ChessBreaker: a smaller AlphaZero for Endgames

Marcello Ceresini
Department of Computer Science and Engineering
Alma Mater Studiorum Università di Bologna
mceresini.97@gmail.com

Abstract

Chess is one of the oldest games known to man, with its roots dating back to more than a thousand years ago. Despite its longevity, the enormously vast search space makes it an insurmountable task both for a human to play perfectly, but also for a computer to evaluate exactly each position, due to a game-tree complexity of roughly 10^{123}. In this paper, I try to follow in Deepmind's footsteps and tackle chess, keeping in mind the obvious limitations in personnel, knowledge and hardware. For this reason ChessBreaker, the MonteCarlo Tree Search Self-Play algorithm that i will display, is limited to only playing endgames of maximum 7 pieces.

1 Introduction

1.1 Disclaimer

Since this is the reproduction of a state of the art paper, most of the intuitions and technical measures are directly taken from David Silver & Al. and his work with Deepmind in various previous papers [1] [2] [3] [4].

1.2 Chess

Chess is a game of perfect information, so an optimal value function that determines the outcome of a game from every board position given perfect play from both players must exist. The search, however, would require the full expansion of the game tree, that has an upper bound on the number of nodes given by b^d, with b being the branching factor and d the depth of the tree. In chess, $b \sim 35$ and $d \sim 80$ [5], bringing the game tree complexity to $\sim 10^{123}$. The search, however, can be reduced by artificially reducing both b and d. In this paper, through a deep network we truncate the descent at a certain depth and replace the subtree with an approximate value, while also sampling actions from a policy to reduce the breadth of the search.

1.3 MonteCarlo Tree Search

When confronted with the [1] & the [2] MCTS algorithms, we chose the second option, both because it has shown better performance and also for ease of implementation. MCTS algorithms exploit rollouts to refresh the action value of each node, improving it with each rollout. In the meanwhile, through children selection, the policy used to select actions is improved.

In fact, the algorithm used to choose a single move can so be described: The search starts from the root node, the position from which we need to decide the next move. Each node in the tree stores a prior probability $P(s, a)$, a visit count $N(s)$ and an action value $Q(s, a)$. The following points will be repeated for *n* rollouts, with increasing *n* improving the quality of the policy:

1. At every step, the algorithm descends the tree by choosing the best child, the one that maximizes an upper confidence bound $UCB = Q + c * P/(1 + N)$ (where c is an exploration parameter) until a leaf is encountered,.
2. The leaf is expanded and each child is given a P value through the prediction of a neural network.
3. If the search reaches the end of a match, the outcome of the match becomes the Q of that node. Otherwise, a maximum depth is reached, and in this case the evaluation of the neural network is used as Q value instead.
4. In both cases, the Q value is back-propagated through the visited nodes. At each iteration, every node's Q value is the average of all the Q values of the leaf nodes reached by passing through it.

At the end *n* repetitions, the next move is chosen through the policy $\pi_a \propto N(s,a)^{1/\tau}$, where N is the visit count of the direct children of the root node, and τ is a temperature parameter influencing exploration. In this way, MCTS can be seen as a complex operator that given the weights θ of the neural network and the initial state s, returns a policy that improves the original policy of the network.

1.4 Neural Network

Following the findings in [2], the network is implemented as a ResNet: an initial convolution that passes the input to the spine composed by ResNet blocks with constant spatial dimensions and channels. From the spine two heads emerge: a policy head that outputs logits for all legal (and illegal) moves, and a value head that outputs a single value, the state value of the current position.

1.5 Self-play

The learning process interleaves two steps: a playing phase and a training phase. During the playing phase, many games are played, and each move is decided by the MCTS augmented policy. At the end of the game, each tuple of position, chosen move and final outcome (common to every position of the same game) is saved in an experience buffer. To speed things up, games are played in parallel, since the same fixed model is used for playing. During the training phase, the same network used during the MCTS is trained to guess both the chosen move and the final outcome, given the current position. The data is collected by randomly sampling batches of data from the experience buffer.

As a loss for the network we fix, with the same weight, a cross-entropy for the policy head (move choice) and MSE for the value head, plus an L2 regularization parameter.

At the end of the training step, the new improved weights are then used in the successive playing step, in turn improving the following MCTS policy.

1.6 Evaluation

For the evaluation we used the standard Elo rating system, in particular we faced off the same model at different steps in training and used Bayesian logistic regression to estimate the relative strengths of the models. We used the BayesElo program [6] [7] that derives Elo ratings through the regression of this formula $P(a \text{ defeats } b) = \frac{1}{1+exp(c_{elo}(e(b)-e(a)))}$ where $e(a)$ and $e(b)$ are the Elos and c_{elo} is a standard constant $= 1/400$.

2 Deviation from original papers

2.1 Problem selection

Since the hardware and the abilities at our disposal were orders of magnitude less than the ones used by Deepmind, we decided to reduce considerably the search space of the problem. In fact, in the beginning we tried approaching the full problem: since we knew that even small results would have required too much time, we tried exploiting supervised learning to jump-start the network policy through a publicly available Master's Game dataset [8] with $\sim$ 3 million games with their outcome and $\sim$ 300 million recorded moves. This task, however, already required way too much time for the

available hardware to see any improvement in results. Since the supervised learning part was the fastest part of the process, we decided to focus our attention on a smaller and more treatable problem: endgames. An endgame is usually a position with fewer pieces left on the board. We decided to allow up to seven pieces (five random pieces, plus the two kings) maximum. This reduces notably both the mean branching ratio (legal moves) and depth (average number of moves) to end a match.

This small but important change brought cascade consequences all throughout the algorithm.

2.2 MCTS

During a match, each MCTS was set with a n number of repetitions $= 100$, compared to the 800 used in [3]. Moreover, the maximum depth that could be reached by the tree search was set to 4.

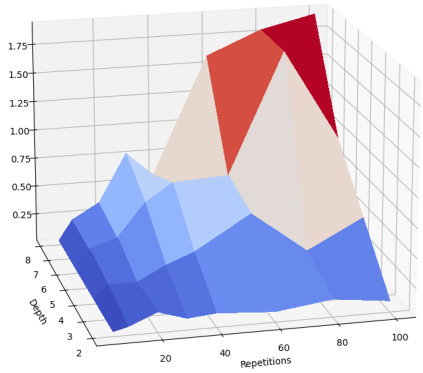


Figure 1: Evaluation time vs. n repetitions and maximum depth

As can be seen in Fig. 1, there is a direct correlation between the total number of evaluations in a match (predictions by the model) and the average time per move. This means that reducing leaf expansions reduces the time needed to play a game, and both depth and n repetitions influence greatly this factor. In particular, depth seems to have a much bigger importance, meanwhile increasing the n repetitions scales less than linearly. This is probably caused by our implementation of the MCTS algorithm, that keeps exploring the same path without expanding new leaves until a maximum depth, if the value of that path is much higher than the others.

Deepmind, on the other side, does not limit the maximum depth reachable by the search. This, however, increases drastically the number of leaf expansions in one search. We decided to sacrifice some forecasting ability of the model to gain in speed, our main bottleneck.

Other notable changes derive from the lack of need in exploration given by the ever-changing nature of our initial state: since we are only playing endgames, the initial position is always different. For this reason, we removed the Dirichlet noise added to the policy priors of the first move, and set the temperature parameter to 1/5, forcing the network to almost always choose the most visited child of the root node as following move.

However, we left the c exploration parameter (that depends on the current number of the rollout) in the UCB calculation, because in each MCTS we want to rely more heavily on priors in the first rollouts, while at the end of the search we want to shift our focus on action values, a much more reliable prediction if a node is a winning position. We linearly anneal the parameter from 1 when $n = 0$ to 0.2 when $n = 100$.

The last change we applied is strictly linked with our Neural Network choice: we choose the child with max UCB white moves and min UCB for black moves: this means that all nodes favourable to black will hold negative values, and vice-versa.

2.3 Neural Network

Adhering to our lower scale strategy, our ResNet only holds 8 blocks in its spine, with respect to the 19 or 39 in [3]. We introduce, however, a slight variation: instead of showing the board always in the direction of the current player, we keep the format of the input fixed for both players, and instead

swap the signs of the logits in the policy head when it's black turn. This should help the network have a more coherent understanding of the chess board, and also of the importance of the current player for maximizing the correct logit value.

Both the input and output formats are the same as described in the Methods section of [3]

2.4 Self-play

With regard to the actual training phase, everything follows the original paper, but scaled down:

- The experience buffer holds the latest 40,000 moves played by the learning network, instead than 500,000
- The batch size is limited to 64, instead of 4,096
- The Evaluator used in [2] isn't present, following the findings in [3], and this allows for the creation of only one model that first plays a lot of games and then computes many learning steps, decreasing time and memory utilization
- Training proceeded for 20,000 training steps, instead of 700,000, with a checkpoint every 4000 training steps
- The playing step consisted of 80 games, and the following training step of 100 (or 200) weight updates with batch 64, with every batch randomly sampled from the experience buffer.

We started with a learning rate of 2×10^{-2}, reducing it to 2×10^{-3} and 2×10^{-4} at 3000 and 8000 learning steps respectively.

2.5 Dataset creation

Since we decided to focus on endgames, the starting positions had to be taken into consideration: for this reason we created an endgame generator that, given the number of pieces, generates a random combination that is both legal and not automatically won or lost by either player. Moreover, given the total number of positions required, it generates them with an equal distribution of turns (first move for white or for black) and also with an equal distribution of number of pieces, from 3 to 7. So each position has a 50% possibility of being white or black to move, and a 20% possibility of holding any number of pieces from 3 to 7.

Aside, we created a similar but smaller dataset for the evaluation of the different checkpoints of the network.

2.6 Evaluation

The evaluation is exactly the same as in the original papers, but with the difference in starting position: for this reason, every checkpoint is faced against all the others in all of the starting positions, both from the white and from the black side. The results of all of the matches are collected and then passed to the BayesElo program to calculate the respective Elos of all the checkpoints.

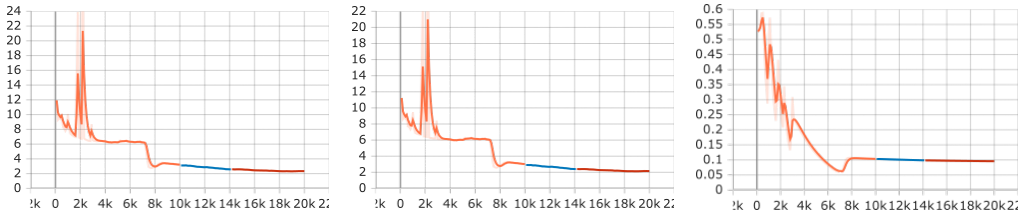
For our main evaluation we only use raw neural network policies to decide the following move, resulting in way worse playing strength but much faster evaluation. For better statistical evidence, all the networks play 100 games against all other checkpoints, from both sides (200 games between each pair of checkpoints). Since the number of matches grows quadratically, utilizing MCTS during evaluation would have required almost half as much time as the training itself (for 6 checkpoints, 6000 games, compared to the total $\sim 16k$ for training). Nonetheless, MCTS is just an improving operator over the policy of the network, so the difference in strength should still be noticeable.

We also add a comparison between our last checkpoint with and without MCTS, to prove the big augmentation that MCTS brings over the same base network it ues.

3 Results

The training lasted 20'000 steps and took more than a week, with 2 intermediate interruptions. During the last phase (red) the training steps were increased to 200 each 80 games played, instead of the

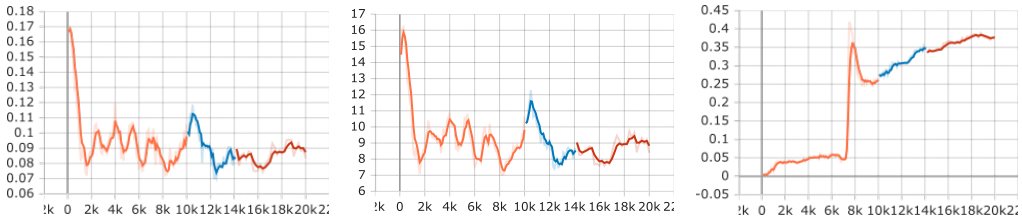
usual 100, to speed up the training. Most probably it could have been increased even more without risking overfitting, but with limited time we couldn't experiment it.



(a) Total loss (b) Policy loss (c) L2 loss

Figure 2: Training data

As can be seen in Fig. 2a and 2b, the main contribution to the total loss was from the policy head. This is a strictly numerical reason, because, at least at the beginning, cross-entropy on an 8*8*73 array has a mean value of $-\frac{1}{k}\sum_{i=1}^{k}\log 1/k = \log k$ that in our case is $\log_2 4672 \sim 12$, while the value loss is an MSE in a range $[-1, 1]$, so always < 4, and in our case, with only $\sim 10\%$ of decisive outcomes, is < 1.



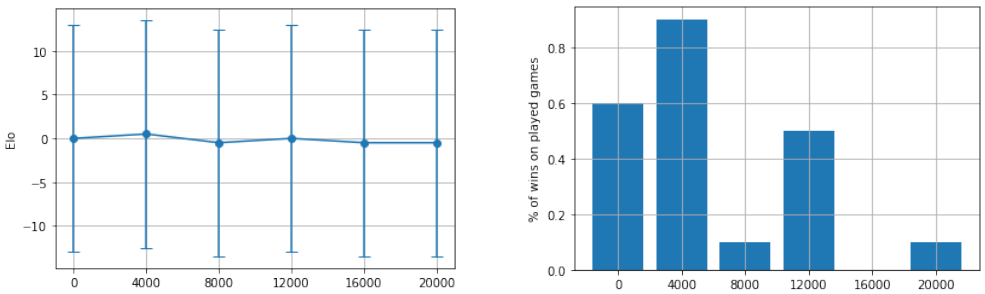
(a) Value loss (b) % of decisive games vs. step (c) Accuracy

Figure 3: More training data

We can also see from Fig. 3a and Fig. 3b that the value head most probably collapsed into only predicting zeros, because the value of the loss seems to be strictly correlated to the number of decisive outcomes (not draws). This surely hinders the augmenting operation of the MCTS, that bases child selection also on the backpropagation of action values from expanded leaf nodes (nodes that are not ending positions).

However, as can be seen in Fig. 3c, the accuracy in the prediction kept increasing, meaning that the model is slowly learning the improved MCTS policy.

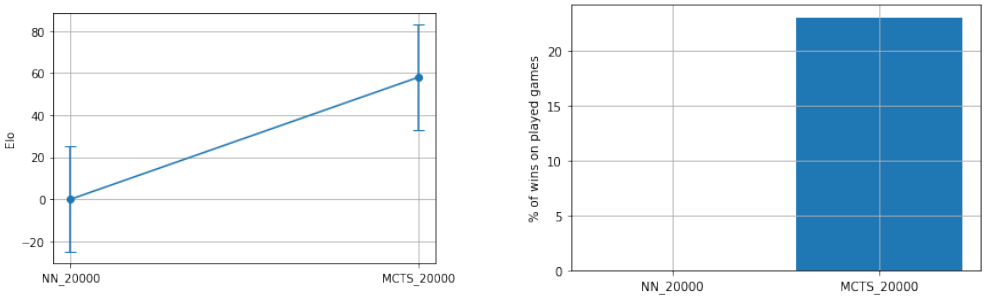
Looking at Fig. 4a, we can see the bare models did not improve with respect to the random initialization. This, however, was to be expected. In fact, even in [4] (Fig. 1A) the Elo of Deepmind's MCTS augmented model was at 0 for the first ~ 15000 steps, even if they used a bigger model, batch size, and a higher learning rate.



(a) Elo vs. steps (b) % of checkpoint wins vs. steps

Figure 4: Evaluation of bare Neural Networks

The big uncertainty on the Elos is given by the small percentage of decisive results (always less than 1%, Fig. 3b) with respect to draws.



(a) Elo MCTS vs. NN, 20k steps

(b) % MCTS wins over NN, 20k steps

Figure 5: Evaluation comparing the best Neural Network to its MCTS augmentation

However, Fig. 5b shows that MCTS augmentation does translate in higher winning chances. We can see that MCTS wins almost 25% of the games against the base NN, gaining almost 60 Elo (fig. 5a).

This means that the strong improvement in policy from MCTS is evident, but is very difficult to learn for the bare model, probably because of the complex input and the difficulty in discriminating between 4672 different possible moves.

3.1 Future Work and Considerations

Surely this project was a massive task, and many errors were made, but a lot was learned for the future:

- As a first consideration, from the trends that can be seen in the loss graph (fig. 2a) and the accuracy graph (fig. 3c), together with the clear evidence of stronger play from MCTS with respect to the bare NN (fig. 5b), we can derive that the model is indeed learning to play as its augmented counterpart, i.e. with higher Elo. However, in chess, one good move ~ 3 (30% accuracy) and two random ones will hardly allow you to checkmate your opponent. This explains why the graph in 4b shows such an erratic behavior, hinting to a almost completely random play from all the checkpoints. The difficulty in checkmating the opponent is solved instead by the Montecarlo rollouts, that will choose with no doubts a move that leads to a checkmate over any other, exploiting the backpropagation of a strong action value (+1 or -1) from the winning position.
- If we were to restart the project, the first change would be to the MCTS. Our implementation sacrificed depth to gain in speed, but, in hindsight, a stronger search would have probably reduced the total training time by increasing the strength of play more quickly.
- When evaluating, surely a comparison between different checkpoints augmented by MCTS would have been more representative, reducing greatly the random play of the bare model. However, just one evaluation of the best model against its MCTS counterpart took over one hour, so we couldn't afford to do the full evaluation between all the chekpoints with MCTS.

To summarize, we think that with simply more time or better hardware at our disposal, the results could have shown a notable improvement, and with some changes in implementation and some more experimenting, the training process could have sped up considerably.

Putting together all of the trends available, the bare model is indeed slowly learning to play as its MCTS augmentation, and the MCTS policy is much stronger than the model. This means that the loop of self-improving policies is confirmed by the data, and over time performance will most surely increase.

References

[1] David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489, Jan 2016.

[2] 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, Oct 2017.

[3] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, Timothy P. Lillicrap, Karen Simonyan, and Demis Hassabis. Mastering chess and shogi by self-play with a general reinforcement learning algorithm. *CoRR*, abs/1712.01815, 2017.

[4] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, Timothy Lillicrap, Karen Simonyan, and Demis Hassabis. A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science*, 362(6419):1140–1144, 2018.

[5] L.V. Allis. *Searching for solutions in games and artificial intelligence*. PhD thesis, Maastricht University, January 1994.

[6] Rémi Coulom. Whole-history rating: A bayesian rating system for players of time-varying strength. In H. Jaap van den Herik, Xinhe Xu, Zongmin Ma, and Mark H. M. Winands, editors, *Computers and Games*, pages 113–124, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.

[7] https://www.remi-coulom.fr/Bayesian Elo/. Bayesian elo rating, 2010.

[8] https://database.nikonoel.fr/. Download the lichess elite database, 2020.