TNM114 - Artificial Intelligence for Interactive Media
November 3, 2024

Xiàngqí AI with Deep Q-Networks

Shaoxuan Yin[1], Oskar Tengvall[2]

Abstract
The boardgame *Xiàngqí* (i.e. *Chinese Chess*) shares many properties with *International Chess* which has been extensively researched in the field of AI. This report will study the learning-rate in Xiàngqí of an AI-player controlled by reinforcement-learning with Deep Q-Networks using the development-tool Qt. Results were inconclusive due to limited time for the AI to train and be fine tuned, and limited types of opponents to have the AI train against.
Source code: https://github.com/Qervas/cn_chess_ai

Authors
[1]*Computer Science Student at Linköping University, shayi783@student.liu.se*
[2]*Media Technology Student at Linköping University, oskte817@student.liu.se*

Keywords: Deep Q-Networks (DQN) — Self-Play — Cuda

Contents

1 Introduction 1
1.1 Xiàngqí . . . 1
1.2 AI in Board Games . . . 1
1.3 Aim . . . 2
1.4 Research questions . . . 2
1.5 Deliminations . . . 2
2 Theory 2
2.1 Xiàngqí Board . . . 2
2.2 Deep-Q-Learning . . . 2
2.3 Q-Learning . . . 2
2.4 Deep-Q-Network . . . 3
2.5 Experience Replay . . . 3
2.6 Target Network . . . 3
2.7 Loss Function . . . 3
3 Method 3
3.1 Steps . . . 3
3.2 GUI . . . 3
3.3 Chess Logic Module . . . 3
3.4 Chess AI . . . 4
3.5 DQN with CUDA . . . 5
Neural Network Architecture ▪ CUDA Optimization ▪ Training Process
3.6 Training Module . . . 6
4 Result 6
5 Discussion & Analysis 6
6 Conclusion 7
References 7

1. Introduction

AI systems bring a unique approach to the interactive game genre. By introducing AI as agents which play as human roles, it demonstrates the computability and possible operations of a wide range of games. Training data is vital for learning-based experiments, meaning the performance of an AI limited by the data it derives decision-making from. Xiàngqí is a chess-like game in which two players compete using strategy and is thus a candidate for AI-powered play.

1.1 Xiàngqí

Xiàngqí (象棋, i.e. *Chinese Chess* or *Elephant Chess*) is a variant of chess that is very popular in Southeast Asia [1]. While related and overall similar to International Chess, it has a unique rule set, layout and pieces with different moves.

1.2 AI in Board Games

AI-driven systems have a long history of optimizing play for competitive board games. *Deep Blue* is a chess-playing expert system developed in 1985 that beat the concurrent chess-grandmaster Garry Kasparov in 1996 [2]. AlphaZero is an AI computer program made to master Chess, Shogi and using a generic reinforcement-learning algorithm via self-play [3]. While games like International Chess shares many similarities with Xiàngqí, Xiàngqí still presents different challenges in its unique state space and piece dynamics.

1.3 Aim

The aim of this report is to study the development of an AI for Xiàngqí that continuously improves through self-play using *DQN* (Deep Q-Networks). This is envisioned through the following approach:

- 1. Implementing the Xiàngqí game-logic.
- 2. Designing a DQN-based AI.
- 3. Utilizig self-play for training, akin to AlphaZero.

The intended outcome is an AI competent in Xiàngqí from having improved over time.

1.4 Research questions

- Can the trained AI agent defeat a human or another AI of reasonable competence?
- Will an AI with DQN-learning have time to train despite Xiàngqí's taxing state-space?

1.5 Deliminations

The development of the AI is limited to a study-period of around a month, leaving time for training the AI to approximately a week. Expectations of how much the AI learns in that time are set accordingly. The AI will be trained on a desktop a computer with a *12th Gen Intel Core* processor and a *NVIDIA GeForce GTX 1650* graphics-card, capping the rate of complex calculations, and thus, the learning-rate.

2. Theory

This chapter describes the theory behind all respective components that concern this project.

2.1 Xiàngqí Board

Xiàngqí (much like International Chess) consists of a board where the two player's set of pieces are arranged on opposite sides in a mirrored fashion as seen in Figure 1. Pieces can be on and be moved to the corners of spaces. While functionally inconsequential, the pieces may have different names depending on their team. These pieces are:

- 將/帥 *(General/King)* - If stuck in a checkmate, the opponent wins.
- 士/仕 *(Advisor)*
- 象/相 *(Elephant/Bishop)*
- 車 *(Chariot/Rook)*
- 馬 *(Horse/Knight)*
- 炮 *(Catapult/Cannon)*
- 卒/兵 *(Soldier/Pawn)*

Notable differences of the board compared to International-Chess is that Xiàngqí has tiles with unique properties. Across the middle of the board, horizontal to each player are tiles constituting a *river*, which some pieces cannot cross. Around each player's general-piece is a *palace*, which the general and advisor-pieces cannot exit.

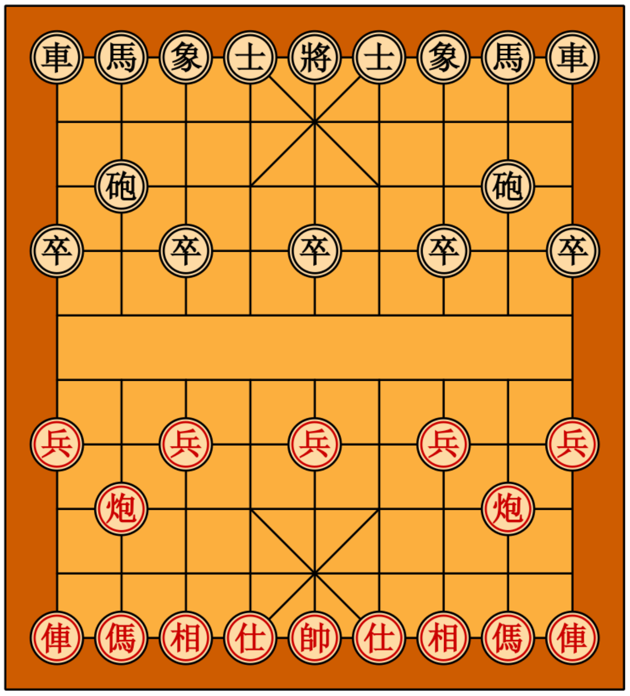

Figure 1. Starting setup of a Xiàngqí board.

Xiàngqí has fewer possible game-states than International Chess, arguably making it slightly simpler to learn and master for an AI that adapts to game-states though trial and error.

2.2 Deep-Q-Learning

Deep-Q-Learning, conceptually speaking, consists of several components interacting. It is notably a combination of *Q-learning* using *Experience Replay* to update *Deep-Q-Network* based on a *Target Network*. [4].

2.3 Q-Learning

Q-learning is a kind of reinforcement-learning that assigns values to all potential actions an agent can take in a given state by storing values in a matrix called a *Q-table* [5]. Its algorithm follows Equation 1.

$$Q(s,a) = \mathbb{E}\left[r + \gamma \max_{a'} Q(s',a') \mid s,a\right] \tag{1}$$

- $Q(s,a)$: The Q-value, represents the expected reward for taking action a in state s.
- r: The immediate reward received after taking action a.

- γ: The discount factor, which determines the importance of future rewards.
- $\max_{a'} Q(s',a')$: The maximum future reward possible from the next state s'.

2.4 Deep-Q-Network

A *Deep*-Q-Network (DQN), unlike a Q-table, is a neural network instead of a table that finds an approximate for each state-action pair (the most fitting action for any given state). This is a useful way to store values for large-state spaces that would otherwise require an impractically massive Q-table [6].

2.5 Experience Replay

With *Experience Replay*, an AI bases its next action on a randomly sampled prior action instead of its latest one, as well as only updating Q-values periodically. This helps break correlations with false-positives and avoids developing misguided patterns from misunderstood coincidences [7].

2.6 Target Network

A less frequently updated network, called a *target network*, is used as further error measure to stabilize training, by functioning as an informed backlog of actions that are not yet tampered by short-sighted updating. [4].

2.7 Loss Function

A loss function maps the values of variables relating to a potential cost/error of an event [4]. Its algorithm follows Equation 2.

$$L(\theta) = \mathbb{E}_{(s,a,r,s')\sim \mathrm{U(D)}}\left[\left(r + \gamma \max_{a'} Q(s',a';\theta^-) - Q(s,a;\theta)\right)^2\right] \tag{2}$$

The variables are as defined as follows:

- $U(D)$: Experience Replay buffer.
- θ: Parameters of the Q-Network.
- θ^-: Parameters of the Target Network.
- α: Learning rate.
- r: Immediate reward, reflecting the immediate outcome of taking action a in state s.
- γ: Discount factor. Indicates the importance of future rewards. A value closer to 1 emphasizes future rewards more heavily.
- $\max_{a'} Q(s',a';\theta^-)$: The maximum predicted Q-value from the target network for the next state s'. It represents the best possible future reward.
- $Q(s,a;\theta)$: The current Q-network's predicted Q-value for taking action a in state s.

The goal of a loss function is to minimize it, which if successfully done removes error and aligns Q-Network predictions with target Q-values.

To prevent endless games, *Gradient Descent* is used, defined by Equation 3 [4].

$$\theta \leftarrow \theta - \alpha \nabla_\theta L(\theta) \tag{3}$$

3. Method

DQN (Deep Q-network) reinforcement-learning was used to train two AI opponents in Xiàngqí. After certain outcomes, each AI was guided through being rewarded or deducted points from its score that immediately factor into its decision-making on its next turn. The implementation followed a modular approach with several key components working in conjunction.

3.1 Steps

First, a program was created to have a *GUI* (graphical user interface). It was made to visualize a digital board of Xiàngqí with all its pieces. Then a *chess logic module* was created to allow user input to control the chess pieces. The module was also made to manage the game's rules, update the game-state and validate the pieces' moves. An AI was made to make decisions based on *Deep Q-Network* predictions, to then be given control over player input. The program was running with 2 identical AI agents playing against each other in order to model training iterations via self-play. These implementation steps are summarized in Figure 2.

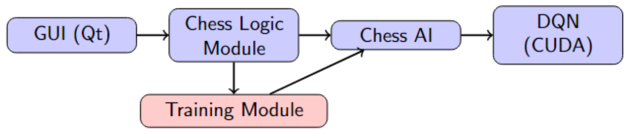

Figure 2. Scheme of the implementation steps.

3.2 GUI

To create an environment to train AI with a simple *GUI* (graphical user interface) for easier tracking, the development tool *Qt Creator* was used and compiled with *CMake*. Qt is an efficient cross-platform open-source library for GUI in C++. This made it easier to verify that the AI follows the rules and that only valid moves are played. It intuitively provides three game modes: Human-vs-Human, Human-vs-AI and AI-vs-AI. The interface also tracks the scores of each player. The interface is seen in Figure 5.

3.3 Chess Logic Module

The Chess Logic Module was added to handle the fundamental layer of the system, comprising thorough game

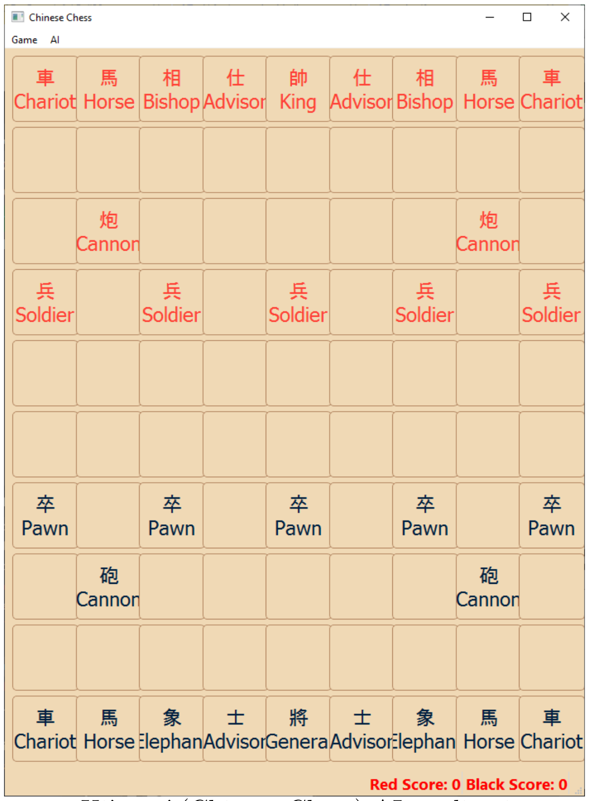

Figure 3. Xiàngqí (Chinese Chess) AI application setup.

state management and rule enforcement for Xiàngqí. Fundamentally, the module optimizes memory access patterns and state updates by using a 10x 9 grid implemented as a flat vector, therefore using an effective board representation scheme. Every point in this grid preserves comprehensive information on piece kind and color, therefore facilitating quick state evaluation and move validation.

Move validation is accomplished using a sophisticated suite of specialized tools, each designed to fit the particular movement patterns of various piece kinds. These roles uphold the complex laws of Xiàngqí, including rigorous palace border checks limiting Generals and Advisors' mobility to their own palaces. The system controls intricate path obstruction identification for pieces like the Horse and Elephant as well as river crossing limits that limit some pieces' mobility across the center divide. The special capture mechanisms of the Cannon needs particular focus as it depends on exact validation of intervening parts during capture motions.

Game state control involves several linked systems cooperating. A strong piece tracking system regularly updates score computations depending on material advantages and accurately records acquired pieces. The module guarantees correct game termination when win criteria are satisfied by using advanced detection algorithms for game-ending circumstances including checkmate and stalemate scenarios. Turn management is controlled by a state machine guaranteeing move legality in the present game environment and preserving player alternation. Maintaining computational efficiency, this all-encompassing approach to game state management guarantees that all elements of the game advance in line with conventional Xiàngqí guidelines.

3.4 Chess AI

Extending the Chess Logic Module as an AI-operated player-agent in Xiàngqí, the Chess AI component was built on an abstract `IChessAgent` interface. This architecture keeps a consistent interface for game interaction while allowing flexible application of several AI methods. The system effectively codes both piece locations and types by use of a complex state representation technique using a 90x14 dimensional state vector. Together with positional assessment measures that reflect strategic piece placement concerns, this representation is enhanced with additional state elements like normalized game scores, move counts, and player turn information.

By means of dynamic exploration rate adjustment and ε-greedy strategy, the action selection mechanism balances exploitation of known effective methods with investigation of new opportunities. The system guarantees only legal movements are taken into account by using the `getAllValidActions()` technique for move validation. While the move assessment system regards both immediate tactical benefits and possible future rewards, a comprehensive action space mapping mechanism closes the gap between board coordinates and DQN outputs.

With the General valued at 1000 points, Chariots at 90, Cannons at 45, Horses at 40, Advisors and Elephants at 20, and Soldiers at 10 points the reward system uses a finely calibrated scoring method based on component values. Position bonuses that reward clever piece placement, terminal state rewards that offer feedback on game outcomes, and move count restrictions that promote effective play and prohibit endless games augment this fundamental material evaluation.

The AI's decision-making process integrates multiple evaluation factors:

$$\text{Evaluation} = w_m \cdot \text{Piece} + w_p \cdot \text{Position} + w_m \cdot \text{Mobility} \quad (4)$$

This equation combines piece evaluation—considering material values and captures—positional evaluation —accounting for strategic board control—and mobility assessment—analyzing the amount of accessible legal moves. The weights are constantly changed depending on the game state and learnt patterns, therefore enabling the artificial intelligence to change its approach as the game advances.

By use of piece-square tables that offer dynamic position assessment depending on piece type and placement, the evaluation function performs complex strategic analysis. These tables change depending on the game level and include specific benefits for managing important squares, therefore enabling subtle positional play. Advanced game phase recognition—that which detects various game phases depending on piece count and position —is another feature of the system. Whether opening, middlegame, or endgame, this awareness helps to choose adaptable strategies and adjust assessment weights suitable for every game phase.

Training gathers experience by means of state-action-reward sequences using both self-play and opponent-based learning strategies. Maintaining thorough records of game statistics including victory rates, average game time, and piece capture patterns, the system supports ongoing performance monitoring and optimization. By allowing interrupted training sessions to be continued and trained models to be stored and loaded for assessment or deployment, model persistence features help to guarantee effective use of training time and resources.

The AI was made to base future moves on a score that is updated according to the game's state. For example, it increases as the AI player takes an opponent piece or wins, while decreasing if the AI player loses pieces or the game.

3.5 DQN with CUDA

The AI was designed to improve over time by feeding *Q-learning* values into a deep *DQN* (Deep Q-network) with *Experience Replay* and a *Target Network* using the parallel computing platform *CUDA*. The DQN estimates Q-values for state-action pairs according to Equation 1 in conjunction with the *Loss Function* according to Equation 2.

3.5.1 Neural Network Architecture

Deep neural networks especially meant for the Xiàngqí state space were implemented in the network architecture. Comprising 1,263 nodes ($90 \times 14 + 3$), the input layer capture thes board state as well as other game elements. This dense representation guarantees the preservation of all pertinent game information for use in decision-making. The network maintains computational efficiency while nevertheless offering enough complexity to capture significant game trends by using a hidden layer of 128 nodes with ReLU activation functions. With 8,101 nodes (90×90), the output layer represents every conceivable move combination in the game and lets the network analyze all possible actions concurrently.

Two essential components of steady learning were included into the DQN implementation. First, from a history of 10,000 state-action-reward transitions, an Experience Replay Buffer randomly samples batches of 32 events for training. This randomizing technique guarantees steady convergence and helps to decorate the learning samples thereby prevents overfitting to recent events. Second, every 100 training steps a Target Network technique is used to synchronize with the main network. With $\tau = 0.001$, this target network uses a soft update technique to progressively incorporate new learning while eliminating oscillations in the learning process and keeps constant learning goals.

3.5.2 CUDA Optimization

By use of advanced memory management and kernel optimizations, the solution made use of CUDA's parallel processing capacity. The memory management system guaranteed effective memory allocation and automatic cleanup by use of a proprietary CUDA memory allocator based on CudaUniquePtr. To reduce data transmission overhead between CPU and GPU, this is matched by well-crafted host-device memory transfer techniques and pinned memory for asynchronous tasks.

Maximizing GPU use via various important techniques was the main emphasis of kernel optimizations. Frequently accessed data is mostly accessible via shared memory, so global memory access latency is much lowered. For effective parallel reduction operations, warp-level primitives are used; memory access patterns are carefully crafted to guarantee coalesced memory access, hence optimizing memory bandwidth use.

By use of parallel CUDA kernels, the Q-value estimation procedure executes Equation 1 thereby facilitating simultaneous assessment of many state-action pairs.

The loss function is used in a parallel-optimized manner in the network training process (Equation 2. Forward propagation enables batch-wise state processing and efficient parallel matrix multiplication using cuBLAS, hence optimizing activation function computations. Similar simultaneous gradient computation with effective error propagation and batch-normalized updates drives the backpropagation process. This parallel technique greatly speeds up the training process such that the network may simultaneously handle several game states and preserve numerical stability.

3.5.3 Training Process

The training approach used a well-adjusted set of learning parameters best for the Xiàngqí domain. The approach guaranteed consistent weight updates by using a conservative learning rate (α) of 0.001, therefore preserving the capacity of the network to learn from fresh events. With a discount factor (γ) of 0.99 the network may efficiently balance long-term strategic benefits with instantaneous rewards. An ε-greedy approach managed the exploration-exploitation trade-off by dynamically changing the exploration rate from 1.0 to 0.1 across training, hence progressively moving from widespread exploration to targeted exploitation of learned methods.

Using mini-batch gradient descent, the optimization plan effectively updated network weights by analyzing limited sets of experiences. The optimizer with momentum helped to further this by adjusting the learning rate for every parameter depending on the size of recent gradients. The method used gradient clipping to limit weight updates under realistic bounds, hence preserving numerical stability and preventing training divergence.

Parallel processing capabilities enabled simultaneous computation of several training components, therefore attaining significant performance improvement. To drastically cut training time while yet preserving the stability needed for deep Q-learning, the system parallelized state representation computations, Q-value estimations, batch experience processing, and network weight updates. This CUDA-accelerated method efficiently solved the computing problems presented by Xiàngqí's large state space.

3.6 Training Module

The AI's learning process was coordinated in the training module by use of a complex mix of self-play and opponent-based training tactics. The system used self-play learning throughout a four-day training session, in which the artificial intelligence kept developing by challenging itself and getting instantaneous feedback via a whole reward system. These events helped the DQN to grow by iteratively improving its decision-making capacity, hence progressively refining its capabilities.

Parallel game simulations enabled self-play training, hence optimizing computing efficiency and hastening learning. Maintaining thorough records of state-action-reward cycles, the system created a comprehensive collection of gaming experiences. While they provided benchmarks for tracking development over time, regular model assessment and checkpointing provided training stability and helped to enable recovery from possible disturbances.

4. Result

This section showcases the results of implementing the scheme from Figure 2 to an application that runs 1000 simulated Xiàngqí matches, letting the AI train in each to consecutively improve. The win-rate for the AI against an opponent that only makes random (legal) moves throughout 1000 matches is plotted in Figure 4. The win-rate for the AI against another AI of identical learning algorithm (self-play) is plotted in Figure 5.

5. Discussion & Analysis

Observed from the results was that the AI steadily improved its decision-making over time. The AI made less suboptimal moves and stabilized its strategy as the AI had more experience to draw from. The AI developed a strategy that maximizes long-term rewards, adapts to scenarios and sets up advantageous scenarios for itself.

However, the improvement rate stagnated firmly after only a few hundred games, which is likely a mix of two main reasons:

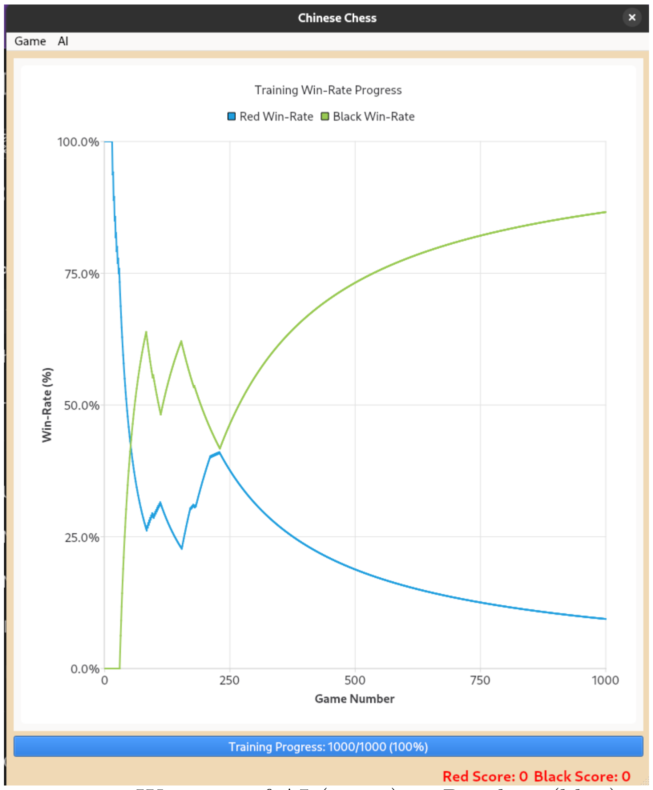

Figure 4. Win-rate of AI (green) vs Random (blue).

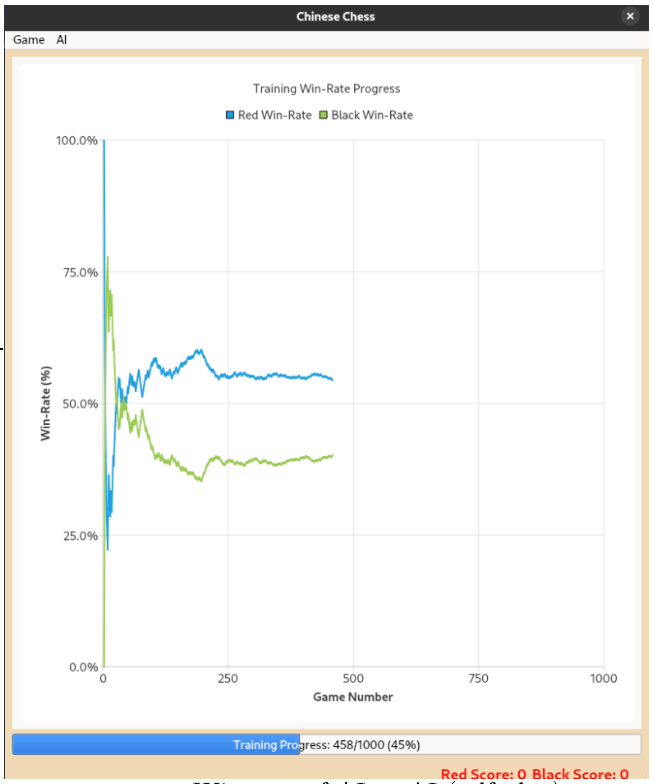

Figure 5. Win-rate of AI vs AI (self-play).

- The AI did not strike a practical balance of exploration versus exploitation during training.
- The opponent was too incompetent to punish sub-optimal moves which would incentivize avoiding them.

The implementation of DQN networks proved a functional way for an AI to continuously improve, even if later stagnating and, in this particular experiment, not ending up competent enough to pose a challenge to a human opponent.

A challenge encountered early on was Xiàngqí's taxing state-space, which significantly slowed down training. Thanks to a software like CUDA which off-loaded heavy calculations to the GPU, the training time became magnitudes faster.

To optimize the large size of state-space, a Monte Carlo tree search would significantly reduce the computation cost by pruning off numerous branches and only exploring those most promising branches. It offers a balance between exploration and exploitation and it doesn't rely solely on the current policy estimates to make decisions. This is particularly useful in games or problems with sparse rewards where DQN might struggle with adequate exploration.

This project has proved a framework with evident potential, but with room for improvement in its neural network architecture. This project only featured training versus a very limited set of opponents, which could be expanded upon with multi-agent training to ensure the AI is equipped to counter any kind of counter-play.

6. Conclusion

The aim of having a Xiàngqí AI using Deep Q-Networks was achieved, but with inconclusive success.

Using CUDA and C++ greatly accelerated computations, solving the problem of Xiàngqí's taxing state-space making an AI with DQN-networks too slow. However, the lack of implementation of Experience Replay leads to an AI with a stagnant strategy, ill fit to defeat a human or another AI of reasonable competence. Future work could employ multi-agent training for more diverse strategies.

References

[1] Hans L. Bodlaender and Fergus Duniho, **The Chess Variant Pages**, *Xiàngqí (象棋): Chinese Chess* (2001). Accessed on December 10th 2024. Url: `https://www.chessvariants.com/xiangqi.html`

[2] **CHESScom**, *Kasparov vs. Deep Blue | The Match That Changed History* (Oct 12, 2018). Accessed on December 10th 2024. Url: `https://www.chess.com/article/view/deep-blue-kasparov-chess`

[3] David Silver, **Google DeepMind**, *AlphaZero: Shedding new light on chess, shogi, and Go* (Dec 6, 2018). Accessed on December 10th 2024. Url: `https://deepmind.google/discover/blog/alphazero-shedding-new-light-on-chess-shogi-and-go/`

[4] Volodymyr Mnih, **LETTER**, *Human-level control through deep reinforcement learning* (Sep 1, 2024). Accessed on December 10th 2024. Url: `https://web.stanford.edu/class/psych209/Readings/MnihEtAlHassibis15NatureControlDeepRL.pdf`

[5] Dhanoop Karunakaran, **Medium**, *Q-learning: a value-based reinforcement learning algorithm* (Sep 17, 2020). Accessed on December 10th 2024. Url: `https://medium.com/intro-to-artificial-intelligence/q-learning-a-value-based-reinforcement-learning-algor`

[6] Samina Amin, **Medium**, *Deep Q-Learning (DQN)* (Sep 1, 2024). Accessed on December 10th 2024. Url: `https://medium.com/@samina.amin/deep-q-learning-dqn-71c109586bae`

[7] Samina Amin, **Medium**, *Deep Q-Learning (DQN)* (Sep 1, 2024). Accessed on December 10th 2024. Url: `https://paperswithcode.com/method/experience-replay`