SAI
a Sensible Artificial Intelligence that plays Go

F. Morandin, G. Amato, R. Gini, C. Metta, M. Parton, G. C. Pascutto

September 12, 2018

Abstract

We propose a multiple-komi modification of the AlphaGo Zero/Leela Zero paradigm. The winrate as a function of the komi is modeled with a two-parameters sigmoid function, so that the neural network must predict just one more variable to assess the winrate for all komi values. A second novel feature is that training is based on self-play games that occasionaly branch –with changed komi– when the position is uneven. With this setting, reinforcement learning is showed to work on 7×7 Go, obtaining very strong playing agents. As a useful byproduct, the sigmoid parameters given by the network allow to estimate the score difference on the board, and to evaluate how much the game is decided.

1 Introduction

The longstanding challenge in artificial intelligence of playing Go at professional human level has been succesfully tackled in recent works [5, 7, 6], where software tools (AlphaGo, AlphaGo Zero, AlphaZero) combining neural networks and Monte Carlo tree search reached superhuman level. A recent development was Leela Zero [4], an open source software whose neural network is trained over millions of games played in a distributed fashion, thus allowing improvements within reach of the resources of the academic community.

However, all these programs suffer from a relevant limitation: it is impossible to target their victory margin. They are trained with a fixed komi of 7.5 and they are built to maximize just the winning probability, not considering the score difference.

This has several negative consequences for these programs: when they are ahead, they choose suboptimal moves, and often win by a small margin; they cannot be used with komi 6.5, which is also common in professional games; they show bad play in handicap games, since the winning probability is not a relevant attribute in that situations.

In principle all these problems could be overcome by replacing the binary reward (win=1, lose=0) with the game score difference, but the latter is known to be less robust [3, 8] and in general strongest programs use the former since the seminal works [1, 3, 2].

Truly, letting the score difference be the reward for the AlphaGo Zero method, where averages of the value are computed over different positions, would lead to situations in which a low probability of winning with a huge margin could overcome a high probability of winning by 0.5 points in MCTS search, resulting in weaker play.

An improvement that would ensure the robustness of estimating winning probabilities, but at the same time would overcome these limitations, would be the ability to set the initial bonus for white player (komi) to an arbitrary value. The agent would then maximize the winning probability with a variable virtual bonus/malus, resulting in a flexible play able to adapt to positions in which it is ahead and behind taking into account implicit information about the score difference.

The first attempt in this direction gave unclear results [9].

In this work we propose a model to pursue this strategy, and as a proof-of-concept of its value we apply it to a game of Go on a 7×7 goban. We discuss the result of our experiment and propose a way forward.

The source code of the SAI fork of Leela Zero and of the corresponding server can be found on GitHub at `https://github.com/sai-dev/sai` and `https://github.com/sai-dev/sai-server`.

2 General ideas

In this section we explain the general ideas of our project. Many details, both theorical and applied are deferred to the sections after this.

2.1 Probability of winning

The probability of victory ρ of the current player depends on the state s. For the sake of generality we include a second parameter, i.e. a number $x \in \mathbb{Z}$ of virtual bonus points the player is given. So we will have $\rho = \rho(s, x) = \rho_s(x)$, with the latter being our standard notation. When trying to win by some amount of points n, the agent may let $x = -n$ to ponder its chances. When trying to recover from a losing position (maybe because of handicap) where it estimates being n points behind, the agent may initially let x be a positive integer less than n to try to recover some points to start.

Since $\rho_s(x)$ as a function of x must be increasing and map the real line onto $[0, 1]$, a family of sigmoid functions is a natural choice:

$$\rho_s(x) = \sigma(x + \bar{k}_s, \alpha_s, \beta_s) \tag{1}$$

Here we set

$$\sigma(x, \alpha, \beta) := \frac{1}{1 + \exp(-\beta(\alpha + x))} = \frac{1}{2}\tanh\left(\frac{1}{2}\beta(\alpha + x)\right) + \frac{1}{2} \tag{2}$$

The number $\bar{k}_s$ is the signed komi, i.e. if the real komi of the game is k, we set $\bar{k}_s = k$ if at s the current player is white and $\bar{k}_s = -k$ if it is black.

The number $\alpha = \alpha_s$ is a shift parameter: since $\sigma(-\alpha, \alpha, \beta) = 1/2$, it represents the expected difference of points on the board from the perspective of the current player.

The number $\beta = \beta_s$ is a scale parameter: the higher it is, the steeper is the sigmoid, generally meaning that the result is set.

The highest meaningful value of β is of the order of 10, since at the end of the game, when the score on the board is set, ρ must go from about 0 to about 1 by increasing its argument by one single point.

The lowest meaningful value of β for the full 19×19 board is of the order of $10/2/361 \approx 0.01$, since at the start of the game, even for a very weak agent it would be impossible to lose with a 361.5 points komi in favor.

2.2 Neural network

AlphaGo, AlphaGo Zero, AlphaZero and Leela Zero all share the same core structure, with a neural network[1] that for every state s gives

- a probability distribution over the possible moves p_s (the *policy*);
- a real number v_s (the *value*).

The policy is trained as to choose the most promising moves for searching the tree of subsequent positions.

The value is trained to estimate the probability of winning for the current player. (In some programs v is mapped to $[-1, 1]$ instead of $[0, 1]$ but this doesn't change its meaning and use.)

We propose a modification of Leela Zero neural network that for every state s gives the usual policy p_s, and the two parameters α_s and β_s described above instead of v_s.

The net will feature the usual main structure, based on a 3×3 residual convolutional tower, topped with 3 heads, for the 3 outputs to estimate.

The value head in Leela Zero is formed by a 1×1, 1-filter convolution (with batch normalization and ReLu as is usual), followed by a 256-unit fully-connected layer, ReLu, then 1-unit fully-connected layer and finally a tanh transformation. It is trained against the outcome of the game with l^2 loss function.

We will introduce several slightly different net structures, but the basic idea is to duplicate the value head up to the single unit output. These two heads would yield the final value of α_s and a raw output β_s^* that is transformed to β_s by

$$\beta_s = c \exp(\beta_s^*). \tag{3}$$

The exponential transform imposes the natural condition that β is always positive. The constant c is clearly redundant when the net is fully trained, but the first numerical experiments show that it may be useful to tune the training process at the very beginning, when the net weights are almost random, because otherwise β would be of the order of 1, which is much too large for random play, yielding training problems.

The two outputs will be trained with the usual l^2 loss function but with the value v_s substituted with

$$\rho_s(0) = \sigma\big(\bar{k}_s, \alpha_s, \beta_s\big).$$

Remark 1. *It must be noted that from a theoretical point of view, for each state s we are trying to train two parameters α_s and β_s from a single output (i.e. the game's outcome). This is expected to work somewhat, thanks to the generalizing abilities of neural networks, but it makes the training more difficult, in particular at the beginning of the reinforcement learning pipeline.*

We stress that it would be much better to have at least two finished games (with different komi) for many training states s. More on this in the next section.

[1] Actually, this is done with two neural networks in the case of AlphaGo.

Remark 2. *The basic approach proposed here is was chosen to be simple and lightweight. The komi value of the game is exclusively used in a rigid way to compute the probability of winning. In particular it is not used to modify the estimated policy probabilities. This may have to be changed in the future. A different choice would have been for example to feed the komi value into the input planes.*

2.3 Training data

Training of Go neural networks with multiple komi evaluation is a challenge on its own.

Supervised approach appears unfeasible, since large databases of games have typically standard komi values of 6.5, 7.5 or so and moreover it's not possible to estimate final territory reliably for them.

Unsupervised learning asks for the creation of millions of games even when the komi value is fixed. If that had to be made variable, then theoretically millions of games would be needed *for each komi value*[2].

Moreover, games started with komi very different from the natural values may well be weird, wrong and useless for training, unless one is able to provide agents with different strength.

Finally, the fact that we need to estimate two values instead of one would certainly work better if there were two games with different komi starting from every position, because of the issue mentioned in Remark 1.

We propose a solution to this problem, by dropping the usual choice that self-play games for training always start from the initial empty board position.

The proposed procedure is the following.

1. Start a game from the empty board with random komi near the natural one.
2. For each state in the game, take note of the estimated value of α.
3. After the game is finished, look for states r in which $d := |\bar{k}_r + \alpha_r|$ is large: these are positions in which one of the sides was estimated to be ahead of d points.
4. With some probability start a new game from states s_* with the komi corrected by d points, in such a way that the new game starts with even chances of winning, but with a komi very different from the natural one.
5. Iterate from the start.

With this approach games branch when they become uneven, generating fragments of games with natural situations in which a large komi may be given without compromising the style of game.

Moreover, the starting fuseki positions, that, with the typical naive approach, are greatly over-represented in the training data, are in this way much less frequent.

Finally, not all but many training states are in fact branching points for which there exists two games with different komi, thus expected to yield easier training.

[2]The argument that one can play the games to the end and then score under multiple komi does not work here because this doesn't allow to estimate the β parameter. Moreover that approach would rely on the agent of the self-plays to converge to *score*-perfect play, while the current approach is satisfied with convergence to *winning*-perfect play.

2.4 Agent behaviour

In Leela Zero, the agent chooses moves according to information obtained during a Monte Carlo Tree Search (MCTS) inside the tree of possible futures. The MCTS is guided by the policy and value estimated for each state. The precise algorithm, that will be detailed in Section 4.3.1, is often referred to as *multi-armed bandit* and the tree which it defines iteratively is called *upper confidence tree* (UC tree).

The procedure is actually quite noisy, due to the useful but very imprecise network evaluation of states. MCTS exploration with very high visit number is then able to mitigate this potential weakness thanks to two smart choices:

- the evaluation of the winning probability of an intermediate state s is the *average* of the value v over the subtree of states rooted at s, instead of the typical *minimax* that is expected in these situations;
- the final selection of the move to play is done, at the root of the MCTS tree, by maximizing the number of playouts instead of the winning probability.

Both these choices are very robust to defects of the neural network and hence we chose to incorporate them in our new agent, which is designed to be able to win by large score differences. In other words, we maintained this procedure and just designed a new v. We actually designed a parametric family of scores $\nu = \nu_\lambda(s)$, $\lambda \in [0, 1]$ containing v as a special case, as follows.

The idea is to define ν_λ as the average of $\sigma(x, \alpha, \beta)$ for x on an interval ranging from $\bar{k}$ to a level of bonus/malus points $\bar{x}_\lambda$ that would make the game closer to be even, therefore under- or over-estimating the probability of victory.

Let

$$\nu_0 = \rho(0) = \sigma(\bar{k}, \alpha, \beta) \approx v.$$

Notice that if the signed komi was $\bar{k}' = -\alpha$, then

$$\sigma(\bar{k}', \alpha, \beta) = \frac{1}{2}$$

We want the point $\bar{x}_\lambda$ to be chosen inside $[\bar{k}', \bar{k}]$ in such a way that it is equal to $\bar{k}$ when $\lambda = 0$ and to $\bar{k}'$ when $\lambda = 1$. To this end, let

$$\pi_\lambda := (1 - \lambda)\rho(0) + \lambda\frac{1}{2}, \qquad \lambda \in [0, 1]$$

that is, π_λ is a probability intermediate between the true $\pi_0 = \rho(0)$ (with komi $\bar{k}$) and the even probability $\frac{1}{2}$, and is closer to $\frac{1}{2}$ if λ is higher. Then, let $\bar{x}_\lambda = \bar{x}_\lambda(s)$ be the unique real number such that

$$\sigma(\bar{x}_\lambda, \alpha, \beta) = \pi_\lambda$$

that is, the level of bonus/malus points that would make the probability of victory π_λ.

Finally, let

$$\nu_\lambda(s) := \frac{1}{\bar{k}_s - \bar{x}_\lambda(s^*)} \int_{\bar{x}_\lambda(s^*)}^{\bar{k}} \sigma(x, \alpha_s, \beta_s)dx$$

where s^* is the root node of the UC tree for which the value of $\nu_\lambda(s)$ is needed (more details in Section 4.3).

As a result we have a family of agents, parametrized by λ. If $\lambda = 0$ the agent simply tries to maximize the winning probability, as the Leela Zero agent did – even though this probability is estimated using α and β instead of v. If $\lambda > 0$, the behaviour depends on whether the current player is winning or not, that is, if π_0 is bigger or smaller than $\frac{1}{2}$.

- When the current player is winning, that is $\pi_0 > \frac{1}{2}$, agents with a high λ try to get a higher probability of winning even with a score malus ($\bar{x}_\lambda < \bar{k}$), thus they aim at maximizing the final score.
- When the current player is losing, that is $\pi_0 < \frac{1}{2}$, agents with a high λ try to get a higher probability of winning under the assumption that they have a score bonus ($\bar{x}_\lambda > \bar{k}$), thus they embark on a calm plan to recover points, instead of trying desperate moves to subvert the game.

We observe that the first situation is useful during games with weaker opponents, because agents with high λ would try very strong moves also in the endgame and late middle-game. The second situation is useful to produce a strong game when the software plays with some handicap, in stones or komi.

It is natural to imagine to tune the value of λ dynamically depending on the situation, the opponent's style, or the moment of the game. It would be even possible to apply values of λ outside of $[0, 1]$, provided $\pi_\lambda \in [0, 1]$.

3 Setting the stage: 7×7 Leela Zero

To provide a benchmark for the developement of SAI, we adapted Leela Zero to 7×7 Go board and performed several runs of training from purely random play to a level at which further improvement wasn't expected.

3.1 Scaling down Go complexity

Scaling the Go board from size n to size ρn with $\rho < 1$ yields several advantages:

- Average number of legal moves at each position scales by ρ^2.
- Average length of a game scales by ρ^2.
- The number of visits in the UC tree that would result in a similar understanding of the total game, scales at an unclear rate, nevertheless one may naively infer from the above two, that it may scale by about ρ^4.
- The number of resconv layers in the ANN tower scales by ρ.
- The fully connected layers in the ANN are also much smaller, even if it is more complicated to estimate the speed contribution.

All in all it is reasonable that the total speed improvement for self-play games is of the order of ρ^9 at least.

Since the expected time to train 19×19 Go on reasonable hardware has been estimated to be in the order of several hundred years, we anticipated that for 7×7 Go this time should be in the order of weeks.

In fact, with a small cluster of 3 personal computers with average GPUs we were able to complete most runs of training in less than a week each.

We always used networks with 3 residual convolutional layers of 128 filters, the other details being the same as Leela Zero.

The number of visits corresponding to the standard value of 3200 used on the regular Go board would scale to about 60 for 7×7. We initially experimented with 40, 100 and 250 visits and then went with the latter, which we found to be much better.

The Dirichlet noise α parameter has to be scaled with the size of the board, according to [6] and we did so, testing with the (nonscaled) values of 0.02, 0.03 and 0.045.

The number of games on which the training is performed was assumed to be quite smaller that the standard 250k window used at size 19, and after some experimenting we observed that values between 8k and 60k generally give good results.

3.2 Measuring playing strength

When doing experiments with training runs of Leela Zero, we produce many networks, which must be tested to measure their playing strength, so that we can assess the performance and efficiency of each run.

The simple usual way to do so is to estimate an Elo/GOR score for each network[3]. The idea which defines this number is that if s_1 and s_2 are the scores of two nets, then the probability that the first one wins against the second one in a single match is

$$\frac{1}{1+e^{(s_2-s_1)/c}}$$

so that $s_1 - s_2$ is, apart from a scaling coefficient c (traditionally set to 400), the log-odds-ratio of winning.

This model is so simple that is actually unsuitable to deal with the complexity of Go and Go playing ability. Even for size 7 board.

In fact in several runs of Leela Zero 7×7 we observed that each training phase would produce at least one network which solidly won over the previous best, and was thus promoted to new best. This process would continue forever, or at least as long as we dared keep the run going, even if from some point on, the observed playing style was not evolving anymore. When some match was tried between nonconsecutive networks, we saw that the strength inequality was not transitive, in that it was easy to find cycles of 3 or more networks that regularly beat each other in a directed circle. Even with very strong margins.

We even tried to measure the playing strength in a more refined way, by performing round-robin tournaments between nets and then estimating Elo score by maximum likelihood methods. This is much heavier to perform and still showed poor improvement in predicting match outcomes.

It must be noted that this appears to be an interesting research problem in its own. The availability of many artificial playing agents with different styles, strengths and weaknesses will open new possibilities in collecting data and experimenting in this field.

[3] In fact the neural network, is just one of many components of the playing software, which depends also on several other important choices, such as the number of visits, fpu policies and all the other parameters. Rigorously the strength should be defined for the playing *agent* (each software implementation of Leela Zero), but to ease the language and the exposition, we will speak of the strength of the *network*, meaning that the other parameters were fixed at some value for all matches.

Remark 3. *It appears that this problem is mainly due to the peculiarity of 7×7 Go and only relevant to it.*

In the official 19×19 Leela Zero project the Elo estimation is done with respect to previous best agent only and it is known that there is some Elo inflation, but tests against a fixed set of other opponents or against further past networks have shown that real playing strength does improve.

3.2.1 Panel evaluation and elicitation

A different approach which is both robust and refined and is easy to generalize is to use a panel of networks to evaluate the strength of each new candidate.

We chose 15 networks of different strength from the first 5 runs of Leela Zero 7×7. Each network to be evaluated is opposed to each of these in a 100 games match. The result is then a vector of 15 sample winning rates, which contains useful multivariate information on the playing style, strengths and weaknesses of the tested net.

To summarize this information in one rough (but scalar) score number, it would not do to simply sum the winning rates, since the elements of the panel have different strength and the results against them will certainly be correlatend in a complex way. Some sort of weighted sum is recommended, thus allowing for a sort of *elicitation* of the panel nets.

An imperfect but simple solution is to use principal component analysis. We performed covariance PCA once for all on the match results of the first few hundreds of good networks, determined the principal factor and used its components as weights[4]. Hence the score of a network is the principal component of its PCA decomposition.

This value, which we call *panel evaluation*, correlates well with the maximum likelihood estimation of Elo by round-robin matches, but is much easier and quick to compute and convenient to use on large sets of networks from different runs.

3.3 Effect of visits and Dirichlet parameter

After a few experimental runs to ensure that the system was working well, we decided to study the influence of the two main factors: visits and Dirichlet's α.

3.3.1 Design of experiment

We performed 5 runs (Taguchi L4 design plus central point).

Number of visits. It is the total number of nodes of the UC tree which is built for every position in the game to decide the next move. The higher it is, the slower are the games, because each move takes more time. Because of tree reuse, if the net is strong, only a small fraction of the nodes will be computed anew on each move, and hence the time dependence on this parameter is not expected to be linear.

[4]By the properties of PCA, the principal factor will find the maximum distinguisher between results againts the panel networks. In the 15-dimensions space of results this will be the direction in which the networks analysed are more different from one another. The coefficients of this factor resulted to be all positive numbers ranging from 0.040 for very weak networks, to 0.415 for the strongest one, thus confirming that these weights represent a reasonable measure of strength.

Moreover if a larger value is used, then the agent will play better, generations will improve faster and so fewer generations of nets are expected to be required to reach the same level of play. In this sense we expected some trade-off in the total effort required and possibly the existence of an optimal value for this parameter.

The values used for these experiments are 40, 100 and 250. (Logarithmically spaced.)

Dirichlet noise parameter. It is denoted by $\alpha > 0$ and is used in the generation of a random probability distribution η on the moves (itself Dirichlet distributed), which is used as a perturbation of the policy distribution p estimated by the neural network to get the noisy policy P:

$$P := (1 - \varepsilon)p + \varepsilon\eta$$

Here $\varepsilon = 0.25$, $\eta \sim \mathrm{Dir}(\alpha^*)$ and $\alpha^* = \alpha\frac{19^2}{7^2}$ is rescaled according to [6].

By the properties of the Dirichlet distribution, if m is the number of legal moves, it is expected that most of the probability mass of η will be concentrated on $m\alpha^*$ moves. (This number is ≈ 10 at the start of the game, for the default value $\alpha = 0.03$. Hence these randomly selected moves will have an average probability bonus of about $\frac{\varepsilon}{m\alpha^*}$, or 2.5%.)

It is expected that this parameter impacts on the quality and quantity of exploration during self-play and that it may interact in a nonlinear way with the number of visits, since if the latter is too low then these probability bonus can have small to no effect.

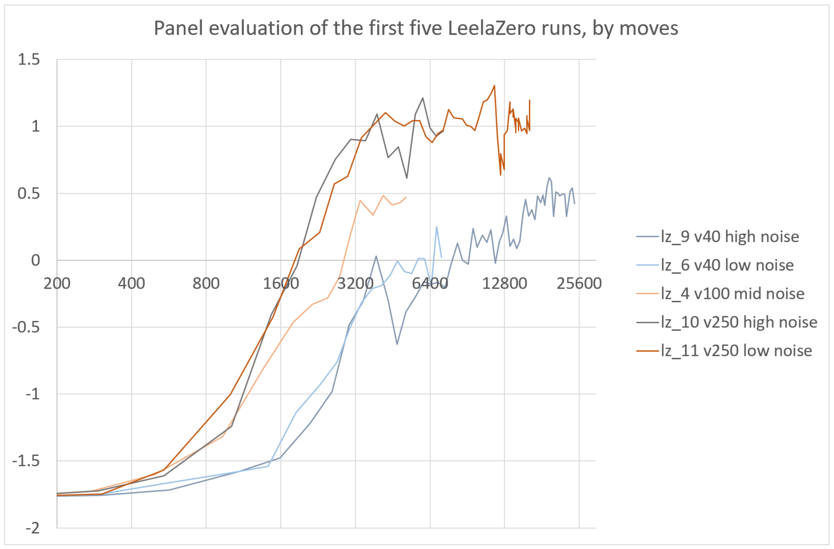
The values used for these experiments are 0.02, 0.03, 0.045. (Logarithmically spaced.)

Other parameters were fixed at their default value. In particular, the number of moves played more randomly at the beginning of self-play games was set to 4, which seems a suitable rescaling of the default value of 30 which is used on size 19 board.

3.3.2 Experimental protocol

After some trial and error the following choices were made.

- AlphaGo Zero promotion criterion: at least 55% wins of 400 games against current best network.
- Fixed number of 10240 self-play games per generation.
- Training window of 1 to 5 previous generations (10240 to 51200 games) according to whether the last generations improved much or not.
- Minibatch size 512. Training rate 0.05, training steps 8000. New candidate network to test every 1000 steps.
- Training starts from current best.
- If no new network gets promoted, replicate the last training once, then try again up to 10 times by changing the number and choice of generations in the training window.
- Stop the run if unable to find a network that gets promoted, or after waiting at least 10 more generation if there is evidence that the playing strength against the panel is not improving. (Even if new networks get promoted each time.)



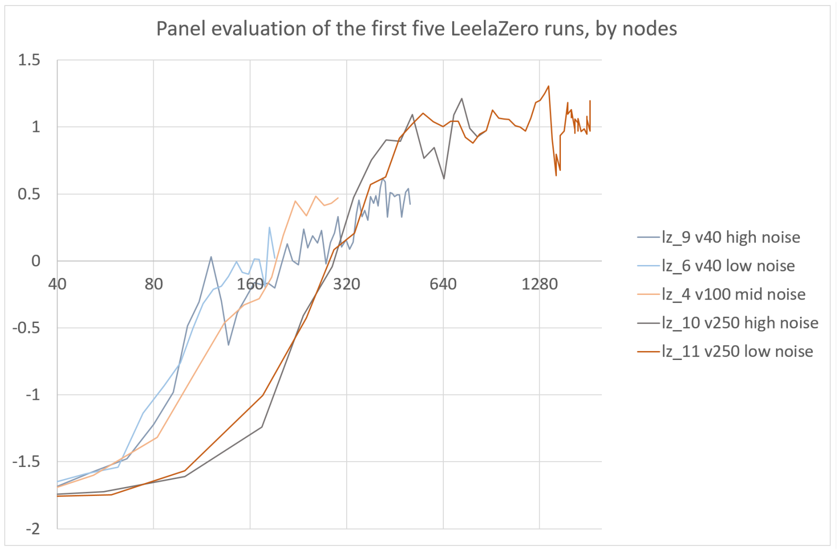


Figure 1: The first plot x axis is expressed in thousand moves played. Runs with high visits use less moves (hence less games) to learn. However, moves determined with high visits number are slower to compute, hence the second plot, where the panel evaluation is expressed as a function of the millions nodes computed, is more useful. Runs with low and mid visits are faster to growth, but stall before reaching the maximum level of play of the others.

3.3.3 Results

The results of the five runs are in the plots of Figure 1. The Dirichelet's α parameter has apparently no effect in this range of values. The number of visits instead has a major importance, in that not only it impacts the speed of learning, but it also affects the final level of playing that is reached by the best networks of the run.

3.4 Effect of net structure and AlphaZero protocol

After the first experiments, we decided to fix the number of visits to 250 and the Dirichelet's parameter to 0.02 and studied the dependence on different choices in the runs protocol.

AlphaZero promotion criterion. There is only one new network for each generation. Every network plays 1280 self-play games. The training window is of the last 16 generations, for a total of at most 20480 games. The next generation's network is produced after 1000 training steps and automatically promoted best network.

AlphaZero randomness. All moves of self-play games are played more randomly.

Augmented filters. The net structure is slightly augmented by raising from 1 to 2 the number of filters for the $1{\times}1$ convolutions of the value head.

3.4.1 Results

The results of the five runs are in the plots of Figure 2.

It is apparent that AlphaZero promotion criterion gives good fast improvement and even seems more uniform than promotion conditioned on winning.

AlphaZero randomness seems to give more stability but slows quite a bit the learning at the beginning.

The augmented filters version seems harder to train and somewhat of similar performances.

4 Proof of concept: 7×7 SAI

After gaining a proper understanding of the learning process of 7×7 Leela Zero we started experimenting with SAI.

4.1 Neural network structure

As explained in Section 2.2, Leela Zero's neural network provides for each position two outputs:

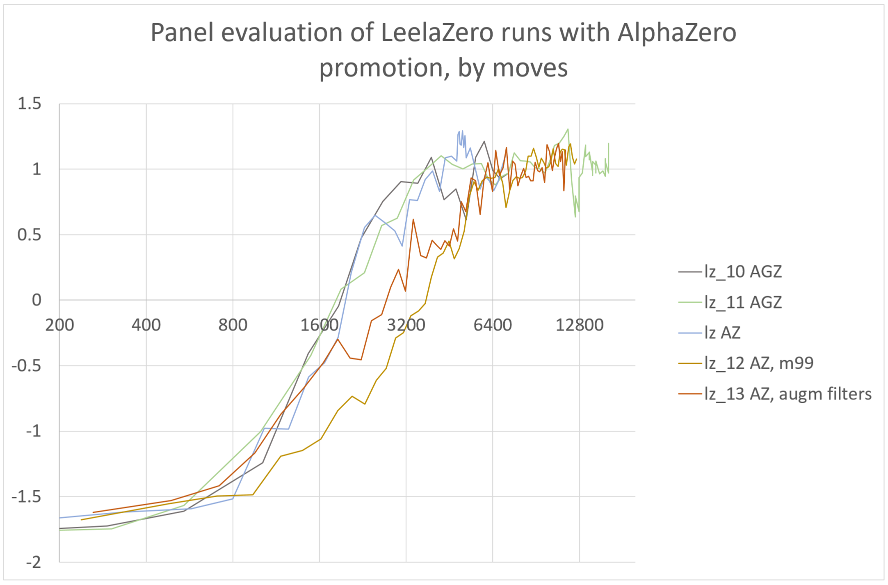
policy – is a probability distribution over the existing moves which predicts the moves that the engine should read with higher priority;

winrate – is an estimate of the probability of winning of the current player.

SAI's neural network should provide for each position three outputs: the policy as before and the two parameters α and β of a sigmoid function which would allow to estimate the winrate for different komi values with a single computation of the net.

It is unclear whether the komi itself should be provided as an input of the neural network: it may help the policy adapt to the situation, but could also make the other two parameters unreliable[5]. For the initial experiments the komi will not be provided as an input to the net.

[5]As will be explained soon, the training is done at the level of winrate, so in principle, knowing the komi, the net could train α and β to any of the infinite pairs that, with that komi, give the right winrate.



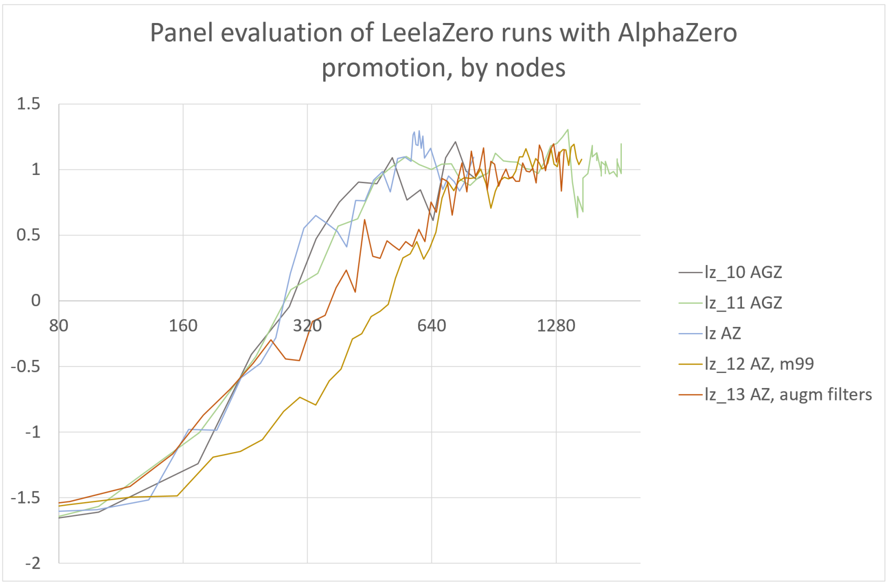


Figure 2: Effect of net structure and AlphaZero promotion criterion. The three new runs all implement automatic promotion of new networks. The first one is otherwise identical to the previous ones. The second one has all the moves played more randomly. The third one has one more filter in the value head (see text). The best runs of the previous set, `lz_10` and `lz_11` are reported as a comparison.

With the above premises, the first structure we propose for the network is very similar to Leela Zero's one, with the value head substituted by two identical copies of itself devoted to the parameters α and β^*. The latter is then mapped to β by equation (3).

We call the following *type V* structure.

- One 3×3 convolutional layer with 18 input planes[6] and 128 output filters, followed by batch normalization and ReLU.
- A tower of three 3×3 residual convolutional layers with 128 inputs and 128 filters each, followed by batch normalization and ReLU.
- One 1×1 convolutional layer with 128 inputs and 2 filters, followed by batch normalization and ReLU.
- One fully connected layer with 2×49 inputs and $49 + 1$ filters, followed by softmax gives the policy distribution.
- Attached again to the output of the tower of residual layers, one 1×1 convolutional layer with 128 inputs and 1 filter, followed by batch normalization and ReLU.
- One fully connected layer with 49 inputs and 256 filters, followed by ReLU, and then a fully connected layer with 256 inputs and 1 filter, and finally ReLU, gives the parameter α.
- Attached again to the output of the tower of residual layers, one 1×1 convolutional layer with 128 inputs and 1 filter, followed by batch normalization and ReLU.
- One fully connected layer with 49 inputs and 256 filters, followed by ReLU, and then a fully connected layer with 256 inputs and 1 filter, and finally ReLU, gives the parameter β^*.

Some natural alternatives to the above are *type Y* and *type T* structures, which substitute the last four points of the list above respectively with:

- Attached again to the output of the tower of residual layers, one 1×1 convolutional layer with 128 inputs and 2 filters, followed by batch normalization and ReLU.
- One fully connected layer with 2×49 inputs and 128 filters, followed by ReLU, and then a fully connected layer with 128 inputs and 1 filter, and finally ReLU, gives the parameter α.
- Attached again to the output of the last convolutional layer, one fully connected layer with 2×49 inputs and 128 filters, followed by ReLU, and then a fully connected layer with 128 inputs and 1 filter, and finally ReLU, gives the parameter β^*.

and with:

- Attached again to the output of the tower of residual layers, one 1×1 convolutional layer with 128 inputs and 2 filters, followed by batch normalization and ReLU.

[6]Two input planes are either all ones or all zeros according to the color of the current player. The other 16 layers represent the last 8 positions of the board as bit planes describing the stones of current/other player.

- One fully connected layer with 2×49 inputs and 256 filters, followed by ReLU, and then a fully connected layer with 256 inputs and 2 filters, and finally ReLU, gives the parameters α and β^*.

4.2 Training

To train the network we included the komi value into the training data used by SAI. The training is then performed the same way as for Leela Zero, with the loss function given by the sum of regularization term, cross entropy for the policy and l^2 norm for the winning rate.

The winning rate is computed with the sigmoid function given by equations (1) and (2), in particular we set $v(s) = \rho_s(0)$ and backpropagate gradients through these functions.

4.2.1 On generating good training data

To train the neural network it is clearly necessary to have different komi values in the data set. It would be best to have *very* different komi values, but when the agent starts playing well enough, only few values around the correct komi[7] make the games meaningful.

To adapt the komi values range to the ability of the current network, when the server assign a self-play match to a client, it chooses a komi value randomly generated with distribution given by the sigmoid itself. Formally,

$$K = 0.5 + \lfloor \rho_s^{-1}(U) \rfloor \tag{4}$$

where $\rho_s(x) = \sigma(x, \alpha_s, \beta_s)$, s is the initial empty board state, α_s and β_s are the computed values with current network and $U \sim \text{unif}(0, 1)$, thus giving to K an approximate logistic distribution.

As the learning goes on, we expect α_s to converge to the correct value of 9, and β_s to increase, narrowing the range of generated komi values.

To deal with this problem we implemented the possibility for the server to assign self-play games starting from any intermediate position.

After a standard game is finished, the server looks to each of the game's positions and from each one may branch a new game (independently and with small probability). The branched game starts at that position with a komi value that is considered even by the network. Formally,

$$k' = 0.5 + \lfloor \pm\alpha_s \rfloor$$

where s is the branching position and $\pm\alpha_s$ is the value of α at position s, as computed by the current network, with the sign changed if the current player was white.

The branched game is then played until it finishes and then all its positions starting from s are stored in the training data, with komi k' and the correct information on the winner of the branch.

[7]The correct komi for 7×7 Go is known to be 9, in that with that value both players can obtain a draw. Since we didn't want to deal with draws, for 7×7 Leela Zero we chose a 9.5 komi, thus giving victory to white in case of a perfect play. In fact we noticed that with a komi of 7.5 or 8.5 (equivalent by chinese scoring) the final level of play of the agents didn't seem to be as subtle as it appears to be for the 9.5 komi.

This procedure should produce branches of positions with unbalanced situations and values for the komi that are natural to the situation but nevertheless range on a wide interval of values.

4.3 Sensible agent

When SAI plays, it can estimate the winning probability for all values of the komi with a single computation of the neural network. In fact, getting α and β it knows the sigmoid function that gives the probability of winning with different values of the komi for the current position.

We propose the generalization of the original agent of Leela Zero as introduced in Section 2.4. Here we give further details.

The agent behaviour is parametrized by a real number λ which will be usually chosen in $[0, 1]$. The expected behaviour is to simply maximize the probability of winning when $\lambda = 0$ (as Leela Zero does) and try to win but also increase the score when $\lambda > 0$.

4.3.1 Formalization of the UC tree

To describe rigorously the agent, we need to introduce some more mathematical notation.

Games, moves, trees. Let $\mathcal{G}$ be the set of all legal game states, with $\varnothing \in \mathcal{G}$ denoting the empty board starting state.

For every $s \in \mathcal{G}$, let $\mathcal{A}_s$ the set of legal moves at state s and for every $a \in \mathcal{A}_s$, let $s_a \in \mathcal{G}$ denote the game state reached from s by performing move a. This clearly induces a directed graph structure on $\mathcal{G}$ with no directed cycles (which are not legal because of superko rule) and with root $\varnothing$. This graph can be uplifted to a rooted tree by taking multiple copies of the states which can be reached from the root by more than one path. From now on we will identify $\mathcal{G}$ with this rooted tree and denote by $\to$ the edge relation going away from the root.

For all $s \neq \varnothing$ let $\bar{s}$ denote the unique state such that $\bar{s} \to s$.

For all $s \in \mathcal{G}$, let $\mathcal{R}_s = \{r \in \mathcal{G} : s \to r\}$ denote the set of states reachable from s by a single move. We will identify $\mathcal{A}_s$ with $\mathcal{R}_s$ from now on.

For any subtree $T \subset \mathcal{G}$, let $|T|$ denote its size (number of nodes) and for all $s \in T$, let T_s denote the subtree of T rooted at s.

Values, preferences and playouts. Suppose that we are given a policy P and two value functions u, v with the following properties:

- the *policy* P, defined on $\mathcal{G}$ with values in $[0, 1]$ and such that

$$\sum_{r \in \mathcal{R}_s} P(r) = 1, \qquad s \in \mathcal{G};$$

- the *value* v, defined on $\{(s, r) : s \in \mathcal{G}, r \in \mathcal{G}_s\}$ with values in $[0, 1]$;
- the *first play urgency* u, defined on $\mathcal{G}$ with values in $[0, 1]$.

Then for any non-empty subtree T and node s not necessarily inside T we can define the *evaluation* of s over T, as

$$Q_T(s) := \begin{cases} u(s) & \text{if } s \notin T \\ \dfrac{1}{|T_s|} \displaystyle\sum_{r \in T_s} v(s, r) & \text{if } s \in T \end{cases}$$

It should be noted here that u may well also depend on the subtree T. In fact two proposed choices for u are the following:

$$u(s) \equiv 0.5 \qquad \text{(AlphaGo Zero)}$$

$$u_T(s) = v(\bar{s}, \bar{s}) - C_{\text{fpu}} \sqrt{\sum_{r \in T_{\bar{s}}} P(r)} \qquad \text{(Leela Zero)}$$

We can then define the *UC urgency* of s over T, as

$$U_T(s) := Q_T(s) + C_{\text{puct}} \sqrt{|T_{\bar{s}}| - 1} \frac{P(s)}{1 + |T_s|}$$

Finally, the *playout* over T, starting from $s \in T$ is defined as the unique path on the tree which starts from s and at every node r chooses the node $t \in \mathcal{R}_r$ that maximizes $U_T(t)$.

Definition of v. In the case of Leela Zero, the value function $v(s, r)$ does not depend on s and is simply the output of the value head of the neural network, passed through an hyperbolic tangent and rescaled in $(0, 1)$.

In the case of SAI, as explained in Section 2.4 we compute the average winrate at r over a range of komi values that depends on s.

Formally, for any state s, let

$$\sigma_s(x) := \sigma(x, \alpha_s, \beta_s), \qquad x \in \mathbb{R}$$

Let k be the real komi value of the game and let

$$\bar{k}_s := \begin{cases} k & \text{if at } s \text{ the current player is white} \\ -k & \text{if at } s \text{ the current player is black} \end{cases}$$

Then the estimated winrate for the current player at s is $\rho_s(0) := \sigma_s(\bar{k}_s)$ and it is natural to generalize this value to the family of averages

$$\mu_s(y) := \begin{cases} \rho_s(0) & \text{if } y = \bar{k}_s \\ \dfrac{1}{\bar{k} - y} \displaystyle\int_y^{\bar{k}} \sigma_s(x) dx & \text{otherwise} \end{cases} \tag{5}$$

Ideally the right value of y should be not too far from $\bar{k}$, in particular if the estimated winning rate is not far from $1/2$. The winning rate estimated at y should not be too different from the one estimated at $\bar{k}$.

To this end, a natural choice is to define

$$\pi_\lambda := (1 - \lambda)\rho_s(0) + \lambda \frac{1}{2}$$

(not too different from $\rho_s(0)$ and on the same side of $1/2$ for $\lambda < 1$) and to let

$$\bar{x}_s := \sigma_s^{-1}(\pi_\lambda)$$

(not too far from $\bar{k}$ when the state is critical, but possibly very far from it when the winning rate is high or low).

The final definition is then

$$v(s, r) := \mu_r(\bar{x}_s)$$

where the value is computed at state r but the range of the average is decided at state s.

Remark 4. *We bring to the attention of the reader that a simple rescaling shows that the quantity $\mu_r(\bar{x}_r)$ would be somewhat less useful, because it depends on α_r and β_r only through $\rho_r(0)$.*

Remark 5. *Notice that the integral in equation (5) can be computed analitically and easily implemented in the software.*

Tree construction and move choice. Suppose we are at state $t \in \mathcal{G}$ and the agent has to choose a move in $\mathcal{A}_t$. This will be done by defining a suitable *decision* subtree $\mathcal{T}$ of $\mathcal{G}$, rooted at t, and then choosing the move s randomly inside $\mathcal{R}_t$ with probabilities proportional to

$$\exp(C_{\text{temp}}^{-1} |\mathcal{T}_s|), \qquad s \in \mathcal{R}_t$$

where C_{temp} is the Gibbs temperature which is defaulted to 1 for the first moves of self-play games and to 0 (meaning that the move with highest $|\mathcal{T}_s|$ is chosen) for other moves and for match games.

The decision tree $\mathcal{T}$ is defined by an iterative procedure. In fact we define a succession of trees $\{t\} =: T^{(1)} \subset T^{(2)} \subset \ldots$ and stop the procedure by letting $\mathcal{T} := T^{(N)}$ for some N (usually the number of *visits* or when the thinking time is up).

The trees in the succession are all rooted at t and satisfy $|T^{(n)}| = n$ for all n, so each one adds just one node to the previous one:

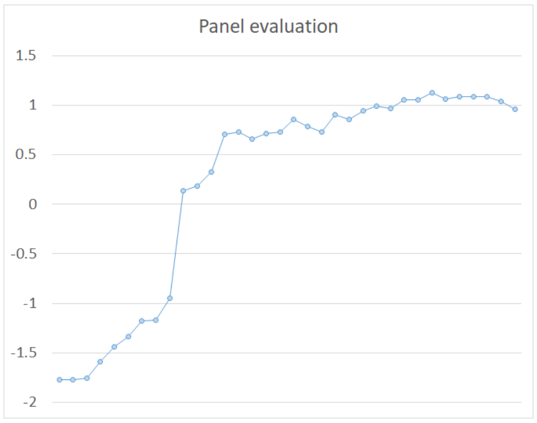
$$T^{(n)} = T^{(n-1)} \cup \{t_n\}$$

The new node t_n is defined as the first node outside $T^{(n-1)}$ reached by the playout over $T^{(n-1)}$ starting from s.

4.4 Results

4.4.1 Obtaining a strong SAI

The first experimental run was done before the generation of branching games was implemented, and it showed that SAI could learn to play at the same level as Leela Zero, attaining a maximum evaluation of 1.12 with many networks above 1, meaning that the winrate was correctly estimated. Nevertheless the two parameters α and β were not estimated very well and even the stronger networks failed to discover that the correct komi is 9. (See Figure 3



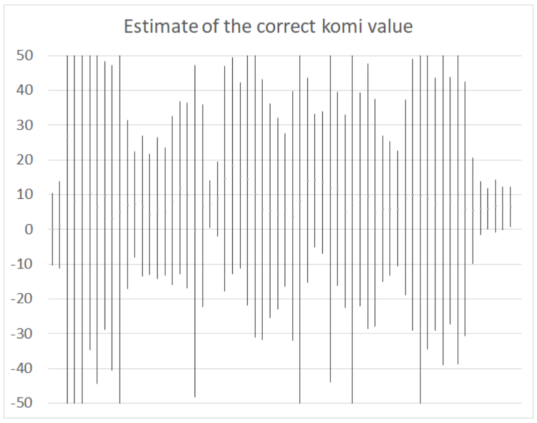


Figure 3: First explorative run of 7×7 SAI.
The left plot shows the panel evaluation of several of the networks in training order. The number of moves/nodes was not reported, because this run didn't follow a strict protocol and they wouldn't be meaningful. The playing strength of some of the last nets is above 1, similar to the typical levels of Leela Zero. The right plot shows the "correct" komi value estimate as given by all the trained networks. The 95% confidence interval was obtained by computing $\alpha_\varnothing$ and $\beta_\varnothing$ for the empty board state $\varnothing$ and then plotting $\alpha_\varnothing \pm \frac{2}{\sqrt{3}\beta_\varnothing}$ (because we consider a logistic distribution). It is apparent that without the generation of branching games the networks have a poor understanding of the two parameters.

In the subsequent runs, plotted in Figure 4, we experimented with several parameters: the promotion protocol (AlphaGo Zero vs AlphaZero), the network type (V, Y or T), the value of λ for self-play games (0 or 0.5) and the first play urgency definition (Leela Zero or AlphaGo Zero).

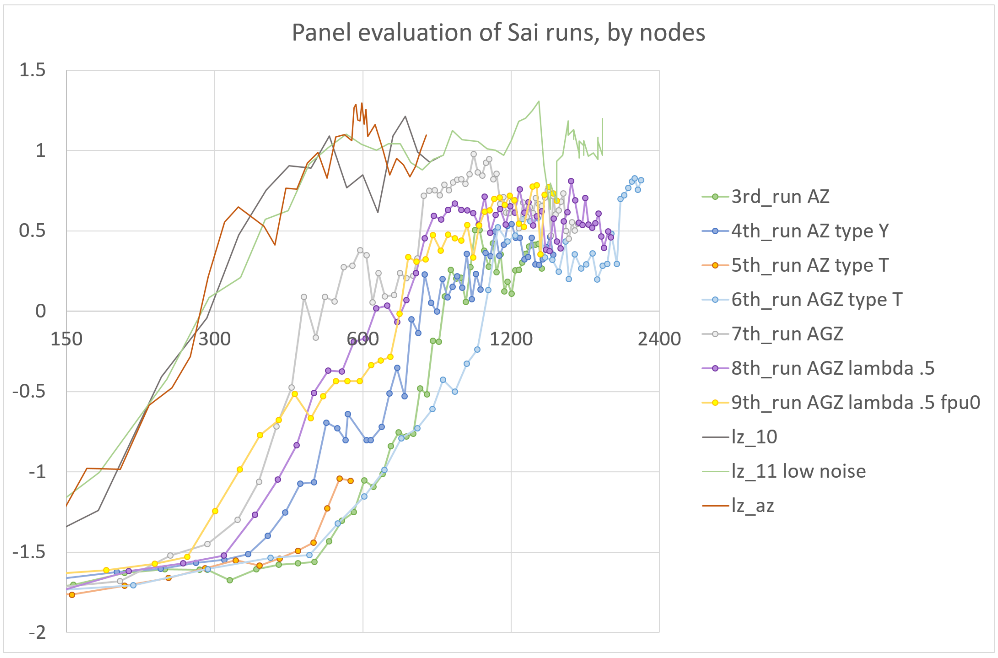
These experiments were done with 250 visits and Dirichelet's parameter 0.02. The number of first move played more randomly was changed between 4 and 15.

Generation of branching games was implemented in all these runs, with probability of branching $C_{\text{branch}}[1-4w(1-w)]$, where $w = \rho_s(0)$ is the estimated probability of winning at state s and the constant is set to 0.025. The rationale behind this choices was that we thought it to be more useful to branch in very unbalanced situations, and since the average length of a game is around 40 moves, this formula gives approximatively one branch per standard game.

All the matches with the evaluation panel were done with $\lambda = 0$ and Leela Zero's definition of first play urgency, with the intent to attain the maximum strength.

These experiments showed that with branching generation it is indeed possible to train α and β correctly, and to learn very sharply that the correct komi is 9. (See Figure 5.) Nevertheless the maximum playing strength was somewhat lower than Leela Zero and the learning process have a tendency to get stuck at evaluations values in the range $[0.5, 1]$.

Analysis of the losing match games that affected panel evaluation mostly showed that SAI nets were unable to anticipate some tesuji moves of Leela Zero panel nets in complex situations. Further study of the nets evaluations and of the tree search



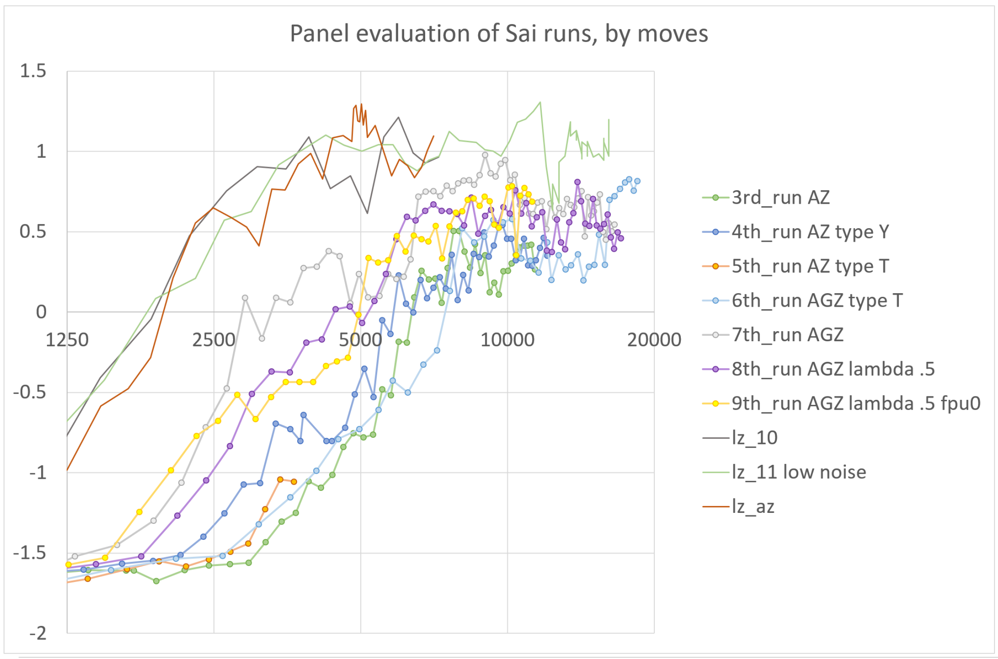


Figure 4: Panel evaluation of all the current runs of SAI. The first one as a function of the millions nodes computed, the second one as a function of the thousand moves played. The performance of the best three runs of Leela Zero is reported as a landmark.

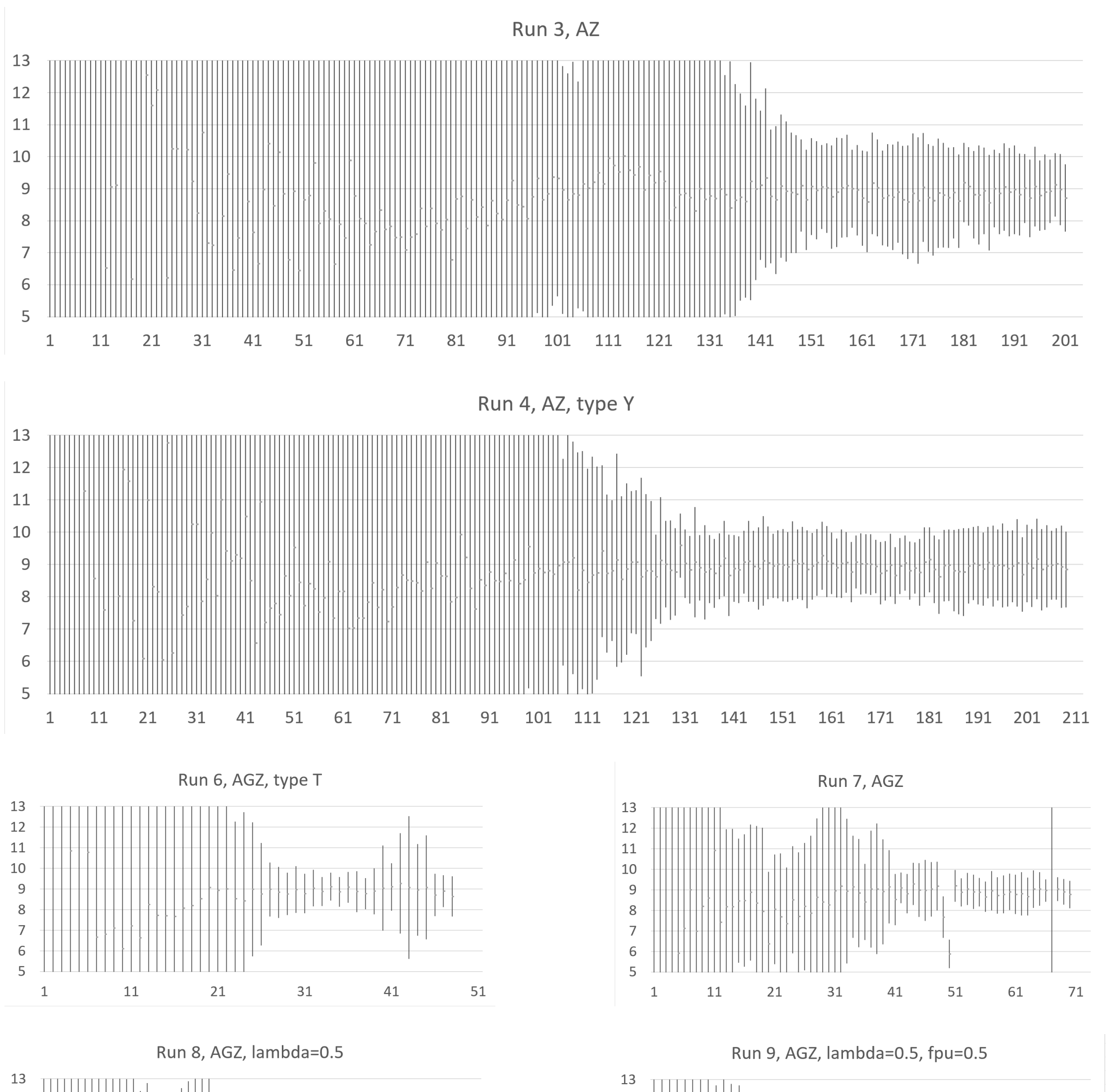


Figure 5: Estimate of the "correct" komi value, as given by all the networks of different runs. The confidence interval was obtained as in Figure 3, but notice that the scale is very different.

development in those positions suggested that SAI nets had a poor accuracy in estimating α in near even positions.

Considering this, we tried to simplify the formula for the branching probability, giving constant probability of branching $C_{\text{branch}} = 0.025$ for all states, thus giving higher chance of branching in balanced situation.

We applied this change at the end of the 9th run of SAI to see if it was able to produce immediate benefit, and indeed the improvement was almost immediate, steady and important, with many networks above 1 with a maximum of 1.11. See Figure 6.

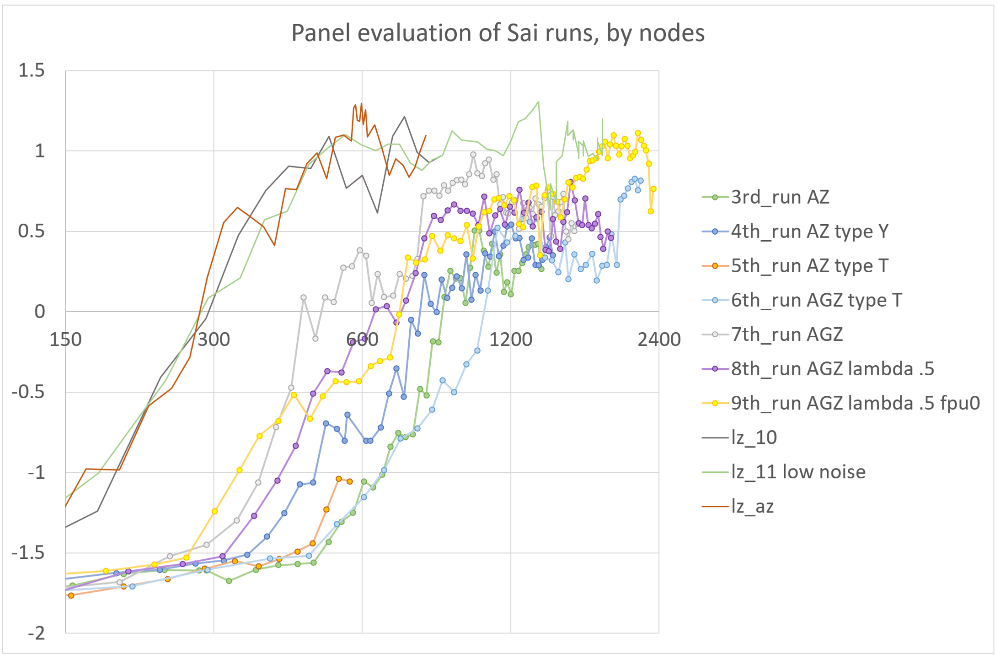


Figure 6: The 9th run of SAI, after the introduction of the constant branching probability, showed a clear improvement in playing strength, reaching the values of Leela Zero.

4.4.2 Evaluation of positions by Leela Zero and by SAI

To illustrate the ability of SAI to understand the winning probability in a more complex fashion, we chose 3 meaningful positions which are shown in Figure 7.

For each position we plotted SAI's sigmoid evaluations of the winrate (black curves) and Leela Zero's point estimates at standard komi (blue dots). Every one of these plots shows a sample of 41 Leela Zero and 76 SAI nets from different runs, chosen among the strongest ones.

There appears to be much variability, showing that even strong nets do not have a clear understanding of single complicated positions, this being mitigated by MCTS when choosing the next move. It is important to observe that the distributions of the winrates seem to agree for the two groups at standard komi, indicating that SAI's estimates have similar accuracy and precision as Leela Zero's.

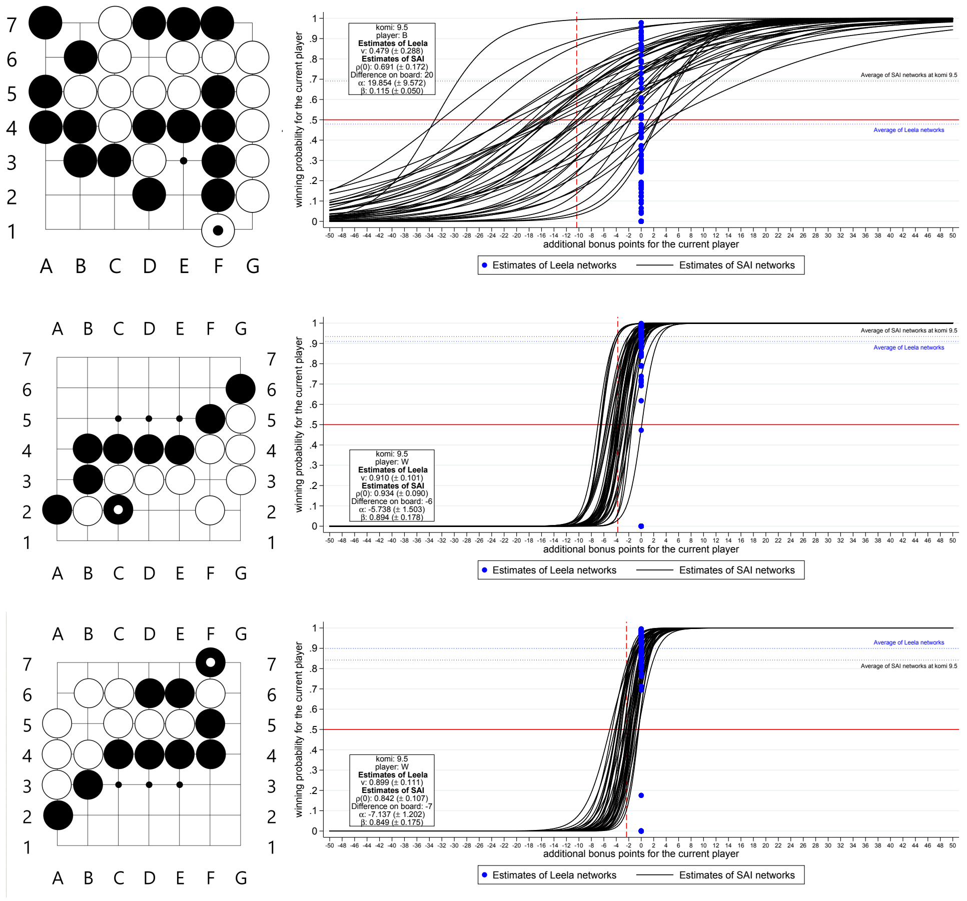


Figure 7: Evaluation of three positions by a sample of strong Leela Zero and SAI nets.

Remark 6. *It would be really important for the playing agent to be able to estimate this variability for each position, that is, have some knowledge of how much precise is its current estimate of the winrate. This is particularly relevant, because this variability is highly dependent on the position.*

SAI networks have this possibility: these plots show that in the more complex first position, the value of β is small (consistently across the nets), while in the latter two positions, β is much larger.

Thus, the estimate of β can be considered as a measure of the precision of both α and the winrate estimate, hence allowing a single network to understand if the situation is more or less complicated and allowing for subtle tuning of future agents based on this kind of networks.

Finally, we emphasize that the SAI nets also provide an estimate of the difference of points between the players.

Position 1. Here it is black turn and black is ahead of 13 points on the board, thus, with komi 9.5, black margin is 3.5 points. However the position is difficult, because there is a seki (quite uncommon in our 7×7 games) which may be poorly interpreted as dead (black ahead by 49 points on the board) or as alive (black ahead by 5 points on the board). In agreement with this analysis, the sample of SAI nets gives a low and sharp estimate for β with average 0.115 and standard deviation 0.050 and a wild estimate for α, with average 19.9 and standard deviation 9.6, with values ranging from 5 to 42. The sample of Leela Zero nets gives winrate estimates which are almost uniformly distributed in $[0, 1]$: many of these nets have a wrong understanding of the position and are not aware of this. SAI nets on the other hand are aware of the high level of uncertainty.

Position 2. Here it is white turn and white is behind by 5 points on the board, thus, with komi 9.5, white is winning by 4.5 points. Following the policy, which recognizes a frequent shape here, many nets will consider cutting at F6 instead than E5, so losing two points. Accordingly, the estimate of α ranges approximately from -3 to -10 with average -5.74 and standard deviation 1.50. The sample of β has average 0.89 and standard deviation 0.18, thus showing that α is to be considered precise to plus or minus one unit.

Position 3. Here the situation is very similar to the previous one: white is behind by 7 points on the board, thus, with komi 9.5, white is winning by 2.5 points. Following the policy, which recognizes a frequent shape here, many nets will consider cutting at B2 instead than C3, so losing two points. Accordingly, the estimate of α ranges approximately from -5 to -9 with average -7.14 and standard deviation 1.20. The sample of β has average 0.85 and standard deviation 0.18, thus showing that α is to be considered precise to plus or minus one unit.

4.4.3 Experimenting different agents for SAI

Finally, we experimented on how the attitude parameter λ of SAI affects the playing strength and the style of the agent. This was done by computing panel evaluation of the same nets when playing matches with $\lambda > 0$ (normally panel matches for SAI nets are played with $\lambda = 0$).

We computed the panel evaluation at $\lambda = 0.5$ of all the nets of the 9th run of SAI. We expected to obtain a slightly lower strength than with $\lambda = 0$, since the latter agent should in principle maximize the winning probability, while the former should try to balance winning probability and a higher score margin, which, against an almost perfect player, could result in running the risk of losing more often. Moreover, when in disadvantage, an agent with $\lambda > 0$ overestimates its winning probability, thus leading to a weaker game against an almost perfect player.

The results are illustrated in Figure 8. We found that the strength of most networks was not affected by setting $\lambda = 0.5$. Only one group of weak consecutive networks showed a clear decrease in strength, while the other differences were compatible with pure statistical fluctuations.

This is probably due to the fact that margins of victory are almost non-existent in the 7×7 setting, therefore the performance of a strong net may be independent on λ in this context. On larger boards, agents with a variable value of λ may be experimented to test their ability to target higher margins of victory and avoid sub-optimal moves, while still maintaining a strong game.

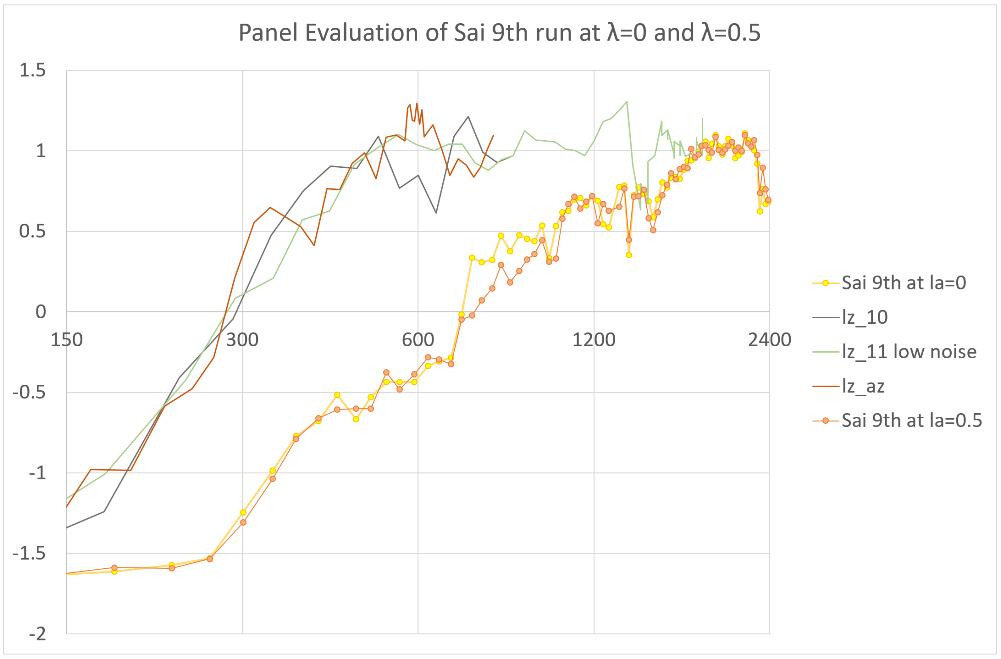


Figure 8: The same nets of the 9th run of SAI with agents $\lambda = 0$ and with $\lambda = 0.5$ are evaluated with matches against the panel.

5 Conclusions

We introduced SAI, a model incorporating a variable level of bonus points in the traditional neural network and Monte Carlo tree search strategy. We trained several nets on the simplified 7×7 goban, exploring several sets of parameters and settings, and we could obtain nets able to play at almost perfect level. We showed that the estimates of the winning probability of our nets at standard komi are compatible with those of Leela Zero, but we showed that SAI's bonus-dependent estimates provide a deeper understanding of the game situation. SAI also provides an estimate of the current point difference between players.

Due to the limitations of the 7×7 goban, it was not possible to assess whether our model allowed to target higher margins of victory, but the results were promising.

We posit that implementing SAI in a distributed effort could produce a software tool able to provide a deeper understanding of the potential of each position, to target high margins of victory and play with handicap in the 9×9 and full 19×19 board, thus providing an opponent for human players which never plays sub-optimal moves, and ultimately progressing towards the optimal game.

References

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

[2] 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.

[3] Sylvain Gelly, Yizao Wang, Rémi Munos, and Olivier Teytaud. Modification of UCT with patterns in Monte-Carlo go. Research Report RR-6062, INRIA, Nov 2006.

[4] Gian-Carlo Pascutto and contributors. Leela Zero, 2018. [Online; accessed 17-August-2018].

[5] 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, 2016.

[6] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, et al. Mastering chess and shogi by self-play with a general reinforcement learning algorithm. *arXiv preprint arXiv:1712.01815*, 2017.

[7] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of Go without human knowledge. *Nature*, 550(7676):354, 2017.

[8] David Silver, Richard Sutton, and Martin Müller. Reinforcement learning of local shape in the game of go. In *IJCAI*, volume 7, pages 1053–1058, 2007.

[9] Ti-Rong Wu, I Wu, Guan-Wun Chen, Ting-han Wei, Tung-Yi Lai, Hung-Chun Wu, Li-Cheng Lan, et al. Multi-Labelled Value Networks for Computer Go. *arXiv preprint arXiv:1705.10701*, 2017.