0.1 Page 3

Each search starts from the root state following the most promising actions, selected using the utility value U(s, a), proportional to $Q(s,a) + \frac{P(s,a)}{1+N(s,a)}$.

0.2 Page 4

Once the self-play game has been finished and the final outcome has become known, every step of the game is added to the training dataset, which is a list of tuples (s_t, π_t, r_t), where s_t is the game state, π_t is the action probabilities calculated from MCTS search sampling and r_t is the games outcome from the perspective of the player at step t.

With this at hand, our training is simple: we sample minibatches from replay buffer of training examples and minimizing the MSE between the value head prediction and actual position value and cross-entropy loss between predicted probabilities and sampled probabilities π.

0.3 Page 11

The final function in the class returns the probability of actions and the action values for game state using the gathered statistics during the MCTS searches. There are two modes of probability calculation, specified by the parameter τ. If it equals to zero, the selection becomes deterministic, as we select the most frequently visited action. In other cases, the distribution given by $\frac{N(s,a)^{\frac{1}{\tau}}}{\sum_k N(s,k)^{\frac{1}{\tau}}}$ is used, which, again, improves exploration.

0.4 Page 12

The function accepts lots of parameters, including MCTS class instance (one or two, as we potentially want to use separate MCTS statistics for different neural networks), optional replay buffer, networks to be used during the game, amount of game steps need to be taken before the τ parameter used for the action probability calculatation will be changed from 1 to 0. Other options are amount of MCTS searchers to perform and other flags.