UCB, Monte Carlo Tree Search, AlphaZero

Milan Straka

December 05, 2022
Revisiting multi-armed bandits with $\varepsilon$-greedy exploration, we note that using the same epsilon for all actions in $\varepsilon$-greedy method seems inefficient.

One possible improvement is to select action according to upper confidence bound (instead of choosing a random action with probability $\varepsilon$):

$$A_{t+1} \overset{\text{def}}{=} \arg \max_a \left[ Q_t(a) + c \sqrt{\ln t \over N_t(a)} \right],$$

where:

- $t$ is the number of times any action has been taken;
- $N_t(a)$ is the number of times the action $a$ has been taken;
- if $N_t(a) = 0$, the right expression is frequently assumed to have a value of $\infty$.

The updates are then performed as before (e.g., using averaging or fixed learning rate $\alpha$).
Motivation Behind Upper Confidence Bound

Actions with little average reward are probably selected too often. Instead of simple \( \varepsilon \)-greedy approach, we might try selecting an action as little as possible, but still enough to converge.

Assuming that random variables \( X_i \) are bounded by \([0, 1]\) and denoting \( \bar{X} = \frac{1}{N} \sum_{i=1}^{N} X_i \), (Chernoff-)Hoeffding's inequality states that

\[
P(\bar{X} \leq \mathbb{E}[\bar{X}] - \delta) \leq e^{-2N\delta^2}.
\]

Our goal is to choose \( \delta \) such that for every action,\n
\[
P(Q_t(a) \leq q^*(a) - \delta) \leq \left(\frac{1}{t}\right)^{\alpha}.
\]

We can fulfil the required inequality if \( e^{-2N_t(a)\delta^2} \leq \left(\frac{1}{t}\right)^{\alpha} \), which yields

\[
\delta \geq \frac{\alpha}{2} \cdot \sqrt{\frac{\ln t}{N_t(a)}}.
\]
Asymptotical Optimality of UCB

We define \textit{regret} as the difference of maximum of what we could get (i.e., repeatedly using the action with maximum expectation) and what a strategy yields, i.e.,

$$\text{regret}_N \overset{\text{def}}{=} N \max_a q_*(a) - \sum_{i=1}^N \mathbb{E}[R_i].$$

It can be shown that regret of UCB is asymptotically optimal, see Lai and Robbins (1985), \textit{Asymptotically Efficient Adaptive Allocation Rules}; or the Chapter 8 of the 2018 \textit{Bandit Algorithms Book} available online at \url{https://banditalgs.com/}.
Upper Confidence Bound Multi-armed Bandits Results

![Graph showing UCB and ε-greedy performance](image)

UCB $c = 2$

ε-greedy $ε = 0.1$

Average reward

Steps

Figure 2.4 of “Reinforcement Learning: An Introduction, Second Edition”.
Multi-armed Bandits Comparison

Average reward over first 1000 steps

$\epsilon$-greedy

gradient bandit

$\text{UCB}$

greedy with optimistic initialization $\alpha = 0.1$

Figure 2.6 of "Reinforcement Learning: An Introduction, Second Edition".
Monte Carlo Tree Search

Repeated X times

Selection → Expansion → Simulation → Backpropagation

The selection function is applied recursively until a leaf node is reached.

One or more nodes are created.

One simulated game is played.

The result of this game is backpropagated in the tree.

Figure 1 of “Monte-Carlo Tree Search: A New Framework for Game AI” by Guillaume Chaslot et al.
Monte Carlo Tree Search – Selection

When selecting a child to visit, we apply the **UCT** (Upper Confidence Bounds applied for Trees) method, which chooses an action according to

\[
a^* = \arg \max_a \left[ Q(s, a) + c \sqrt{\frac{\ln N(s)}{N(s, a)}} \right],
\]

where:

- \( Q(s, a) \) denotes the average result of playing action \( a \) in state \( s \) in the simulations so far,
- \( N(s) \) is the number of times the state \( s \) has been visited in previous iterations,
- \( N(s, a) \) is the number of times the action \( a \) has been sampled in state \( s \).

The \( Q(s, a) \) values are often normalized to the \([0, 1]\) range, and the constant \( c \) is game dependent, but some papers propose to try out \( \sqrt{2} \) first.
Monte Carlo Tree Search

The MCTS is by nature asymmetrical, visiting promising actions more often.

Expansion
Once a chosen action does not correspond to a node in the search tree, a new leaf is added to the game tree.

Simulation
The simulation phase is the “Monte Carlo” part of the algorithm, and is heavily game-specific. It reaches a terminal state and produces final game outcome.

Backpropagation
The obtained outcome updates the $Q(s, a)$ values in the nodes along the path from the expanded node to the root of the search tree.
Alpha* Gradual Development

- **AlphaGo**
  - Mar 2016 – beat 9-dan professional player Lee Sedol

- **AlphaGo Master** – Dec 2016
  - beat 60 professionals, beat Ke Jie in May 2017

- **AlphaGo Zero** – 2017
  - trained only using self-play
  - surpassed all previous version after 40 days of training

- **AlphaZero** – Dec 2017 (Dec 2018 in Nature)
  - self-play only, defeated AlphaGo Zero after 30 hours of training
  - impressive chess and shogi performance after 9h and 12h, respectively

- **MuZero** – Dec 2020 in Nature
  - extends AlphaZero by a *trained model*
  - later extended into Sampled MuZero capable of handling continuous actions

- **AlphaTensor** – Oct 2022 in Nature
  - automatic discovery of faster algorithms for matrix multiplication
On 7 December 2018, the AlphaZero paper came out in Science journal. It demonstrates learning chess, shogi and go, *tabula rasa* – without any domain-specific human knowledge or data, only using self-play. The evaluation is performed against strongest programs available.

Figure 2 of "A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play" by David Silver et al.
AlphaZero uses a neural network predicting \((p(s), v(s)) \leftarrow f(s; \theta)\) for a given state \(s\), where:

- \(p(s)\) is a vector of move probabilities, and
- \(v(s)\) is expected outcome of the game in range \([-1, 1]\).

Instead of the usual alpha-beta search used by classical game playing programs, AlphaZero uses Monte Carlo Tree Search (MCTS).

By a sequence of simulated self-play games, the search can improve the estimate of \(p\) and \(v\), and can be considered a powerful policy improvement operator – given a network \(f\) predicting policy \(p\) and value estimate \(v\), MCTS produces a more accurate policy \(\pi\) and better value estimate \(w\) for a given state \(s\):

\[(\pi(s), w(s)) \leftarrow \text{MCTS}(p(s), v(s), f)\] for \((p(s), v(s)) = f(s; \theta)\).
AlphaZero – Overview

The network is trained from self-play games.

A game is played by repeatedly running MCTS from a state \( s_t \) and choosing a move \( a_t \sim \pi_t \), until a terminal position \( s_T \) is encountered, which is then scored according to game rules as \( z \in \{-1, 0, 1\} \). (Note that the range \([0, 1]\) is also often used, for example in MuZero.)

Finally, the network parameters are trained to minimize the error between the predicted outcome \( v \) and the simulated outcome \( z \), and maximize the similarity of the policy vector \( \pi \) and the search probabilities \( \pi \) (in other words, we want to find a fixed point of the MCTS):

\[
\mathcal{L} \overset{\text{def}}{=} (z - v)^2 - \pi^T \log p + c\|\theta\|^2.
\]

The loss is a combination of:

- a mean squared error for the value functions;
- a crossentropy/KL divergence for the action distribution;
- \( L^2 \)-regularization.
MCTS keeps a tree of currently explored states from a fixed root state. Each node corresponds to a game state and to every non-root node we got by performing an action $a$ from the parent state. Each state-action pair $(s, a)$ stores the following set of statistics:

- visit count $N(s, a)$,
- total action-value $W(s, a)$,
- mean action value $Q(s, a) \equiv W(s, a)/N(s, a)$, which is usually not stored explicitly,
- prior probability $P(s, a)$ of selecting the action $a$ in the state $s$. 

$$Q(s, a) = \frac{W(s, a)}{N(s, a)}$$
Each simulation starts in the root node and finishes in a leaf node $s_L$. In a state $s_t$, an action is selected using a variant of PUCT algorithm as

$$a_t = \arg\max_a \left( Q(s_t, a) + U(s_t, a) \right),$$

where

$$U(s, a) \overset{\text{def}}{=} C(s)P(s, a) \frac{\sqrt{N(s)}}{1 + N(s, a)},$$

with $C(s) = \log \left( \frac{1 + N(s) + c_{\text{base}}}{c_{\text{base}}} \right) + c_{\text{init}}$ being slightly time-increasing exploration rate.

The paper uses $c_{\text{init}} = 1.25$, $c_{\text{base}} = 19652$ without any (public) supporting experiments.

The reason for the modification of the UCB formula was never discussed in any of the AlphaGo/AlphaZero/MuZero papers and is not obvious. However, in June 2020 a paper discussing the properties of this modified formula was published.
Additionally, exploration in the root state \( s_{\text{root}} \) is supported by including a random sample from Dirichlet distribution,

\[
P(s_{\text{root}}, a) = (1 - \varepsilon)p_a + \varepsilon \text{Dir}(\alpha),
\]

with \( \varepsilon = 0.25 \) and \( \alpha = 0.3, 0.15, 0.03 \) for chess, shogi and go, respectively.

Note that using \( \alpha < 1 \) makes the Dirichlet noise non-uniform, with a smaller number of actions with high probability.

The Dirichlet distribution can be seen as a limit of the Pólya’s urn scheme, where in each step we sample from a bowl of balls (with initial counts \( \alpha \)) and return an additional ball of the same color to the bowl.

To sample from a symmetric Dirichlet distribution, we can:

- sample \( x_i \) from the Gamma distribution \( x_i \sim \text{Gamma}(\alpha) \),
- normalize the sampled values to sum to one, \( p_i = \frac{x_i}{\sum_j x_j} \).
When reaching a leaf node $s_L$, we:

- evaluate it by the network, generating $(p, v)$,
- add all its children with $N = W = 0$ and the prior probability $p$,
- in the backward pass for all $t \leq L$, we update the statistics in nodes by performing
  - $N(s_t, a_t) \leftarrow N(s_t, a_t) + 1$, and
  - $W(s_t, a_t) \leftarrow W(s_t, a_t) \pm v$, depending on the player on turn.

\[ N(s_t, a_t) = \max \{ N(s_t, a_t) \pm v \} \]

\[ W(s_t, a_t) = \max \{ W(s_t, a_t) \pm v \} \]

Figure 2 of “Mastering the game of Go without human knowledge” by David Silver et al.
The Monte Carlo Tree Search runs usually several hundreds simulations in a single tree. The result is a distribution proportional to exponentiated visit counts $N(s_{\text{root}}, a)^{\frac{1}{\tau}}$ using a temperature $\tau$ ($\tau = 1$ is mostly used), together with the predicted value function.

The next move is chosen as either:

- proportional to visit counts $N(s_{\text{root}}, \cdot)^{\frac{1}{\tau}}$:
  
  $$\pi_{\text{root}}(a) \propto N(s_{\text{root}}, a)^{\frac{1}{\tau}},$$

- deterministically as the most visited action
  
  $$\pi_{\text{root}} = \arg\max_a N(s_{\text{root}}, a).$$

During self-play, the stochastic policy is used for the first 30 moves of the game, while the deterministic is used for the rest of the moves. (This does not affect the internal MCTS search, there we always sample according to PUCT rule.)
Visualization of the 10 most visited states in a MCTS with a given number of simulations. The displayed numbers are predicted value functions from the white's perspective, scaled to [0, 100] range. The border thickness is proportional to a node visit count.

Figure 4 of "A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play" by David Silver et al.
AlphaZero – Network Architecture

The network processes game-specific input, which consists of a history of 8 board positions encoded by several $N \times N$ planes, and some number of constant-valued inputs.

Output is considered to be a categorical distribution of possible moves. For chess and shogi, for each piece we consider all possible moves (56 queen moves, 8 knight moves and 9 underpromotions for chess).

The input is processed by:

- initial convolution block with CNN with 256 $3 \times 3$ kernels with stride 1, batch normalization, and ReLU activation,
- 19 residual blocks, each consisting of two CNN with 256 $3 \times 3$ kernels with stride 1, batch normalization, ReLU activation, and a residual connection around them,
- policy head, which applies another CNN with batch normalization, followed by a convolution with $73/139$ filters for chess/shogi, or a linear layer of size 362 for go,
- value head, which applies another CNN with one $1 \times 1$ kernel with stride 1, followed by a ReLU layer of size 256, and a final $\tanh$ layer of size 1.
## AlphaZero – Network Inputs

<table>
<thead>
<tr>
<th>Feature</th>
<th>Go Planes</th>
<th>Chess Planes</th>
<th>Shogi Planes</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1 stone</td>
<td>1</td>
<td>6</td>
<td>14</td>
</tr>
<tr>
<td>P2 stone</td>
<td>1</td>
<td>6</td>
<td>14</td>
</tr>
<tr>
<td>Repeatations</td>
<td>2</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>Colour</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Total move</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Castling</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>No-progress</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

| Total         | 17        | 119          | 362          |

*Table S1 of “A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play” by David Silver et al.*
## AlphaZero – Network Outputs

<table>
<thead>
<tr>
<th>Feature</th>
<th>Chess</th>
<th>Shogi</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Planes</td>
<td>Planes</td>
</tr>
<tr>
<td>Queen moves</td>
<td>56</td>
<td>64</td>
</tr>
<tr>
<td>Knight moves</td>
<td>8</td>
<td>2</td>
</tr>
<tr>
<td>Underpromotions</td>
<td>9</td>
<td>64</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Promoting queen moves</td>
<td></td>
<td>2</td>
</tr>
<tr>
<td>Promoting knight moves</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Drop</td>
<td></td>
<td>7</td>
</tr>
<tr>
<td>Total</td>
<td>73</td>
<td>139</td>
</tr>
</tbody>
</table>

*Table S2 of “A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play” by David Silver et al.*
Training is performed by running self-play games of the network with itself. Each MCTS uses 800 simulations. A replay buffer of one million most recent games is kept.

During training, 5000 first-generation TPUs are used to generate self-play games. Simultaneously, network is trained using SGD with momentum of 0.9 on batches of size 4096, utilizing 16 second-generation TPUs. Training takes approximately 9 hours for chess, 12 hours for shogi and 13 days for go.
Figure 1 of "A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play" by David Silver et al.

Table S3 of "A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play" by David Silver et al.

<table>
<thead>
<tr>
<th></th>
<th>Chess</th>
<th>Shogi</th>
<th>Go</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mini-batches</td>
<td>700k</td>
<td>700k</td>
<td>700k</td>
</tr>
<tr>
<td>Training Time</td>
<td>9h</td>
<td>12h</td>
<td>13d</td>
</tr>
<tr>
<td>Training Games</td>
<td>44 million</td>
<td>24 million</td>
<td>140 million</td>
</tr>
<tr>
<td>Thinking Time</td>
<td>800 sims</td>
<td>800 sims</td>
<td>800 sims</td>
</tr>
<tr>
<td></td>
<td>~ 40 ms</td>
<td>~ 80 ms</td>
<td>~ 200 ms</td>
</tr>
</tbody>
</table>

Table 53 of "A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play" by David Silver et al.
According to the authors, training is highly repeatable.

Figure S3 of “A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play” by David Silver et al.
In the original AlphaGo Zero, symmetries (8 in total, using rotations and reflections) were explicitly utilized, by

- randomly sampling a symmetry during training,
- randomly sampling a symmetry during MCTS evaluation.

However, AlphaZero does not utilize symmetries in any way (because chess and shogi do not have them).

Figure S1 of "A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play" by David Silver et al.
During inference, AlphaZero utilizes much less evaluations than classical game playing programs.

<table>
<thead>
<tr>
<th>Program</th>
<th>Chess</th>
<th>Shogi</th>
<th>Go</th>
</tr>
</thead>
<tbody>
<tr>
<td>AlphaZero</td>
<td>63k (13k)</td>
<td>58k (12k)</td>
<td>16k (0.6k)</td>
</tr>
<tr>
<td>Stockfish</td>
<td>58,100k (24,000k)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Elmo</td>
<td></td>
<td>25,100k (4,600k)</td>
<td></td>
</tr>
<tr>
<td>AlphaZero</td>
<td>1.5 GFlop</td>
<td>1.9 GFlop</td>
<td>8.5 GFlop</td>
</tr>
</tbody>
</table>

Table S4 of "A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play" by David Silver et al.
## AlphaZero – Ablations

<table>
<thead>
<tr>
<th>Fig.</th>
<th>Match</th>
<th>Start Position</th>
<th>AlphaZero</th>
<th>Opponent</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>Book</td>
<td>Main</td>
</tr>
<tr>
<td>2A</td>
<td>Main</td>
<td>Initial Board</td>
<td>No</td>
<td>3h</td>
</tr>
<tr>
<td>2B</td>
<td>1/100 time</td>
<td>Initial Board</td>
<td>No</td>
<td>108s</td>
</tr>
<tr>
<td>2B</td>
<td>1/30 time</td>
<td>Initial Board</td>
<td>No</td>
<td>6min</td>
</tr>
<tr>
<td>2B</td>
<td>1/10 time</td>
<td>Initial Board</td>
<td>No</td>
<td>18min</td>
</tr>
<tr>
<td>2B</td>
<td>1/3 time</td>
<td>Initial Board</td>
<td>No</td>
<td>1h</td>
</tr>
<tr>
<td>2C</td>
<td>latest Stockfish</td>
<td>Initial Board</td>
<td>No</td>
<td>3h</td>
</tr>
<tr>
<td>2C</td>
<td>Opening Book</td>
<td>Initial Board</td>
<td>No</td>
<td>3h</td>
</tr>
<tr>
<td>2D</td>
<td>Human Openings</td>
<td>Figure 3A</td>
<td>No</td>
<td>3h</td>
</tr>
<tr>
<td>2D</td>
<td>TCEC Openings</td>
<td>Figure S4</td>
<td>No</td>
<td>3h</td>
</tr>
</tbody>
</table>

Table S8 of "A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play" by David Silver et al.

<table>
<thead>
<tr>
<th>Fig.</th>
<th>Match</th>
<th>Start Position</th>
<th>AlphaZero</th>
<th>Opponent</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>Book</td>
<td>Main</td>
</tr>
<tr>
<td>2A</td>
<td>Main</td>
<td>Initial Board</td>
<td>No</td>
<td>3h</td>
</tr>
<tr>
<td>2B</td>
<td>1/100 time</td>
<td>Initial Board</td>
<td>No</td>
<td>108s</td>
</tr>
<tr>
<td>2B</td>
<td>1/30 time</td>
<td>Initial Board</td>
<td>No</td>
<td>6min</td>
</tr>
<tr>
<td>2B</td>
<td>1/10 time</td>
<td>Initial Board</td>
<td>No</td>
<td>18min</td>
</tr>
<tr>
<td>2B</td>
<td>1/3 time</td>
<td>Initial Board</td>
<td>No</td>
<td>1h</td>
</tr>
<tr>
<td>2C</td>
<td>Aperyqhapaq</td>
<td>Initial Board</td>
<td>No</td>
<td>3h</td>
</tr>
<tr>
<td>2C</td>
<td>CSA time control</td>
<td>Initial Board</td>
<td>No</td>
<td>10min</td>
</tr>
<tr>
<td>2D</td>
<td>Human Openings</td>
<td>Figure 3B</td>
<td>No</td>
<td>3h</td>
</tr>
</tbody>
</table>

Table S9 of "A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play" by David Silver et al.
AlphaZero – Ablations

Figure 2 of "A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play" by David Silver et al.
AlphaZero – Ablations

Figure 4 of "Mastering the game of Go without human knowledge" by David Silver et al.
AlphaZero – Preferred Chess Openings

Figure S2 of "A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play" by David Silver et al.