Proceedings of the 5th International Conference on Signal Processing and Machine Learning
DOI: 10.54254/2755-2721/108/2025.21192

Overview of Machine Learning Bots Capable of Achieving Top-level or Superhuman Performance in 2D Competitive Games

Xiaoyu Li[1,a,*]

[1]*Shijiazhuang Foreign Language School, Shijiazhuang, China*
a. stormorli@outlook.com
**corresponding author*

Abstract: With recent developments in machine learning, bots have conquered many games that were previously believed to be too difficult for computers. We evaluated 4 machine-learning bots that mastered 4 different games: AlphaGo for Go, Stockfish for chess, AlphaStar for StarCraft, and Juewu for Honor of Kings. We summarized and compared their fundamental algorithms and discovered two potential patterns: an increase in generalizability causes an increase in the amount of computation needed, and an increase in the complexity of the game environment causes a decline in the neural networks' performance. To conclude, we discuss the implications of machine learning game bots on neural networks aimed at handling real-life scenarios and artificial general intelligence (AGI).

Keywords: Machine learning, Supervised learning, Reinforcement learning, Neural network, Heuristic algorithm, Game algorithm

1. Introduction

In 1997, IBM's Deep Blue defeated Garry Kasparov by 3½ to 2½ in a game of chess. Its victory deeply influenced not only chess but also the game industry as a whole, suggesting that computers could defeat top human players in competitive games.

Despite this, humans continued to dominate most competitive games for almost 20 years. In Go, for example, heuristic algorithms significantly increased the playing strength of bots, but the best bots were still weaker than the worst professional players.

Machine learning is the scientific study of algorithms and statistical models that computer systems use to perform a specific task without being explicitly programmed and describes the capacity of systems to learn from problem-specific training data to automate the process of analytical model building and solve associated tasks[1,2]. In the early 2010s, major breakthroughs in machine learning greatly boosted the efficiency of neural networks, allowing them to actually solve complex problems for the first time. The discovery that GPUs' parallel computing capabilities can be utilized to train neural networks greatly increased training efficiency[3,4]. Since then, many research teams have used machine learning to give bots the capability of competing with top human players. In this paper, we review, compare, and contrast 4 machine learning bots developed in the last decade.

DOI: 10.54254/2755-2721/108/2025.21192

1.1. Challenges of creating a powerful game bot

1.1.1. Complex board games

With complex board games, there can be hundreds of choices available at each step. A complete search is mostly out of consideration due to the exponentially growing search time. To reduce this complexity, most bots use decision tree pruning, which reduces the total number of searches needed by removing unfavorable search branches, or heuristic algorithms, which determine a near-optimal solution instead of the optimal solution to reduce computation time. However, both of these algorithms' effectiveness depends on their evaluation algorithm. In a complex board game, it may be possible to devise a reasonably good evaluation algorithm, but such an algorithm is insufficient for competing with top human players.

1.1.2. real-time digital games

The game environment of real-time digital games poses a challenge to bots. Firstly, since the gameplay is real-time, the bot must respond to its environment quickly. Secondly, since the game environment is diverse and can change, the bot must be able to accept different input features. Thirdly, since digital games usually contain a large number of in-game states and actions, the action space for the bot is high-dimensional, making exhaustive search difficult. Last but not least, it was difficult to obtain a comprehensive training set that contains every possible situation in the game.[2]

Traditional bots for real-time digital games usually use predetermined, man-made strategies and responses, making them static, inflexible, and generally weaker than human players in terms of strategy, though they may be able to defeat humans with their faster reflection times. For the purpose of this paper, we will discuss bots that defeat humans not with superior speed but with more robust strategies.

1.2. Machine learning

Machine learning is a subfield of artificial intelligence which enables computers to learn knowledge that is not explicitly programmed. Machine learning can be roughly classified into these types: reinforcement learning, supervised learning, unsupervised learning, and semi-supervised learning. In this overview, reinforcement learning and supervised learning are our main focus.

1.2.1. Reinforcement learning

Reinforcement learning is a form of machine learning that uses samples to optimize behavior and function approximations to describe complex environments. in the process of reinforcement learning, an intelligence performs an action in the environment, which is converted by the environment into a reward and a state representation that is subsequently fed back to the intelligence. It learns through interaction and feedback with the environment. It is a part of the indispensable factors to achieve human-level or super-human AI systems [5].

1.2.2. Supervised learning

Supervised learning is another type of machine learning mostly used to classify inputs into predetermined types. Supervised learning is usually done by training the neural network with a training set consisting of labeled inputs and outputs. Then, the neural network is provided with a test set consisting of new inputs along with their desired outputs. If the neural network fails to produce the correct output, it will modify its model accordingly. This process is repeated multiple times until the neural network has reached a satisfying level of accuracy[6].

DOI: 10.54254/2755-2721/108/2025.21192

1.3. Heuristic algorithms

Although traditional algorithms can return the optimal solution, they have to traverse the entirety of the search tree, thus making them slow when given a large input. Heuristic algorithms aim to provide an acceptable result in a feasible period of time. Heuristic search does not search every possibility but instead uses a heuristic function to approximate the optimal solution and decide which branches to search. Heuristic algorithms always involve a trade-off between optimality, completeness, accuracy, and execution time. Heuristic algorithms may not always produce optimal results, but they can be orders of magnitude faster than traditional algorithms[7].

2. Overview and analysis of ML bots

2.1. AlphaGo

2.1.1. Go

Go is a 2-player game typically played on a 19 by 19 board, though other smaller board sizes exist. There are only two types of stones, black and white, corresponding to the two players. There are a few limitations on where a stone can or cannot be placed. Due to the vast number of choices, Go is considered to be much harder for machines to master than chess[8]. Previous Go programs could only reach a strong amateur player's level of skill.

DeepMind's research team developed a machine learning Go program named AlphaGo[9]. AlphaGo defeated Fan Hui in 2015, becoming the first Go program to defeat a professional Go player in a fair game. In 2016, AlphaGo won 4 to 1 against Lee Sedol, one of the best Go players at the time.

The versions of AlphaGo, which defeated Fan Hui and Lee Sedol, were correspondingly named AlphaGo Fan and AlphaGo Lee. They used Monte Carlo tree search and were trained with supervised learning and reinforcement learning.

These two versions use two neural networks: A policy network, which predicts the probability of actions taken at a game state, and a value network, which evaluates the probability of a player winning from a game state when both players use a certain policy.

2.1.2. Algorithm Design

a) Monte Carlo tree search: Monte Carlo tree search is a heuristic search algorithm that does not traverse the entire tree but instead selects favorable nodes to search using random sampling [10]. Monte Carlo tree search has been used by many Go programs before[11, 12], but none of the previous Go programs were able to defeat professional players because they used traditional Go evaluation. AlphaGo, on the other hand, uses neural networks to guide its search. AlphaGo's search method is as follows.

The root node is the current board state. Each leaf node is a new board state created by a possible move. Each edge of the search tree contains an action value Q (how favorable this move is), a visit count N (how many times this edge has been traversed), and a prior probability P (how likely this move is to be played from the previous state).

The tree is searched using a simulation starting from the root node. Figure 1 shows the search process. At each step, AlphaGo sums the action value Q of the edge with a bonus proportional to $\frac{P}{1+N}$ and selects the node with the highest sum. The bonus is added to encourage exploration. Without this bonus, AlphaGo would repeatedly search the leaf node that was deemed best at the first step and ignore other possible choices.

If a new leaf node is selected, AlphaGo will expand the leaf node by running the policy network once to produce a set of moves and their corresponding probabilities.

DOI: 10.54254/2755-2721/108/2025.21192

At the end of the simulation, the leaf node is evaluated in two ways: First, the value network evaluates the game state. Second, a random rollout is played from the game state using a fast rollout policy to the end of the game. The winner of the rollout is calculated with the reward function r. The value network's evaluation and the rollout result are combined to produce the final action value Q of the leaf node.

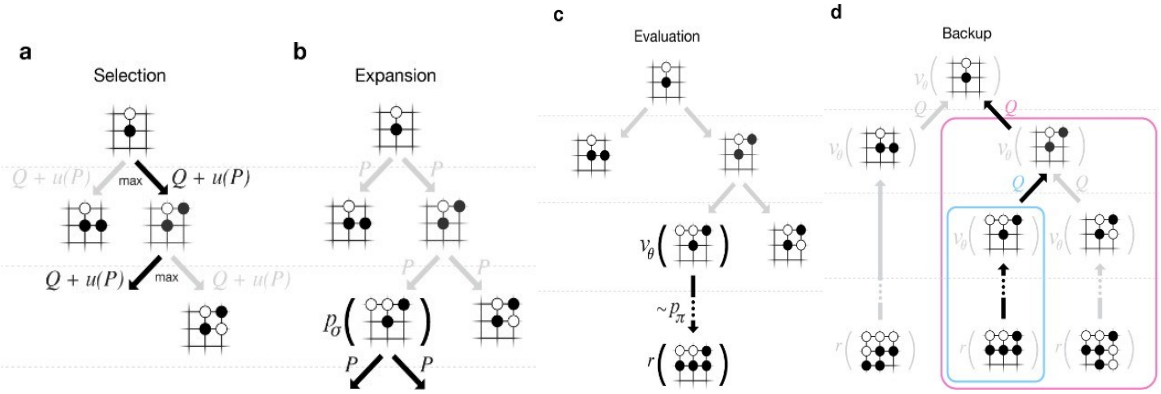

Figure 1: Monte Carlo tree search

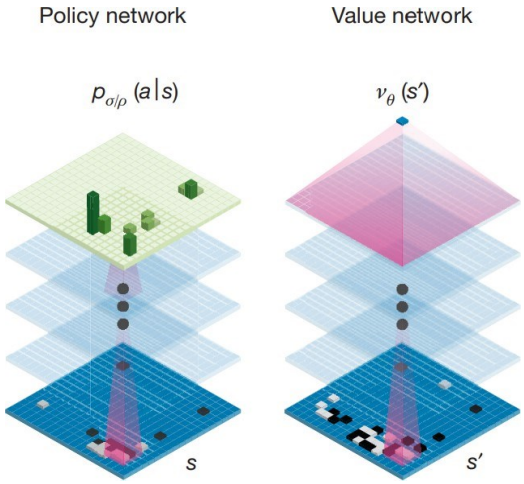

Figure 2: Visualization of AlphaGo's policy and value networks

Each node's action value Q tracks the mean value of the evaluations of all the nodes within its subtree. Therefore, when a leaf node's Q value is updated, the Q value of every node from the leaf node to the root node is also updated.

Finally, AlphaGo selects the edge with the most visit count from the root node as its next move.

b) Policy network: As *shown* in Figure 2, AlphaGo's policy network takes a board state as input and outputs a set of moves and their corresponding possibilities.

The policy network is initially trained with supervised learning to predict expert moves. It is trained using random state-action pairs from 30 million positions of expert games on the KGS Go Server. Testing shows that the SL policy network has an accuracy of 57.0% with all input features, and 55.7% with only board positions and move history as inputs.

A rollout policy network is also trained to be used for evaluation during the Monte Carlo tree search. Its accuracy is only 24.2%, but it only takes 2 microseconds to compute an action, whereas the policy network takes 3 milliseconds. The rollout policy network compensates for its low accuracy with its high speed since it is run multiple times when evaluating every board state.

DOI: 10.54254/2755-2721/108/2025.21192

The second stage of training used reinforcement training. The RL policy network is initialized to be identical to the SL policy network. Then, it repeatedly plays games against previous iterations of itself. At the end of each game, weights at each time step are updated by stochastic gradient ascent based on the result of the game.

c) Value network: As shown in Figure 2, AlphaGo's value network takes a board state as input and outputs a single value denoting how likely it is to win from this board state.

A major problem with training the value network is that game states from the same Go game are highly correlated. Two successive game states often differ by just one stone. If the value network is trained on multiple state-outcome pairs from the same game, overfitting may occur. When the network is trained on the KGS dataset, the mean squared error is only 0.19 on the training set but 0.34 on the test set, indicating that overfitting has indeed occurred.

To prevent this, the state-outcome pairs were selected from games played between the policy network and itself. 30 million pairs were sampled, each from a separate game. When trained with this input, the mean squared error on the training set was 0.226, and the mean squared error on the test set was 0.234. The position evaluation accuracy of the value network was consistently higher than Monte Carlo rollouts using the fast rollout policy. It approached the accuracy of Monte-Carlo rollouts with the RL policy network, despite requiring 15000 times less computation.

d) AlphaGo Zero: The research team of AlphaGo developed an improved version called AlphaGo Zero[13]. AlphaGo Zero com- bines the policy and value networks into one neural network that gives both a set of move probabilities and a value when given a game state as input. It still uses the Monte Carlo tree search.

AlphaGo Zero is not given any samples from human games. It is trained solely using reinforcement learning and self-play. During the training process, it discovered many josekis (opening moves humans optimized over the years) on its own. Later, it invented new openings unseen by humans.

2.1.3. Performance and contributions

AlphaGo played against 4 of the strongest Go programs at the time: Zen, Crazy Stone, Pachi, and Fuego. It won 494 out of 495 games. The estimated elo ratings for AlphaGo Fan, AlphaGo Lee, and AlphaGo Zero are respectively 3144, 3739, and 5185. In comparison, Shin Jin-seo, the current highest elo player, has a rating of 3867.

AlphaGo was not the first Go program to use Monte Carlo tree search, but it was the first to incorporate neural networks into the search process. AlphaGo's success brought widespread attention to the AI community and possibly sped up the development of other ML bots.

AlphaGo's algorithm framework is much more generalizable than traditional bots which focus on one specific game. In 2018, Al- phaGo's research team released AlphaZero[14], a general-purpose bot that can play Go, chess, and shogi. AlphaZero's algorithmic structure is mostly the same as AlphaGo Zero. AlphaZero defeated AlphaGo Zero by 60 to 40 and won against the best chess and shogi programs at the time, demonstrating that ML bots can perform extremely well across a wide range of games.

2.2. Stockfish

2.2.1. Chess

Chess is a two-player game in which two sides play white and black pieces, with 16 pieces of each color, including 8 pawns, 2 knights, 2 bishops, 2 rooks, a queen, and a king. All pieces move on an 8*8 board. Different types of pieces have different rules of movement, and the initial positions of all

DOI: 10.54254/2755-2721/108/2025.21192

pieces are fixed. The object of the game is to kill the opponent's king. Players need to use proper strategies to move, kill, or sacrifice pieces to achieve this target. Computers have performed well in chess since the 20th century. IBM's Deep Blue was able to defeat Garry Kasparov, the best chess player at the time. Deep Blue used alpha-beta pruning and search algorithms to enhance its speed but did not use machine learning at all[15]. Although chess bots do not necessarily need machine learning to beat humans, machine learning can help increase their playing strength even further, as shown in the case of Stockfish.

2.2.2. Algorithm design

Stockfish uses a clean alpha-beta pruning search and a neural network valuation scheme, applying an alpha-beta pruning searching algorithm to find possible scenarios for each turn. A turn taken by a player is used to represent the search depth which is increased gradually by iterative deepening depth-first search(IDFS). Stockfish also applied *forward pruning*, *futility pruning,* and *reduction* to reduce the search space.

Firstly, *forward pruning* is aimed at cutting subgraphs that are unlikely to lead to a good result in the game tree search, including weak moves or clearly unfavorable moves[16].

Secondly, *futility pruning* is used when the current assessment of the player's situation is already very unfavorable to cut out some branches, as these choices are unlikely to change the unfavorable trend of the situation[17].

Thirdly, reduction technologies such as late Move Reduction reduce the depth of some subgraphs instead of cutting the entire subgraphs.

Whenever the search algorithm reaches a node, the heuristic function is used to evaluate the current situation. In Stockfish 12, the coding approach based on chess concepts used prior to version 11 was abandoned in favor of using an Efficiently Updatable Neural Network (NNUE) with about twice the efficiency[18].

a) Alpha-beta pruning search algorithm: Alpha-beta pruning is one of the most fundamental optimization techniques for the Minimax algorithm that helps increase the speed of the game drastically. [19]. In this algorithm, when a node's subgraphs' value is determined not to influence the final decision, any further searching would be unnecessary, and the subgraphs would be cut. Alpha-beta pruning maintains two values, Alpha and Beta. alpha denotes the current best option found by the Max player, while Beta denotes the current best option found by the Min player. During the search process, if the value of a node is less than or equal to Alpha or greater than or equal to Beta, it can be pruned because the value of that node will not affect the final decision. As a result, the search efficiency is increased because the number of nodes to be searched decreases.

b) Efficiently Updatable Neural Network: Efficiently Up-datable Neural Network (NNUE) was added to the Stockfish repo in 2020 and has largely increased the capacity and accuracy[20], which uses a 4-layer neural network as a *valuation* function model and proposes a supervised learning method for training the valuation function. It is designed to limit the number of pieces affected by each move by representing the board by a combination of king and other piece locations. This is done so that incremental updates can be made. The neural network can learn new data samples incrementally instead of retraining the entire network. This saves computational resources and adapts to new data faster. Combined with the fact that it is optimized specifically for the CPU, its computation time is very short, making it easy to combine with other search algorithms and pruning algorithms.

DOI: 10.54254/2755-2721/108/2025.21192

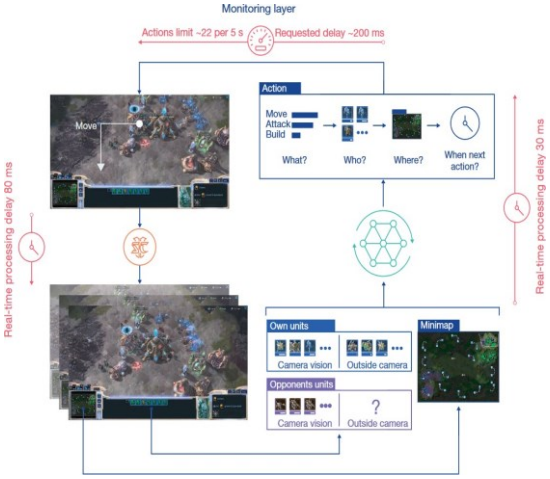

Figure 3: AlphaStar's system design

2.2.3. Performance and contribution

The Top Chess Engine Championship(TCEC) is a computer chess tournament that has been held since 2010, attracting nearly all the world's top chess engines. Stockfish has won multiple TCEC championships and currently has the most championships of any chess program. By 2022, Stock-fish has finished first 13 times, second 7 times, and third once in the 23 seasons of the TCEC. In the computer chess championship(CCC) which has been held since 2018, Stockfish also got the top four spots in all 27 competitions through 2022, getting first place 21 times, second place 4 times, and third and fourth place once each.

Stockfish applied and improved multiple algorithms and demonstrated the extreme power of machine learning in chess. Compared to most other machine learning programs, it requires a very small amount of computational power and can be run on just one single CPU. It can serve as an example of implementing powerful neural networks in portable devices.

2.3. AlphaStar

2.3.1. StarCraft

StarCraft is a real-time strategy game that involves an in-game economy, individual control of hundreds of units, and an incomplete knowledge of the whole game state due to camera view range limits and unexplored areas. The game is challenging because it requires the player to use dynamic strategies and counter-strategies against their opponents.

2.3.2. Algorithm Design

a) General structure: DeepMind's research team developed a StarCraft bot named AlphaStar using machine learning[21]. The general algorithmic structure is shown in Figure 3. Its inputs include the player's units inside the camera view, the player's units outside the camera view, the enemy's units inside the camera view, as well as the minimap. The enemy units outside the camera view are not visible to the player. The outputs include the action to take, the target or targets of said action, the location of the action, and when to take the next action. AlphaStar is given an action cap to prevent it from beating humans by simply performing moves faster. It can perform at most 22 actions in 5 seconds, which makes it slower than humans.

DOI: 10.54254/2755-2721/108/2025.21192

b) Self-play: AlphaStar's agent parameters were initially trained by supervised learning. They were given replays of games played by humans to learn to imitate human strategies.

To further improve AlphaStar's skill level, reinforcement learning via self-play was used. Similar to AlphaGo, AlphaStar would repeatedly play games against itself and adjust the weights of the neural network based on the results.

AlphaStar uses league-based training to ensure the diversity of its opponents. The league consists of 3 types of agents: The main agents, the main exploiter agents, and the league exploiter agents.

The main agents are the agents trained to play against humans in the end. They play against both previous versions of themselves and the other two types of special agents.

The main exploiter agents are designed to find weaknesses in the main agents and thereby make them improve their strategies. Main exploiter agents only play against main agents to better develop strategies specifically aimed at defeating main agents. They're reinitialized once in a while to encourage them to devise different strategies against the main agents.

The league exploiter agents are trained similarly to the main agents but do not play against the main exploiter agents. Their goal is to find the weaknesses of the entire league instead of just the main agents.

2.3.3. Performance and contribution

The final version of Al-phaStar was played on an anonymous account against human players in StarCraft II. It achieved a 6275 Match Making Rating for Protoss, 6048 for Terran, and 5835 for Zerg, higher than 99.8% of human players, placing it at the Grandmaster level for all 3 races. It was the first bot to reach Grandmaster in StarCraft.

AlphaStar innovatively used league-based training and special purpose agents to improve the training effectiveness. Specifically training certain agents to find the weaknesses of other agents resulted in more diverse and flexible strategies and playstyles. This framework can be adapted to train not only other ML bots but also neural networks aimed at handling complex real-world scenarios.

2.4. Juewu

2.4.1. Honor of Kings

Honor of Kings is a Multiplayer Online Battle Arena (MOBA) game developed by Tencent Timi Studio Group. This paper's discussion about this game is limited to 1v1 mode only.

Players need to control a hero (in-game character) to fight against other heroes within the fixed limited map of the game. Players perform attacks and other combat actions by moving in all 2D directions and unleashing three hero-unique (or more for some heroes) skills and two generic skills. Players also need to defeat the game units(e.g., enemy heroes, creeps, turrets) to get gold and exp to buy equipment and upgrade skills to improve the hero's ability. In the map, the player and the enemy each have a crystal, and although it is generally accepted that the destruction of one side's crystal counts as a victory for the other side, for the study of Juewu AI[22], it is mainly about losing or winning battles with other heroes.

According to Tencent's official paper on Juewu [22], the Juewu system has four parts: the Reinforcement Learning (RL) Learner, the Artificial Intelligence (AI) Server, the Dispatch Module, and the Memory Pool. Figure 4 shows Juewu's system design.

The AI server is used to implement interaction between the AI modules and the actual game. In the AI server, the game episodes are self-generated using mirrors, and game state features are extracted and used to predict the possible actions of the heroes by Boltzmann Exploration, which are then sent to the scheduling module for compression and packaging for further sending to memory

DOI: 10.54254/2755-2721/108/2025.21192

pools used for samples storage and data sampling. After that, data is sent to the most central and important component, the RL learner.

2.4.2. Algorithm design

a) Boltzmann Exploration: Boltzmann exploration is a classic strategy for sequential decision-making under uncertainty and is one of the most standard tools in Reinforcement Learning [23]. In uncertain environments, intelligent agents need to learn to find and use the best strategy possible, and in uncertain environments where the goodness of actions is unknown, the intelligent agents will need to explore the unknown actions to find a better one, and at the same time, the intelligent agent will need to juggle the actions of the moment. Boltzmann exploration is an algorithm that solves this kind of problem. It assigns an energy level to each behavior to indicate the estimated value of the behavior and introduces a value called the temperature parameter to control how flat the probability distribution of all actions is, with flatter emphasizing exploration and less emphasizing exploration. Behaviors are selected by calculating the resulting probability distribution. The setting of the temperature parameter helps to progressively reduce exploration during the learning process, leading to better utilization of known effective strategies while ensuring that sufficient exploration of unknown strategies is possible at an early stage.

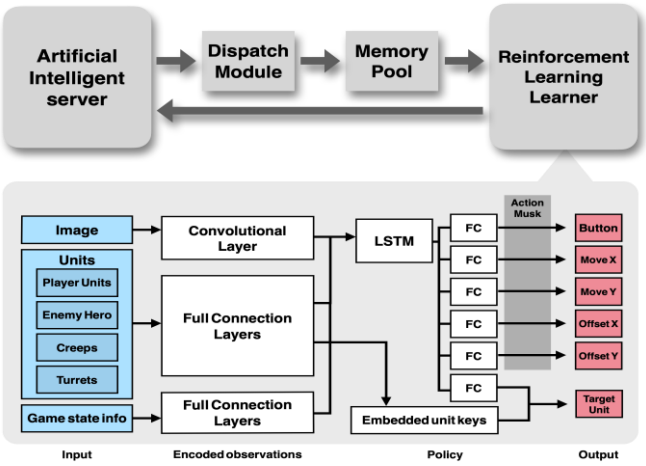

Figure 4: The system of Juewu

b) The RL Learner: As shown in Figure 5, the RL learner module has two portions: the encoded observation portion and the policy portion. At input, there are three data types inputted to the RL learner: in-game units, the camera view, and the game state information.

The unit data or the vector features are passed through multiple full connections (FC) layers, Rectified Linear Unit(ReLU) activation functions, and a max-pooling layer. The data is split into two parts: the unit's representation and the target's attention keys. The image feature is encoded using convolutions, and the game state information is encoded via an FC layer. At the end of the encoded observation process, features are represented as the encoded vector of an observable game state and then mapped to a Long Short Term Memory(LSTM) cell, which takes temporal information into further consideration.

LSTM is used to learn skill combos for heroes, which are critical to creating high burst damage. To address the difficulty of simulating the correlations between different types of in-game actions (e.g., movement direction, and skill selection), each tag of action is treated independently to decouple their correlations. This decoupled objective simplifies the policy structure and increases the diversity of actions. However, it also further increases the complexity of policy training.

DOI: 10.54254/2755-2721/108/2025.21192

To improve the training efficiency, prior knowledge of experienced human players is used to prune the exploration of RL. For example, if the hero wants to move in the direction of an obstacle that obstructs movement, this aspect of exploration will be eliminated. After this action mask, the RL learner finally outputs actions, including movement, skill selection, skill offset, and target unit selection.

c) Proximal Policy Optimization and Improved Version: Proximal Policy Optimization (PPO) is a reinforcement learning algorithm proposed by OpenAI in 2017 [24]. It is a type of policy gradient-based method for optimizing the policies of intelligence to learn and perform tasks in the environment. It adopts interactive sampling of data and proposes a new alternative objective function that allows multiple small batch updates using a stochastic gradient ascent algorithm.

Compared to the standard policy gradient method, PPO can perform more model updates with the same sample size, thus improving learning efficiency. PPO limits the magnitude of each policy update through proximal optimization, thus avoiding overly drastic updates that can lead to unstable learning. This is achieved by introducing a truncation term or proximal constraint that controls the size of the policy update during the optimization process. This constraint ensures that each policy update does not move too far away from the previous policy, thus maintaining the stability and convergence of the learning. Usually, PPO performs relatively well in many environments, and in the area of deep learning, it is commonly used. However, in the Juewu application setting, the training is massive, and the movements are sampled from multiple sources; the current policy might be greatly different from the previous policy, so the deviation from the old and new policies can be so large that standard PPO is unable to handle it.

Therefore, researchers proposed a dual-clipped PPO algorithm and applied it in the RL learner to support large-scale distributed training which further clips the ratio with a lower bound. The dual-clipped version of the PPO algorithm is proposed to guarantee convergence with large and deviated batches.

d) Long Short Term Memory: Long Short Term Memory (LSTM) is a special kind of Recurrent Neural Network(RNN) [25]. RNN is a type of neural network used to process sequential data. Different from a normal neural network, it is able to handle data that has its meaning dependent on the sequence it is in. For example, the meaning of a certain word will be different depending on the context. RNN is able to solve this kind of problem very well.

LSTM is mainly used to solve the problem of gradient vanishing and gradient explosion during the training of long sequences. LSTM can perform better in longer sequences than ordinary RNNs. LSTM introduced gating functions, including a cell state, an input gate, a forget gate, and an output gate to control the flow of information in the memory cells. The input gate decides which information needs to be stored in the cell state, the forget gate decides which old information needs to be forgotten, and the output gate determines the output information. The cell state is responsible for the transfer and storage of memory information throughout the memory cell.

This gating mechanism allows the LSTM to deal with long-term dependencies between input data and thus achieve good results when dealing with sequential data. These features made LSTM a hot research topic in the field of deep learning. It has achieved significant success in many applications.

2.4.3. Performance and contributions

Researchers invited 5 top-level professional Honor of Kings human players to play best-of-5 matches against Juewu. The result was that Juewu used various types of heroes to defeat all the top human players in an almost one-sided victory, even when the human players were using the heroes they were best at. Juewu won by 3 to 0 against 4 professional players, and only one professional player managed

DOI: 10.54254/2755-2721/108/2025.21192

to kill Juewu a single time, resulting in a score of 3 to 1. From 2 August to 5 August 2019, public matches were held in Shanghai, where the public was allowed to face off against Juewu. Juewu achieved a 99.81% win rate among 2100 matches, with only 4 games lost. Five of the eight heroes maintained a 100% win rate throughout hundreds of matches.

This paper designed the dual-clip PPO algorithm which can be useful for machine learning in different scenarios and made outstand- ing contributions to building AI for 1v1 MOBA games. Although the released version of Juewu focused on the Honor of Kings, Juewu's framework can theoretically be used for any MOBA game.

3. Comparison & Contrast

3.1. AlphaGo and Stockfish

AlphaGo's neural networks are not specifically designed to play Go and can evaluate other board games as well, as shown by AlphaZero's success in Go, chess, and shogi. Stockfish, on the other hand, uses traditional algorithmic evaluation to some extent, limiting it to only chess. However, Stockfish's sole fixation on chess made it smaller and required much less computation than any iteration of AlphaGo. This demonstrates that although generalizable neural networks that evaluate games from a more fundamental level can master multiple games at once, they're not nearly as efficient as programs designed to handle one single game.

AlphaZero is stronger than Stockfish 8 and Stockfish 12. Against Stockfish 8, it won 28 games and drew the remaining 72. Against Stockfish 12, it won 290 games, lost 24 games, and drew 886. It is unclear whether AlphaZero is stronger than the current version of Stockfish, Stockfish 16. However, another chess bot modeled after AlphaZero, Leela Chess Zero[26], is quite close to Stockfish in strength but only defeated Stockfish once in TCEC in 2020.

3.2. AlphaStar and Juewu

AlphaStar and Juewu employed different training strategies. Juewu did not use special exploiter agents in its training process like Al- phaStar did, mainly because Honor of Kings emphasizes fast reaction, accurate prediction, and precise timing over flexible strategies. Instead, Juewu optimized its reinforcement learning algorithms to maximize training efficiency and effectiveness.

The different levels of complexity in the two games led to different performances. AlphaStar was placed in the top 0.2% of human players but could not defeat the best professional players. Juewu, on the other hand, won overwhelmingly against amateur and professional players alike. Juewu is not concerned with off-camera information or individual control of hundreds of units. Honor of Kings' in-game economy is also much simpler than StarCraft. The lower level of strategic complexity placed higher importance on direct combat, which Juewu excelled at.

1) Board game bots and digital game bots: All four of the bots utilize machine learning to perform regression analysis. Reinforcement learning is most predominantly used.

In terms of playing strength, AlphaStar and Juewu did not surpass humans nearly as much as AlphaGo and Stockfish did. The game states and actions of StarCraft and Honor of Kings are more complex than board games like Go and Chess. The former two have high-dimensional and continuous input space and action space, whereas the latter two have low-dimensional and discrete input space and action space.

There are multiple ways to solve the problem of complexity. One is to train the neural network with less specific information and aim at a single generalized neural network that can process many different types of inputs, similar to AlphaZero. Another way is to divide the game into several aspects and design neural networks more focused on one specific aspect to improve efficiency, which allows for faster decisions and reactions.

DOI: 10.54254/2755-2721/108/2025.21192

4. Discussion

The 4 ML bots discussed in this paper have largely solved the challenges illustrated in the introduction & background section.

With Monte Carlo tree search guided by neural networks, AlphaGo was able overwhelmingly defeat previous Go programs and could defeat even the best human players, proving that machine learning can be highly effective despite the high complexity of the action space. The action space in chess is simple compared to Go due to the lower number of possible moves. Therefore, it is easier to evaluate chess positions using traditional man-made algorithms. Despite this, Stockfish still incorporated neural networks and deep learning to evaluate closely matched boards, demonstrating the superior evaluation capabilities of machine learning even in less complex board games.

AlphaStar and Juewu also solved the challenges of real-time digital games. Their neural networks can accept a wide variety of input features, and AlphaStar only requires one high-end consumer GPU to run. Since AlphaStar and Juewu use a policy function to decide their actions, they do not search the action tree at all and bypass this problem.

The problem of training data is the only issue that poses a serious challenge to AlphaStar and Juewu. A vast amount of data is required to train the neural networks, and it would be difficult to gather enough data from matches between top players. Matches from regular players were used instead. However, this poses another problem, as the bot would not be able to perform well if it was trained solely based on data from regular players. Therefore, self-play was used to generate a large number of matches, and reinforcement learning was used to enable the bots to improve from those matches. AlphaStar's approach to self-play is particularly interesting as it uses league-based training with agents designed specifically to target and find weaknesses of other agents. This method would allow for the creation of more robust and flexible bots.

AlphaZero's algorithm stands out due to its ability to play multiple games. It approaches board games from a fundamental level. Its policy network takes a game state as input and outputs possible moves. This method of processing game information can be used to master any board game.

AlphaGo and Stockfish demonstrate that neural networks can make decisions faster and far better than humans. Although real-world decision-making is not as simplistic as Go or chess, their algorithms can inspire those who wish to develop an AI for tasks such as customized recommendations.

Solving the problem of complex digital games also has significant real-world implications. The physical world, similar to a digital game, has a continuous and high-dimensional input space and action space, with the main difference being that more factors need to be taken into consideration in a real-world task. Currently, AI is still unable to perform a large number of such tasks reliably, but game bots like AlphaStar and Juewu demonstrated the strength of reinforcement learning and devised innovative methods and technologies. They're fast, reliable, and flexible, all of which are essential for AI concerned with tasks such as driving. Therefore, they can serve as a valuable example.

5. Conclusion

The 4 ML bots demonstrate machine learning's potential in gaming. AlphaGo and Stockfish play turn-based board games, whereas AlphaStar plays a real-time strategy game, and Juewu plays a MOBA game.

In terms of playing strength, AlphaGo and Stockfish are significantly better than even the best human players. Juewu is stronger than professional players, but AlphaStar is only stronger than 99.8% of human players.

DOI: 10.54254/2755-2721/108/2025.21192

In terms of computational power required, AlphaGo Lee needs 48 TPUs, whereas AlphaGo needs 4 TPUs. AlphaStar needs 1 single high-end consumer GPU. Stockfish has the lowest requirement for computational power, as it can be run on one single CPU. Juewu's requirements are not specified.

References

[1] *Mahesh, B. Machine Learning Algorithms - A Review. IJSR 9, 381–386 (2020).*
[2] *Janiesch, C., Zschech, P. & Heinrich, K. Machine learning and deep learning. Electron Markets 31, 685–695 (2021).*
[3] *Steinkraus, D., Buck, I. & Simard, P. Y. Using GPUs for machine learning algorithms in Eighth International Conference on Document Analysis and Recognition (IC- DAR'05) (2005), 1115–1120.*
[4] *Schlegel, D. Deep machine learning on Gpu. University of Heidelber-Ziti 12 (2015).*
[5] *Matsuo, Y. et al. Deep learning, reinforcement learning, and world models. Neural Networks 152, 267–275 (2022).*
[6] *Nasteski, V. An overview of the supervised machine learning methods. Horizons. b 4, 51–62 (2017).*
[7] *Desale, S., Rasool, A., Andhale, S. & Rane, P. Heuristic and meta-heuristic algorithms and their relevance to the real world: a survey. Int. J. Comput. Eng. Res. Trends 351, 2349–7084 (2015).*
[8] *Burmeister, J. & Wiles, J. The challenge of Go as a domain for AI research: a comparison between Go and chess in Proceedings of Third Australian and New Zealand Conference on Intelligent Information Systems. ANZIIS-95 (1995), 181–186.*
[9] *Silver, D. et al. Mastering the game of Go with deep neural networks and tree search. nature 529, 484–489 (2016).*
[10] *Chaslot, G., Bakkes, S., Szita, I. & Spronck, P. Monte-Carlo tree search: A new framework for game ai in Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment 4 (2008), 216–217.*
[11] *Coulom, R. The Monte Carlo revolution in Go in The Japanese-French Frontiers of Science Symposium (JFFoS 2008), Roscoff, France 115 (2009).*
[12] *Chaslot, G., Saito, J.-T., Bouzy, B., Uiterwijk, J. & Van Den Herik, H. J. Monte-carlo strategies for computer go in Proceedings of the 18th BeNeLux Conference on Artificial Intelligence, Namur, Belgium (2006), 83–91.*
[13] *Silver, D. et al. A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play. Science 362, 1140–1144 (2018).*
[14] *Silver, D. et al. Mastering the game of go without human knowledge. Nature 550, 354–359 (2017).*
[15] *Modi, E. & Acuna, K. The Effects of Computer and AI Engines on Competitive Chess. J Stud Res 12, (2023).*
[16] *Smith, S. J. & Nau, D. S. An analysis of forward pruning in AAAI (1994), 1386–1391.*
[17] *Thakur, S. et al. Designing a Next-Generation Chess Engine with Advanced AI and Neural Networks. (2024)*
[18] *Maharaj, S., Polson, N. & Turk, A. Chess AI: competing paradigms for machine intelligence. Entropy 24, 550 (2022).*
[19] *Mandadi, S. Implementation Of Sequential And Parallel Alpha-Beta Pruning Algorithm. 7(2020).*
[20] *A. M. Chitale, A. M. Cherian, A. Singh & P. P, Implementing the Chess Engine using NNUE with Nega-Max Algorithm(2024).*
[21] *Vinyals, O. et al. Grandmaster level in StarCraft II using multi-agent reinforcement learning. Nature 575, 350–354 (2019).*
[22] *Ye, D. et al. Mastering complex control in moba games with deep reinforcement learning in Proceedings of theAAAI Conference on Artificial Intelligence 34 (2020), 6672–6679.*
[23] *Marullo, C. & Agliari, E. Boltzmann Machines as Generalized Hopfield Networks: A Review of Recent Results and Outlooks. Entropy 23, 34 (2020)*
[24] *Schulman, J., Wolski, F., Dhariwal, P., Radford, A. & Klimov, O. Proximal Policy Optimization Algorithms. arXiv: 1707.06347[cs] (2017).*
[25] *Yu, Y., Si, X., Hu, C. & Zhang, J. A Review of Recurrent Neural Networks: LSTM Cells and Network Architectures. Neural Computation 31, 1235–1270 (2019)*
[26] *Klein, D. Neural Networks for Chess. arXiv preprint arXiv:2209.01506 (2022).*