1 System Details

1.1 System Owner

This may be the designer deploying the system, a larger agency or body, or some combination of the two. The entity completing the report should also be indicated.

This system was developed by the DeepMind core Reinforcement Learning Team members. More information about AlphaGo's development can be found at the project website (https://deepmind.com/research/case-studies/alphago-the-story-so-far) as well as DeepMind's GitHub repository

1.2 Dates

The known or intended timespan over which this reward function & optimization is active.

Development of AlphaGo began about two years prior to the matches against Lee Sedol in spring 2016, shortly after DeepMind's acquisition by Google [Ribeiro(2016)]. Development of AlphaZero, based entirely on self-play, followed AlphaGo and was completed prior to October 2017. Development of MuZero, also based on self-play, followed AlphaZero and was first described in a preliminary paper in 2019 [Schrittwieser et al.(2020)].

1.3 Feedback & Communication

Contact information for the designer, team, or larger agency responsible for system deployment.

Any correspondence should be directed to `press@deepmind.com`.

1.4 Other Resources

Where can users or stakeholders find more information about this system? Is this system based on one or more research papers?

There is little additional disclosed information.

2 Optimization Intent

2.1 Goal of Reinforcement

A statement of system scope and purpose, including the planning horizon and justification of a data-driven approach to policy design (e.g. the use of reinforcement learning or repeated retraining). This justification should contrast with alternative approaches, like static models and hand-designed policies. What is there to gain with the chosen approach?

Go, and general game-playing at a human level, was long defined as one of the "grand challenges" of AI. For AlphaGo, the use of reinforcement to learn both the policy and value networks beyond the abilities of a human expert.

For AlphaZero, the sole use of reinforcement learning without any human data was important validation of its potential as a more general learning procedure [Silver et al.(2017)]. The algorithm additionally incorporated lookahead search (Monte Carlo Tree Search) inside the training loop.

For MuZero, the use of model-based reinforcement learning without any prior knowledge of the game dynamics was further indication of RL's potential to develop planning capabilities in more challenging or complex domains [Schrittwieser et al.(2020)]. The learned model performed well in both classic game environments (Go, chess, shogi) as well as canonical video game environments (57 distinct Atari games).

2.2 Defined Performance Metrics

A list of "performance metrics" included explicitly in the reward signal, the criteria for why these metrics were chosen, and from where these criteria were drawn (e.g. government agencies, domain precedent, GitHub repositories, toy environments). Performance metrics that are used by the designer to tune the system, but not explicitly included in the reward signal should also be reported here.

As with most game-playing systems, the performance metric is defined as a win rate among games. In other games, score is used, but in

one-versus-one games win rate is the only direct metric. To better capture the uncertainty of playing varying opponents, this win rate is translated into a running Elo rating system.

2.3 Oversight Metrics

Are there any additional metrics not included in the reward signal but relevant for vendor or system oversight (e.g. performance differences across demographic groups)? Why aren't they part of the reward signal, and why must they be monitored?

Some other performance metrics are not included in the specification, but are monitored for the purpose of evaluating system effects on the domain:

- Absolute opponents' world rankings - following their public games, versions of AlphaGo and AlphaZero were considered to possibly improve the skill levels of expert human opponents, as measured by those players' absolute world ranking. If humans played better after playing AlphaGo, this was to be seen as a positive effect of the system's influence on the game of Go. Fan Hui, following his games against AlphaGo, claimed it made him a better played and accredits his world ranking jump from 600 to 300 in three months to training against it [Murgia(2016)].

- Qualitative changes in playstyle - following their public games, versions of AlphaGo were considered to possibly influence the playstyle of expert human opponents, as interpreted by the wider community of expert players. If expert humans played differently, more creatively or unpredictably, or expressed surprise after AlphaGo's public performances, this was to be seen as a positive effective of the system's influence on the game in question. Garry Kasparov, following his observation of AlphaZero play, was impressed that it appeared to be "a very sharp and attacking player" given that almost all computer programs have a conservative playstyle [Ingle(2018)]. While not integral in any way for system performance, AlphaGo's performance and playstyle have had a noticeable impact on the strategies of expert human players.

2.4 Known Failure Modes

A description of any prior known instances of "reward hacking" or model misalignment in the domain at stake, and description of how the current system avoids this.

Monte Carlo search limitations. In the fourth match (of five) against Lee Sedol in spring 2016, the system failed to recognize move 78 by Sedol. The Monte Carlo search tree, which was designed to prune sequences of moves considered to be irrelevant for maximizing odds of victory, failed to recognize this move. This is because that move was so far outside the distribution of prior game situations that the AlphaGo system failed to accurately calculate its significance for determining the odds of victory [Ormerod(2016)]. The result was a sequences of moves 79-87 by AlphaGo that were considered poor by expert human players, a function of Monte Carlo's myopic look-ahead search following move 78. AlphaGo subsequently conceded the game at move 178, at which point it evaluated its own odds of victory as lower than 20 percent [Metz(2016)].

3 Institutional Interface

3.1 Deployment Agency

What other agency or controlling entity roles, if any, are intended to be subsumed by the system? How may these roles change following system deployment?

The AlphaGo system was developed by DeepMind. This version played against Fan Hui in 5 matches held at DeepMind headquarters in October 2015. These matches were secret and not revealed until the publication of results in January 2016 [Silver et al.(2016)]. A later version of the same system, AlphaGo Lee, played Lee Sedol in March 2016 in 5 matches in Seoul, South Korea. This match was overseen by the

Korea Baduk Association. A yet more sophisticated version of the same system, AlphaGo Master, played against Ke Jie at the Future of Go Summit in Wuzhen, China in May 2017. An earlier version of AlphaGo Master, dubbed Master, had already won 60 straight online games against top pro players, including against Ke Jie [Silver and Hassabis(2017)]. This version was awarded a professional 9-dan title by the Chinese Weiqi Assocation.

3.2 Stakeholders

What other interests are implicated in the design specification or system deployment, beyond the designer? What role will these interests play in subsequent report documentation? What other entities, if any, does the deployed system interface with whose interests are not intended to be in scope?

Compared to other prominent automated game-playing systems like Stockfish (open-source chess engine) or CrazyStone (offline Go engine based on deep learning), versions of AlphaGo perform much much better with additional computational power. The versions of AlphaGo that played against Fan Hu, Lee Sedol, and Ke Jie all made use of distributed CPUs and GPUs. AlphaGo Zero, based entirely on reinforcement learning and self-play, became stronger than AlphaGo Lee after 3 days and stronger than AlphaGo Master after 21 days. Its self-play training time was stopped after 40 days, at which point it was stronger than any known Go player (human or program) as measured by Elo rating in October 2017 [Silver and Hassabis(2017)].

AlphaZero, in its initial chess games against Stockfish, was criticized by expert human chess players has having unfair computational advantages over the opponent [Doggers(2018)].

MuZero's learning has been made more efficient in follow-up work, dubbed EfficientZero [Ye et al.(2021)].

3.3 Explainability & Transparency

Does the system offer explanations of its decisions or actions? What is the purpose of these explanations? To what extent is the policy transparent, i.e. can decisions or actions be understood in terms of meaningful intermediate quantities?

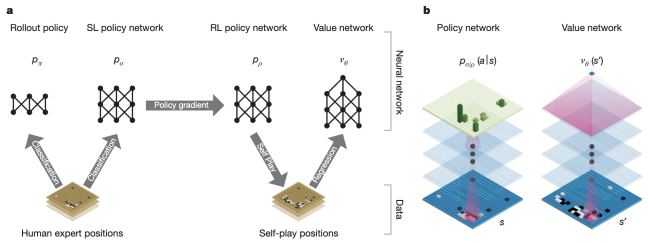

Figure 1: The AlphaGo game playing system architecture.

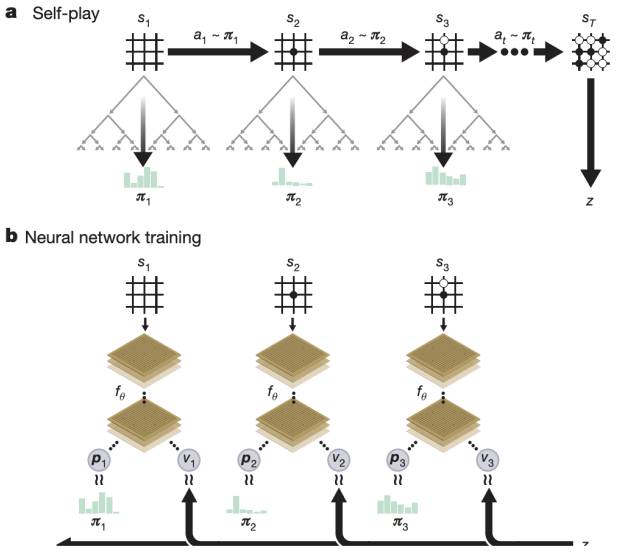

Figure 2: The AlphaZero game playing system architecture.

The MuZero system offers few tools for transparency in its current form. While the learning process develops a structured model for the game dynamics, it is not done in a way that is accessible by engineers or external parties.

3.4 Recourse

Can stakeholders or users contest the decisions or actions of the system? What processes, technical or otherwise, are in place to handle this?

N/A

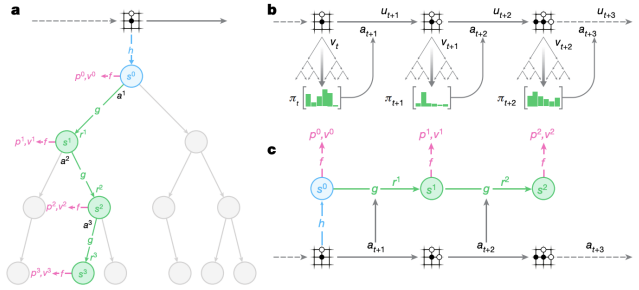

Figure 3: The MuZero general game playing system.

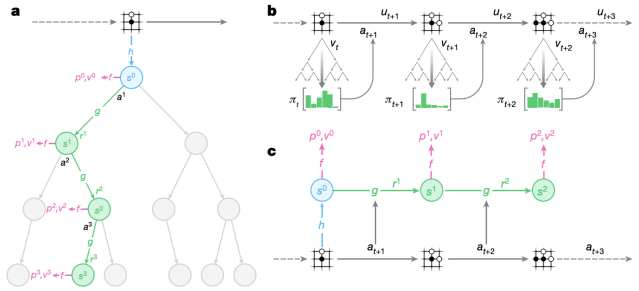

Figure 4: The MuZero general game playing system.

4 Implementation

4.1 Reward Details

How was the reward function engineered? Is it based on a well-defined metric? Is it tuned to represent a specific behavior? Are multiple terms scaled to make one central loss, and how was the scaling decided?

The reward function is entirely prescribed as win rate, and the resulting Elo rating. An important sub-component that will be referenced later is the value function estimating game state. This is an internal representation of reward central to training and evaluation.

4.2 Environment Details

Description of states, observations, and actions with reference to planning horizon and hypothesized dynamics/impact. What dynamics are brought into the scope of the optimization via feedback? Which dynamics are left external to the system, as drift? Have there been any observed gaps between conceptualization and resultant dynamics?

The original environment is the full game of Go which is constrained by finite rules, but other games with visual states were added.

4.3 Measurement Details

How are the components of the reward and observations measured? Are measurement techniques consistent across time and data sources? Under what conditions are measurements valid and correct? What biases might arise during the measurement process?

The measurements differ across games from the full gameboard to a visual rendering of the world. Extracting information from pixels is substantially less efficient than directly from the game state.

4.4 Algorithmic Details

The key points on the specific algorithm(s) used for learning and planning. This includes the form of the policy (e.g. neural network, optimization problem), the class of learning algorithm (e.g. model-based RL, off-policy RL, repeated retraining), the form of any intermediate model (e.g. of the value function, dynamics function, reward function), technical infrastructure, and any other considerations necessary for implementing the system. Is the algorithm publicly documented and is code publicly available? Have different algorithms been used or tried to accomplish the same goal?

The key algorithm feature is the use of Monte Carlo Tree Search (MCTS). MCTS is used to search over board states (by planning over actions) and parses the value representation. The value function is represented by a deep neural network mapping from game state to value.

The second crucial element to training is self play. Here gameplaying agents evaluate their performance versus past training snapshots. This synergistic mechanism is crucial to reaching superhuman performance. In MuZero, and learned model is used to used to improve performance in games without complete information (such as visual states) by constraining the policy optimization. At each turn, the model is used to predict the correct policy, the value

function, and the reward received by the move (in games that have an intermediate score). The model is updated in an end-to-end fashion, so it is included in the same training loop in the agent architecture.

Fully algorithmic details and open source code are not released.

4.5 Data Flow

How is data collected, stored, and used for (re)training? How frequently are various components of the system retrained, and why was this frequency chosen? Could the data exhibit sampling bias, and is this accounted for in the learning algorithm? Is data reweighted, filtered, or discarded? Have data sources changed over time?

Data flow is not well documented, but it relies on Google's distributed training and deployment infrastructure.

4.6 Limitations

Discussion and justification of modeling choices arising from computational, statistical, and measurement limitations. How might (or how have) improvements in computational power and data collection change(d) these considerations and impact(ed) system behavior?

4.7 Engineering Tricks

RL systems are known to be sensitive to implementation tricks that are key to performance. Are there any design elements that have a surprisingly strong impact on performance? For example, state-action normalization, hard-coded curricula, model-initialization, loss bounds, or more?

Not documented.

5 Evaluation

5.1 Evaluation Environment

How is the system evaluated (and if applicable, trained) prior to deployment (e.g. using simulation, static datasets, etc.)? Exhaustive details of the offline evaluation environment should be provided. For simulation, details should include description or external reference to the underlying model, ranges of parameters, etc. For evaluation on static datasets, considering referring to associated documentation (e.g. Datasheets *[Gebru et al.(2021)]).*

For games, the simulator is reality so evaluation is matched to training.

5.2 Offline Evaluations

Present and discuss the results of offline evaluation. For static evaluation, consider referring to associated documentation (e.g. Model Cards *[Mitchell et al.(2019)]). If applicable, compare the behaviors arising from counterfactual specifications (e.g. of states, observations, actions).*

Multiple internal evaluations of the agent were performed prior to high-profile, public matches with the worlds best players.

5.3 Evaluation Validity

To what extent is it reasonable to draw conclusions about the behavior of the deployed system based on presented offline evaluations? What is the current state of understanding of the online performance of the system? If the system has been deployed, were any unexpected behaviors observed?

5.4 Performance standards

What standards of performance and safety is the system required to meet? Where do these standards come from? How is the system verified to meet these standards?

N/A.

6 System Maintenance

6.1 Reporting Cadence

The intended timeframe for revisiting the reward report. How was this decision reached and motivated?

While this system is evaluated in closed-world games, updates are not anticipated.

6.2 Update Triggers

Specific events (projected or historic) significant enough to warrant revisiting this report, beyond the cadence outlined above. Example triggers include a defined stakeholder group empowered to demand a system audit, or a specific metric (either of performance or oversight) that falls outside a defined threshold of critical safety.

This report will be revisited upon release of each new game-playing AI from DeepMind.

6.3 Changelog

Descriptions of updates and lessons learned from observing and maintaining the deployed system. This includes when the updates were made and what motivated them in light of previous reports. The changelog comprises the central difference between reward reports and other forms of machine learning documentation, as it directly reflects their intrinsically dynamic nature.

N/A (v1)

References

[Doggers(2018)] Peter Doggers. 2018. AlphaZero Chess: Reactions From Top GMs, Stockfish Author. https://www.chess.com/news/view/alphazero-reactions-from-top-gms-stockfish-author. [Online; accessed 8-January-2022].

[Gebru et al.(2021)] Timnit Gebru, Jamie Morgenstern, Briana Vecchione, Jennifer Wortman Vaughan, Hanna Wallach, Hal Daumé Iii, and Kate Crawford. 2021. Datasheets for datasets. *Commun. ACM* 64, 12 (2021), 86–92.

[Ingle(2018)] Sean Ingle. 2018. 'Creative' AlphaZero leads way for chess computers and, maybe, science. https://www.theguardian.com/sport/2018/dec/11/creative-alphazero-leads-way-chess-computers-science. [Online; accessed 8-January-2022].

[Metz(2016)] Cade Metz. 2016. Go Grandmaster Lee Sedol Grabs Consolation Win Against Google's AI. https://www.wired.com/2016/03/go-grandmaster-lee-sedol-grabs-consolatio [Online; accessed 8-January-2022].

[Mitchell et al.(2019)] Margaret Mitchell, Simone Wu, Andrew Zaldivar, Parker Barnes, Lucy Vasserman, Ben Hutchinson, Elena Spitzer, Inioluwa Deborah Raji, and Timnit Gebru. 2019. Model cards for model reporting. In *Proceedings of the conference on fairness, accountability, and transparency*. 220–229.

[Murgia(2016)] Madhumita Murgia. 2016. Humans versus robots: How a Google computer beat a world champion at this board game - and what it means for the future. http://s.telegraph.co.uk/graphics/projects/go-google-computer-game/. [Online; accessed 8-January-2022].

[Ormerod(2016)] David Ormerod. 2016. Lee Sedol defeats AlphaGo in masterful comeback – Game 4. https://web.archive.org/web/20161116082508/https://gogameguru.com/lee-sedol-defeats-alphago-masterful-comeb [Online; accessed 8-January-2022].

[Ribeiro(2016)] John Ribeiro. 2016. AlphaGo's unusual moves prove its AI prowess, experts say. https://www.pcworld.com/article/420054/alphagos-unusual-moves-prove-its-ai-prowe html. [Online; accessed 8-January-2022].

[Schrittwieser et al.(2020)] Julian Schrittwieser, Ioannis Antonoglou, Thomas Hubert, Karen Simonyan, Laurent Sifre, Simon Schmitt, Arthur Guez, Edward Lockhart, Demis Hassabis, Thore Graepel, et al. 2020. Mastering atari, go, chess and shogi by planning with a learned model. *Nature* 588, 7839 (2020), 604–609.

[Silver and Hassabis(2017)] David Silver and Demis Hassabis. 2017. AlphaGo Zero: Starting from scratch. https://deepmind.com/blog/article/alphago-zero-starting-scratch. [Online; accessed 8-January-2022].

[Silver et al.(2016)] 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. 2016. Mastering the game of Go with deep neural networks and tree search. *nature* 529, 7587 (2016), 484–489.

[Silver et al.(2017)] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. 2017. Mastering the game of go without human knowledge. *nature* 550, 7676 (2017), 354–359.

[Ye et al.(2021)] Weirui Ye, Shaohuai Liu, Thanard Kurutach, Pieter Abbeel, and Yang Gao. 2021. Mastering atari games with limited data. *Advances in Neural Information Processing Systems* 34 (2021).