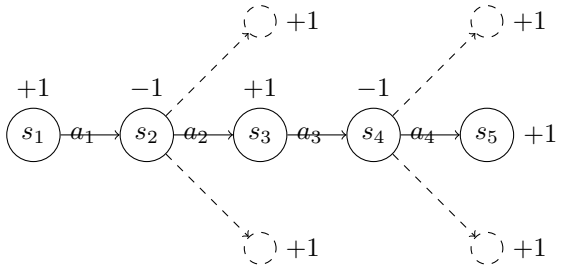

Figure 3: A game implies extra data via the AND constraint.

minimize the mean square error between the predicated values and observed game results. Combining policy loss and L_2 regularization, the loss function is expressed as:

$$\hat{L}(\hat{f}_\theta;\mathcal{D}) = \sum_{(s,a,z_s)\in\mathcal{D}} \left(w(z_s - v(s))^2 - \log p(a|s) + c||\theta||^2 \right) \tag{1}$$

Here, $0 < w \leq 1$ is a weighting factor that used to control the relative weight of the value loss. c is a constant for the level of L_2 regularization. (s, a) is a state-action pair from dataset $\mathcal{D}$, and z_s is the game result with respect to s.

It seems much more data intensive to train a three-head neural net, since all action-values must be available. We show that by using the consistency between a parent and a child node in minimax, we can augment large number of action-values to train 3HNN on the same training data as for 2HNN.

Assume a game has only two outcomes, either first player win or second player win. Then, for any given game state s, the following relations hold:

- The OR constraint: if s is winning, then at least one action leads to a losing state for the opponent.
- The AND constraint: if s is losing, then *all* actions lead to a winning state for the opponent.

Let dataset $\mathcal{D}$ be a collection of games obtained from some strong players. Under ideal condition, if the games in $\mathcal{D}$ were played by perfect players, for each game $g \in \mathcal{D}$, g implies a tree rather than a single trajectory. See Figure 3.

Reflecting in the loss function, we introduce the following loss term:

$$L_Q(f_\theta;\mathcal{D}) = \sum_{(s,a,z_s)\in\mathcal{D}} \frac{\max(-z_s, 0)}{|\mathcal{A}(s)|} \sum_{a'\in\mathcal{A}(s)} (z_s + q(s,a'))^2 \tag{2}$$

where $\mathcal{A}(s)$ is the action set at s.

A two-player alternate-turn zero-sum game can be formulated as an Alternating Markov Game (AMG) [Littman, 1996]. For an AMG, suppose the value function $v(s)$, $q(s, a)$ are with respect to the player to play at s or (s, a). The optimal Bellman equation can be expressed as:

$$v^*(s) = -\min_a q^*(s,a), s \in \mathcal{S}, \tag{3}$$

Here, v^* and q^* are the optimal state-value and action-value functions. The above optimal Bellman equation expresses the consistency of state and action values under an optimal policy.

Since in our three-head neural net, there are both state- and action-value outputs, to force the state and action values to satisfy the optimal consistency, we augment our loss function by adding the optimal Bellman error as a quadratic penalty.

$$L_P(f_\theta;\mathcal{D}) = \sum_{(s,a,z_s)\in\mathcal{D}} (\min_{a'} q(s,a') + v(s))^2 \tag{4}$$

Combining the usual state-, action-value and policy losses, we propose the following loss function:

$$\begin{aligned} L(f_\theta;\mathcal{D}) = \sum_{(s,a,z_s)\in\mathcal{D}} \Bigg(& w\Big(\frac{1}{2}(z_s - v(s))^2 \\ & + \frac{1}{2}(z_s + q(s,a))^2\Big) \\ & - \log p(a|s) + c||\theta||^2 \Bigg) + wL_Q + wL_P \end{aligned} \tag{5}$$

Note that we assume the game is with deterministic transition, which means $q(s, a)$ is equivalent to $v(s')$ if taking a at s leads to s'. w and c are weighting parameters as in (1), $v(s)$, $q(s, a)$ and $p(a|s)$ are predictions from 3HNN f_θ.

3 Related Work

Monte Carlo Tree Search has been an active research domain after the seminal work by [Coulom, 2006] for Computer Go. [Kocsis and Szepesvári, 2006] introduces UCT search which uses upper confidence bound in the selection phase, striking a balance between exploration and exploitation.

Deep neural nets can provide strong domain knowledge. The PV-MCTS algorithm [Silver *et al.*, 2016; 2017] uses a deep policy net to set prior move probabilities, and a deep value net for estimating the value of leaf nodes in the search. The resulting AlphaGo program [Silver *et al.*, 2016] convincingly beat top human professional players [Silver *et al.*, 2016] in 19×19 Go. The updated version AlphaGo Zero [Silver *et al.*, 2017] adopts essentially the same search framework, in which the Monte Carlo rollout was totally abandoned. The better performance is much due to better quality of neural nets obtained by an iterated training style that optimizes the neural net based on data generated by PV-MCTS with previous neural net models. The success of PV-MCTS in Go relies on advanced hardware TPU [Jouppi and al, 2017] for fast neural net inference. It is still a research question how to further improve the efficiency of PV-MCTS on regular hardware, and obtain large number of high quality expert games with limited computation resources.

[Nash, 1952] proved that the game theoretic result of Hex is the first player win, but explicit winning strategy is unknown. Hex has been a challenging domain in Artificial Intelligence research since Shannon's seminal work [Shannon, 1953]. Similar to Go, the problem of lacking reliable hard-crafted evaluation function was sidestepped by Monte Carlo tree search [Arneson *et al.*, 2010; Huang *et al.*, 2013]. Deep neural nets that learn on expert MCTS

games have also been applied to Hex [Gao *et al.*, 2017; Anthony *et al.*, 2017]. MoHex-CNN [Gao *et al.*, 2017] was the state-of-the-art computer player on 13×13 Hex, winning against MoHex 2.0 [Huang *et al.*, 2013; Pawlewicz *et al.*, 2015].

4 Experiments

In this section, we present experimental results on 13×13 Hex, the largest board size that has been adopted in computer program competitions. We first present the neural net training results, then show its performance when combined with MCTS. We give detailed comparisons between two and three heads architectures trained using similar loss functions. Finally, we present results after strengthening the neural critic by considering both state- and action- value estimates.

4.1 Setup

Dataset

We use the publicly available training dataset of MoHex-CNN [Gao *et al.*, 2017], generated from MoHex 2.0 self-play[1], containing about 10^6 distinct state-action-value examples. Each game is an alternating sequence of black and white moves, along with game result.

Neural Net Design

We adopt a thin residual neural net [He *et al.*, 2016a] that has 10 blocks, each block with two convolution layers, each layer with 32 3×3 filters. We use pre-activation [He *et al.*, 2016b] in each residual block which applies batch normalization [Ioffe and Szegedy, 2015] and ReLU before convolution. The input features have 4 binary planes that contain only basic board state information, i.e., black stones, white stones, empty points, and toplay plane. In Hex, each player owns two of the board's four sides: to indicate this, we pad each board's side with a row of stones of the appropriate color. See Figure 4 for detailed neural network design.

By contrast, [Gao *et al.*, 2017] used 9 feature planes, e.g. adding precomputed small virtual connections called bridges, which improved net accuracy. In this paper, rather than adopt game-specific features, we take a zero-knowledge approach, using only basic features, and relying on the deep neural net to extract more meaningful features.

Implementation

The neural nets are implemented with Tensorflow, trained by Adam optimizer [Kingma and Ba, 2014] using default learning rate with mini-batch size of 128 for 100 epochs. We execute experiments on the same Intel i7-6700 CPU computer with a single GTX 1080 GPU and 32 GB RAM. For loss function (5), we set L_2 regularization constant c to 10^{-5}, and value loss weight w to 0.01.

4.2 Prediction Accuracies

As in [Gao *et al.*, 2017], the dataset is partitioned into training and testing sets, where examples from testing set do not appear in the training set. The testing set contains about 10^5 positions.

[1]The trained neural net models of MoHex-CNN and dataset are publicly available https://drive.google.com/drive/folders/18MdnvMItU7O2sEJDlbmk_ZzUhZG7yDK9.

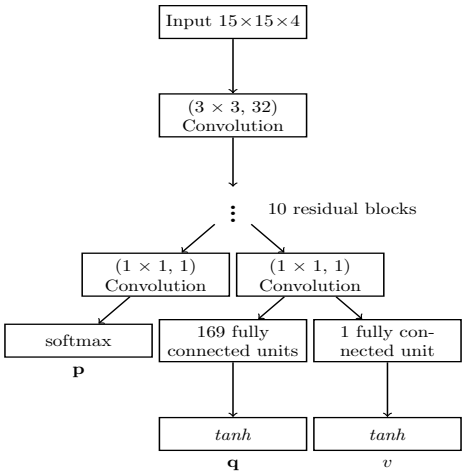

Figure 4: Neural net architecture. Each residual block repeats twice batch normalization, ReLU, convolution using 32 3 × 3 filters, then adds up original input before leaving the block.

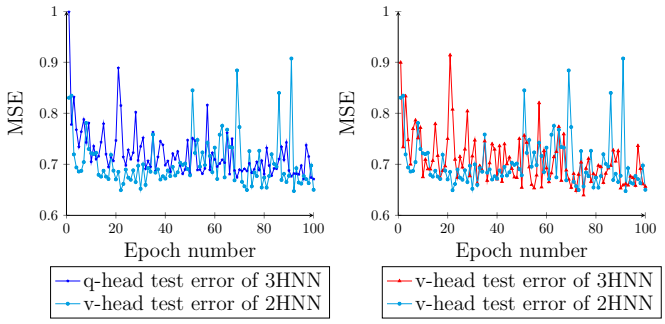

Figure 5: Mean Square Errors of two and three heads residual nets.

To show the learning progress of the neural nets, at the end of each epoch, model parameters are saved and evaluated on the test data. We then plot the move prediction accuracies and mean square errors (MSEs) in Figures 5 and 6. The possible range of MSE is $[0, 4.0]$.

We also train a 2HNN with identical residual architecture as our 3HNN except that it has only policy and state-value heads. This can be emulated by ignoring the action-value head in Figure 4, optimized with loss function (1).

Figures 5 (left) shows that the MSEs from the newly introduced action-value heads in 3HNN achieved similar accuracies with the state-value head of 2HNN. As learning epoch increases, both state- and action-value heads in 3HNN appear to have lower variances than 2HNN, presumably because of the quadratic penalty term in function (5) helped to regularize 3HNN. Although 3HNN did not produce significantly better predictions than 2HNN, it is advantageous in the sense that it gives an additional vector of all action-values with a single neural net forward pass.

Figure 6 contains the comparison of the move prediction accuracies. The 2HNN and 3HNN have almost indistinguishable learning curves, much because they are trained on the

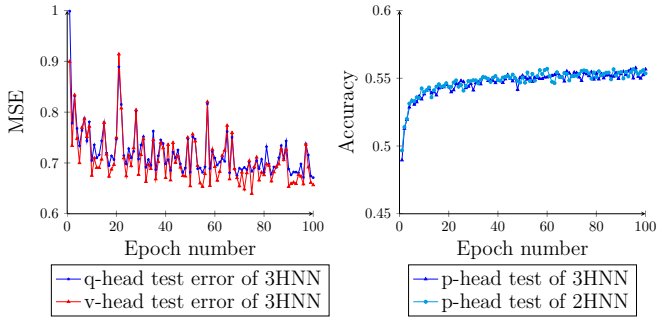

Figure 6: MSEs (left) and top one move prediction accuracies (right) of two and three heads residual nets.

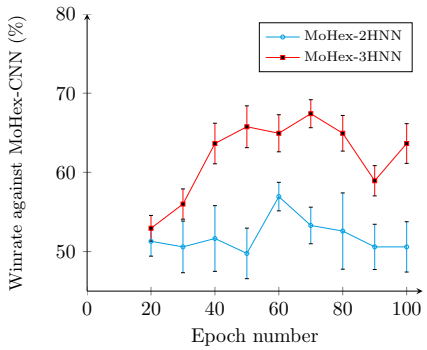

Figure 7: Winrates against MoHex-CNN of PV-MCTS-2H and PV-MCTS-3H. All programs use the same 1000 simulations per move. PV-MCTS-2H uses playout results when there is no node expansion. Notice that after epochs 70 and 60, PV-MCTS-3HNN and PV-MCTS-2HH's performance decreased, possibly due to over-fitting of the neural nets: Figures 5 and 6 show that around epoch 70, the value heads of 3HNN generally achieve smaller errors than epochs around 80 and 90.

same dataset with identical policy loss.

The training times of two and three heads neural nets are similar, all within 20 hours on the GTX1080 machine.

4.3 Evaluating the Integration in PV-MCTS

In this section, we present experimental comparisons between two and three heads nets in the same search framework. We build our programs upon code base MoHex 2.0 [Huang *et al.*, 2013; Pawlewicz *et al.*, 2015; Enzenberger *et al.*, 2010], and use MoHex-CNN [Gao *et al.*, 2017] as a benchmark to measure the relative strength of the new programs. We list the differences between the new programs, MoHex 2.0 and MoHex-CNN in below:

- We use in-tree formula,

$$score(s,a) = (1-\omega)\left(Q(s,a) + c_b\sqrt{\frac{\ln N(s)}{N(s,a)}}\right) + \omega R(s,a) + c_{pb}\frac{p(s,a)}{\sqrt{N(s,a)+1}}$$

  from MoHex 2.0, where the prior probability $p(s,a)$ in MoHex 2.0 is from pattern weights. For other programs, $p(s,a)$ is computed by different neural nets. $R(s,a)$ is the RAVE [Gelly and Silver, 2011] value. We leave parameters ω, c_b, c_{pb} unchanged. Therefore, the relative performance of those programs depends on the quality of neural nets and the efficiency in using them. In previous report [Gao *et al.*, 2017], with those default parameters, this formula achieved better results than a PUCT [Rosin, 2011] variant.

- Leaf node evaluation. Default expansion threshold in MoHex 2.0's code base is 10, hence before the leaf node's visit count reaches expansion threshold, PV-MCTS-2HNN still uses playout result while PV-MCTS-3HNN backs up previously stored action-value. MoHex-CNN always uses playout result, since it does not have a value net. To be consistent with the code base of MoHex, before backing up, neural value estimates in range $[-1,+1]$ are linearly scaled to range $[0,1]$. For all the programs, even when playout result is not back-propagated in tree, to keep consistent RAVE updates, playout was always conducted.

Since we implement PV-MCTS-2HNN and PV-MCTS-3HNN upon MoHex, we call the new programs MoHex-2HNN and MoHex-3HNN. The same as MoHex-CNN [Gao *et al.*, 2017], at each node expansion, the new programs evaluate the neural net in parallel with MoHex's children prior pruning. Because of this implementation, the computation overhead with neural nets become small. We note that on the GTX1080 machine, each forward pass costs about 0.001 seconds for both two and three heads neural nets, MoHex-CNN's neural net model is shallower and generally 0.0001 seconds faster.

We sample 10 neural net models at epochs 10,20,. . .,100. Each neural net model is then combined with MoHex's MCTS and played against MoHex-CNN with the same 1000 simulations per move. Following a practice in the literature [Gao *et al.*, 2017; Huang *et al.*, 2013], the matches are played by iterating all openings (Symmetric cells were removed). Each match between two players consists of 5 rounds, in each round each player plays an opening twice as black and white, therefore a round consists of 170 games in total. No swap rule was applied.

Figure 7 shows the results, which indicate that PV-MCTS-3H is clearly better than PV-MCTS-2H under this setting. Both PV-MCTS-3H and PV-MCTS-2H achieve winrates larger than 50% against MoHex-CNN, presumably because MoHex-CNN only uses neural net as prior knowledge during *in-tree* phase. As explained by Proposition 1, given that neural net estimates more accurate leaf value than random playout, PV-MCTS-3H is better than PV-MCTS-2H, because it always uses neural net value estimation as the critic while PV-MCTS-2H has to use playout result when the leaf is below expansion threshold.

We then compare PV-MCTS-2H to PV-MCTS-3H using expansion threshold of 0. To have a fair comparison, we give both programs the same search time $10s$ per move. The results are summarized in Table 1. As a reference, we also present result from PV-MCTS-2H with the default expansion

Player	Player as black	Player as white	Overall winrate
MoHex-3HNN	76.5%	70.6%	73.5%
MoHex-2HNN threshold 0	65.9%	57.6%	61.8%
MoHex-2HNN default threshold	69.4%	56.5%	62.9%

Table 1: Winrates against Mohex-CNN with the same time per move. For best performance, MoHex-3HNN and MoHex-2HNN respectively use the neural net models at epochs 70 and 60.

Opponent	Opponent as black	Opponent as white	Overall winrate
MoHex-3HNN	48.2%	68.2%	58.2%

Table 2: Winrates of MoHex-3HNN$^\circ$ against MoHex-3HNN with the same time $10s$ per move. Both use the neural net models at epoch 70.

threshold.

Consistent with Figure 7, results in Table 1 show that MoHex-3HNN still achieves the best performance, significantly outplaying MoHex-CNN. We found that with the default expansion threshold, on average MoHex-3HNN took 0.3 milliseconds to execute one simulation. MoHex-2HNN with expansion threshold 0 used 1.8 milliseconds per simulation, about 6 times slower, which explains MoHex-3HNN's better performance given the same time per move.

For direct comparison, under the same setting using $10s$ per move, MoHex-3HNN's winrates against MoHex-2HNN using 0 and default expand thresholds are respectively 64.1% and 70.6%; MoHex-3HNN won 82.4% games against MoHex 2.0.

4.4 Strengthen the Leaf Estimate

Different from PV-MCTS-2H, when expanding a node s, PV-MCTS-3H receives an additional vector of action-values. Rather than solely using the state-value head, a natural question is whether the leaf estimate can be improved by considering the action-values?

Our first attempt is combining the minimum action-values and state-value at s when they are both available, by the following formula:

$$v^\circ(s) = \frac{1}{2}v(s) + \frac{1}{2}(-\min_{a\in\mathcal{A}(s)} q(s,a)) \tag{6}$$

Same as the previous practice, MoHex-3HNN$^\circ$ adopts all the default parameters from MoHex 2.0, but combines the critic by (6) when expanding a node. Table 2 contains the results of MoHex-3HNN$^\circ$ against the original MoHex-3HNN, which shows MoHex-3HNN$^\circ$ slightly outplayed MoHex-3HNN; however, we found its winrate against MoHex-CNN dropped to 68.8%.

The second approach is to utilize the OR constraint, i.e., one losing child is enough to prove that the parent state s is winning. This means, if there are $k > 1$ action-values close to -1, the confidence that $v^*(s) = +1$ would be high, since this can only be denied by the fact that all k action-value predications are "wrong". More precisely, we may say an action-value prediction from $\mathbf{q}$ is correct when $q(s,a) < \delta \wedge q^*(s,a) = -1$. Suppose there are k action-values below δ, and each of them is correct with probability $\bar{p}$, then the chance that $v^*(s) = +1$ is $1-(1-\bar{p})^k$, implying a desirable property for high-confidence estimations. Setting $\delta = -0.5$, our second attempt is to let MoHex-3HNN replace $v(s)$ with $\frac{1}{2}(1+v(s))$ when at least k action-values from $\mathbf{q}$ are *correct*. We name the new program as Mohex-3HNN* and list the playing results against MoHex-3HNN in Table 3 varying $k \in \{3,4,5,6\}$.

Opponent	$k=3$	$k=4$	$k=5$	$k=6$
MoHex-3HNN	47.6%	52.4%	54.7%	54.7%

Table 3: Different winrates of MoHex-3HNN* against MoHex-3HNN by varying k. Both use the same neural net model at epoch 70 and are allocated with the same $10s$ per move.

Results in Table 3 show that this also slightly beat MoHex-3HNN when $k \geq 4$. As those modifications are based on the assumption that value predictions from neural nets are with small error with respect to optimal, the usefulness of them may increase when the training data become more accurate.

5 Conclusions and Future Work

We have described a new three-head neural net architecture for Monte Carlo Tree Search. Learning an additional q-value improves the efficiency of MCTS by making action-values available to the search. The action head can be trained without extra training data by exploiting minimax consistency between a parent and its children in AND nodes. The action-value head achieves similarly high prediction accuracy as the state-value head. We evaluated several algorithms based on this approach in the game of Hex. Our new programs are able to significantly outplay the previous state-of-art Hex player MoHex-CNN on 13×13 Hex.

However, MoHex-3HNN was trained on a fixed dataset. Previous work has shown that iteratively training the neural net on stronger games generated by improved PV-MCTS is an viable approach for continually improving the playing performance [Silver *et al.*, 2017; Anthony *et al.*, 2017]. So far, we have only investigated the advantage of a three-head neural net for one iteration of such an iterative procedure. To further improve the playing performance, a topic for future work is building a closed loop system, which continues to improve MoHex-3HNN by learning from self-play games generated from using recent neural network parameters. The improved efficiency of the search would hopefully lead to faster convergence.

References

[Anthony *et al.*, 2017] Thomas Anthony, Zheng Tian, and David Barber. Thinking fast and slow with deep learning and tree search. *arXiv preprint arXiv:1705.08439*, 2017.

[Arneson *et al.*, 2010] Broderick Arneson, Ryan B Hayward, and Philip Henderson. Monte carlo tree search in Hex. *IEEE Transactions on Computational Intelligence and AI in Games*, 2(4):251–258, 2010.

[Browne *et al.*, 2012] Cameron B Browne, Edward Powley, Daniel Whitehouse, Simon M Lucas, Peter I Cowling, Philipp Rohlfshagen, Stephen Tavener, Diego Perez, Spyridon Samothrakis, and Simon Colton. A survey of monte carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in games*, 4(1):1–43, 2012.

[Clark and Storkey, 2015] Christopher Clark and Amos Storkey. Training deep convolutional neural networks to play Go. In *International Conference on Machine Learning*, pages 1766–1774, 2015.

[Coulom, 2006] Rémi Coulom. Efficient selectivity and backup operators in Monte-Carlo tree search. In *International conference on computers and games*, pages 72–83. Springer, 2006.

[Enzenberger *et al.*, 2010] Markus Enzenberger, Martin Müller, Broderick Arneson, and Richard 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*, 2(4):259–270, 2010.

[Gao *et al.*, 2017] Chao Gao, Ryan B Hayward, and Martin Müller. Move prediction using deep convolutional neural networks in Hex. *IEEE Transactions on Games*, 2017.

[Gelly and Silver, 2007] Sylvain Gelly and David Silver. Combining online and offline knowledge in UCT. In *Proceedings of the 24th international conference on Machine learning*, pages 273–280. ACM, 2007.

[Gelly and Silver, 2011] Sylvain Gelly and David Silver. Monte-Carlo tree search and rapid action value estimation in computer Go. *Artificial Intelligence*, 175(11):1856–1875, 2011.

[He *et al.*, 2016a] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[He *et al.*, 2016b] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Identity mappings in deep residual networks. In *European Conference on Computer Vision*, pages 630–645. Springer, 2016.

[Huang *et al.*, 2013] Shih-Chieh Huang, Broderick Arneson, Ryan B Hayward, Martin Müller, and Jakub Pawlewicz. Mohex 2.0: a pattern-based MCTS Hex player. In *International Conference on Computers and Games*, pages 60–71. Springer, 2013.

[Ioffe and Szegedy, 2015] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International Conference on Machine Learning*, pages 448–456, 2015.

[Jouppi and al, 2017] Norman P. Jouppi and et. al. In-datacenter performance analysis of a tensor processing unit. In *Proceedings of the 44th Annual International Symposium on Computer Architecture*, ISCA '17, pages 1–12. ACM, 2017.

[Kingma and Ba, 2014] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations*, 2014.

[Kocsis and Szepesvári, 2006] Levente Kocsis and Csaba Szepesvári. Bandit based Monte-Carlo planning. In *European conference on machine learning*, pages 282–293. Springer, 2006.

[LeCun *et al.*, 2015] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015.

[Littman, 1996] Michael Lederman Littman. *Algorithms for sequential decision making*. PhD thesis, Brown University Providence, RI, 1996.

[Maddison *et al.*, 2015] Chris J Maddison, Aja Huang, Ilya Sutskever, and David Silver. Move evaluation in Go using deep convolutional neural networks. In *International Conference on Learning Representations*, 2015.

[Mnih *et al.*, 2015] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

[Nash, 1952] John Nash. Some games and machines for playing them. Technical Report D-1164, RAND Corporation, Feb 1952.

[Pawlewicz *et al.*, 2015] Jakub Pawlewicz, Ryan Hayward, Philip Henderson, and Broderick Arneson. Stronger virtual connections in Hex. *IEEE Transactions on Computational Intelligence and AI in Games*, 7(2):156–166, 2015.

[Rosin, 2011] Christopher D Rosin. Multi-armed bandits with episode context. *Annals of Mathematics and Artificial Intelligence*, 61(3):203–230, 2011.

[Shannon, 1953] Claude E Shannon. Computers and automata. *Proceedings of the IRE*, 41(10):1234–1241, 1953.

[Silver *et al.*, 2016] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.

[Silver *et al.*, 2017] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, Yutian Chen, Timothy Lillicrap, Fan Hui, Laurent Sifre, George van den Driessche, Thore Graepel, and Demis Hassabis. Mastering the game of Go without human knowledge. *Nature*, 550(7676):354–359, 2017.

[Tian and Zhu, 2015] Yuandong Tian and Yan Zhu. Better computer Go player with neural network and long-term prediction. In *International Conference on Learning Representations*, 2015.