Using Deep Convolutional Neural Networks in Monte Carlo Tree Search

Tobias Graf(✉) and Marco Platzner

University of Paderborn, Paderborn, Germany
tobiasg@mail.upb.de, platzner@upb.de

Abstract. Deep Convolutional Neural Networks have revolutionized Computer Go. Large networks have emerged as state-of-the-art models for move prediction and are used not only as stand-alone players but also inside Monte Carlo Tree Search to select and bias moves. Using neural networks inside the tree search is a challenge due to their slow execution time even if accelerated on a GPU. In this paper we evaluate several strategies to limit the number of nodes in the search tree in which neural networks are used. All strategies are assessed using the freely available cuDNN library. We compare our strategies against an optimal upper bound which can be estimated by removing timing constraints. We show that the best strategies are only 50 ELO points worse than this upper bound.

1 Introduction

Deep Convolutional Neural Networks (DCNNs) have changed Computer Go substantially [5,11,12,14]. They can predict expert moves at such a high quality that they even can play Go themselves at a reasonable level [14]. Used in Monte Carlo Tree Search (MCTS) [2] to select and bias moves they can increase playing strength by hundreds of ELOs. During the writing of this paper Google DeepMind has released their program AlphaGo [12] which uses neural networks not only for move prediction but also for positional evaluation. For the first time in Computer Go their program has beaten a professional player and is going to challenge one of the best players in the world.

DCNNs achieved remarkable improvements but they pose a challenge for MCTS as their execution time is too slow to be used in the whole search tree. While a remedy is to use several GPUs [12] this paper focuses on single GPU scenarios where not all nodes in the search tree can use the DCNN as a move predictor. To decide which nodes profit the most from DCNN knowledge several strategies are possible. This paper evaluates four typical strategies to replace knowledge from fast classifiers with DCNN predictions. All strategies are assessed within the same Go program to decide which is best. Moreover, we construct an upper bound on playing strength by using an equal test environment but removing timing constraints. We then compare the strategies with this upper bound to show the loss in playing strength resulting from the use of replacement strategies.

A. Plaat et al. (Eds.): CG 2016, LNCS 10068, pp. 11–21, 2016.
DOI: 10.1007/978-3-319-50935-8_2

The contributions of our paper are as follows.

- We demonstrate that replacing traditional move prediction knowledge in Computer Go programs can yield remarkable improvements in playing strength.
- We investigate the scalability of knowledge in MCTS, i.e., in how far do better neural networks lead to stronger MCTS-players.
- As DCNNs are too slow to be used in the complete search tree we explore several strategies to decide which nodes profit the most from DCNNs.
- We look into technical aspects of using GPUs inside MCTS.

The remainder of this paper is structured as follows: In Sect. 2 we describe the settings and architectures of the deep convolutional neural networks we use in the paper. In Sect. 3 we outline several replacement strategies for an efficient application of slow knowledge in MCTS. In Sect. 4 we show the results of several experiments regarding the quality of DCNNs and replacement strategies. In Sect. 5 we present related work. Finally, in Sect. 6 we draw our conclusion and point to future directions.

2 Deep Convolutional Neural Networks

In this section we outline the Deep Convolutional Neural Networks which are used in this paper. The architecture of our DCNNs is similar to [11]. We use several convolutional layers (3, 6 or 12) with 5×5 filter in the first one and 3×3 in the others. The width of each layer is 128, 192 or 256. After all convolutional layers we add an extra 3×3 convolutional layer with one output feature followed by a softmax layer. The position is encoded with black to move (if white moves the colors of the stones are reversed). The 20 input features of the neural network are:

- Black, White, Empty, Border
- Last 5 moves
- Legality
- Liberties (1, 2, 3, $\geq$4)
- Liberties after move (1, 2, 3, 4, 5, $\geq$6)

We used the Caffe framework [10] to train all DCNNs. We trained the networks with plain SGD with mini-batch size 128 for 3 million iterations (one iteration is one mini-batch). The learning rate is 0.05 for the first 1.5 million iterations and then halved for the rest of the training every 500,000 iterations. We used a weight decay of 1e-6 and initialized all weights with the msra-filler [8]. As dataset of Go games we used KGS games[1] with players having at least 4 dan strength using only no-handicap games which have at least 150 moves. The positions are split into a validation set with 1,280,000 positions and a training set with 60,026,402 positions. Positions of both sets are from distinct games. The positions in the training set are randomly rotated and mirrored to one of 8 possible orientations.

[1] http://u-go.net/gamerecords-4d/.

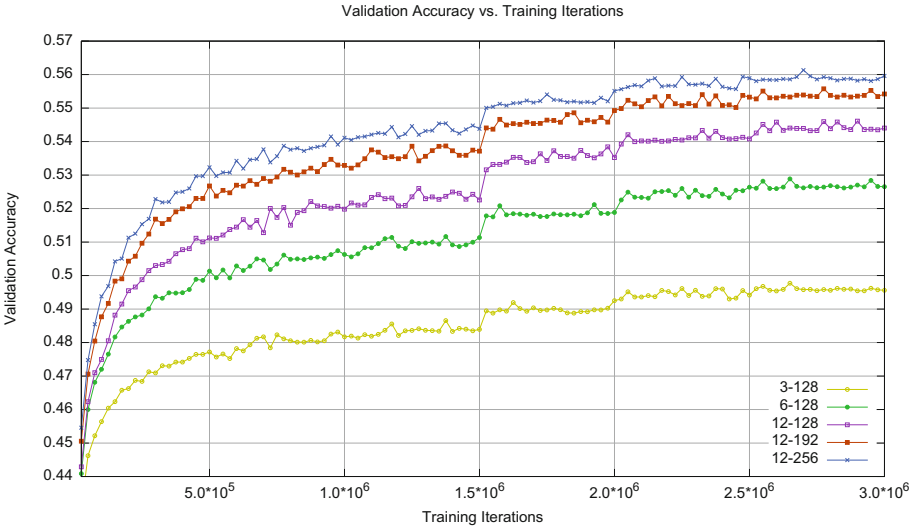

Fig. 1. Accuracy on validation set during training

Figure 1 shows the accuracy on the validation set during training. Accuracy is the percentage of positions where the top model prediction equals the move of the expert. After 1.5, 2.0 and 2.5 million iterations sharp increases in accuracy due to the learning-rate schedule can be observed. The achieved validation accuracy after training is comparable to those reached in [11].

3 Integration of DCNNs into MCTS

Deep Convolutional Neural Networks need considerably more computing time than conventional models used in MCTS. This section surveys several techniques to include DCNNs into MCTS without hampering the speed of the search.

3.1 Selection Formula

To include knowledge into MCTS we use the following formula which includes RAVE [6] and progressive bias [3].

$$(1-\beta)\cdot Q_{\mathrm{Uct}}(s,a)+\beta\cdot Q_{\mathrm{Rave}}(s,a)+K\frac{\pi(s,a)}{\sqrt{\mathrm{visits}(s,a)}}$$

where $\pi(s,a)\in[0,1]$ is the output of the move prediction model.

3.2 Using GPUs Inside MCTS

To include deep convolutional neural networks into MCTS we make use of the cuDNN library version 3.0[2] of Nvidia [4]. The GPU-accelerated library contains primitives for deep neural networks which are highly tuned. It supports multi-threading and allows using separate streams. While the library is much more low level than the Caffe framework it provides the necessary functionality for an efficient use inside MCTS.

We use a batch-size of one for each DCNN execution on the GPU. To increase the utilization of the GPU each thread of the MCTS gets a dedicated CUDA stream. In this way memory transfers and kernels from different threads can be executed concurrently. Moreover, in case of asynchronous replacement strategies we use CUDA events. This allows to efficiently continue the work on the CPU while the GPU evaluates the DCNN.

Table 1 shows the execution times of all DCNNs from the previous section on a system with two Intel Xeon E5-2670 (16 cores, 2.6 GHz) and a Tesla K20 GPU. In contrast to the baseline which only uses shape and common fate graph patterns [7] larger DCNNs are more than 10 times slower in execution time and achieve less than half the playout-rate.

Table 1. Execution time, playout-rate in MCTS and accuracy

	Execution time	Playout-rate MCTS	Accuracy validation-set
Baseline	0.38 ms	11552 p/s	42.1%
DCNN-3-128	0.94 ms	10734 p/s	49.6%
DCNN-6-128	1.70 ms	8939 p/s	52.7%
DCNN-12-128	3.23 ms	5458 p/s	54.4%
DCNN-12-192	7.52 ms	3111 p/s	55.4%
DCNN-12-256	10.07 ms	2338 p/s	55.9%

3.3 Replacement Strategies

In this paper we explore four replacement strategies for knowledge inside MCTS. We assume that a fast move predictor (e.g., [7,13]) is available in addition to the slower DCNN. This allows to specify different strategies to decide which knowledge can be used. All replacement strategies try to predict which nodes inside the search tree are important. In these nodes they apply the slow knowledge as soon as possible. All strategies can be formulated in a synchronous and an asynchronous version. On the one hand, the advantage of the synchronous version is that MCTS does not waste iterations with low quality knowledge. On the other hand, asynchronous versions can continue with the search. They will use more low quality knowledge in the beginning but in return can search faster and build a deeper search tree.

[2] We also tested the release candidate of version 4. We observed faster single execution times but a small slowdown when used in parallel MCTS.

Replace by Depth. This strategy decides which node gets DCNN knowledge by the depth of each node in the search tree. We specify a parameter D and every node with depth $\leq D$ gets DCNN knowledge while all others nodes only use the fast classifier. At an extreme with $D = 0$ only the root node receives DCNN knowledge. The reasoning behind this strategy is that decisions near the root are the most important and should use the best knowledge available. Disadvantages are that the parameter D is highly dependent on the overall time spent for the search and thus has to be changed for different thinking times. Moreover, MCTS builds up a very irregular search tree where some promising branches are searched very deeply while others are not. On the one hand, specifying an overall depth threshold cannot capture this important aspect of MCTS. On the other hand, this strategy does its decision at node initialization so that knowledge can be fully utilized.

The strategy can be turned into an asynchronous version by initializing each node with fast knowledge and for nodes with depth $\leq D$ immediately a request is sent to the GPU. Once the DCNN execution has been finished it replaces the fast knowledge of the node.

Replace in Principal-Variation. Beginning from the root node we can follow in each node of the search tree the move which has been investigated most. The sequence of moves resulting from this is called the principal variation and represents best play from both sides. The following strategy tries to identify the principal variation of the search and initializes all nodes of this variation with slow DCNN knowledge. All other nodes are interpreted as less important and are using fast knowledge. In MCTS the principal variation changes during the search quite often so we also want to include variations which are close. This leads to the following strategy with the parameter $\epsilon \in [0, 1]$: When starting MCTS at the root we set a flag $PV \leftarrow \mathit{true}$. If the move a is selected and the count of the move n_a is smaller than $\epsilon \cdot max_a n_a$ then $PV \leftarrow \mathit{false}$ else it is unchanged. When a new node is expanded we initialize the node with DCNN knowledge if PV is true. Otherwise, the node is initialized with the fast classifier. Moreover, if we encounter nodes during tree traversal which do not have DCNN knowledge we replace it with DCNN knowledge if PV is true. In the synchronous version we wait until the knowledge is available. In the asynchronous version we continue the work.

The advantage of the strategy is that DCNN knowledge can be utilized early in the search as important nodes are identified before expansion. In contrast to the depth-replacement strategy it is also independent of the overall search time and adapts to the irregular shape of the search tree. The disadvantage is that if the principal variation is not followed early on in the search, abrupt changes can occur. Then all nodes in the new principal variation do not have the DCNN knowledge and are thus promoted only now which can be very late in the search.

Replace by Threshold. This strategy initializes the knowledge in each node with the fast classifier. If a node is searched more than T times the fast knowledge

is replaced by the DCNN. In the synchronous version a node is locked for other threads and the current thread waits for the GPU to finish the DCNN execution. In the asynchronous version a node is not locked for other threads and the current thread just sends a request to the GPU and continues the MCTS. Once the GPU has finished work the DCNN knowledge is used in the node.

The advantage of this strategy is that the threshold is mostly independent of the overall search time and can thus be easily tuned. Moreover, the more a node is searched by MCTS the more important it is. So this strategy identifies all significant nodes. The disadvantage is that this only happens quite late so that DCNN knowledge cannot be fully utilized in early stages.

Increase Expansion Threshold. MCTS expands nodes after a certain amount of simulations have passed through the node. The default value of Abakus is 8, i.e., if a move has more than 8 simulations a new node is expanded. While the value of 8 is optimized for a fast classifier we can increase the value to fit the slow DCNN. The synchronous version of this strategy initializes each node by DCNN knowledge and controls the rate at which nodes are expanded with a threshold E. The asynchronous version initializes each node with the fast classifier and immediately sends a request to the GPU and replaces the knowledge once the DCNN data is available.

The disadvantage of this strategy is that smaller trees are searched when the expansion threshold E is set too high. However, the DCNN knowledge can be exploited in each node from the beginning.

4 Experiments

In this section we show the results of our experiments. We run several tournaments of our program Abakus against the open source program Pachi [1]. Abakus makes use of RAVE [6], progressive widening [9], progressive bias [3] and a large amount of knowledge (shape and common fate graph patterns [7]) in the tree search part. With the addition of DCNNs it holds a 5-Dan rank on the internet server KGS[3].

As Pachi is weaker than Abakus we used handicap games to level the chances. One challenge for the experiments was the great range of strength which results from using DCNNs. Therefore, we used a handicap of 7 stones and komi of 0.5 in all the experiments.

In our first experiments we wanted to illustrate the raw strength improvement one can get by using DCNNs. The DCNN knowledge is used whenever a new node in the search tree is expanded. In this way the shallow knowledge is never used. To achieve a direct comparison we performed games with a fixed amount of playouts. This can also be seen as the maximum strength improvement possible by using the specific DCNN. In practice, these gains cannot be achieved as application of DCNNs; they need considerably more time than the shallow knowledge.

[3] www.gokgs.com.

Table 2. Playing strength of ABAKUS (white) against PACHI (black, 7 handicap stones), 512 games played for each entry, 95% confidence intervals, ABAKUS 11,000 playouts/move, PACHI 27,500 playouts/move

	Winrate vs. PACHI	ELO vs. PACHI	ELO vs. baseline	Average speed
Baseline	9.8% ±2.6	−386 [−444, −341]	0	12,092 Playouts/s
DCNN-3-128	49.3% ± 4.3	−5 [−35, 25]	381	11,349 Playouts/s
DCNN-6-128	67.4% ± 4.1	126 [95, 159]	512	9,277 Playouts/s
DCNN-12-128	78.9% ± 3.5	229 [194, 269]	615	5,661 Playouts/s
DCNN-12-192	81.9% ± 3.3	263 [226, 305]	649	3,258 Playouts/s
DCNN-12-256	85.6% ± 3.0	310 [271, 358]	696	2,456 Playouts/s

The number of playouts per move was chosen as 11,000 for ABAKUS and 27,500 for PACHI. This is approximately the same amount of playouts which each program can achieve in 1 s on an empty board. In this way the experiments are comparable to later experiments which use 1 s thinking time.

The results are shown in Table 2. The better the DCNN is the stronger the program plays against PACHI. But we can also see that the strength improvement declines for the last DCNNs. Moreover, the average speed reduces quickly as more powerful networks are used (which here is not taken into account as the number of playouts is fixed per move).

The next experiments evaluate the four replacement schemes by using a fixed amount of time. We used 1 s per move so that the above results give an approximate upper bound on the playing strength. As the gain by large networks diminishes we used the DCNN-12-128 for the following experiments as it gives a good trade-off between quality and execution time.

In Table 3 we see the results for the replacement scheme depth. The column "DCNN Apply/Coun" shows the average number of simulations of a node when the DCNN knowledge is applied and how often this is done during a search. The depth replacement strategy applies knowledge once a node is expanded but as the search-tree is reused on the next move several old nodes are upgraded with the knowledge. This explains the quite high number of $D = 0$ for apply, whereas the application only uses knowledge in the root.

In Table 4 we see the results for the strategy to increase the expansion threshold to lower the rate of new nodes in the search tree. As long as E is not set too high this strategy achieves as good results as the threshold strategy. It's advantage is that knowledge is applied very early (at about 8 simulations on average) but the search tree is not as big as usual.

In Table 5 we see the results for the principal variation replacement scheme. While the scheme tries to use the DCNN as soon as possible knowledge is often applied quite late (e.g., in the synchronous case for $\epsilon = 0.5$ if the DCNN is used

Table 3. Replace by depth: evaluation with DCNN-12-128 and various parameters D, playing strength of Abakus against Pachi, 512 games played for each entry, 95% confidence intervals, 1 s/move

	D	Winrate	ELO	ELO vs UB	DCNN apply/count	Playouts/s
Upper bound		78.9% ± 3.5	229 [194,269]	0		
Synchronous	0	42.6% ± 4.3	−52 [−83, −22]	−281	2169.1/1.2	12077 p/s
	4	59.6% ± 4.3	67 [37, 99]	−162	28.8/201	10892 p/s
	8	63.4% ± 4.2	95 [65, 127]	−134	3.0/539	7694 p/s
	12	60.4% ± 4.2	74 [43, 105]	−155	0.6/629	6125 p/s
Asynchronous	0	41.2% ± 4.3	−62 [−93, −31]	−291	2233.3/1.2	12109 p/s
	4	64.7% ± 4.1	106 [75, 138]	−123	35.9/228	11804 p/s
	8	68.2% ± 4.0	132 [101, 166]	−97	10.7/637	9180 p/s
	12	62.5% ± 4.2	89 [58, 121]	−140	8.1/704	7252 p/s

in a node on average 46 simulations have already passed through it) which shows that the principal variation often changes during a search.

In Table 6 we see the results for the replacement scheme threshold. As soon as the threshold is sufficiently high to not disturb the search the winrate stays quite high. Only for large thresholds the winrate starts to drop as knowledge is applied too late in the nodes.

In conclusion, the strategies to replace knowledge by a simulation threshold or to increase the expansion threshold of MCTS achieve the best results. The depth replacement scheme cannot adapt to the search tree which results in worse playing strength. Using knowledge exclusively in the principal variation accomplished better results but it seems difficult to identify the final principal variation in a search. All strategies performed better when executed asynchronously.

Table 4. Increase expansion-threshold: evaluation with DCNN-12-128 and various parameters E, playing strength of Abakus against Pachi, 512 games played for each entry, 95% confidence intervals, 1 s/move

	E	Winrate	ELO	ELO vs UB	DCNN apply/count	Playouts/s
Upper bound		78.9% ± 3.5	229 [194, 269]			
Synchronous	8	58.1% ± 4.3	57 [27, 88]	−172	0.0/634	5513 p/s
	16	64.8% ± 4.1	106 [75, 139]	−123	0.0/449	9551 p/s
	24	69.1% ± 4.0	140 [109, 174]	−89	0.0/313	10743 p/s
	32	72.2% ± 3.9	166 [133, 201]	−63	0.0/236	11167 p/s
Asynchronous	8	67.1% ± 4.1	124 [93,157]	−105	7.5/710	6669 p/s
	16	72.7% ± 3.9	170 [137, 205]	−59	7.4/531	12002 p/s
	24	70.4% ± 4.0	151 [119, 185]	−78	8.1/344	12329 p/s
	32	65.8% ± 4.2	114 [82, 148]	−115	9.1/252	12443 p/s

Table 5. Replace in principal-variation: evaluation with DCNN-12-128 and various parameters ϵ, playing strength of ABAKUS against PACHI, 512 games played for each entry, 95% confidence intervals, 1 s/move

	ϵ	Winrate	ELO	ELO vs UB	DCNN apply/count	Playouts/s
Upper bound		$78.9\% \pm 3.5$	229 [194,269]	0		
Synchronous	0.1	$63.1\% \pm 4.2$	93 [62, 125]	-136	3.1/590	6827 p/s
	0.2	$68.1\% \pm 4.0$	131 [100, 165]	−98	8.3/473	8514 p/s
	0.3	$63.6\% \pm 4.2$	97 [66, 129]	−132	15.8/338	9838 p/s
	0.4	$67.3\% \pm 4.1$	125 [94, 159]	−104	27.3/228	10639 p/s
	0.5	$68.2\% \pm 4.0$	132 [101, 166]	−97	46.0/148	11180 p/s
Asynchronous	0.1	$61.7\% \pm 4.2$	83 [52,115]	−146	11.6/691	7947 p/s
	0.2	$65.6\% \pm 4.1$	112 [81, 145]	−117	17.8/592	9728 p/s
	0.3	$71.3\% \pm 3.9$	158 [126, 193]	−71	26.0/447	10810 p/s
	0.4	66.6 ± 4.1	120 [89, 153]	−109	36.7/317	11375 p/s
	0.5	$64.3\% \pm 4.2$	102 [71, 134]	−127	51.2/219	11707 p/s

Table 6. Replace by threshold: evaluation with DCNN-12-128 and various parameters T, playing strength of ABAKUS against PACHI, 512 games played for each entry, 95% confidence intervals, 1 s/move

	T	Winrate	ELO	ELO vs UB	DCNN apply/count	Playouts/s
Upper bound		$78.9\% \pm 3.5$	229 [194,269]			
Synchronous	0	$58.1\% \pm 4.3$	57 [27, 88]	−172	0.0/634	5513 p/s
	8	$67.0\% \pm 4.1$	123 [92, 156]	−106	8.0/493	8026 p/s
	16	$70.3\% \pm 4.0$	150 [118, 184]	−79	16.0/382	9174 p/s
	32	$67.6\% \pm 4.1$	128 [96, 161]	−101	32.0/256	10052 p/s
	64	$66.5\% \pm 4.1$	119 [88, 152]	−110	64.0/153	10678 p/s
	128	$68.8\% \pm 4.0$	137 [105, 171]	−92	128.0/85	11195 p/s
Asynchronous	0	$67.1\% \pm 4.1$	124 [93, 157]	−105	7.5/710	6669 p/s
	8	$73.2\% \pm 3.8$	175 [142, 211]	−54	16.1/653	10886 p/s
	16	$69.0\% \pm 4.0$	139 [108, 173]	−90	23.5/484	11711 p/s
	32	$71.5\% \pm 3.9$	160 [127, 195]	−69	40.6/307	11981 p/s
	64	$70.8\% \pm 3.9$	154 [122, 189]	−75	74.8/175	12066 p/s
	128	$66.9\% \pm 4.1$	122 [91, 155]	−107	141.7/94	12118 p/s

5 Related Work

Deep Convolutional Neural Networks have been first used as stand-alone players [5] without using MCTS. Later DCNNs were used inside MCTS [11] with the help of asynchronous node evaluation. A large mini-batch size of 128 taking 150ms for evaluation is used and every node in the search tree is added to the mini-batch

in FIFO order. Once the mini-batch is complete it is submitted to the GPU. The disadvantage of the method is a large lag due to using a big mini-batch. According to the authors the main reason for using such a large mini-batch size was that reducing the size was not beneficial in their implementation. As shown in this paper using the freely available cuDNN library of Nvidia allows to reduce the mini-batch size to one which substantially reduces the lag.

The DARKFOREST [14] program uses a synchronized expansion. Whenever a node is added the GPU evaluates the DCNN while the MCTS waits for the result and only then expands the search tree (Synchronous Replace by Threshold with $T = 0$).

ALPHAGO [12] uses the strategy which we call in our paper Increase-Expansion-Threshold. Knowledge inside the MCTS is initialized with a fast classifier and asynchronously updated once the GPU has evaluated the DCNN. They use a threshold of 40 which in relation to our experiments is quite large but they use DCNNs for move prediction and positional evaluation which results in twice as many neural networks to evaluate.

6 Conclusions and Future Work

In this paper we demonstrated that using Deep Convolutional Neural Networks in Monte Carlo Tree Search yields large improvements in playing strength. We showed that in contrast to the baseline program which already uses a great deal of knowledge DCNNs can boost the playing strength by several hundreds of ELO. Ignoring execution time better move predictors led to better playing strength with improvements close to 700 ELO.

Because DCNNs have slow execution times we suggested to use the cuDNN library of Nvidia to accelerate them on the GPU. Using different CUDA streams for each MCTS search thread fully utilizes the GPU. CUDA events allowed to asynchronously execute the DCNN on the GPU while continuing with the tree search on the CPU.

To decide which nodes in the search tree profit most from DCNN knowledge we investigated several replacement strategies. The results show that the best strategy is to initialize the knowledge used inside MCTS with a fast classifier and when sufficient simulations have passed through a node in the search tree replace it with the DCNN knowledge. A second possibility is to increase the expansion threshold inside MCTS. As long as the threshold is not large the results were close to the best strategy. In the experiments in all replacement schemes asynchronous execution on the GPU yielded better results than synchronous execution. This shows that it is important to not disturb the speed of search even if DCNN knowledge is of much higher quality than the initial knowledge.

All replacement strategies in this paper focus on using neural networks for move predictions inside MCTS. Future work includes extending these schemes for positional evaluation as well. As the amount of work for the GPU doubles strategies for the efficient use of DCNNs get even more important.

References

1. Baudiš, P., Gailly, J.: PACHI: state of the art open source go program. In: Herik, H.J., Plaat, A. (eds.) ACG 2011. LNCS, vol. 7168, pp. 24–38. Springer, Heidelberg (2012). doi:10.1007/978-3-642-31866-5_3
2. Browne, C., Powley, E., Whitehouse, D., Lucas, S., Cowling, P., Rohlfshagen, P., Tavener, S., Perez, D., Samothrakis, S., Colton, S.: A survey of Monte Carlo tree search methods. IEEE Trans. Comput. Intell. AI Games **4**(1), 1–43 (2012)
3. Chaslot, G., Winands, M., Uiterwijk, J., van den Herik, H., Bouzy, B.: Progressive strategies for Monte-Carlo tree search. New Math. Nat. Comput. **4**(3), 343–357 (2008)
4. Chetlur, S., Woolley, C., Vandermersch, P., Cohen, J., Tran, J., Catanzaro, B., Shelhamer, E.: cuDNN: efficient primitives for deep learning (2014). http://arxiv.org/abs/1410.0759
5. Clark, C., Storkey, A.: Training deep convolutional neural networks to play go. In: Proceedings of The 32nd International Conference on Machine Learning, pp. 1766–1774 (2015)
6. Gelly, S., Silver, D.: Combining online and offline knowledge in UCT. In: Proceedings of the 24th International Conference on Machine Learning (ICML 2007), New York, NY, USA, pp. 273–280 (2007)
7. Graf, T., Platzner, M.: Common fate graph patterns in monte carlo tree search for computer go. In: 2014 IEEE Conference on Computational Intelligence and Games (CIG), pp. 1–8, August 2014
8. He, K., Zhang, X., Ren, S., Sun, J.: Delving deep into rectifiers: surpassing human-level performance on imagenet classification. In: IEEE International Conference on Computer Vision (2015)
9. Ikeda, K., Viennot, S.: Efficiency of static knowledge bias in Monte-Carlo tree search. In: Herik, H.J., Iida, H., Plaat, A. (eds.) CG 2013. LNCS, vol. 8427, pp. 26–38. Springer, Heidelberg (2014). doi:10.1007/978-3-319-09165-5_3
10. Jia, Y., Shelhamer, E., Donahue, J., Karayev, S., Long, J., Girshick, R., Guadarrama, S., Darrell, T.: Caffe: convolutional architecture for fast feature embedding. arXiv preprint arXiv:1408.5093 (2014)
11. Maddison, C., Huang, A., Sutskever, I., Silver, D.: Move evaluation in go using deep convolutional neural networks. In: International Conference on Learning Representations (2015)
12. Silver, D., Huang, A., Maddison, C.J., Guez, A., Sifre, L., van den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., Dieleman, S., Grewe, D., Nham, J., Kalchbrenner, N., Sutskever, I., Lillicrap, T., Leach, M., Kavukcuoglu, K., Graepel, T., Hassabis, D.: Mastering the game of Go with deep neural networks and tree search. Nature **529**(7587), 484–489 (2016)
13. Stern, D., Herbrich, R., Graepel, T.: Bayesian pattern ranking for move prediction in the game of go. In: Proceedings of the 23rd International Conference on Machine Learning, pp. 873–880 (2006). http://dx.doi.org/10.1038/nature16961
14. Tian, Y., Zhu, Y.: Better computer go player with neural network and long-term prediction. In: International Conference on Learning Representations (2016)

http://www.springer.com/978-3-319-50934-1

Computers and Games
9th International Conference, CG 2016, Leiden, The Netherlands, June 29 – July 1, 2016, Revised Selected Papers
Plaat, A.; Kosters, W.; van den Herik, J. (Eds.)
2016, XX, 225 p. 68 illus., Softcover
ISBN: 978-3-319-50934-1