TRƯỜNG ĐẠI HỌC GIAO THÔNG VẬN TẢI

**KHOA CÔNG NGHỆ THÔNG TIN**



**ĐỒ ÁN TỐT NGHIỆP**

ĐỀ TÀI

**NGHIÊN CỨU CÁC THUẬT TOÁN TÌM KIẾM TRONG CỜ VUA VÀ MÔ HÌNH HỌC SÂU XÂY DỰNG CHESS BOT AI**

Giảng viên hướng dẫn : TS Cao Thị Luyên

Sinh viên thực hiện : Trần Công Thành

Mã sinh viên : 201240331

Lớp : Công nghệ thông tin 6

Khóa : 61

**Hà Nội – 2024**

TRƯỜNG ĐẠI HỌC GIAO THÔNG VẬN TẢI

**KHOA CÔNG NGHỆ THÔNG TIN**



**ĐỒ ÁN TỐT NGHIỆP**

ĐỀ TÀI

**NGHIÊN CỨU CÁC THUẬT TOÁN TÌM KIẾM TRONG CỜ VUA VÀ MÔ HÌNH HỌC SÂU XÂY DỰNG CHESS BOT AI**

Giảng viên hướng dẫn : TS Cao Thị Luyên

Sinh viên thực hiện : Trần Công Thành

Mã sinh viên : 201240331

Lớp : Công nghệ thông tin 6

Khóa : 61

**Hà Nội – 2024**

**LỜI CẢM ƠN**

\*\*\*

Em chân thành cảm ơn **Cô Cao Thị Luyên** trong thời gian qua đã giúp đỡ, hướng dẫn em để em có thể hoàn thành đề tài này một cách tốt nhất.

Qua đây, em cũng xin được gửi lời cảm ơn đến tất cả các thầy cô công tác tại khoa Công nghệ thông tin - Trường Đại học Giao thông vận tải đã dìu dắt, chỉ bảo cho em trong suốt quá trình học tập tại trường.

Cuối cùng, em xin gửi lời cảm ơn đến gia đình và tất cả người thân, bạn bè, những người đã luôn động viên, ủng hộ, tạo điều kiện cho em trong thời gian hoàn thành đồ án.

Mặc dù em đã cố gắng rất nhiều nhưng cũng không tránh khỏi thiếu sót và hạn chế trong quá trình thực hiện. Em rất mong nhận được sự thông cảm và chỉ bảo cũng như những đóng góp chân thành của các thầy cô và bạn bè để bài đồ án được tốt hơn. Em xin chân thành cảm ơn!

Hà Nội, Ngày 07 tháng 06 năm 2024

*Sinh viên thực hiện:*

**Thành**

**Trần Công Thành**

**MỤC LỤC**

[LỜI MỞ ĐẦU 1](#_Toc168542342)

[CHƯƠNG 1. TỔNG QUAN ĐỀ TÀI 2](#_Toc168542343)

[1.1. Mô tả bài toán 2](#_Toc168542344)

[1.1.1. Lý do lựa chọn đề tài 2](#_Toc168542345)

[1.1.2. Mục tiêu của đề tài 2](#_Toc168542346)

[1.1.3. Phạm vi của đề tài 2](#_Toc168542347)

[1.2. Lịch sử và sự phát triển của trí tuệ nhân tạo trong cờ vua 2](#_Toc168542348)

[1.3. So sánh giữa Minimax và Monte Carlo Tree Search 5](#_Toc168542349)

[1.4. Mạng neuron nhân tạo trong cờ vua 7](#_Toc168542350)

[1.4.1. Mạng neuron nhân tạo 8](#_Toc168542351)

[1.4.2. AlphaZero 9](#_Toc168542352)

[1.4.3. Efficiently Updatable Neural Networks (NNUE) 11](#_Toc168542353)

[1.5. Kết luận 13](#_Toc168542354)

[CHƯƠNG 2. CÁC PHƯƠNG PHÁP ĐÁNH GIÁ VÀ TÌM KIẾM NƯỚC ĐI TRONG CỜ VUA 14](#_Toc168542355)

[2.1. Bộ sinh nước đi 14](#_Toc168542356)

[2.1.1. Tổng quan về bộ sinh nước đi 14](#_Toc168542357)

[2.1.2. Biểu diễn bàn cờ 14](#_Toc168542358)

[2.1.3. Nước đi giả lập (Pseudo-legal moves) 15](#_Toc168542359)

[2.1.4. Quân cờ không trượt (Non-sliding Pieces) 16](#_Toc168542360)

[2.1.5. Quân cờ trượt (Sliding Pieces) 17](#_Toc168542361)

[2.1.6. Các nước đi đặc biệt 20](#_Toc168542362)

[2.1.7. Biểu diễn và lưu trữ nước đi 21](#_Toc168542363)

[2.1.8. Sinh các nước đi hợp lệ 21](#_Toc168542364)

[2.1.9. Phát hiện chiếu hết và hòa cờ 22](#_Toc168542365)

[2.2. Cây tìm kiếm 23](#_Toc168542366)

[2.2.1. Tổng quan về cây tìm kiếm 23](#_Toc168542367)

[2.2.2. Negamax và cắt tỉa Alpha-Beta 24](#_Toc168542368)

[2.2.3. Quiescence Search 25](#_Toc168542369)

[2.2.4. Null-move pruning (NMP) 26](#_Toc168542370)

[2.2.5. Zobrist Hashing và Transposition Table 27](#_Toc168542371)

[2.2.6. Tìm kiếm biến thể chính (Principal Variation Search) 29](#_Toc168542372)

[2.2.7. Iterative deepening và Aspiration Windows 29](#_Toc168542373)

[2.2.8. Killer heuristic và History heuristic 30](#_Toc168542374)

[2.2.9. Thứ tự sắp xếp nước đi 31](#_Toc168542375)

[2.2.10. Late move reductions (LMR) 31](#_Toc168542376)

[2.3. Hàm lượng giá bàn cờ tĩnh 32](#_Toc168542377)

[2.3.1. Tổng quan về hàm lượng giá 32](#_Toc168542378)

[2.3.2. Giai đoạn cờ (Game phases) 32](#_Toc168542379)

[2.3.3. Quân và vị trí 33](#_Toc168542380)

[2.3.4. Khả năng di chuyển của quân (Mobility) 34](#_Toc168542381)

[2.3.5. Độ an toàn của quân Vua (King safety) 34](#_Toc168542382)

[2.3.6. Cột mở và nửa mở 34](#_Toc168542383)

[2.3.7. Cấu trúc tốt (Pawn structure) 35](#_Toc168542384)

[2.4. Hàm lượng giá bàn cờ sử dụng mô hình học sâu 35](#_Toc168542385)

[2.4.1. Tổng quan về học sâu 35](#_Toc168542386)

[2.4.2. Xây dựng mô hình 47](#_Toc168542387)

[2.5. Kết luận chương 54](#_Toc168542388)

[CHƯƠNG 3. XÂY DỰNG CHESS BOT 56](#_Toc168542389)

[3.1. Website Lichess.org 56](#_Toc168542390)

[3.1.1. Giới thiệu 56](#_Toc168542391)

[3.1.2. Lichess API 56](#_Toc168542392)

[3.1.3. Lichess Bot 57](#_Toc168542393)

[3.2. Kết nối Chess bot với website Lichess.org 58](#_Toc168542394)

[3.2.1. Thư viện Berserk 58](#_Toc168542395)

[3.2.2. UCI Protocols 58](#_Toc168542396)

[3.2.3. Cấu trúc của chương trình 59](#_Toc168542397)

[KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN 61](#_Toc168542398)

[TÀI LIỆU THAM KHẢO 62](#_Toc168542399)

**DANH MỤC KÝ HIỆU VÀ CHỮ VIẾT TẮT**

|  |  |  |  |
| --- | --- | --- | --- |
| **STT** | **Từ viết tắt** | **Tên đầy đủ** | **Giải thích** |
| 1 | AI | Artificial Intelligence | Trí tuệ nhân tạo |
| 2 | ANN | Artificial Neural Network | Mạng nơ-ron nhân tạo |
| 3 | CCRL | Computer Chess Rating List | Tổ chức xếp hạng phần mềm cờ vua |
| 4 | CEGT | Chess Engines Grand Tournament | Tổ chức xếp hạng phần mềm cờ vua |
| 5 | CNN | Convolutional Neural Network | Mạng nơ-ron tích chập |
| 6 | CP | Centipawn | 1 quân tốt = 100 centipawn |
| 7 | DL | Deep Learning | Học sâu |
| 8 | DNN | Deep Neural Network | Mạng nơ-ron sâu. |
| 9 | DRL | Deep Reinforcement Learning | Học sâu tăng cường |
| 10 | FEN | Forsyth-Edwards Notation | Quy ước cờ vua mô tả thế cờ/bàn cờ |
| 11 | LMR | Late Move Reductions | Cắt giảm nước đi “muộn” |
| 12 | LSTM | Long-Short Term Memory | Bộ nhớ dài-ngắn hạn |
| 13 | MCTS | Monte-Carlo Tree Search | Cây tìm kiếm Monte Carlo |
| 14 | ML | Machine Learning | Học máy |
| 15 | MLP | Multi-Layer Perceptron | Mạng Perceptron nhiều lớp |
| 16 | MVV-LLA | Most Valuable Victim – Least Valuable Aggressor | Quân cờ bị ăn có giá trị cao nhất và quân ăn có giá trị thấp nhất |
| 17 | NMP | Null Move Pruning | Nhường lượt cho đối thủ |
| 18 | NN | Neural Network | Mạng nơ-ron |
| 19 | NNUE | Efficiently Updatable Neural Networks | Mạng nơ-ron có khả năng cập nhật |
| 20 | PGN | Portable Game Notation | Định dạng tập tin ghi lại bàn cờ |
| 21 | ProbCut | Probability Cut | Cắt tỉa xác suất |
| 22 | PST | Piece Square Table | Bảng đánh giá quân cờ theo ô |
| 23 | PUCT | Probabilistic Upper Confidence bounds applied to Trees | Giới hạn độ tin cậy dựa trên xác suất |
| 24 | PV | Principal Variation | Biến thể chính |
| 25 | PVS | Principal Variation Search | Tìm kiếm biến thể chính |
| 26 | QS | Quiescence Search |  |
| 27 | RL | Reinforcement Learning | Học tăng cường |
| 28 | RNN | Recurrent Neural Network | Mạng nơ-ron hồi quy |
| 29 | TPT | Transposition Table | Bảng di chuyển đơn vị |
| 30 | UCI | Universal Chess Interface | Giao diện giao tiếp cờ vua phổ thông |

**DANH SÁCH HÌNH ẢNH**

[Hình 1.1. Bàn cờ vua tiêu chuẩn 13](#_Toc168541943)

[Hình 1.2. Cây tìm kiếm Minimax 16](#_Toc168541944)

[Hình 1.3. Monte-Carlo Tree Search 17](#_Toc168541945)

[Hình 1.4. Mối quan hệ giữa AI, ML, DL 18](#_Toc168541946)

[Hình 1.5. Mạng neuron nhân tạo 19](#_Toc168541947)

[Hình 1.6. Cấu trúc mạng neuron của AlphaGo Zero 20](#_Toc168541948)

[Hình 1.7. Cấu trúc của NNUE 22](#_Toc168541949)

[Hình 2.1. Biểu diễn bàn cờ bằng kí hiệu các quân 24](#_Toc168541950)

[Hình 2.2. Biểu diễn bàn cờ vua bằng bitboard 25](#_Toc168541951)

[Hình 2.3. Các nước đi giả lập khiến vua bị chiếu của bên đen 26](#_Toc168541952)

[Hình 2.4. Biểu diễn các nước đi của quân Mã 26](#_Toc168541953)

[Hình 2.5. Biểu diễn các nước đi của quân Vua 27](#_Toc168541954)

[Hình 2.6. Biểu diễn nước ăn chéo của quân tốt 27](#_Toc168541955)

[Hình 2.7. Các nước đi hợp lệ của quân Xe 27](#_Toc168541956)

[Hình 2.8. Biểu diễn các nước đi của các quân trượt 28](#_Toc168541957)

[Hình 2.9. Biểu diễn mặt nạ nước đi của quân xe 29](#_Toc168541958)

[Hình 2.10. Ví dụ minh họa về Magic bitboard 30](#_Toc168541959)

[Hình 2.11. Biểu diễn nước đi bằng các bit 31](#_Toc168541960)

[Hình 2.12. Logic kiểm tra nước đi hợp lệ 32](#_Toc168541961)

[Hình 2.13. Biểu diễn nước đi đầu tiên của bên trắng trên cây tìm kiếm 33](#_Toc168541962)

[Hình 2.14. Cây Negamax 34](#_Toc168541963)

[Hình 2.15. Thuật toán cắt tỉa Alpha-Beta 35](#_Toc168541964)

[Hình 2.16. Zugzwang 37](#_Toc168541965)

[Hình 2.17. Zobrist Hashing 38](#_Toc168541966)

[Hình 2.18. Đánh giá tapered 43](#_Toc168541967)

[Hình 2.19. So sánh ML và DL 46](#_Toc168541968)

[Hình 2.20. Quá trình lan truyền ngược trong mạng. 47](#_Toc168541969)

[Hình 2.21. Mạng MLP 48](#_Toc168541970)

[Hình 2.22. Mạng CNN 48](#_Toc168541971)

[Hình 2.23. Mạng RNN 50](#_Toc168541972)

[Hình 2.24. Đồ thị của các hàm activation Sigmoid, Tanh, ReLU, LeakyReLU 55](#_Toc168541973)

[Hình 2.25. Các mức độ phù hợp với dữ liệu 56](#_Toc168541974)

[Hình 2.26. Chuyển đổi giá trị từ Centipawn sang tỉ lệ WDL 59](#_Toc168541975)

[Hình 2.27. Mô hình NNUE đơn giản. 60](#_Toc168541976)

[Hình 2.28. Mô hình HalfKP đơn giản 61](#_Toc168541977)

[Hình 2.29. Training Loss và Validation Loss 63](#_Toc168541978)

[Hình 2.30. Dự đoán và thực tế trên 150 mẫu ngẫu nhiên từ tập valiation 63](#_Toc168541979)

[Hình 2.31. Biểu đồ hồi quy mô hình trên 150 mẫu ngẫu nhiên 64](#_Toc168541980)

[Hình 3.1. Chơi cờ trên website Lichess.org 66](#_Toc168541981)

[Hình 3.2. API thực hiện một nước đi cho Bot 67](#_Toc168541982)

[Hình 3.3. Cấu trúc của chương trình 69](#_Toc168541983)

[Hình 3.4. Chess Bot xây dựng đánh bại bản thân 70](#_Toc168541984)

**DANH SÁCH BẢNG BIỂU**

[Bảng 1.1. Một số phần mềm cờ vua và kỹ thuật tìm kiếm nổi bật 15](#_Toc168541985)

[Bảng 1.2. So sánh giữa Minimax và MTCS 17](#_Toc168541986)

[Bảng 2.1. Điểm thứ tự nước đi 41](#_Toc168541987)

[Bảng 2.2. So sánh tốc độ, tổng số nút duyệt, và nước đi tốt nhất của mô hình học sâu và hàm lượng giá tĩnh 65](#_Toc168541988)

# LỜI MỞ ĐẦU

**\*\*\***

Cờ vua đã luôn là một nguồn cảm hứng vô tận cho em trong hành trình khám phá trí tuệ nhân tạo. Từ những di chuyển nhỏ nhặt đến những chiến thuật phức tạp, trò chơi này không chỉ thách thức trí tuệ mà còn là một lĩnh vực thú vị để nghiên cứu và phát triển.

Trong phạm vi đề tài, em sẽ tập trung vào việc phân tích các thuật toán tìm kiếm trong cờ vua, như Negamax và Alpha-Beta Pruning, để hiểu rõ hơn về cách chúng hoạt động và cách áp dụng chúng vào cây trò chơi cờ vua. Đồng thời, em cũng sẽ nghiên cứu cách sử dụng mạng nơ-ron để đánh giá bàn cờ, cùng với phương pháp đánh giá bàn cờ truyền thống, hướng đến các đánh giá chính xác hơn.

Mục tiêu của đề tài không chỉ là xây dựng một Chess bot AI chơi cờ vua mạnh mẽ, mà còn là hiểu rõ hơn về cách kết hợp giữa thuật toán tìm kiếm và học sâu trong việc giải quyết các vấn đề phức tạp. Hy vọng rằng nghiên cứu này sẽ đóng góp vào việc phát triển các hệ thống AI thông minh và hiệu quả hơn, không chỉ trong lĩnh vực cờ vua mà còn trong nhiều ứng dụng thực tế khác.

Bố cục của đề tài gồm: phần mở đầu, phần thân (3 chương), phần kết luận và tài liệu tham khảo. Kết quả chính của đề tài tập trung ở chương 2 và chương 3. Cụ thể:

**Chương 1:** Trình bày tổng quan về đề tài, lịch sử và một số các công nghệ nổi bật của các thuật toán và AI trong cờ vua.

**Chương 2:** Phân tích các phương pháp đánh giá bao gồm bộ sinh nước đi, cây tìm kiếm, hàm lượng giá bàn cờ tĩnh hoặc sử dụng mô hình học sâu để đánh giá.

**Chương 3:** Xây dựng và cài đặt Chess Bot.

**Kết luận:** Trình bày tóm tắt được những kết quả mà đề tài đạt được, chỉ ra những điểm hạn chế, đưa ra những hướng phát triển trong tương lai.

# TỔNG QUAN ĐỀ TÀI

## Mô tả bài toán

### Lý do lựa chọn đề tài

Lý do chọn đề tài này xuất phát từ niềm đam mê của bản thân đối với cờ vua và các thuật toán trong trí tuệ nhân tạo, đồng thời em cũng muốn thử sức với lĩnh vực học sâu, một lĩnh vực đang phát triển rất nhanh ở thời điểm hiện tại. Cờ vua không chỉ là một trò chơi trí tuệ mà còn là một môn nghệ thuật của tư duy chiến lược, đòi hỏi khả năng phân tích và dự đoán. Chính những yếu tố này đã thu hút em không ngừng học hỏi và tìm hiểu sâu hơn. Khi tiếp cận với trí tuệ nhân tạo, em nhận thấy sự phát triển của AI trong cờ vua đã đạt được những bước tiến vượt bậc. Các máy tính đã vượt qua khả năng của các kỳ thủ hàng đầu trên thế giới, điều này đã khơi dậy sự tò mò của em trong việc muốn tìm hiểu rõ hơn về các thuật toán và công nghệ đứng sau những thành công này.

### Mục tiêu của đề tài

Trong đề tài này, em sẽ tập trung vào một số mục tiêu sau:

* Nghiên cứu cách sinh nước đi trong cờ vua nhanh với bitboard.
* Nghiên cứu về các thuật toán tìm kiếm trong cờ vua như Negamax, cắt tỉa Alpha-Beta, ...
* Nghiên cứu về các hàm lượng giá tĩnh cũng như xây dựng một mạng neuron sâu để có thể đánh giá bàn cờ.
* Áp dụng nghiên cứu để xây dựng Chess Bot AI chạy trên website Lichess.org

### Phạm vi của đề tài

Phạm vi nghiên cứu của đề tài tập trung nghiên cứu vào các kĩ thuật, thuật toán, hàm lượng giá, mạng neuron sâu liên quan đến cờ vua. Sau cùng Chess Bot sẽ có thể giao tiếp với người dùng thông qua website Lichess.org trên nền tảng áp dụng các nghiên cứu đã trình bày.

## Lịch sử và sự phát triển của trí tuệ nhân tạo trong cờ vua

Cờ vua là một trò chơi chiến lược hai người chơi trên một bàn cờ hình vuông với 64 ô, Mỗi người chơi bắt đầu với 16 quân cờ gồm một vua, một hậu, hai xe, hai mã, hai tượng và tám quân tốt. Mục tiêu của trò chơi là chiếu hết vua của đối thủ, đồng thời bảo vệ vua của chính mình không bị chiếu hết. Cờ vua được cho là có nguồn gốc từ Ấn Độ vào thế kỷ thứ 6 và đã trở thành một trong những trò chơi phổ biến nhất trên thế giới.



###### **Hình 1.1.** Bàn cờ vua tiêu chuẩn

1. **Giai đoạn tiền AI (trước năm 1950):**

* **Thế kỷ XVI**: Các nhà toán học bắt đầu nghiên cứu các vấn đề liên quan đến cờ vua, đặt nền tảng cho việc lập trình máy tính chơi cờ vua.
* **Thế kỷ XVIII**: Máy móc cơ học được chế tạo để mô phỏng các nước đi cờ vua, nhưng không có khả năng suy luận hay đưa ra quyết định.

1. **Giai đoạn sơ khai của AI (1950 – 1970):**

* **1950**: Alan Turing đề xuất một cách đo lường trí tuệ nhân tạo.
* **1957**: Allen Newell và Herbert Simon phát triển chương trình máy tính đầu tiên chơi cờ vua, General Problem Solver (GPS), sử dụng phương pháp cây tìm kiếm.
* **1968:** Chương trình máy tính Mac Hack VI của Richard Greenblatt đánh bại nhà vô địch cờ vua Hoa Kỳ Arnold Denker, sử dụng phương pháp đánh giá thế cờ và tìm kiếm bằng alpha-beta.

1. **Giai đoạn phát triển thuật toán tìm kiếm (1970 – 1990):**

* **1970:** Peter Cheng phát triển chương trình máy tính đầu tiên đánh bại một nhà vô địch cờ vua cấp tỉnh, sử dụng phương pháp tìm kiếm bằng alpha-beta nâng cao.
* **1978:** Chương trình máy tính máy tính Máy tính Belle của Larry Atkin và David Beal đánh bại nhà vô địch thế giới cờ vua Anatoly Karpov trong một ván đấu không chính thức, sử dụng phương pháp tìm kiếm Minimax.
* **1980:** Chương trình máy tính máy tính Mephisto của Hans Berliner trở thành chương trình máy tính đầu tiên đạt được mức Elo 2300, sử dụng mạng nơ-ron nhân tạo để đánh giá thế cờ.

1. **Giai đoạn bùng nổ phần cứng và phần mềm (1990 – 2010):**

* **1997:** Deep Blue của IBM đánh bại nhà vô địch thế giới Garry Kasparov trong một trận đấu sáu ván, sử dụng phần cứng máy tính mạnh mẽ và thuật toán tìm kiếm Minimax được tối ưu hóa.
* **2006:** Deep Fritz đánh bại nhà vô địch thế giới Vladimir Kramnik, khẳng định sự thống trị của AI trong cờ vua đỉnh cao, sử dụng chương trình học máy để cải thiện khả năng chơi cờ.
* **2007:** Chương trình máy tính Rybka của Vasik Rajlich và Jacek Urbaniec trở thành chương trình máy tính đầu tiên đạt được mức Elo 2800.

1. **Giai đoạn AI học máy (2010 – nay):**

* **2017:** AlphaZero của Google DeepMind đánh bại các nhà vô địch thế giới cờ vua, cờ tướng và cờ vây, sử dụng học tăng cường để tự học cách chơi cờ từ đầu.
* **2018:** Stockfish 8 trở thành chương trình máy tính đầu tiên đạt được mức Elo 3300, sử dụng kết hợp các kỹ thuật tìm kiếm, đánh giá và học máy tiên tiến**.**
* **2020:** Leela Zero, một chương trình máy tính cờ vua được huấn luyện bằng mạng nơ-ron nhân tạo, đánh bại Stockfish 8 trong một trận đấu, thể hiện tiềm năng to lớn của học sâu trong cờ vua.

Ta nhận thấy rằng, các thuật toán, các phần mềm cờ vua được phát triển không ngừng theo thời gian. Điều này một phần là do sự yêu thích của mọi người đối với trò chơi cờ vua, phần khác do cờ vua là một trò chơi tuyệt vời để có phát triển các thuật toán trong lĩnh vực trí tuệ nhân tạo. Theo hệ thống Fide, kỳ thủ cao nhất có mức ELO cờ cổ điển là Magnus Carlsen: 2834.5 ELO, nhưng những phần mềm cờ vua đã dễ dàng vượt qua con số này từ lâu.

Dựa trên kết quả tìm kiếm và tổng hợp từ xếp hạng các phần mềm cờ vua năm 2021 [1]:

**Bảng 1.1.** Một số phần mềm cờ vua và kỹ thuật tìm kiếm nổi bật

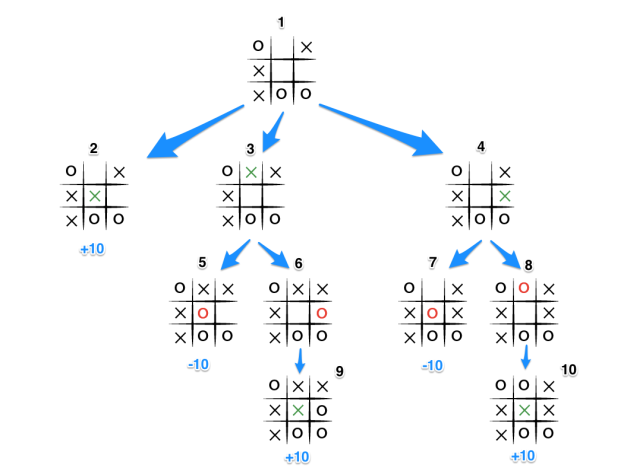
|  |  |  |  |
| --- | --- | --- | --- |
| **STT** | **Phần mềm** | **ELO  (ước tính)** | **Kỹ thuật tìm kiếm nổi bật** |
| 1 | Stockfish | 3550+ | NNUE, Alpha-Beta Pruning |
| 2 | AlphaZero | 3500+ | MCTS, Deep Reinforcement Learning (ResNet) |
| 3 | LC0 (Leela Chess Zero) | 3400+ | MCTS, Deep Learning (ResNet) |
| 4 | Komodo Dragon | 3350+ | Alpha-Beta Pruning, Deep Learning Integration |
| 5 | Houdini | 3200+ | Alpha-Beta Pruning, Selective Search |
| 6 | Fire | 3150+ | Alpha-Beta Pruning, Multi-Cut Search |
| 7 | Ethereal | 3100+ | Alpha-Beta Pruning, Late Move Reductions (LMR) |
| 8 | Xiphos | 3050+ | Principal Variation Search (PVS), LMR |
| 9 | Rybka | 3000+ | Alpha-Beta Pruning, Selective Search |
| 10 | Shredder | 2950+ | Alpha-Beta Pruning, ProbCut |

## So sánh giữa Minimax và Monte Carlo Tree Search

Trong lĩnh vực trí tuệ nhân tạo và trò chơi, các thuật toán tìm kiếm trên cây đóng vai trò quan trọng trong việc ra quyết định chiến lược. Mặc dù có rất nhiều loại cây tìm kiếm khác nhau, hai phương pháp phổ biến và nổi bật hơn cả vẫn là Minimax của phần mềm Stockfish và Monte Carlo Tree Search với trí tuệ nhân tạo AlphaZero. Ta sẽ nói khái quát một chút về hai phương pháp này.

Minimax là một thuật toán tìm kiếm cây được sử dụng trong trò chơi hai người chơi như cờ vua, cờ caro, tic-tac-toe, cờ tướng, cờ Shogi… Thuật toán này tập trung vào việc tìm kiếm không gian của trò chơi bằng cách đánh giá các nước đi tiếp theo và chọn ra nước đi tốt nhất cho người chơi hiện tại, giả định rằng đối thủ sẽ luôn chọn nước đi tốt nhất của mình. Quá trình này được thực hiện đệ quy, tạo thành một cây tìm kiếm với các nút là trạng thái của cây trò chơi và các cạnh là các nước đi. Thuật toán Minimax cần phải duyệt qua toàn bộ cây tìm kiếm để tìm ra nước đi tối ưu.

Tuy Minimax có thể tìm ra được nước đi tối ưu trong một trò chơi cụ thể, nhưng đối với các trò chơi có không gian tìm kiếm lớn, việc duyệt toàn bộ cây trò chơi là không khả thi. Do đó, Minimax thường được kết hợp với các kĩ thuật cắt tỉa như Alpha-Beta Pruning để cải thiện hiệu suất tính toán.

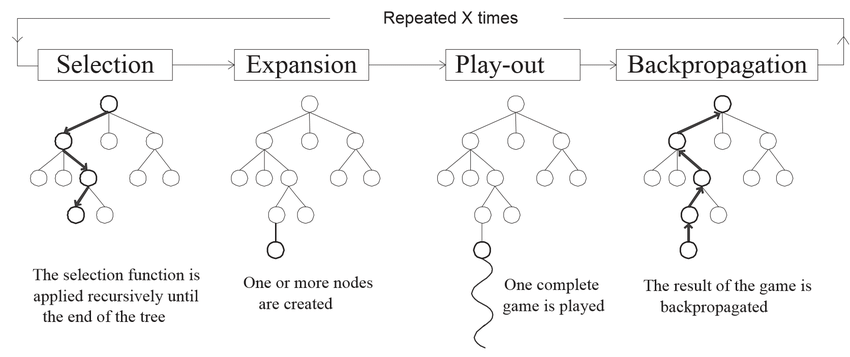


###### **Hình 1.2.** Cây tìm kiếm Minimax [2]

Monte-Carlo Tree Search (MCTS) là một phương pháp tìm kiếm cây dựa trên việc mô phỏng các lượt chơi ngẫu nhiên để xác định nước đi tiếp theo trong trò chơi. MCTS không yêu cầu đánh giá tất cả các nước đi, mà chỉ cần mô phỏng một số lượng lớn lượt chơi từ các trạng thái từ trờ chơi. Quá trình này được lặp đi lặp lại nhiều lần để xây dựng cây tìm kiếm, trong đó mỗi nút là một trạng thái của trò chơi và các cạnh là các nước đi.

MCTS bao gồm các bước chính như sau:

* **Chọn (Selection)**: bắt đầu từ trạng thái ban đầu, MCTS chọn các nút để mở rộng dựa trên một chiến lược chọn nút.
* **Mở rộng (Expansion)**: Một hoặc nhiều nút mới được thêm vào cây tìm kiếm, tương ứng với các nước đi chưa được khám phá từ nút được chọn.
* **Mô phỏng (Simulation)**: Từ các nút mới mở rộng, MCTS thực hiện một số lượng chơi ngẫu nhiên, gọi là mô phỏng để đánh giá chất lượng của nước đi.
* **Cập nhật (Backpropagation)**: Kết quả của các lượt chơi được truyền ngược lên cây tìm kiếm, giúp cập nhật ước lượng về chất lượng các nút.



###### **Hình 1.3.** Monte-Carlo Tree Search

MCTS không yêu cầu kiến thức về cấu trúc của trò chơi và thích hợp cho các trò chơi và thích hợp cho các trò chơi có không gian tìm kiếm lớn hoặc không xác định.

**Bảng 1.2.** So sánh giữa Minimax và MTCS

|  |  |  |
| --- | --- | --- |
| **Tiêu chí** | **Minimax** | **MCTS** |
| Hiệu suất tính toán | Đòi hỏi tính toán tất cả các nước đi | Chọn một số nước đi để mở rộng cây và lặp lại quá trình này |
| Hiệu suất trong trò chơi | Hiệu quả trong các trò chơi có không gian tìm kiếm giới hạn. | Thường được ưa chuộng trong các trò chơi với không gian tìm kiếm lớn hoặc không xác định |
| Khả năng đối phó với không gian tìm kiếm lớn | Gặp hạn chế với các trò chơi có không gian tìm kiếm lớn | Có khả năng xử lý tốt các trò chơi có không gian tìm kiếm lớn |
| Độ chính xác của quyết định | Có thể tìm ra nước đi tối ưu với số lượng nước đi đủ lớn | Không đảm bảo tìm ra nước đi tối ưu nhưng quyết định được đưa ra trong thời gian hợp lý |

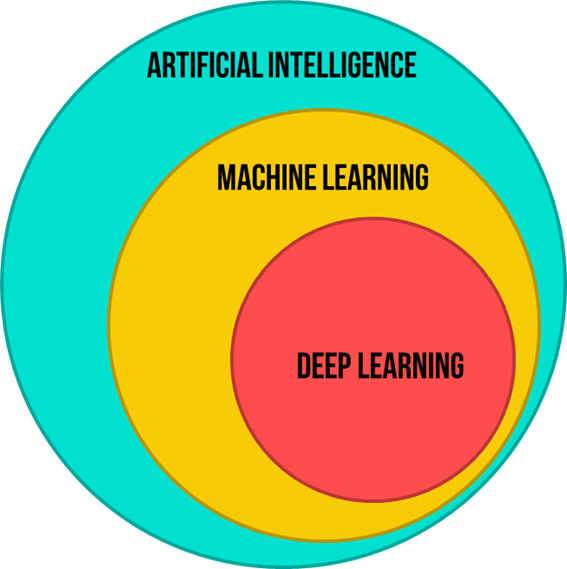
## Mạng neuron nhân tạo trong cờ vua

Mạng neuron nhân tạo (ANN) đã chứng minh sức mạnh của mình trong việc cải thiện khả năng chơi cờ vua của các chương trình máy tính. Trong đó có hai cái tên đặc biệt được chú ý nhiều nhất là AlphaZero và NNUE.

Mạng neuron đã trở thành một công cụ quan trọng trong lĩnh vực cờ vua, không chỉ để đánh giá trạng thái bàn cờ mà còn để cải thiện khả năng đánh giá và tìm kiếm nước đi của các chương trình chơi cờ.

### 1.4.1. Mạng neuron nhân tạo

Trong những năm trở lại đây, các khái niệm như học máy (Machine learning - ML), hay học sâu (Deep Learning – DL) đã trở nên phổ biến trong đời sống. Chúc đều thuộc về lĩnh vực trí tuệ nhân tạo (Artificial Intelligence). Cụm từ AI bắt đầu được nói đến nhiều sau hội nghị Dartmouth vào năm 1956. Từ đó trở đi lĩnh vực AI đã dành được sự thu hút lớn và phát triển cho đến tận ngày nay.



###### **Hình 1.4.** Mối quan hệ giữa AI, ML, DL

Ban đầu, các thuật toán áp dụng cho mô hình còn sơ khai, ít dữ liệu, các nhà khoa học chỉ áp dụng ML như một cách thực hiện phân tích dữ liệu. Một số mô hình của ML là thuật toán Linear Regression, Logistic Regression, Support Vector Machines (SVM), … Tuy nhiên hiệu quả của những mô hình này không cao, dễ bị nhiễu (noise). Để giải quyết vấn đề này các nhà khoa học đã sáng tạo ra mạng neuron (Neural network-NN), một mạng được đưa ra theo cơ chế của bộ não con người. Tuy nhiên, không hoàn toàn giống như mạng thần kinh của con người, NN đã được cải thiện hơn, nó có các lớp rời rạc, các kết nối, hướng truyền dữ liệu nhằm mở rộng và giải quyết nhiều nhu cầu.



###### **Hình 1.5.** Mạng neuron nhân tạo

Một Neural Network sẽ có 3 tầng chính là:

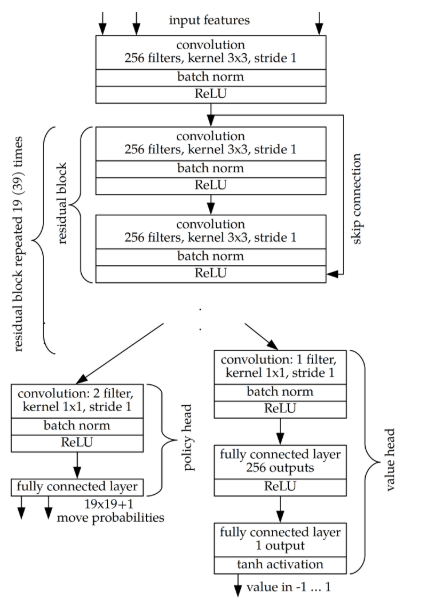
* **Tầng vào** (Input layer): là tầng cho các đầu vào của mạng
* **Tầng ra** (Output layer): là tầng cho các đầu ra của mạng.
* **Tầng ẩn** (Hidden layer): là tầng giữa tầng vào và tầng ra, thể hiện cho việc tính toán,suy luận của mạng để tìm ra đặc trưng.

### 1.4.2. AlphaZero

AlphaZero là một chương trình trí tuệ nhân tạo (AI) do DeepMind, một công ty con của Google, phát triển. Được giới thiệu lần đầu vào năm 2017, AlphaZero là phiên bản cải tiến từ AlphaGo và AlphaGoZero, những chương trình AI đã đạt được những thành công vượt trội trong trò chơi cờ vây. AlphaZero có khả năng chơi nhiều trò chơi khác nhau bao gồm cờ vua, cờ vây và shogi (cờ Nhật Bản) chỉ từ những quy tắc cơ bản mà không cần bất kỳ dữ liệu huấn luyện nào từ các trận đấu của con người.

Điểm đột phá của AlphaZero nằm ở khả năng học tập từ đầu (tabula rasa learning). Thay vì được lập trình với các chiến lược hoặc kiến thức cờ vua cụ thể, AlphaZero tự học các chiến lược và quy tắc chơi thông qua hàng triệu trận đấu với chính mình, từ đó liên tục cải thiện và tối ưu hóa lối chơi.

AlphaZero sử dụng một kiến trúc mạng nơ-ron sâu với hai thành phần chính: mạng chính sách (policy network) và mạng giá trị (value network). Các thành phần này được kết hợp trong một cấu trúc chung được gọi là Residual Neural Network (ResNet). Cấu trúc mạng của AlphaZero như sau:



###### **Hình 1.6.** Cấu trúc mạng neuron của AlphaGo Zero [3]

1. **Input Layer:**

* Bàn cờ 8x8 biểu diễn dưới dạng ma trận với các giá trị đại diện cho các quân cờ khác nhau.
* Các tính năng bổ sung như số bước đã đi, số quân cờ còn lại, …

1. **Residual Blocks:**

* AlphaZero sử dụng nhiều khối residual, mỗi khối gồm hai lớp convolutional
* Cấu trúc Residual giúp mô hình học được các đặc trưng phức tạp của bàn cờ mà không phải vấn đề mất mát gradient khi mạng trở nên sâu hơn [3]

1. **Policy head:**

* Một nhánh của mạng được gọi là “policy head” chịu trách nhiệm dự đoán cho nước đi tiếp theo.
* Lớp Convolutional với một số lượng kênh nhất định được sử dụng để trích xuất đặc trưng từ các khối residual.
* Lớp fully connected sau đó sẽ dự đoán xác suất của tất cả các bước đi hộp lệ từ trạng thái hiện tại của bàn cờ.

1. **Value head:**

* Chịu trách nhiệm ước lượng giá trị của bạn cờ hiện tại (tỉ lệ thắng của người chơi).
* Lớp Convolutional và một lớp fully connected để dự đoán giá trị (thắng, hòa, thua).

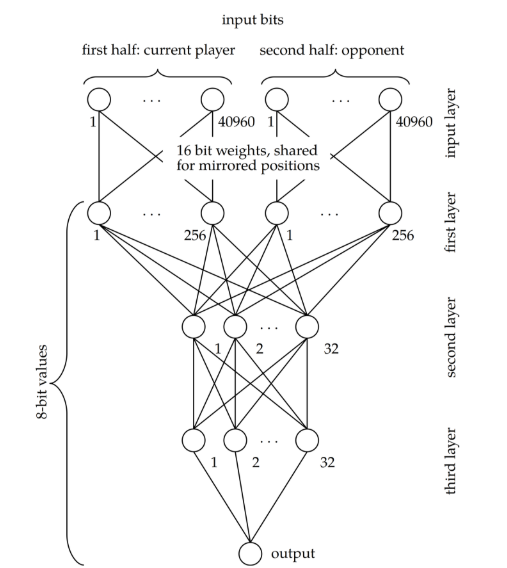
1. **Output:**

* Policy Head cung cấp một xác suất cho mỗi bước đi có thể đi, hướng AlphaZero về nước đi tiếp theo nên chọn.
* Value Head cung cấp một giá trị duy nhất đại diện cho đánh giá của AlphaZero về trạng thái bàn cờ hiện tại.

AlphaZero được huấn luyện thông qua quá trình tự chơi (self-play). Sử dụng thuật toán Monte Carlo Tree Search (MCTS) để khám phá các nước đi và trạng thái cờ tiềm năng. Các kết quả từ MCTS được sử dụng để cập nhật mạng neuron, cải thiện cả chính sách (lựa chọn bước đi) và giá trị (đánh giá trạng thái) của AlphaZero. AlphaZero chính là một ví dụ hoàn hảo về học sâu tăng cường.

### 1.4.3. Efficiently Updatable Neural Networks (NNUE)

NNUE là một cái tiến đặc biệt trong việc áp dụng mạng neuron nhân tạo vào các chương trình cờ vua. NNUE được thiết kế để tối ưu hóa hiệu quả tính toán và khả năng cập nhật, khác với mạng ResNet phức tạp được sử dụng trong AlphaZero và LC0. NNUE được phát triển ban đầu cho Shogi (cờ tướng Nhật Bản) nhưng sau đó đã được áp dụng thành công trong cờ vua, cờ tướng và cờ vậy, đáng chú ý nhất là trong các phiên bản gần đây của Stockfish, một trong những phần mềm chơi cờ mạnh nhất hiện này, sử dụng cây tìm kiếm Minimax kết hợp với thuật toán cắt tỉa Alpha-Beta cũng như biến thể của chúng.



###### **Hình 1.7.** Cấu trúc của NNUE [3]

1. **Đặc trưng đầu vào:**

* NNUE sử dụng các đặc trưng đầu vào đơn giản nhưng quan trọng từ bàn cờ như vị trí và loại quân cờ. Các đặc trưng này được mã hóa thành các vector đầu vào cho mạng.

1. **Các lớp kết nối hoàn toàn:**

* NNUE hướng đến các phần mềm cờ vua có thể hoạt động tốt ngay cả trên các máy có CPU không quá mạnh nên thường sẽ chỉ có các lớp Fully Connected thay vì các lớp Convolutional. Điều này giúp đơn giản hóa cấu trúc mạng và giảm tải tính toán.

1. **Lớp đầu ra**

* Lớp đầu ra của NNUE đưa ra đánh giá về trạng thái bàn cờ, thường là một giá trị duy nhất đại diện cho đánh giá của bàn cờ.

Thay vì phải huấn luyện toàn bộ mạng lại từ đầu với mỗi hình cờ, trong một trận đấu, NNUE sẽ chỉ cập nhật một tham số cụ thể dựa trên dữ liệu mới. Điều này giúp NNUE cập nhật nhanh chóng và linh hoạt hơn. Cụ thể ở đây là cập nhật sau mỗi nước đi cờ.

## Kết luận

Trong chương này, chúng ta đã tìm hiểu sơ qua về lịch sử và sự phát triển của trí tuệ nhân tạo trong lĩnh vực cờ vua. Từ những bước đầu tiên của máy tính chơi cờ cho đến những bước đột phá hiện đại như AlphaZero và NNUE, ta có thấy sự tiến bộ đáng kinh ngạc của công nghệ và thuật toán trong trò chơi này.

Kết quả từ các phần mềm cờ vua hàng đầu như Stockfish, AlphaZero đã chứng minh rằng sức mạnh của máy tính không chỉ giới hạn ở khả năng tính toán mà còn ở khả năng học và cải thiện từ kinh nghiệm chơi. Các phần mềm này đã thách thức và vượt qua những đỉnh cao của trò chơi, mở ra những triển vọng mới trong lĩnh vực trí tuệ nhân tạo và cơ hội cho những nghiên cứu và ứng dụng tiếp theo.

Từ những tìm hiểu sơ bộ này, ta có thể đưa ra kết luận như sau:

* Tất cả các phần mềm cờ vua đều sử dụng một cây tìm kiếm trò chơi cụ thể, có thể đó là cây Minimax với thuật toán cắt tỉa Alpha-Beta của Stockfish, hoặc MCTS của AlphaZero.
* Tất cả các phần mềm cờ vua đều sẽ đánh giá trạng thái cờ theo nhiều cách khác nhau và lựa chọn nước đi tốt nhất từ việc đánh giá hình thái cờ sau nước đi với các độ sâu khác nhau.

Từ đây em xác định được một phần mềm cờ vua mạnh được kết hợp bởi ít nhất hai yếu tố chính đó là: cây tìm kiếm và hàm đánh giá bàn cờ. Tuy vậy với ngôn ngữ sử dụng là Python, việc duyệt nước đi trên cây tìm kiếm có rất nhiều hạn chế và chậm chạp nếu sử dụng thư viện có sẵn. Vì vậy em cũng tìm hiểu thêm và biết được rằng hầu hết các phần mềm cờ vua đều được viết bằng ngôn ngữ C, đồng thời đều sử dụng và biểu diễn các nước đi bằng các phép bit để sinh các nước đi nhanh chóng. Vì tài nguyên của AlphaZero không được công bố cũng như có ít tài liệu nghiên cứu hơn về nó em quyết định xây dựng cây tìm kiếm Negamax (biến thể của Minimax) giống như Stockfish và các thuật toán xoay quanh nó. Tất cả những nghiên cứu của em về cách sinh nước đi, các thuật toán tìm kiếm cũng như là cách đánh giá bàn cờ sẽ được trình bày ở chương 2. Sau cùng, một Chess Bot được xây dựng hoàn chỉnh sẽ được trình bày tại chương 3.

# CÁC PHƯƠNG PHÁP ĐÁNH GIÁ VÀ TÌM KIẾM NƯỚC ĐI TRONG CỜ VUA

## 2.1. Bộ sinh nước đi

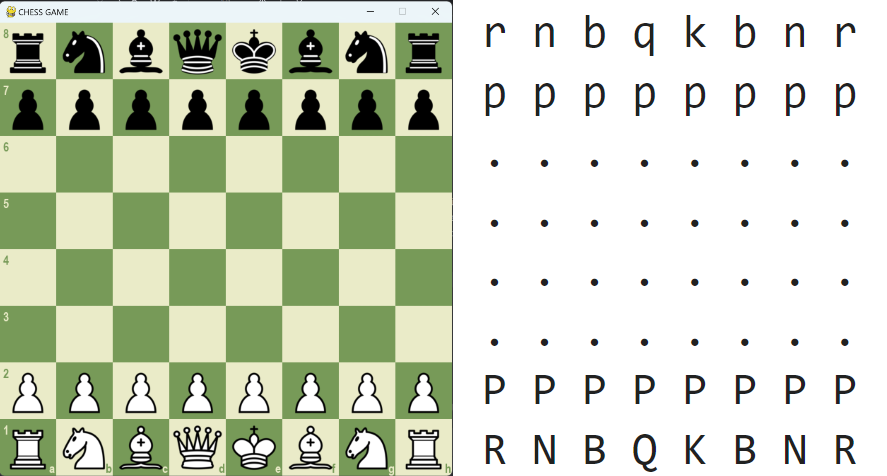
### 2.1.1. Tổng quan về bộ sinh nước đi

Việc sinh nước đi là phần thiết yếu trong các phần mềm cờ vua. Các lập trình viên rất linh hoạt trong việc thiết kế bộ sinh nước đi. Có rất nhiều cách tiếp cận để biểu diễn bàn cờ, dẫn đến các phong cách sinh nước đi khác nhau. Nhìn một cách tổng quan, việc sinh nước đi thường chiếm khoảng 15% tổng tính toán của phần mềm [4]. Điều này có nghĩa là, một phần mềm có thời gian sinh nước đi ngắn có thể nâng cao hiệu suất tính toán đáng kể. Tiếp theo em sẽ trình bày một số kĩ thuật để tối ưu thời gian sinh nước đi.

### 2.1.2. Biểu diễn bàn cờ

Bàn cờ vua gồm 64 ô vuông và có thể chứa tối đa 32 quân cờ, với sáu loại quân khác nhau cho mỗi bên. Bàn cờ có thể được biểu diễn theo nhiều cách. Cách đầu tiên là với mỗi quân cờ khác nhau, lưu tất cả các ô mà loại quân này ở vị trí đó, như vậy với quân tốt sẽ mất 8 ô nhớ, các quân xe, mã tượng sẽ mất 2 ô mỗi quân... Với 32 quân trên bàn cờ, cần 32 ô nhớ.

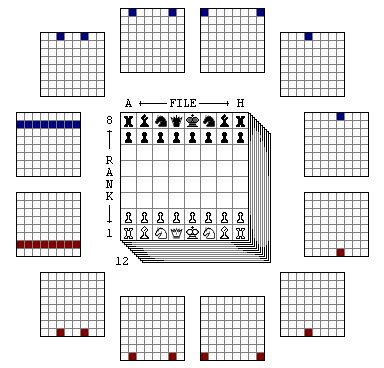
Cách thứ hai hay được sử dụng hơn là tạo một ma trận 8 \* 8, với các quân cờ được kí hiệu bằng số hoặc bằng chữ khác nhau. Dùng các kí hiệu lần lượt là P, N, B, R, Q, K tương ứng với các quân tốt, mã, tượng, xe, hậu, vua trắng, và các quân đen có cùng kí hiệu nhưng viết thường.



###### **Hình 2.****1.** Biểu diễn bàn cờ bằng kí hiệu các quân

Vì chỉ sử dụng một bàn cờ duy nhất để lưu trữ toàn bộ thông tin nên bộ nhớ lưu trữ không lớn. Tuy nhiên cách lưu trữ thứ 2 này mắc phải vấn đề về tốc độ truy xuất thông tin. Ví dụ, để lấy tất cả các quân cờ từ một bên cụ thể, ta phải duyệt qua toàn bộ danh sách, cụ thể ở đây là 64 ô, và việc này được thực hiện lặp đi lặp lại rất nhiều lần trong việc sinh nước đi. Điều này khiến ta cảm thấy cách thứ nhất dường như có hiệu quả, khi cần duyệt và lấy đúng tất cả các ô thuộc bên trắng hoặc bên đen. Ngược lại, nếu muốn xác định quân cờ nào chiếm giữ một ô vuông cụ thể cách thứ 2 dường như có lợi thế hơn. Vì vậy các phần mềm cờ vua hiện đại thường kết hợp cả hai lại.

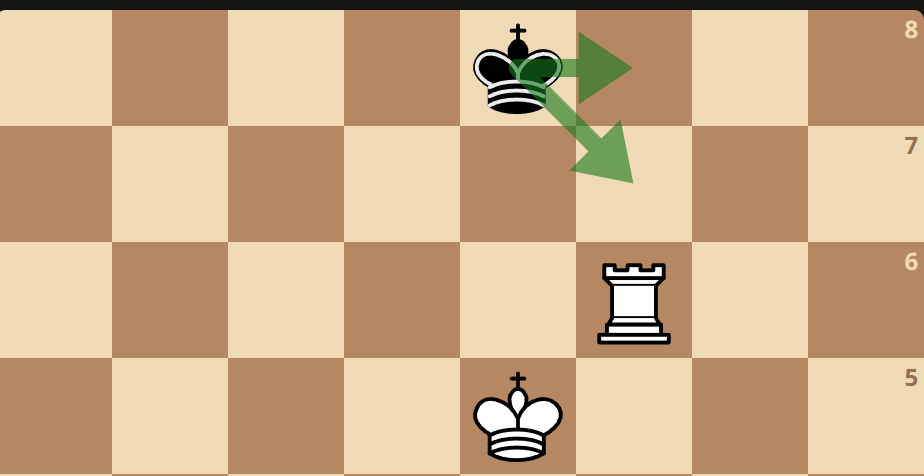
Nhận thấy rằng với cách thứ nhất, thay vì sử dụng 1 mảng để lưu trữ tất cả các vị trí của một loại quân, vì bàn cờ có 64 ô, coi các ô không chiếm đóng là số 0, các ô được loại quân này chiếm đóng là 1, như vậy cần 1 số nguyên 64 bits để biểu diễn vị trí của một loại quân trên bàn cờ. Cách triển khai này được gọi là bitboard. Các phép tính bitwise rất nhanh nên hiệu suất truy xuất thông tin được cải thiện đáng kể. Có 12 loại quân khác nhau, cần 12 số nguyên 64 bits, và nếu như muốn truy xuất vị trí của tất cả quân bên trắng, cần dùng 6 phép AND cho 6 số nguyên của 6 loại quân bên trắng.



###### **Hình 2.2.** Biểu diễn bàn cờ vua bằng bitboard

### 2.1.3. Nước đi giả lập (Pseudo-legal moves)

Việc đầu tiên cần làm khi sinh nước đi chính là tạo các nước đi giả lập. Nước đi giả lập là tất cả các quân có thể thực hiện được ở trên bàn cờ, bao gồm các nước đi khiến cho quân vua bị chiếu.



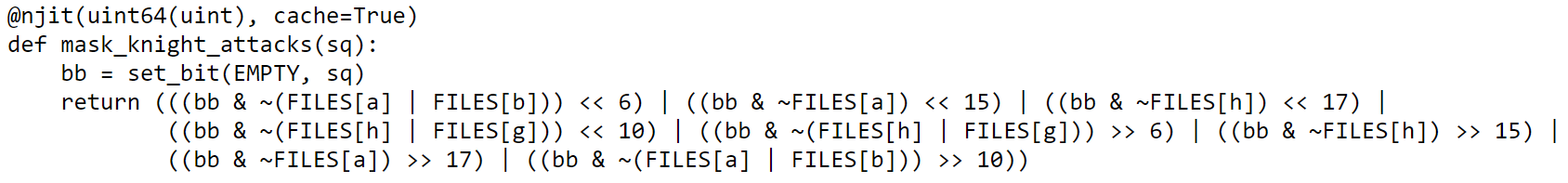
###### **Hình 2.3.** Các nước đi giả lập khiến vua bị chiếu của bên đen

Muốn xác định các nước đi giả lập cho một quân cờ cụ thể, khi đã sử dụng bitboard, có 2 cách để tính toán nước đi giả lập. Cách thứ nhất là sử dụng các phép toán bitwise theo các quy luật nhất định với mỗi lần tính toán. Cách thứ hai là với mỗi loại quân, lưu trữ các ma trận nước đi được tính toán từ trước và truy xuất mỗi khi cần trong runtime, đòi hỏi nhiều bộ nhớ hơn. Hầu hết các phần mềm cờ vua ở thời điểm hiện tại đều sử dụng các ma trận nước đi ở các dạng khác nhau, Với bộ nhớ hạn chế thì buộc phải sử dụng các thứ nhất. Với các quân có thể di chuyển theo quy luật nhất định như tốt, mã, vua có thể dễ dàng sinh các nước đi. Tuy nhiên với các quân cờ có thể di chuyển dọc theo bàn cờ như quân xe, tượng, hậu thì có nhiều khó khăn hơn.

### 2.1.4. Quân cờ không trượt (Non-sliding Pieces)

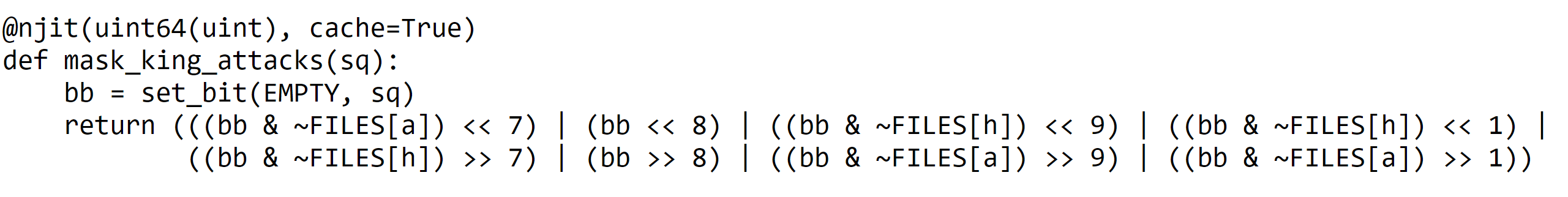
Trong cờ vua, các quân cờ không di chuyển trượt theo các đường bao gồm quân vua, quân mã và quân tốt.

Với quân mã, di chuyển theo hình chữ L, tính được các vị trí mà nó có thể di chuyển bằng cách dịch đi một số lượng bit cụ thể sang trái hoặc phải tùy theo hướng mà quân mã muốn đi. Nhưng trước đó cần kiểm tra xem quân mã có ở các vị trí biên hay không, bởi khi ở những vị trí này, số lượng nước đi của quân mã ít hơn.



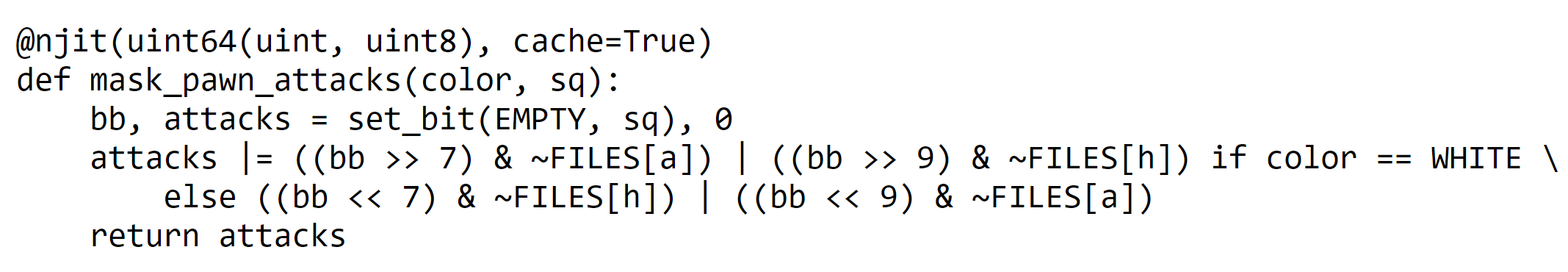
###### **Hình 2.4.** Biểu diễn các nước đi của quân Mã

Quân vua cũng vậy tương tự quân mã, cách di chuyển và ăn quân là giống nhau.



###### **Hình 2.5.** Biểu diễn các nước đi của quân Vua

Quân tốt đặc biệt hơn quân mã và vua ở chỗ nước đi di chuyển và ăn quân không giống nhau, quân tốt có thể di chuyển 2 bước lên phía trước nếu không có quân cản, hoặc một nước nếu như đã rời khỏi hang ngang số 2 hoặc số 7. Để tiến quân tốt trắng tiến lên 1 ô thì cần shift bit của quân tốt trắng này là 8 bits sang trái, hoặc với 2 ô là 16 bits, đồng thời kiểm tra các quân cản ở trước hoặc ở vị trí các bit này. Quân tốt có thể ăn chéo một ô, tuy nhiên phải kiểm tra thêm màu quân để có thể ăn đúng hướng.

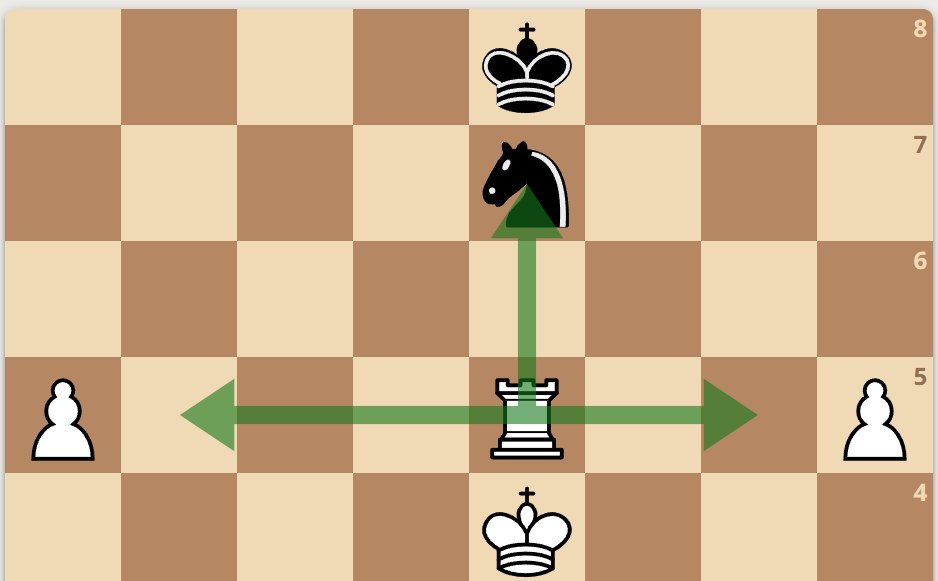


###### **Hình 2.6.** Biểu diễn nước ăn chéo của quân tốt

Với quân mã và vua, kiểm tra ô đến có quân đồng minh không bằng phép bitwise. Với quân tốt khi ăn quân kiểm tra xem ô ăn quân có quân địch hay không thì mới có thể thực hiện nước đi.

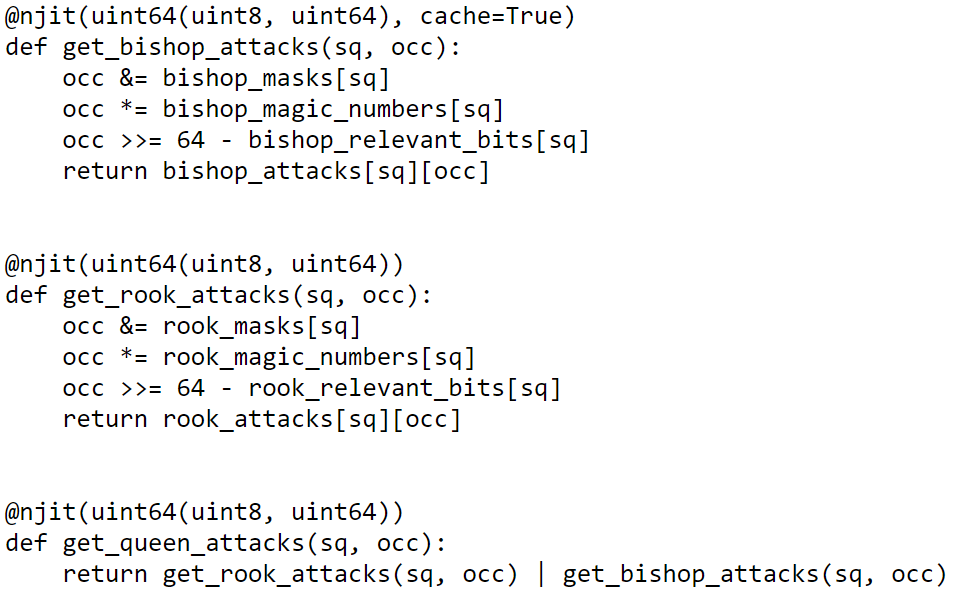
### 2.1.5. Quân cờ trượt (Sliding Pieces)

Bao gồm các quân hậu, xe, tượng. Đây là các quân cờ có thể di chuyển theo một số ô không giới hạn theo các đường ngang, dọc, chéo, bị giới hạn hay bị chặn bởi cả các quân bên mình và bên địch, đối với quân bên địch thì các quân này có thể mở ra thêm 1 ô nhờ việc ăn quân.



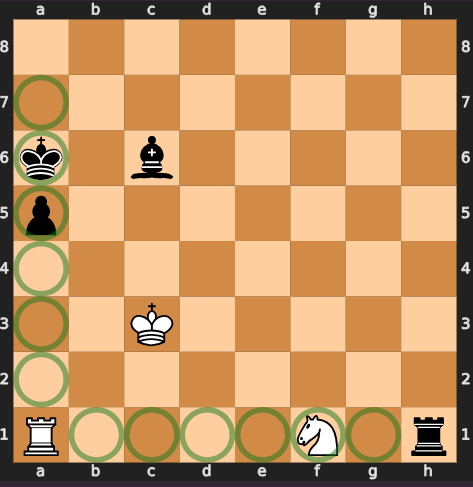
###### **Hình 2.7.** Các nước đi hợp lệ của quân Xe

Sử dụng Magic bitboard [5] để có thể tìm kiếm nước đi cho các quân này. Cách sử dụng Magic bitboard như sau:



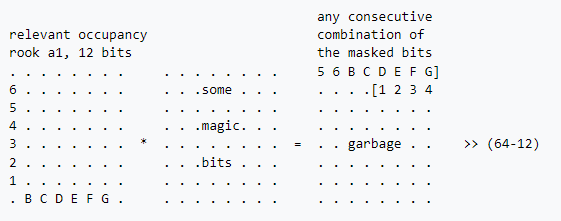
###### **Hình 2.8.** Biểu diễn các nước đi của các quân trượt

* “Mask” ở đây là 1 số nguyên 64 bits đại các ô mà quân, xe, tượng, hậu có thể di chuyển tới bất kể có quân cản hay không. Không cần quan tâm đến các ô ở biên, vì dù có quân ở biên hay không, thì nước trượt của các quân này cũng không bị các vị trí biên ảnh hưởng. Ví dụ như hình dưới đây:



###### **Hình 2.9.** Biểu diễn mặt nạ nước đi của quân xe

* “Occ” ở đây là tất cả các vị trí của các quân trên bàn cờ, cụ thể là các quân ở ô a1, a5, a6, c3, c6, f1, h1. Khi thực hiện phép & với “Mask” của quân xe, ta lấy ra được các bit gây cản trở cho việc trượt của quân xe. Cụ thể ở đây là các vị trí f1, a5, a6. Thực hiện phép & nay giúp lấy ra được các quân cản ở trên cùng trục tung và trục hoành với quân xe.
* Trong trường hợp này quân xe đang ở ô a1, từ b1 đến g1 có 6 bits, tương tự với a2 đến a7. Như vậy ở đây có tối đa 2 ^ 12 = 4096 trường hợp có quân cản hoặc không trong các vị trí bit này. Với mỗi trường hợp cản quân như vậy, lưu trữ 1 số nguyên 64 bits đại diện cho tất cả các vị trí mà quân xe có thể di chuyển đến trong trường hợp đó. Bàn cờ có 64 ô, như vậy cần 64 \* 4096 ô nhớ để có thể lưu trữ tất cả các nước đi của quân xe có thể đi. Với quân tượng là 64 \* 512 ô nhớ. Tuy nhiên với 1 bàn cờ có 64 bit, có cả trường hợp có quân ở ô a7, cần 1 chỉ mục lên đến lớn hơn 2 ^ 58, không thể lưu trữ chỉ với 4096 ô nhớ. Như vậy ta cần tìm cách ánh xạ các số bit lớn sang chỉ mục nhỏ hơn.
* Relevant bits ở đây là chỉ tổng số các ô có thể gây ảnh hưởng đến việc trượt của quân xe hoặc quân tượng. Ví dụ ở ô a1 thì quân xe có 12 ô ảnh hưởng, nhưng nếu ở ô a2, chỉ có 11 ô ảnh hưởng.
* Ở đây ta sẽ sử dụng 64 ô magic bitboards được sinh từ trước, các số này khi nhân với số nguyên biểu diễn các ô chiếm đóng sẽ có kết quả như sau:



###### **Hình 2.10.** Ví dụ minh họa về Magic bitboard [5]

* Sau khi dịch (64 – 12) bits sang phải ta có thể dễ dàng lấy ra được 12 bits cuối cùng có thể gây ảnh hưởng đến nước trượt của quân xe từ đó ánh xạ đến đúng vị trí cần thiết.
* Tương tự với quân tượng. Còn với quân hậu ta chỉ cần cần kết hợp các nước đi của xe và tượng với phép OR.

### 2.1.6. Các nước đi đặc biệt

Các nước đi đặc biệt trong cờ vua bao gồm: Phong cấp, nhập thành, đẩy tốt hai nước và bắt tốt qua đường.

**Phong cấp**: Xảy ra khi quân tốt đến hàng ngang cuối cùng của bên phía đối diện. Bắt buộc phải phong trở thành các quân, mã, tượng, xe, hậu. Lúc này xóa bit 1 trên số nguyên biểu diễn vị trí của quân tốt và thêm bit 1 vào cùng vị trí trên số nguyên của quân được phong.

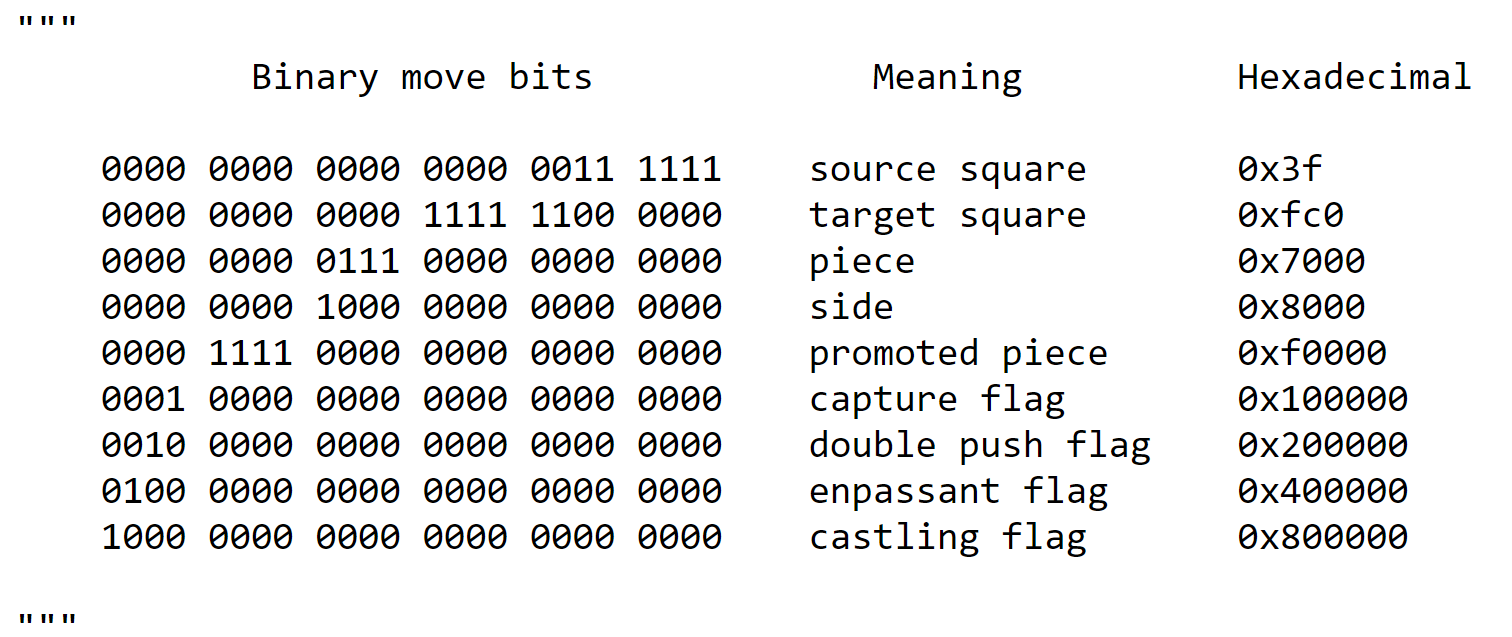
**Đẩy tốt hai nước**: Kiểm tra xem quân tốt có còn ở hàng ngang số 2 hoặc số 7 tùy theo bên quân, đồng thời kiểm tra các bit ở vị trí tiến lên 1 nước, hoặc 2 nước có quân nào khác hay không.

**Bắt tốt qua đường**: Khi một quân tốt được đẩy hai nước, lưu trữ ô phía sau của quân tốt đó là ô có thể thực hiện bắt tốt qua đường, tồn tại trong một nửa nước đi để cho bên còn lại quân tốt có thể thực hiện nước ăn đặc biệt này. Khi thực hiện bắt tốt qua đường, quân tốt được đẩy lên 2 nước trước đó sẽ bị xóa, đồng thời quân ăn sẽ ở vị trí ô vừa được lưu trữ.

**Nhập thành**: Các quân vua và xe chưa di chuyển, đồng thời trên đường vua di chuyển nhập thành không có quân nào chiếu và quân cản. Kiểm tra lần lượt các ô trên đường vua di chuyển có bị chiếu hay cản không nếu như quân vua và xe có thể nhập thành. Cập nhật lại tình trạng nhập thành sau khi thực hiện nước đi vua hoặc xe.

### 2.1.7. Biểu diễn và lưu trữ nước đi

Chỉ với 28 bits, có thể dễ dàng biểu diễn một nước đi trong cờ vua như sau:



###### **Hình 2.11.** Biểu diễn nước đi bằng các bit

Theo thứ tự từ phải sang trái có:

* 6 bits đầu tiên dùng để lưu trữ ô hiện tại (bàn cờ có 64 = 2 ^ 6 ô).
* 6 bits tiếp theo dùng để lưu trữ ô mà quên được di chuyển đến.
* Do mỗi bên có 6 loại quân, sử dụng 3 bits để encode cho quân: p: 0 (000), n: 1 (001), b: 2 (010), r: 3 (011), q: 4 (100), k: 5 (101).
* Sau đó sử dụng 1-bit tiếp theo để encode màu của quân hiện tại.
* Sử dụng 4 bits kế tiếp để lưu trữ quân phong cấp.
* Bit tiếp theo kiểm tra xem nước hiện tại có phải nước ăn quân không.
* Với các nước đi đặc biệt như đẩy tốt hai nước, bắt tốt qua đường, nhập thành, lần lượt dùng thêm 1-bit để kiểm tra.

### 2.1.8. Sinh các nước đi hợp lệ

Để sinh các nước đi hợp lệ trước hết cần sinh hết tất cả các nước đi giả lập và lưu trữ nó trong 1 danh sách. Sau đó chúng thử đi nước vừa rồi, bất kì nước nào khiến cho quân vua của bên mình bị chiếu là nước đi không hợp lệ và bị loại ra khỏi danh sách. Có thể kiểm tra trước các nước đi này trước khi đưa vào danh sách. Dưới đây là cách kiểm tra nước đi hợp lệ:



###### **Hình 2.12.** Logic kiểm tra nước đi hợp lệ

### 2.1.9. Phát hiện chiếu hết và hòa cờ

Trong cờ vua chiếu hết là khi quân của một bên bị bên còn lại chiếu và không có nước đi nào. Kiểm tra xem quân vua có bị chiếu hay không bằng cách kiểm tra vị trí của quân vua hiện tại có nằm trong phạm vi tấn công của các quân phía bên địch không sử dụng các ma trận nước đi được lưu trữ từ trước.

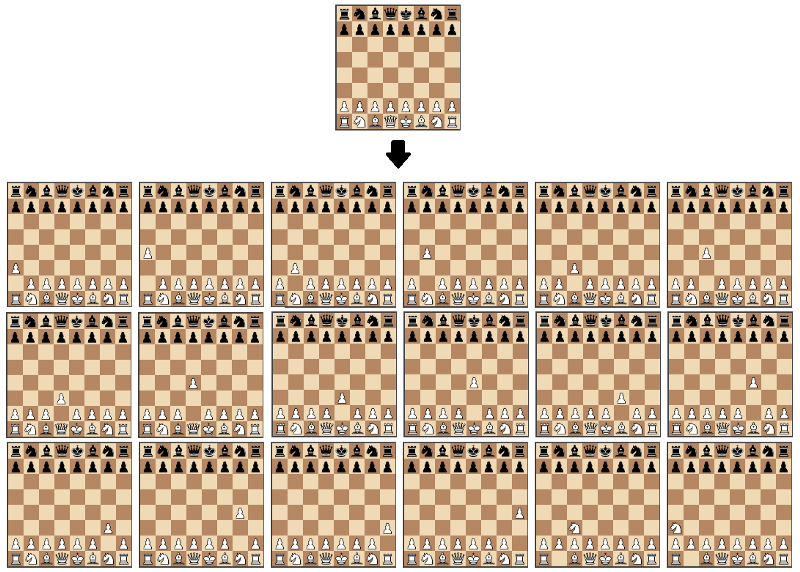
Xét về luật hòa cờ chúng có các luật sau:

* **STALEMATE**: Hòa cờ do một bên hết nước đi hợp lệ nhưng quân vua không bị chiếu. Hòa cờ giống chiếu hết ở chỗ một bên không còn nước đi hợp lệ, chỉ khác là quân vua trong trường hợp này không bị chiếu.
* **50 Moves**: Mỗi khi bên trắng hoặc bên đen đi, được tính là một nửa nước. Khi cả 2 đạt tổng cộng 50 nước đi mà không xảy ra sự ăn quân, hay không có nước đẩy tốt thì buộc hòa cờ. Với mỗi trạng thái cờ lưu trữ thêm một biến đếm và reset nó khi cần để giải quyết vấn đề này.
* **Insufficient material**: Hai bên bị buộc hòa cờ do không bên nào có đủ quân để dành chiến thắng. Ví dụ: 2 bên chỉ còn quân vua và không còn quân nào khác.
* **Threefold repetition**: Lặp trạng thái cờ ba lần. Ở đây có 2 hướng giải quyết, một là với mỗi 1 trận đấu chỉ dùng 1 từ điển duy nhất để lưu trữ toàn bộ các trạng thái cờ đã xảy ra, nếu như vậy thì khi phần mềm duyệt trên cây tìm kiếm không thể phát hiện ra hòa cờ để đánh giá, vì vậy buộc phải lưu lại tất cả các trạng thái cờ trước đó vào 1 trong mỗi trạng thái của bàn cờ. Hậu quả là bộ nhớ sử dụng có thể sẽ rất lớn. Để giải quyết vấn đề này có thể dùng **Zobrist Hashing** để băm bàn cờ ra lưu trữ. Phương pháp này được nói đến ở phần kế tiếp.

## 2.2. Cây tìm kiếm

### 2.2.1. Tổng quan về cây tìm kiếm

Một phần quan trọng kế tiếp của một Chess Bot mạnh chính là xây dựng khung cây tìm kiếm. Trong cờ vua việc tìm kiếm nước đi có thể biểu diễn đơn giản bằng một cây trò chơi hoặc cây tìm kiếm, với mỗi nút là trạng thái bàn cờ, và các nút con là trạng thái bàn cờ sau khi thực hiện các nước đi hợp lệ khác nhau.



###### **Hình 2.13.** Biểu diễn nước đi đầu tiên của bên trắng trên cây tìm kiếm

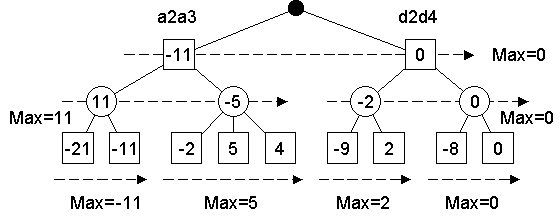
Giống với việc phát triển các bộ sinh nước đi, có không ít các thuật toán khác nhau để cải thiện khả năng tìm kiếm nước đi quân cờ, tuy nhiên việc áp dụng quá nhiều loại thuật toán khác nhau không phải lúc nào cũng đảm bảo sẽ giúp ta tìm được nước đi tốt nhất một cách hiệu quả và chính xác, điều này phụ thuộc rất nhiều vào hàm lượng giá bàn cờ. Trong các phần mềm cờ vua hiện đại, có rất nhiều loại cây tìm kiếm khác nhau, nhưng tựu chung, nổi bật nhất vẫn là sử dụng cây Minimax với phần mềm Stockfish, MCTS và Deep Reinforcement Learning của AI cờ vua AlphaZero [3] như đã nói ở chương 1.

Vì cây tìm kiếm dưới dạng **Minimax/Negamax** được cho là có khả năng tìm kiếm nước đi nhanh hơn nên em đã chọn cây tìm kiếm theo hướng này, đồng thời áp dụng thêm các thuật toán khác để tạo ra được một cây tìm kiếm nước đi hiệu quả. Tiếp theo, em sẽ tiếp tục trình bày chi tiết về các thuật toán được áp dụng trong Chess Bot của bản thân.

### 2.2.2. Negamax và cắt tỉa Alpha-Beta

#### 2.2.2.1. Negamax

Negamax chính là ý tưởng cốt lõi cơ bản trong cây tìm kiếm, tất cả các thuật toán phía sau đều nhằm cải thiện hiệu suất của thuật toán Negamax. Negamax là một thuật toán tìm kiếm cây trạng thái thường được sử dụng trong cờ vua và các trò chơi chiến lược khác. Nó là phiên bản đơn giản của thuật toán Minimax và được phát triển bởi Donald Knuth vào những năm 1970.

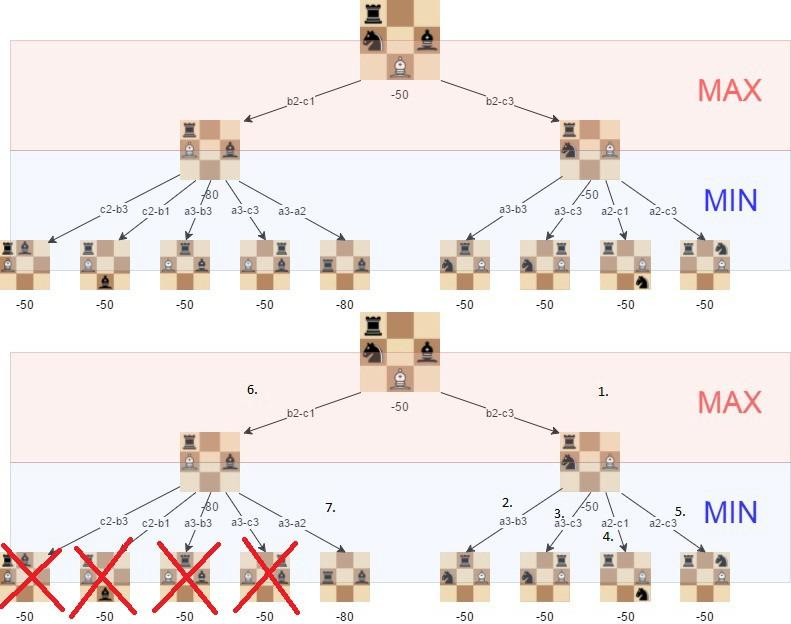


###### **Hình 2.14.** Cây Negamax

Negamax tìm kiếm nước đi tốt nhất bằng cách giả định rằng đối thủ cũng sẽ đi nước tốt nhất, tìm kiếm các nước đi cho đến một độ sâu nhất định. Với mỗi nút trên cây tìm kiếm, các nước đi sẽ tạo nên liên kết đến các nút khác, mỗi nút này sẽ được đánh giá bằng một hàm lượng giá. Trong Negamax, điểm số của đối thủ được tính là âm của điểm số của người chơi hiện tại. Điều này đơn giản hóa việc lập trình vì chỉ cần một hàm đánh giá duy nhất. Điểm mạnh của Negamax là nó sẽ duyệt không bỏ sót trường hợp nào. Điểm yếu của negamax chính là nó rất chậm vì nó luôn tìm kiếm mọi nút cho đến một độ sâu cố định. Chính vì thế Negamax thường được kết hợp với Alpha-beta pruning để cải thiện hiệu suất tìm kiếm.

#### 2.2.2.2. Thuật toán cắt tỉa Alpha-Beta

Alpha-Beta Pruning là một kỹ thuật cắt nhánh được sử dụng để cải thiện hiệu suất của thuật toán Minimax/Negamax trong cờ vua và các trò chơi chiến lược khác. Kỹ thuật này giúp giảm số lượng nút cần phải kiểm tra trong quá trình tìm kiếm, từ đó tăng cường hiệu suất tìm kiếm mà không làm ảnh hưởng đến tính đúng đắn của kết quả.



###### **Hình 2.15.** Thuật toán cắt tỉa Alpha-Beta

Ý tưởng của thuật toán cắt tỉa Alpha-Beta rất đơn giản. Người chơi theo dõi điểm số tốt nhất từ phía đối thủ. Nếu một người chơi gặp phải tình huống mà điểm số của họ vượt quá điểm số tốt nhất từ đối thủ có thể đạt được, họ có thể giả định rằng đối thủ sẽ không muốn người chơi thực hiện nước đi đó vì họ muốn tối đa hóa số điểm của mình. Nói cách khác, ta không cần tìm kiếm và so sánh các nước đi tệ hơn nước đi tốt nhất của đối thủ, hay chính là thực hiện việc cắt tỉa. Ta theo dõi việc cắt tỉa này bằng chính hai biến alpha và beta. Phạm vi tốt nhất của các giá trị đánh giá sẽ nằm trong khoảng giá trị tạo nên bởi hai biến này. Nếu như giá trị đánh giá vượt ra ngoài khoảng, thực hiện việc cắt tỉa, cập nhật alpha và beta tương ứng. Ta dễ dàng nhận thấy, nếu như nước đi tốt nhất được sắp xếp ở trước, thì nhiều cây con sẽ bị cắt tỉa hơn (Cắt tỉa Beta). Điều này chứng minh rằng thứ tự sắp xếp nước đi là vô cùng quan trọng. Thông thường Alpha-Beta sẽ tăng trung bình được 33% độ sâu tính toán xét trên cùng thời gian tìm kiếm.

### 2.2.3. Quiescence Search

Khi tìm kiếm đến một độ sâu nhất định rất có thể sẽ dẫn đến hiệu ứng đường giăng (horizontal effect). Giả sử trước đó với độ sâu 4 máy đánh giá bản thân có lợi thế hơn 300 centipawn sau khi ăn được một quân mã bằng quân hậu và dừng việc tìm kiếm ở đây, nhưng ngay ở nước sau người chơi có thể sử dụng quân tốt để ăn lại quân hậu. Như vậy việc dừng tìm kiếm trực tiếp ở một độ sâu cụ thể sẽ dẫn đến kết quả đánh giá sai một nước đi là mạnh. Quiescence Search (QS) chính là một thuật toán giúp chúng ta giải quyết vấn đề đường giăng [6] .Ý tưởng đằng sau thuật toán này là tìm kiếm cho đến khi nào đạt đến trạng thái “ổn định”. Thông thường “Ổn định” là khi bàn cờ không còn nước ăn quân, nhưng đôi khi có thể là nước chiếu… Thuật toán này không có nhiều khác biệt so với negamax. Sau khi tìm kiếm negamax đến độ sâu yêu cầu, ta tiếp tục thực hiện negamax với các nước đi ăn quân, đây chính là QS. QS vô cùng hiệu quả nếu muốn giải quyết horizontal effect, tuy nhiên cái giá phải trả là đôi khi nó sẽ gây ra bùng nổ không gian tìm kiếm. Mỗi nước đi trong QS đều là nước bắt quân, ta có xu hướng đẩy các nước đi ăn quân lên phía trước trong danh sách các nước đi hợp lệ.

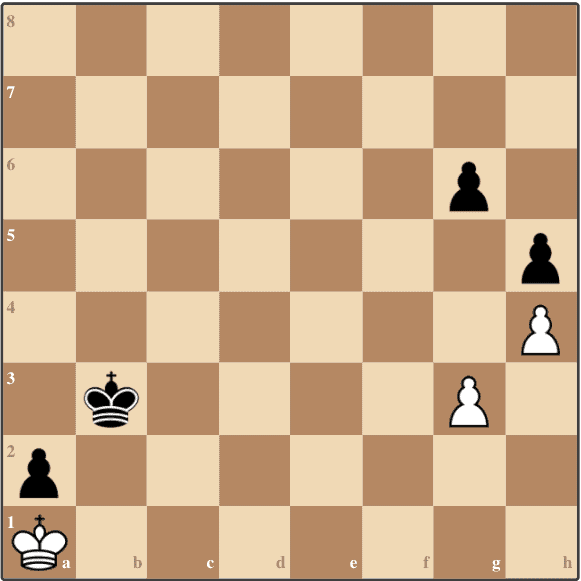
Cách đơn giản nhất để xác định nước đi bắt quân nào tốt hơn thường được biết đến là MVV-LVA. Giống như tên gọi, ta sắp xếp các nước đi bắt quân có giá trị quân ăn thấp nhất và quân bị ăn cao nhất. Hiểu đơn giản là ta sẽ muốn dùng quân tốt ăn quân tốt hơn là dùng quân mã ăn quân tốt. Đây là một kĩ thuật đơn giản nhưng hiệu quả trong việc sắp xếp nước đi bắt quân, nó làm giảm bùng nổ không gian tìm kiếm vì các nước đi tốt nhất sẽ ở đầu danh sách.

### 2.2.4. Null-move pruning (NMP)

Ngoài cắt tỉa Alpha-Beta, chúng ta còn rất nhiều các kĩ thuật để có thể giảm các nút tìm kiếm trên cây, một trong số chúng chính là Null-move pruning (NMP). Giống như tên gọi, ý tưởng của NMP là ta sẽ thực hiện bỏ lượt đi cho một bên. Nếu hiện tại là bên trắng, ta không thực hiện nước đi nào mà nhường lại lượt cho bên đen. Nếu như người không đi nước nào, nhưng mà điểm đủ cao để có thể thực hiện cắt tỉa beta, có thể nói việc chơi nước đi tốt nhất cũng có thể tạo ra cắt tỉa beta. Nếu đặt thuật toán này một mình thì nó sẽ không có hiệu quả, nếu đã tìm thấy nước đi tốt nhất thì có thể nó cũng đã gây ra cắt tỉa beta trước đó. Điểm có lợi của NMP chính là các nút được duyệt dưới NMP sẽ chỉ cần duyệt với độ sâu ít hơn và tất nhiên, ít thời gian hơn. Lý do ta có thể duyệt ít sâu hơn sau NMP chính là việc 2 lượt chơi liên tiếp sẽ khiến cho đối thủ tăng nhanh về hàm lượng giá. Thông thường, độ sâu sẽ giảm đi từ 2 đến 3.

Có rất nhiều tranh cãi xoay quanh NMP, điều này phụ thuộc chủ yếu vào quan điểm của lập trình viên. Một trong số chúng là việc có cho thực hiện các nước đi Null liên tiếp hay không, hoặc cân nhắc khi nào nên sử dụng NMP, khi mà một người chơi bị chiếu, NMP sẽ trở nên không hợp lệ. Đối với bản thân, em chọn cách là kiểm tra xem độ sâu có lớn hơn 3 hay không, đồng thời kiểm tra xem người chơi có đang bị chiếu hay không. Nếu không bị chiếu sẽ thực hiện Negamax với độ sâu giảm đi 2, việc mất mát thông tin có thể xảy ra.

Một lý do nữa cho việc không nên sử dụng NMP chính là khái niệm zugzwang (tất cả các nước đi của bản thân đều bất lợi) [7]



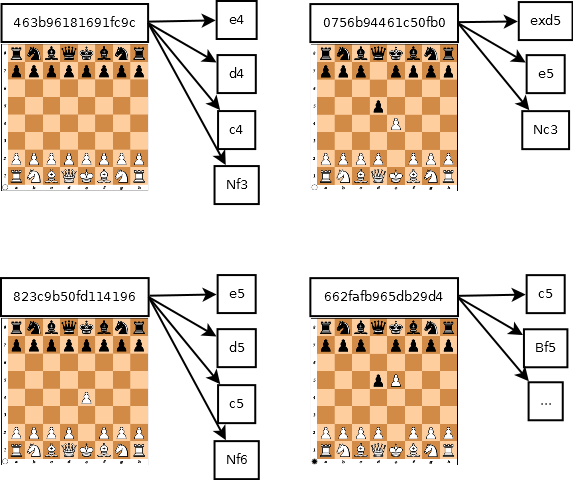
###### **Hình 2.16.** Zugzwang

Ví dụ trong hình cờ này, đến lượt trắng đi, nước đi duy nhất của trắng chính là tốt g4, nhưng điều này sẽ khiến đen có thêm một con tốt thông xuống để phong hậu, nói cách khác trắng không có bất kì nước đi có lợi nào. Như vậy việc nhường nước đi trong trường hợp này là cũng hoàn toàn bất hợp lý. Khi loại bỏ NMP ra khỏi Chessbot, em nhận thấy tốc độ bị giảm đi nhưng độ chính xác không hề tăng, vì vậy em quyết định vẫn áp dụng NMP trong cấu trúc cây tìm kiếm của mình.

### 2.2.5. Zobrist Hashing và Transposition Table

#### 2.2.5.1. Zobrist Hashing

Như đã nói ở phần 2.1.8, Zobrist Hashing chính là 1 kĩ thuật giúp chúng ta lưu trữ toàn bộ thông tin bàn cờ, không bao gồm danh sách các nước đi.



###### **Hình 2.17.** Zobrist Hashing

Zobrist Hashing là một kỹ thuật được sử dụng để ánh xạ trạng thái của bảng cờ với một giá trị hash duy nhất. Kỹ thuật này dựa trên việc sử dụng một bảng các số ngẫu nhiên được tạo ra trước, gọi là Zobrist keys. Mỗi ô trên bảng cờ được ánh xạ với một số ngẫu nhiên trong bảng Zobrist keys. Khi trạng thái của bảng cờ thay đổi, chỉ cần thay đổi giá trị của các ô đã thay đổi và tính toán lại giá trị hash. Việc này giúp nhanh chóng so sánh trạng thái của bảng cờ và phát hiện các trạng thái đã thăm qua trước đó. Về cơ bản Zobrist Keys là một số 64 bits độc nhất. Ban đầu, một trạng thái cờ là một số nguyên ngẫu nhiên. Sau đó với mỗi nước đi ta lại thực hiện phép XOR với các giá trị như key nhập thành, key bắt tốt qua đường, đầu và cuối của một nước đi tùy theo cách lập trình. Kết quả cuối cùng sẽ là một mã 64 bits độc nhất.

#### 2.2.5.2. Transposition Table (TPT)

Chúng ta đã có hàm băm Zobrist Hashing, ta sẽ cần một nơi để lưu trữ những Keys này và tất nhiên sẽ cần những value theo cùng chúng. Ý tưởng cơ bản đằng sau Transposition Table là một bảng tra cứu các thông tin liên quan đến 1 thế cờ được lưu với Zobrist Key. Thông thường khi lưu ZobristKeys, ta sẽ lưu cùng điểm đánh giá, độ sâu mà nó được duyệt trên cây, thông tin về nút (nút cắt tỉa alpha, beta, hay nút exact…). Khi duyệt các nước đi cờ vua với depth 4, có thể có rất nhiều hình cờ trùng lặp với nhau. TPT là 1 công cụ tuyệt vời để lưu trữ tất cả hoặc một số các thế cờ đã đánh giá, từ đó loại bỏ việc tính toán trùng lặp. Dựa trên kết quả đánh giá chủ quan sau khi áp dụng TPT, việc tìm kiếm sẽ có hiệu suất tốt hơn từ 1,5 đến 2,5 lần trước đó. Các nút trên TPT có khái niệm flag. Nếu flag này là alpha thì chỉ ra rằng điểm số là cận dưới, nếu alpha hiện tại vượt quá alpha đã lưu, thì tìm kiếm tại nút này không thể cải thiện và có thể bị cắt. Nếu flag là beta thì chỉ ra rằng đây là điểm giới hạn trên, nếu Beta hiện tại nhỏ hơn thì cũng có thể xảy ra cắt tỉa. Nếu flag là exact thì chỉ ra rằng điểm số là giá trị chính xác, trường hợp này luôn có thể lấy giá trị đã được lưu cùng, tuy nhiên chỉ với trường hợp độ sâu tìm kiếm phải ít nhất bằng độ sâu hiện tại, nếu không, vị trí hiện tại có thể có số điểm cao hơn vì tìm kiếm sâu hơn [8].

### 2.2.6. Tìm kiếm biến thể chính (Principal Variation Search)

Trong Negamax, tất cả các nước đi đều được tìm kiếm trong giới hạn của alpha và beta. Ngoài việc cắt tỉa gán trực tiếp giá trị, đôi khi ta có thể giả định các giá trị alpha beta này. Và đây chính xác là điều mà PVS làm. Thay vì tìm kiếm mỗi nước đi với giới hạn alpha và beta, ta tìm kiếm theo một chuỗi các nước đi với toàn bộ giới hạn. Chuỗi các nước đi này gọi là Principal Variation (PV), chuỗi các nước đi tốt nhất được tìm thấy [9]. Sau đó, tất cả các nút còn lại được tìm kiếm với một giới hạn chặt hơn, được gọi là null windows. Thông thường các bounds này là từ alpha – 1 đến alpha, thay vì từ alpha đến beta. Ý tưởng của PVS là thay vì tìm kiếm nước đi tốt nhất, tìm kiếm cố bác bỏ việc danh sách các nước đi hiện tại là nước đi tốt nhất. Với một giới hạn nhỏ hơn, cắt tỉa beta sẽ diễn ra thường xuyên hơn, tốc độ tìm kiếm nhanh hơn. Tuy nhiên, giá trị của nút có thể vượt ra được null windows hiện tại, chúng ta vẫn phải tìm kiếm lại như bình thường mà không dùng null windows vì giới hạn quá chặt. Sau khi tìm kiếm lặp lại, các giá trị alpha, beta sẽ được điều chỉnh tương ứng, đồng thời thay đổi PV. Việc tìm kiếm lặp đi lặp lại sẽ rất tốn thời gian, vì vậy sử dụng các nước đi tốt nhất được lưu lại trên nhánh PV sẽ là lựa chọn tối ưu hơn. Một phương pháp lưu trữ nhánh các nước đi tốt nhất là sử dụng bảng PV tam giác, dễ hiểu hơn là lưu trữ một mảng cho mỗi độ sâu tìm kiếm, trong đó mảng gồm nhiều nước tốt nhất và điểm tốt nhất.

### 2.2.7. Iterative deepening và Aspiration Windows

#### 2.2.7.1. Iterative deepening

Các Chess bot hầu hết đều chơi dựa trên giới hạn thời gian, điều này đặt ra nhiều vấn đề cho cây tìm kiếm. Nếu cây tìm kiếm đang tìm kiếm đến một độ sâu cố định và hết thời gian, bất kì nước đi nào tìm thấy ở thời điểm đó đều không đáng tin cậy vì không xem xét mọi nước đi. Iterative deepening là một kĩ thuật giải quyết vấn đề này. Ý tưởng đằng sau thuật toán này rất đơn giản, đó là khi được yêu cầu tìm kiếm ở độ sâu D, ta sẽ không bắt đầu tìm kiếm đến độ sâu D ngay lập tức mà thay vào đó ta sẽ tìm kiếm từ độ sâu từ 1 đến D. Trong trường hợp hết thời gian trước độ sâu k, ta có thể lấy kết quả tìm kiếm từ độ sâu k – 1 trước đó đã tìm ra. Thoạt nhìn ta tưởng việc tìm kiếm lặp lại như vậy thường tốn tài nguyên và chắc chắn phải chậm hơn bình thường. Nhưng thực tế không phải vậy, thuật toán này đã cải thiện độ chính xác của Chess bot lên khá nhiều. Lý do cho việc này là do trước đó với PVS và thứ tự các nước đi tốt, đồng thời thông tin của các nút đã có sẵn trong TPT nên việc tính toán lặp lại sẽ không xảy ra, kết quả giống như trò chơi xếp hình, ta lần lượt xếp thứ tự các nước tốt hơn theo từng độ sâu khác nhau. Mặc dù tính toán nhiều hơn, nhưng cây tìm kiếm cũng hiệu quả hơn. Điều này một lần nữa khẳng định thứ tự nước đi được duyệt là vô cùng quan trọng.

#### 2.2.7.2. Aspiration Window

Iterative Deepening bắt đầu tìm kiếm với các giá trị alpha và beta lớn nhất có thể để phản ánh điểm số tệ nhất. Để có thể cải thiện điều này Aspiration Window giả định rằng, các nước đi tiếp theo được tìm thấy, là các nước đi tốt hơn các nước đi trước đó. Vì vậy, ta lấy kết quả từ quá trình lặp sâu trước đó, cộng với một khoảng dương và âm nhỏ xung quanh, tạo ra một cửa sổ trượt. Trong vòng lặp kế tiếp, cửa sổ này hoạt động giống như giá trị bắt đầu của alpha và beta với ý tưởng là cắt tỉa cây đạt được nhanh hơn vì các giá trị alpha-beta gần nhau. Tuy nhiên, nếu một giá trị alpha được trả về nằm ngoài cửa sổ, điều này có nghĩa là giá trị trước đó quá hẹp và chưa chính xác, cửa sổ phải thiết thập lại. Trong quá trình lặp đi lặp lại nhiều lần này, cửa sổ có thể mở rộng dần nên được gọi là “Aspiration”.

### 2.2.8. Killer heuristic và History heuristic

Cho đến thời điểm hiện tại ta đã có một vài tiêu chí để sắp xếp nước đi, được xếp ưu tiên cao nhất chính là các nước đi của Pricinpal Variation (PV) và Most Valuable Victim – Least Valuable Aggressor (MVV-LVA) cho các nước đi bắt quân. Tuy vậy, chỉ với việc bắt quân thôi là chưa đủ nên Killer heuristic và History heuristic sẽ cải thiện việc sắp xếp nước đi hơn nữa. Killer heuristic lưu trữ các nước đi được gọi là “killer”, đây chính là các nước đi gây ra cắt tỉa beta trước đây. Trong quá trình sắp xếp nước đi, nếu một nước đi chưa được sắp xếp bởi các tiêu chí nêu trước đó như ăn quân, PV…, nó sẽ được so sánh với tất cả các “killer moves”, và nếu chúng giống nhau, killer moves sẽ được sắp xếp ưu tiên cao hơn để cố gắng tái tạo lại việc cắt tỉa beta. Hiếm khi một killer move được lưu trữ xuyên suốt quá trình tìm kiếm, vì vậy thay vào đó ta sẽ lưu hai killer moves ở hai độ sâu khác nhau. Mỗi khi cắt tỉa beta xảy ra, killer moves được lưu trữ ở độ sâu tương ứng. Giả sử killer moves chưa được lưu trữ, nó sẽ luôn luôn được sắp xếp ở đầu, đẩy các nước đi còn lại. Đây là lý do thông thường các phần mềm cờ vua sẽ lưu trữ 2 killer moves. Nếu nhiều hơn thì sẽ rất tốn thời gian so sánh và sắp xếp.

Một heuristic khác là History heuristic, mục tiêu giống Killer Heuristic, nó cũng tìm kiếm cắt tỉa beta và cũng sẽ xếp ưu tiên cao hơn cho những nước đi có khả năng dẫn đến cắt tỉa beta. Các điểm số chung được lưu trữ trong một mảng có thông tin bao gồm ô đi và ô đến. hoặc loại quân và ô đến. Bảng thứ nhất được gọi là “Butterfly Board” [10]. Bảng thứ 2 thì sẽ cải tiến hơn về bộ nhớ vì chỉ cần 12 \* 64 ô thay vì 64 \* 64 ô. Mỗi khi bắt đầu tìm kiếm, toàn bộ bảng sẽ được reset về 0. Mỗi khi xảy ra cắt tỉa beta, cộng depth vào ô đến để tăng độ ưu tiên cho nước đi đó. Mặc dù cả hai thường được sử dụng cùng nhau, Killer Heuristic hiệu quả hơn rất nhiều so với History Heuristic. Em gần như không thấy sự khác biệt trước và sau khi thêm History Heuristic, có thể là vì Killer Moves thường đã giữ data từ cắt tỉa beta và được ưu tiên hơn History Heuristic.

### 2.2.9. Thứ tự sắp xếp nước đi

Trong các phần trước có rất nhiều kĩ thuật sắp xếp, ưu tiên nước đi được nhắc đến. Ở đây em sẽ tập trung vào cụ thể từng kĩ thuật được sắp xếp như thế nào trong Chess Bot của bản thân. Mỗi nước đi sẽ đều có được gán những giá trị cụ thể.

**Bảng 2.1.** Điểm thứ tự nước đi

|  |  |
| --- | --- |
| **Phương pháp** | **Điểm** |
| Nút PV | 20000 |
| MVV-LVA | [10000 + 100; 10000 + 605] |
| Killer Move 1 | 9000 |
| Killer Move 2 | 8000 |
| History Heuristic | < 8000 |

Điểm thứ tự nước đi theo giá trị này được sử dụng cho cả Negamax và QS. Ngoài ra các nước đi khác sẽ có giá trị 0 và nằm ở phía sau trong danh sách tìm kiếm nước đi. Như vậy ta đã có đầy đủ thông tin cho việc đánh giá và cho điểm nước đi.

### 2.2.10. Late move reductions (LMR)

Giống như NMP, Late move reductions là một kĩ thuật cắt tỉa, thuật toán cố gắng tìm kiếm các nước đi với độ sâu ít hơn. Ý tưởng của LMR như sau, vì tất cả các nước đi đã được sắp xếp theo độ ưu tiên, các nước đi muộn có ít khả năng ảnh hưởng đến kết quả tìm kiếm và có thể tìm kiếm với độ sâu ít hơn. Đây là một kĩ thuật rất phụ thuộc vào việc sắp xếp nước đi. Một nước đi muộn thường được xem xét là một nước đi sau một số lượng nước đi X nào đó. Giá trị của X thường được xác định thông qua thử nghiệm. Các phần mềm cờ vua khác nhau sẽ có những cách khác nhau để áp dụng kĩ thuật này. Tuy nhiên cách hoạt động của LMR có thể được mô tả như sau:

if (this is a late move)

{

if (this late move satisfies certain conditions)

{

Find a reduced depth to search

Search Alpha-Beta with the reduced depth

Do a full-search if the result is greater than alpha

}

}

* **Late move**: Thường là các nước đi có độ ưu tiên thấp hơn nước đi tốt nhất
* **Conditions**: Chúng ta sẽ không giảm tìm kiếm với các nước đi chiếu, các nước đi bắt quân, phong cấp, nút PV và trong chính 1 không gian LMR khác.’
* **Depth**: Thông thường sẽ xét độ sâu tìm kiếm chỉ còn từ 3 trở xuống, và sẽ không giảm độ sâu tìm kiếm nếu giá trị đang xét lớn hơn bằng giá trị alpha.

Cùng với NMP, LMR cải thiện hiệu suất vô cùng lớn nhờ bỏ đi một số lượng lớn các nước đi ở cuối, tuy nhiên, rủi ro mất mát thông tin là rất lớn do số nút và LMR cắt tỉa là rất nhiều. Tất nhiên ta sẽ lấy tốc độ và độ sâu tìm kiếm cao hơn để bù lại rủi ro mất mát thông tin của LMR.

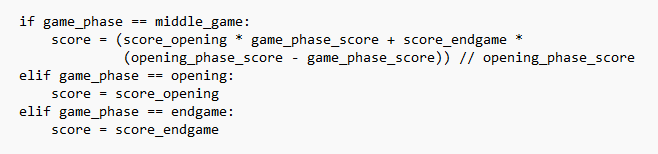
## 2.3. Hàm lượng giá bàn cờ tĩnh

### 2.3.1. Tổng quan về hàm lượng giá

Hàm lượng giá trong cờ vua là công cụ đánh giá một thế cờ nhằm xác định ưu thế của từng bên tại một thời điểm cụ thể. Để đạt được sự đánh giá chính xác, một hàm lượng giá tĩnh thường nhắm đến nhiều yếu tốt như: số lượng, chất lượng của quân cờ, cấu trúc tốt, độ an toàn của vua, khả năng di chuyển… Để có thêm sự linh hoạt hàm lượng giá cần điều chỉnh theo từng giai đoạn của ván cờ (khai, trung, tàn cuộc). Các giá trị này thường được xây dựng và đúc kết trên kinh nghiệm thử nghiệm rất nhiều các giá trị khác nhau và vẫn có những ưu điểm nhất định như: nhanh, ổn định, điểm bất lợi là rất khó để thay đổi hay mở rộng vì cấu trúc của những hàm lượng giá này đã ổn định. Một hàm lượng giá tốt sẽ giúp cây tìm kiếm dễ dàng thực hiện cắt tỉa. Tuy nhiên hàm lượng giá quá phức tạp có thể ảnh hưởng đến tốc độ duyệt cây. Vì vậy việc quyết định độ phức tạp, đa dạng, hay đơn giản nhưng hướng đến tốc độ là phụ thuộc vào lập trình viên. Dưới đây em xin trình bày các hàm lượng giá hay các kĩ thuật em đã áp dụng trong Chess Bot của mình.

### 2.3.2. Giai đoạn cờ (Game phases)

Đối với mỗi giai đoạn cờ, ta có thể điều chỉnh hoặc xây dựng nhiều đánh giá khác nhau. Ba giai đoạn cơ bản của cờ vua gồm khai cuộc, trung cuộc và tàn cuộc. Lý do sử dụng các giá trị hoặc các tính toán khác nhau cho từng giai đoạn chính là cố gắng mở rộng sự linh hoạt cho các hàm lượng giá tĩnh. Ví dụ, trong khai cuộc, quân vua nên sớm nhập thành, quân hậu ít nên di chuyển, quân mã và tốt nên chiếm vị trí ở trung tâm, ưu tiên phát triển mã tượng tốt… Trong giai đoạn đầu, quân xe gần như không có tác động quá nhiều đến ván đấu và thường nó sẽ chỉ có hiệu quả sau 6-8 nước đi đầu tiên. Xác định các giai đoạn của cờ vua dựa theo những giá trị cụ thể cũng rất trừu tượng. Đây chính là lý do mà các phần mềm cờ vua thường dựa đánh giá dựa vào chia tỉ lệ ván cờ theo phần trăm. Ví dụ: 50% trung cuộc và 50% tàn cuộc. Đây chính là phương pháp đánh giá tapered [5], kết hợp các điểm số từ trung cuộc và tàn cuộc theo tỉ lệ phần trăm, đảm bảo sự ổn định và nhất quán trong đánh giá. Tất nhiên có rất nhiều quan niệm cho trung cuộc, nhưng ở đây em định nghĩa đơn giản trung cuộc bằng cách đếm số quân còn lại mà không phải quân tốt và quân vua trên bàn cờ. Ban đầu bàn cờ tổng cộng có 4 quân mã, 2 quân tượng, 2 quân xe và 2 quân hậu, ta lần lượt cộng tổng điểm các quân này vào với nhau tạo thành 1 giá trị scores, đồng thời sử dụng các mốc opening\_score, endgame\_score để định nghĩa giai đoạn cờ. Nếu scores nhỏ hơn endgame\_score, đây sẽ là giai đoạn tàn cục, lớn hơn opening\_score thì sẽ là giai đoạn khai cục, trường hợp còn lại sẽ là giai đoạn trung cuộc. Công thức tính điểm sau cùng như sau:



###### **Hình 2.18.** Đánh giá tapered

### 2.3.3. Quân và vị trí

Hàm lượng giá theo quân chính là cách đơn giản và phổ biến nhất trong các phần mềm cờ vua. Mỗi quân cờ sẽ được gán với các giá trị cụ thể, ví dụ thường thấy nhất tốt là 1 điểm, mã và tượng là 3 điểm, xe là 5 điểm và hậu là 9 điểm đối với bên trắng, còn bên đen thì thường sẽ có giá trị âm. Tuy các điểm số này có thể thay đổi giữa các phần mềm, nhưng nguyên tắc cơ bản nhất chính là bên nào có tổng điểm cao hơn sẽ ưu thế hơn. Tất nhiên giá trị của quân cờ cũng không cố định, như đã nói từ trước, quân xe thường ít có giá trị hơn trong thời gian đầu của trận đấu, ngoài ra quân tượng thường cao điểm hơn quân mã một chút. Kết hợp với giai đoạn cờ, ta sẽ cần 12 giá trị cho quân bên trắng, 12 giá trị cờ cho quân bên đen. (6 cho opening\_score, 6 cho endgame\_score)

Hàm lượng giá theo vị trí quân sẽ đánh giá từng vị trí riêng lẻ. Ví dụ quân vua ở giữa bàn cờ sẽ bị đánh giá thấp hơn, quân tốt sắp được phong cấp sẽ có điểm rất cao. Để tính toán các điểm số theo vị trí ta xây dựng một Piece-Square Tables (Bảng quân-vị trí). Với ý tưởng cơ bản là với mỗi quân trên bàn cờ, theo một vị trí cố định, sẽ được gán một giá trị cụ thể. Không giống như hàm lượng giá theo quân, hàm lượng giá theo vị trí có nhiều điểm linh hoạt hơn. Cụ thể là với mỗi loại quân, ta có thể lưu 64 giá trị khác nhau tương ứng với 64 ô trên bàn cờ, đồng thời lưu thêm 1 bảng nữa cho giai đoạn khai cuộc/tàn cuộc. Điều này rất hữu ích trong việc phân bố quân tùy theo giai đoạn của ván cờ. Bảng giá trị mà em áp dụng được tham khảo tại [11].

### 2.3.4. Khả năng di chuyển của quân (Mobility)

Giống như tiêu đề, hàm lượng giá đánh giá khả năng di chuyển của các quân (mobility), tức là tính số nước đi mà quân cờ có thể thực hiện. Trong khi hàm lương giá theo quân – vị trí chỉ ra hướng đi tốt cho các quân cờ, chúng không hề tính đến được vị trí thực tế trên bàn cờ. Ví dụ như 1 quân tượng không thể đi đến ô e4, ô trung tâm, mặc dù được đánh giá là tốt cho quân tượng nhưng lại bị quân tốt đồng minh chiếm đóng. Ta sẽ phạt hoặc thưởng tùy theo số lượng nước đi mà quân cờ có, điều này giúp chúng ta kiểm soát được việc muốn đưa quân nào ra trận sớm hoặc muộn bằng cách nhân số lượng nước đi với các giá trị hằng số tự định nghĩa. Trong một số phần mềm, khả năng di chuyển của vua và tốt sẽ được tính, nhưng trong Chess Bot của mình em sẽ không tính đến khả năng di chuyển của hai quân này, thay vào đó là sử dụng cấu trúc tốt và độ an toàn của vua để thay thế. Ngoài ra, em cũng không xét đến khả năng di chuyển của quân xe, thay và đó là sử dụng các cột mở và nửa mở để bonus cho quân này

### 2.3.5. Độ an toàn của quân Vua (King safety)

Thông thường các phần mềm cờ vua ít có hàm lượng giá cụ thể cho việc đảm bảo an toàn cho quân vua. Một quân vua nằm giữa bàn cờ và không có che chắn thường sẽ bị đánh giá là không tốt, tuy nhiên trong tàn cuộc, điều này có thể không đúng, vì nếu là tàn cục chỉ có vua và tốt, ta muốn quân vua của mình gần ô trung tâm và có độ linh hoạt nhất có thể. Vì vậy, rất khó để tìm ra được một hàm lượng giá chính xác hoàn toàn để xét đến độ an toàn của quân vua. Sau cùng, vì hướng đến một Chess Bot có xu hướng chậm, ổn chắc và thiếu sai lầm, em quyết định sử dụng hàm bonus cho việc quân vua có bao nhiêu quân chắn trước nó. Cụ thể, em lấy tất các ô mà quân vua có thể di chuyển tới, sau đó kiểm tra xem có quân đồng minh ở đó hay không và áp dụng thưởng cho trường hợp này.

### **2.3.6. Cột mở và nửa mở**

Như đã nhắc đến ở các phần trước, quân xe rất ít di chuyển trong thời gian khai cục và việc thêm điểm cho khả năng di chuyển của quân xe là không cần thiết. Trong cờ vua quân xe thường kiểm soát các đường hàng ngang, hoặc đứng sau bảo vệ quân tốt hoặc tấn công quân tốt địch. Cột nửa mở là cột không có tốt bên mình và chỉ có quân tốt của đối phương, cột mở không có quân tốt nào, khi quân xe được đặt ở các cột này, khả năng của nó được phát huy tối đa trong việc kiểm soát cũng như khả năng di chuyển ít bị giới hạn, vì vậy nếu xe ở những quân này, ta sẽ cho bên đó thêm điểm.

### 2.3.7. Cấu trúc tốt (Pawn structure)

Cấu trúc của các quân tốt là một tiêu chí rất rộng và có nhiều phương pháp đánh giá xoay quanh nó. Thông thường những lợi ích từ cấu trúc tốt không thể thấy ngay lập tức mà chỉ thấy được ở giai đoạn sau của trò chơi, do cây tìm kiếm có độ sâu giới hạn, việc đánh giá một cấu trúc tốt cũng rất khó khăn. Vì vậy em đã lựa chọn một số tiêu chí để đánh giá các cấu trúc tốt.

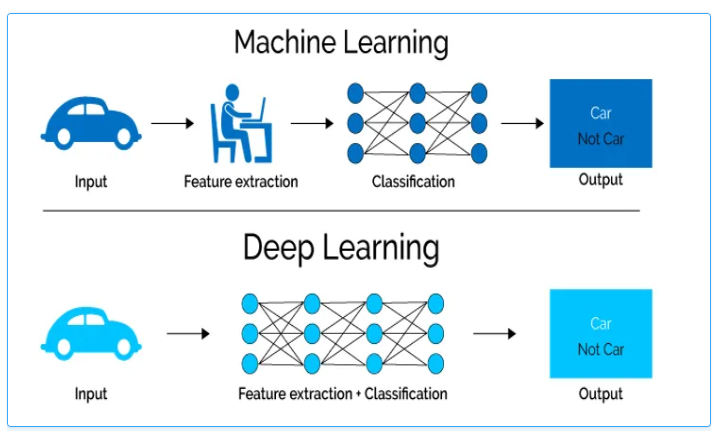
Tiêu chí đầu tiên chính là tốt cô lập. Tốt cô lập chính là quân tốt mà không có quân tốt đồng minh nào bên cạnh. Thông thường chúng ta sẽ muốn các quân tốt được liên kết với nhau, điều này sẽ tạo nên một cấu trúc tốt mạnh mẽ và các quân tốt có thể bảo vệ được lẫn nhau. Nếu có tốt cô lập, ta sẽ trừ điểm. Gần giống tốt cô lập, nếu quân tốt trên đường di chuyển, không bị quân tốt nào cản lại hoặc không thể bị quân tốt nào ăn, quân tốt này rất có khả năng sẽ được phong cấp lên thành quân hạng nặng. Vì vậy, các quân tốt thông sẽ được cộng điểm. Tiêu chí cuối cùng chính là tốt chồng, các quân tốt chồng lên nhau sẽ gây cản trở đến khả năng di chuyển, đồng thời tốt chồng có thể sinh ra cả các tốt cô lập do không có quân bảo vệ, vì vậy ta sẽ trừ điểm cho tiêu chí này.

## 2.4. Hàm lượng giá bàn cờ sử dụng mô hình học sâu

### 2.4.1. Tổng quan về học sâu

#### 2.4.1.1. Giới thiệu về học sâu

Deep learning (học sâu) có thể được xem là một lĩnh vực con của Machine Learning (học máy) - ở đó các máy tính sẽ học và cải thiện chính nó thông qua các thuật toán. DL xây dựng dựa trên các khái phức tạp, chủ yếu hoạt động với các mạng nơ-ron nhân tạo để bắt chước khả năng tư duy và suy nghĩ của con người. Trong những năm gần đây, những tiến bộ trong phân tích dữ liệu lớn (Big Data) đã cho phép ta tận dụng được tối đa khả năng của DL. Các mạng nơ-ron nhân tạo chính là động lực để phát triển Deep learning. Các mạng nơ-ron sâu (DNN) bao gồm nhiều lớp (thường là ba lớp hoặc nhiều hơn), trong đố thông tin được truyền qua nhiều lớp từ đầu vào đến đầu ra.



###### **Hình 2.19.** So sánh ML và DL

**Ưu điểm:**

* Kiến trúc mạng neuron linh hoạt, có thể thay đổi đề phù hợp với nhiều bài toán khác nhau.
* Có khả năng giải quyết nhiều bài toán phức tạp với độ chính xác cao.
* Tính tự động hóa cao, có khả năng tự điều chỉnh và tự tối ưu.
* Có khả năng thực hiện tính toán song song, hiệu năng tốt, xử lý được dữ liệu lớn.

**Nhược điểm:**

* Cần có khối lượng dữ liệu lớn và đa dạng
* Chi phí tính toán cao vì phải xử lý nhiều mô hình phức tạp.
* Chưa có nền tảng lý thuyết mạnh mẽ để lựa chọn các công cụ tối ưu cho DL.

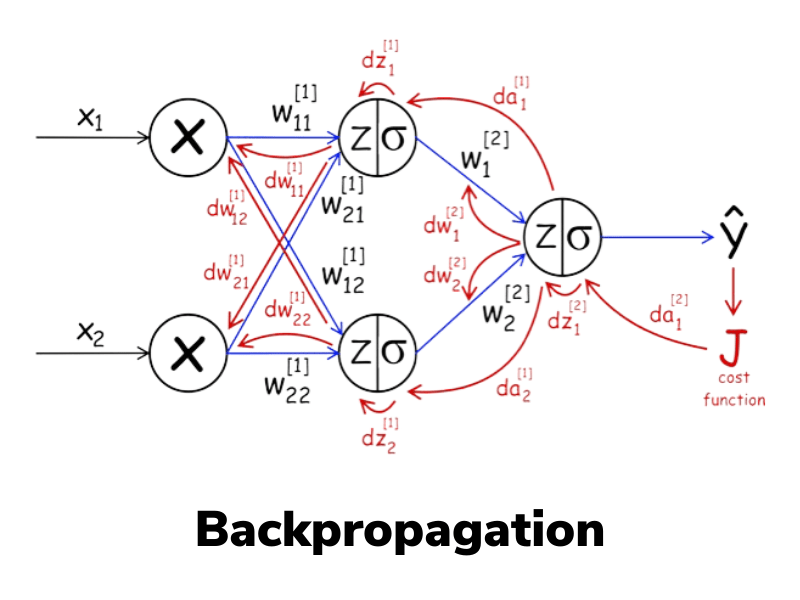
#### 2.4.1.2. Kiến trúc của mạng nơ-ron

Trong chương 1, em đã giới thiệu qua về mạng neuron, ở đây chúng ta sẽ tìm hiểu nguyên lý hoạt động của mạng nơ ron cũng như giới thiệu một số loại mạng neuron thường được bắt gặp nhất.

Mạng neuron hoạt động bằng cách biến đổi các đầu vào thành đầu ra thông qua một chỗi các phép tính toán. Mỗi nơ-ron trong mạng nơ-ron được kích hoạt bởi một tổ hợp tuyến tính của các đầu vào của nó và các trọng số kết nối. Quá trình này có thể được biểu diễn bằng công thức toán học như sau.

Trong đó: là trọng số kết nối giữa đầu vào và nơ-ron. là ngưỡng (bias) của nơ-ron. Hàm kích hoạt thường là một hàm phi tuyến tính như ReLU, Sigmod, hoặc Tanh. Quá trình học của mạng nơ-ron bao gồm việc điều chỉnh các trọng số kết nối và ngưỡng để mô hình có thể học từ dữ liệu huấn luyện và tạo ra dự đoán chính xác cho dữ liệu mới. Quá trình này được thực hiện thông qua việc cập nhật các trọng số và ngưỡng dựa trên độ lỗi giữa đầu ra dự đoán và đầu ra thực tế, và phương pháp tối ưu hóa như gradient descent.

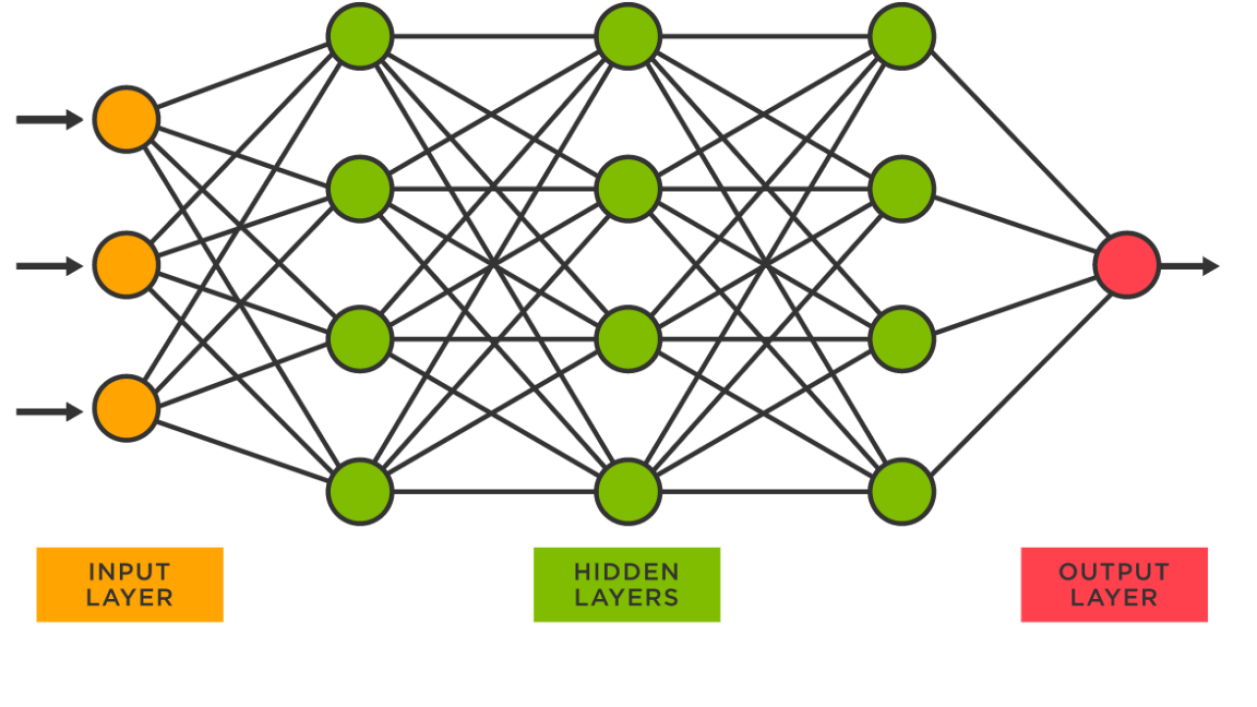
Cụ thể, quá trình học trong mạng neuron thường được thực hiện bằng cách sử dụng giải thuật lan truyền ngược (Backpropagation), trong đó gradient của hàm mất mát được lan truyền ngược qua mạng để tính toán đạo hàm của hàm mất mát theo từng trọng số. Các trọng số sau đó được cập nhật dựa trên gradient này và một hệ số học (learning rate), với mục tiêu là giảm thiểu sai số giữa đầu ra dự đoán và đầu ra thực tế.



###### **Hình 2.20.** Quá trình lan truyền ngược trong mạng.

Một số loại mạng neuron phổ biến trong lĩnh vực học sâu:

1. **Multilayer Perceptron (MLP):**



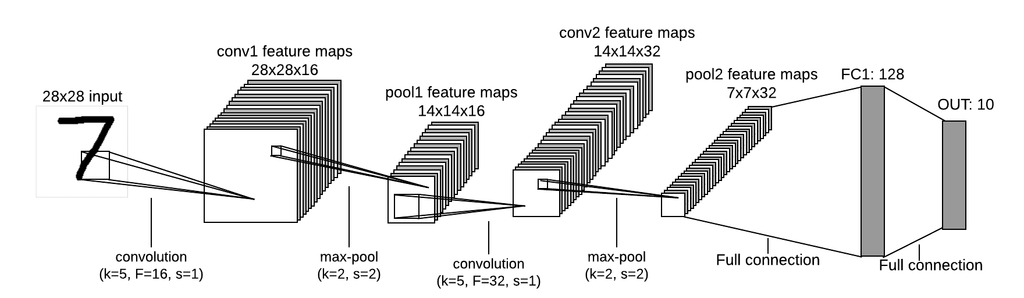
###### **Hình 2.21.** Mạng MLP

Multilayer Perceptron (MLP) là một dạng phổ biến của mạng neuron trong học sâu. Nó bao gồm ít nhất hai lớp neuron - một lớp đầu vào và một lớp đầu ra - cùng với một hoặc nhiều lớp ẩn giữa chúng. Cấu trúc này cho phép MLP học được các biểu diễn phức tạp của dữ liệu và giải quyết các vấn đề phức tạp hơn so với perceptron đơn giản.

* **Ưu điểm**: Có khả năng học biểu diễn phức tạp của dữ liệu, linh hoạt và được sử dụng trong nhiều ứng dụng khác nhau. Dễ huấn luyện thông qua phương pháp lan truyền ngược.
* **Nhược điểm**: Dễ bị overfitting do mô hình phức tạp nhưng không có cơ chế kiểm soát sự phức tạp của mô hình. Mất nhiều thời gian huấn luyện với tập dữ liệu lớn khi có nhiều lớp ẩn.

1. **Convolutional Neural Network (CNN):**

Convolutional Neural Network (CNN) là một loại mạng nơ-ron được thiết kế đặc biệt để xử lý dữ liệu không gian như hình ảnh và video.

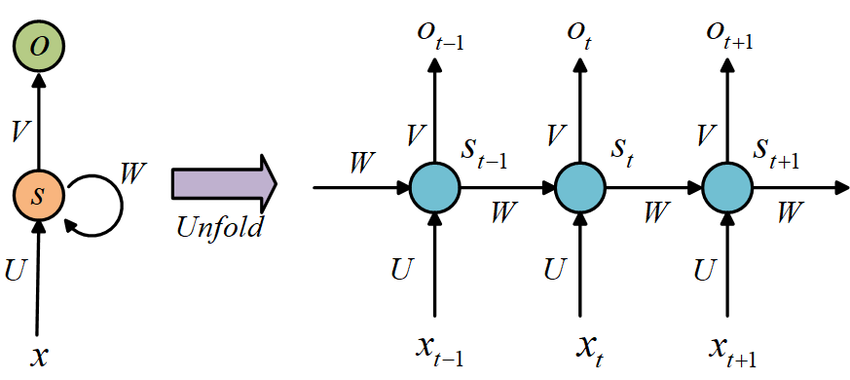


###### **Hình 2.22.** Mạng CNN

* **Lớp Convolutional**: Sử dụng các bộ lọc (kernels) để thực hiện phép tích chập trên dữ liệu đầu vào. Mỗi bộ lọc sẽ di chuyển qua từng vùng của đầu vào và tính tổng tích chập để tạo ra bản đồ đặc trưng (feature map). Số lược và kích thước của các bộ lọc cũng như bước nhảy (stride) được xác định trước và ảnh hưởng đến kích thước và chiều sâu của feature map.
* **Lớp Activation**: Sau khi tính tổng tích chập, một hàm kích hoạt như ReLU thường được áp dụng để tăng tính phi tuyến.
* **Lớp Pooling**: đây là lớp gộp thường được sử dụng để giảm kích thước dữ liệu nhưng vẫn giữ lại thông tin quan trọng. Pooling có thể là Max Pooling (giá trị lớn nhất của vùng), Average Pooling (lấy giá trị trung bình), hoặc các loại pooling khác.
* **Lớp Fully Connected**: Sau các lớp Convolutional và Pooling, các đặc trưng được trải thành một vector và đưa vào các lớp fully connected để phân loại. Các lớp fully connected thường được sử dụng ở cuối mạng để tính xác suất của các lớp đầu ra và thực hiện phân loại.
* **Lớp Dropout**: thường được sử dụng để giảm overfitting bằng cách tắt ngẫu nhiên một số nơ-ron trong quá trình huấn luyện.
* **Lớp Flatten**: được sử dụng để chuyển đổi các tensor đa chiều thành vector một chiều trước khi đưa vào các lớp Fully Connected.
* **Ưu điểm**: Tự động học và trích xuất đặc trưng. Xử lý dữ liệu không gian hiệu quả. Linh hoạt trong ứng dụng.
* **Nhược điểm**: Yêu cầu lượng dữ liệu lớn, tính toán phức tạp và tốn chi phí, phức tạp trong thiết kế.

1. **Recurrent Neural Network (RNN):**

Recurrent Neural Network (RNN) là mộ dạng mạng nơ-ron thích hợp cho việc xử lý dữ liệu tuần tự như chuỗi văn bản, âm nhạc, hoặc chuỗi thời gian. Sự khác biệt chính giữa RNN và các mạng nơ-ron truyền thẳng là RNN có khả năng lặp lại thông tin thông qua các kết nối lặp lại trong kiến thức của chúng, cho phép chúng “nhớ” thông tin từ quá khứ và áp dụng nó cho các dữ liệu hiện tại.



###### **Hình 2.23.** Mạng RNN

* **RNN Cell**: Là đơn vị cơ bản của RNN, xử lý một đầu vào và trạng thái trước đó để tạo ra một đơn vị đầu ra và trạng thái với. Các loại đơn vị RNN phổ biến bao gồm Simple RNN, Long Short-Term Memory (LSTM), và GRU (Gated Recurrent Unit).
* **Hidden State (trạng thái ẩn)**: Trạng thái ẩn của RNN chứa thông tin từ các bước trước đó của chuỗi, giúp mạng “nhớ” thông tin từ quá khứ.
* **Lớp Fully Connected**: thường được dùng ở cuối mạng giống như CNN, thực hiện chức năng tính toán xác suất của các lớp đầu ra và thực hiện phân loại.
* **Lớp Softmax**: thường được sử dụng ở cuối mạng để chuyển đổi các giá trị đầu ra thành phân phối xác suất, giúp trong việc phân loại.
* **Ưu điểm**: Xử lý dữ liệu tuần tự, có khả năng hiểu và học hỏi các mối quan hệ dữ liệu dài hạn thông qua các kết nối lặp lại trong kiến trúc của chúng. Giống như CNN, RNN cũng có tính linh hoạt rất cao và có nhiều ứng dụng
* **Nhược điểm**: RNN có thể gặp vấn đề với việc “quên” thông tin từ các bước trước trong huấn luyện. Việc tính toán các kết nối lặp lại trong RNN yêu cầu nhiều tài nguyên tính toán và làm chậm quá trình huấn luyện. RNN thường không hiệu quả cho các chuỗi dữ liệu quá dài do vấn đề “quên” trong quá trình lan truyền ngược.

#### 2.4.1.3. Optimizer

Trong học sâu, một phần quan trọng của việc huấn luyện mô hình là chọn lựa và cấu hình các phương pháp tối ưu hóa hay còn được gọi là “optimizer”. Optimizer đóng vai trò quan trọng trong việc điều chỉnh các trọng số của mạng nơ-ron dựa trên độ lỗi của dữ liệu huấn luyện. Dưới đây là một số optimizer phổ biến và sử dụng rộng rãi trong thế giới học sâu:

1. **Gradient Descent (GD):**

* **Giới thiệu**: Gradient Descent (GD) à một phương pháp tối ưu hóa cơ bản trong deep learning. Nó cập nhật các trọng số của mạng nơ-ron dựa trên gradient của hàm mất mát tính trên toàn bộ tập dữ liệu huấn luyện.
* **Công thức cập nhật**:

Trong đó:

* là bộ trọng số tại bước thời điểm .
* là learning rate.
* là gradient của hàm mất mát tính tại
* **Ưu điểm:** Dễ triển khai và dễ hiểu, hiệu quả trên các bài toán với dữ liệu nhỏ.
* **Nhược điểm:** Tính toán gradient trên toàn bộ dữ liệu gây tốn kém về tài nguyên, dễ mắc kẹt ở điểm cực tiểu cục bộ.

1. **Stochastic Gradient Descent (SGD):**

* **Giới thiệu**: Stochastic Gradient Descent (SGD) cập nhật các trọng số dựa trên gradient của một mini-batch ngẫu nhiên từ dữ liệu huấn luyện.
* **Công thức cập nhật**:

Trong đó:

* và là mẫu dữ liệu thứ trong mini-batch
* **Ưu điểm:** Phù hợp với dữ liệu lớn. Giảm thời gian tính toán so với GD
* **Nhược điểm:** Dao động lớn do gradient thay đổi liên tục. Có thể cần nhiều thời gian hơn để đến điểm cực tiểu so với GD.

1. **Momentum:**

* **Giới thiệu**: Momentum giúp giảm bớt dao động khi tìm kiếm điểm cực tiểu bằng cách tích lũy đà của gradient trước đó.
* **Công thức cập nhật**:

Trong đó:

* là vectơ momentum tại bước thời điểm .
* là hệ số momentum (thường được chọn trong khoảng từ 0-1).
* **Ưu điểm:** Giúp vượt qua các điểm cực tiểu cục bộ. Tăng tốc độ học
* **Nhược điểm:** Cần chọn tham số momentum phù hợp. Có thể trượt qua điểm cực tiểu nhanh chóng khi momentum quá lớn.

1. **Root Mean Square Propagation (RMSProp):**

* **Giới thiệu**: RMSProp là một phương pháp tối ưu hóa được thiết kế để giảm tỷ lệ học (learning rate) cho từng tham số dựa trên tỷ lệ gradient trung bình của tham số đó.
* **Công thức cập nhật**:

Trong đó:

* ϵ là một số nhỏ để tránh chia cho 0.
* **Ưu điểm:** Hiệu quả trong việc ổn định quá trình học. Giảm nguy cơ mắc kẹt ở điểm cực tiểu cục bộ.
* **Nhược điểm:** Cần lượt chọn tham số tốt cho kích thước cửa sổ trượt.

1. **Adaptive Moment Estimation (Adam):**

* **Giới thiệu**: Adam là một phương pháp tối ưu kết hợp cả Momentum và RMSProp. Nó tự động điều chỉnh tỷ lệ học cho mỗi tham số và tích lũy độ lớn của gradient.
* **Công thức cập nhật**:

Trong đó:

* và là các hệ số momentum và hệ số điều chỉnh cho gradient.
* là khắc phục bias cho
* là khắc phục bias cho
* **Ưu điểm:** Hiệu quả trong quá trình tối ưu hóa hàm mất mát. Tự động điều chỉnh được tỷ lệ học và độ lớn của gradient.
* **Nhược điểm:** Tốn kém về tài nguyên tính toán.

1. **AdamW:**

* **Giới thiệu**: AdamW là một biến thể của phương pháp Adam, chỉ thêm vào một hạng tử điều chuẩn (weight decay) để kiểm soát overfitting trong quá trình huấn luyện.
* **Công thức cập nhật**: Giống với Adam nhưng khác ở bước cuối cùng

Trong đó:

* là hệ số điều chuẩn (weight decay), giúp kiểm soát overfitting.
* **Ưu điểm:** Hiệu suất tối ưu hóa tốt, tự điều chỉnh tốc độ học. Kết hợp cải thiện về regularization. Khả năng chống trượt gradient.
* **Nhược điểm:** Khó kiểm soát và điều chỉnh tham số. Đòi hỏi lượng dữ liệu lớn. Khả năng gặp vấn đề với dữ liệu có nhiễu (noise) lớn.

#### 2.4.1.4. Hàm mất mát và hàm kích hoạt

**Hàm mất mát** (Loss Function): là một đại lượng đô lường mức độ mất mát của mô hình dự đoán so với thực tế. Mục tiêu của việc huấn luyện mô hình học máy hay học sâu là tối thiểu hóa hàm mất mát. Cụ thể, hàm mất mát thường được định nghĩa để đo lường sự khác biệt giữa giá trị thực tế của dữ liệu huấn luyện.

Một số hàm mất mát thông dung:

1. Sai số bình phương trung bình (Mean Squared Error – MSE):

Còn được gọi là L1 loss. Đây là hàm mất phổ biến nhất trong các bài toán hồi quy. Nó tính toán trung bình của bình phương chênh lệch giữa giá trị thực tế và phân phối xác suất thực tế . **Ưu điểm**: dễ tính toán và đô lường hiệu suất mô hình. **Nhược điểm**: Nhạy cảm với giá trị ngoại lai.

1. Mean Absolute Error (MAE)

Còn được gọi là L2 loss. Đây là một hàm mất mát thường được sử dụng trong các bài toán hồi quy để đo lường mức độ trung bình giữa giá trị dự đoán và thực tế. **Ưu điểm**: Không nhạy cảm với giá trị ngoại lại. **Nhược điểm**: Không đạo hàm được tại một số điểm, làm cho việc tối ưu hóa trở nên khó khan hơn so với MSE.

1. Hàm Cross-Entropy:

Thường sử dụng trong các bài toán phân loại nhị phân hoặc đa lớp. Ưu điểm: Đo lường sự tương đồng giữa phân phối xác xuất dự đoán và phân phối xác suất thực tế. Nhược điểm: Cân có một phân phối xác suất định nghĩa rõ rang cho mỗi lớp.

1. Hàm Hingle Loss:

Thường sử dụng trong bài toán phân loại nhị phân hoặc đa lớp với hàm kích hoạt tuyến tính ở lớp đầu ra. Hàm này tạo ra một độ lỗi khi dự đoán của mô hình không đúng hướng. **Ưu điểm**: Hiệu quả trong việc tạo ra một độ lỗi khi dự đoán của mô hình không đúng hướng. **Nhược điểm**: Không thích hợp cho các bài toán phân loại xác suất.

**Hàm kích hoạt** (Actvation Function): được sử dụng để thêm tính phi tuyến vào mạng nơ-ron, cho phép mạng nơ-ron học được các mối quan hệ phức tạp giữa các biến đầu vào và đầu ra. Hàm kích hoạt thường được áp dụng cho kết quả của các lớp trung gian (hidden layeres) trong mạng nơ-ron.

Một số hàm kích hoạt thông dụng:

1. **Hàm Sigmoid:**

Chuyển đổi giá trị đầu vào thành một giá trị trong khoảng từ 0 đến 1, thích hợp cho các tác vụ mà đầu ra cần giới hạn trong một khoảng xác định.

1. **Hàm Tanh:**

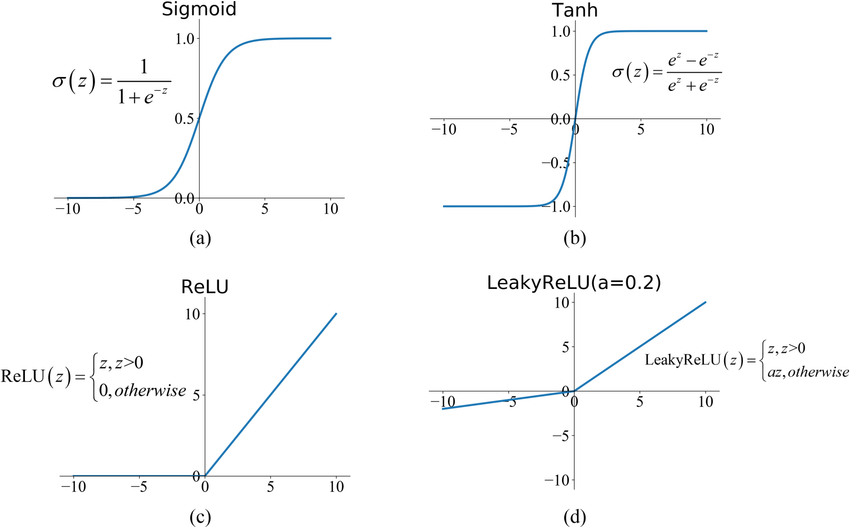
Tương tự như hàm Sigmoid, nhưng chuyển đổi giá trị đầu vào thành một giá trị trong khoảng từ -1 đến 1, thích hợp cho các tác vụ mà đầu ra có thể âm.

1. **Hàm ReLU (Rectified Linear Activation):**

Đây là hàm kích hoạt phổ biến nhất. Cho đầu ra bằng 0 nếu đầu vào là âm và bằng chính nó nếu đầu vào là dương, giúp mạng nơ-ron học được tính phi tuyến.

1. **Hàm Leadky ReLU:**

Giống như ReLU, nhưng cho phép một giá trị nhỏ cho đầu vào âm, giảm nguy cơ bị mất thông tin cho quá trình huấn luyện.



###### **Hình 2.24.** Đồ thị của các hàm activation Sigmoid, Tanh, ReLU, LeakyReLU

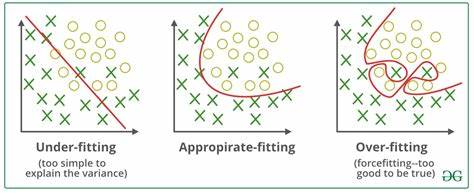
1. **Hàm Softmax:**

Thường được sử dụng trong lớp đầu ra của các mô hình phân loại đa lớp. Nó chuyển đổi giá trị đầu vào thành một phân phối xác suất trên các lớp đầu ra, giúp quyết định lớp cuối cùng được phân vào lớp nào.

Khi thiết kế mạng nơ-ron, việc chọn hàm mất mát và hàm kích hoạt phù hợp là rất quan trọng, ảnh hưởng đến khả năng học và hiệu suất của mô hình.

#### 2.4.1.5. Overfitting và Underfitting

Trong quá trình huấn luyện học sâu hay học máy, hai vấn đề hay gặp phải nhất là Overfitting và Underfitting. Đây là hai trạng thái mô hình mà khiến cho hiệu suất của nó không đạt được sự lý tưởng.



###### **Hình 2.25.** Các mức độ phù hợp với dữ liệu

**Overfitting (Quá khớp):**

* Xảy ra khi mô hình học quá nhiều từ dữ liệu huấn luyện và bắt đầu “ghi nhớ” dữ liệu huấn luyện thay vì “học” từ nó. Điều này dẫn đến việc mô hình trở nên quá phức tạp và không thể tổng quát hóa tốt cho dữ liệu mới không được sử dụng trong quá trình huấn luyện.
* Biểu hiện của Overfitting thường là hiệu suất cao trên tập huấn luyện nhưng hiệu suất thấp trên tập kiểm tra hoặc dữ liệu mới.
* Overfitting có thể xảy ra khi mô hình quá phức tạp hoặc khi dữ liệu huấn luyện không đủ đại diện cho dữ liệu thực tế.
* Để giải quyết tình trạng này, ta sẽ giảm độ phức tạp của mô hình, ví dụ như giảm số lượng tham số, sử dụng mô hình đơn giản hơn. Sử dụng các kỹ thuật regularization như L1, L2 regularization. Tăng dữ liệu huấn luyện. Sử dụng kỹ thuật early stopping để dừng quá trình khi mô hình bắt đầu overfitting. Thực hiện cross-validation để đánh giá hiệu suất mô hình và tránh overfitting. Thêm các lớp Dropout, BatchNorm…

**Underfitting (Quá đơn giản):**

* Ngược lại với Overfitting, Underfitting xảy ra khi mô hình không đủ phức tạp để học được cấu trúc chính xác của dữ liệu huấn luyện. Điều này dẫn đến việc mô hình không thể học và tổng quát hóa dữ liệu nên không thể đạt hiệu suất mong muốn.
* Biểu hiện của Underfitting thường là hiệu suất thấp trên cả tập huấn luyện và tập kiểm tra.
* Underfitting có thể xảy ra khi mô hình quá đơn giản hoặc khi dữ liệu huấn luyện không đủ để học cấu trúc của dữ liệu.
* Để giải quyết ta sẽ tăng độ phức tạp của mô hình: tăng số lớp, số lượng biến… Cải thiện các đặc trưng của dữ liệu đầu vào và tăng dữ liệu huấn luyện.

#### 2.4.1.6. Tầm quan trọng và ứng dụng của học sâu

Học sâu đang trở thành một công cụ quan trọng và mạnh mẽ trong lĩnh vực trí tuệ nhân tạo và khoa học máy tính, ứng dụng rộng rãi trong nhiều lĩnh vực khác nhau:

* **Xử lý ảnh và video**: Deep Learning đã đạt được những tiến bộ đáng kể trong nhận dạng và phân loại hình ảnh, phát hiện và theo dõi đối tượng, tái tạo hình ảnh và video, cũng như trong các ứng dụng như xử lý hình ảnh y tế và giám sát an ninh.
* **Xử lý ngôn ngữ tự nhiên (NLP)**: Trong lĩnh vực NLP, Deep Learning đã mang lại các tiến bộ đáng kể trong nhận dạng và phân loại văn bản, dịch máy, sinh văn bản tự động, và tóm tắt văn bản, giúp cải thiện hiệu suất và độ chính xác của các ứng dụng liên quan đến ngôn ngữ.
* **Xử lý âm thanh và giọng nói**: Deep Learning đang được sử dụng trong nhận dạng và phân tích âm thanh, nhận diện giọng nói, tổng hợp giọng nói, và trong các ứng dụng như ghi chú bằng giọng nói và trợ lý ảo dựa trên giọng nói.
* **Tự lái và xe tự hành**: Trong lĩnh vực tự lái và xe tự hành, Deep Learning đóng một vai trò quan trọng trong việc phát hiện và nhận dạng các đối tượng trên đường, dự đoán hành vi của các phương tiện giao thông, và tự điều khiển các phương tiện di chuyển.
* **Dược phẩm và y học**: Deep Learning được áp dụng trong nhiều ứng dụng y học như dự đoán bệnh lý, phát hiện ung thư từ hình ảnh y khoa, dự đoán phản ứng thuốc, và thiết kế phân tử thuốc.
* **Tài chính và giao dịch**: Trong lĩnh vực tài chính, Deep Learning được sử dụng trong phát hiện gian lận, dự đoán giá cả và xu hướng thị trường, quản lý rủi ro trong các ứng dụng giao dịch tài chính.
* **Năng lượng và môi trường**: Deep Learning có thể được áp dụng để dự đoán tiêu thụ năng lượng, dự báo thời tiết, theo dõi và phân tích dữ liệu môi trường, tối ưu hóa việc sử dụng tài nguyên tự nhiên.

### 2.4.2. Xây dựng mô hình

#### 2.4.2.1. Bộ dữ liệu

Bộ dữ liệu em sử dụng trong bài toán của mình được lấy từ Kaggle:

<https://www.kaggle.com/datasets/ronakbadhe/chess-evaluations>

Dữ liệu gồm 2 cột, đó là FEN và Evaluation

FEN là viết tắt của “Forsyth-Edwards Notation”, một hệ thống ký hiệu được sử dụng để mô trả trạng thái của một bàn cờ vào một thời điểm nhất. định. Nó được sử dụng rộng rãi trong cộng đồng cờ vua để ghi lại phân tích các vị trí trên bàn cờ. FEN bao gồm các thông tin như sau:

1. Vị trí của các quân cờ: Mỗi hàng ngang (từ hàng 8 đến hàng 1) được mô tả từ trái sang phải với các chữ cái đại diện cho các quân cờ (P = tốt, R = xe, N = mã, B = tượng, Q = hậu, K = vua, chữ thường sẽ là quân đen và chữ hoa là quân trắng). Các ô trống được biểu diễn bằng các con số từ 1 đến 8, chỉ số lượng ô trống liên tiếp.
2. Lượt đi: “w” biểu thị lượt đi hiện tại là của quân trắng, “b” biểu thị lượt đi của quân đen.
3. Quyền nhập thành: Quyền nhập thành của quân vua và quân xe được biểu thị bằng các chữ cái K, Q, k, q. Với K - bên trắng vẫn có thể nhập thành cánh vua, Q - bên trắng còn quyền nhập thành cánh hậu, tương tự với k và q cho bên đen. Nếu không có bất kể kí tự nào trong 4 kí tự trên thì không bên nào còn quyền nhập thành và sẽ có kí hiệu “-”.
4. Vị trí bắt tốt qua đường: Nếu có thể bắt tốt qua đường, ô đích sẽ được chỉ định (ví dụ “e3”) nếu không, ký hiệu này là “-”.
5. Số lượt không bắt quân hoặc di chuyển tốt: Dùng để xác định quy tắc 50 nước đi.
6. Tổng số lượt: Số lượt di chuyển của cả hai bên. Mỗi lượt của đen kết thúc một lượt đầy đủ

Ví dụ về dòng FEN thể hiện vị trí bắt đầu của bàn cờ vua:

“rnbqkbnr/pppppppp/8/8/8/8/PPPPPPPP/RNBQKBNR w KQkq - 0 1”

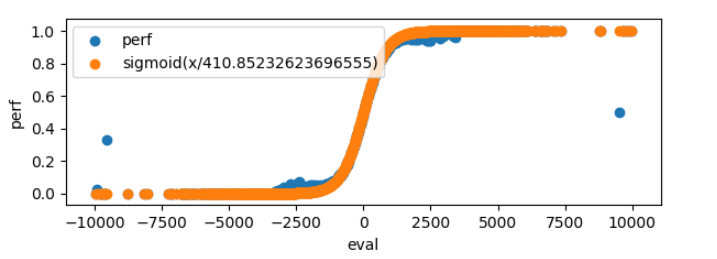
Cột thứ hai bao gồm giá trị được đánh giá dựa trên Stockfish 12 trên 1 hình cờ FEN cụ thể, dưới đơn vị là Centipawn (1 pawn = 100 centipawn), nếu có lợi cho bên trắng, dữ liệu sẽ thể hiện bằng dấu + và số nguyên, nếu có lợi cho bên đen thì sẽ thể hiện bằng dấu trừ và số nguyên. Ngoài ra còn có trường hợp đặc biệt là cột Evaluation có thêm dấu #, nếu có thêm dấu + phía sau thì điều này thể hiện rằng trắng sẽ chiếu hết đen sau k nước đi, ngược lại với bên đen thì sẽ tương ứng với dấu -.

Bộ dữ liệu gồm hơn 12 triệu hình cờ (FEN) khác nhau tương ứng với đó là đánh giá cụ thể của Stockfish 12 về thế cờ đó theo đơn vị Centipawn. Đây sẽ là bộ dữ liệu phù hợp vì có số lượng dữ liệu vừa phải và phục vụ mục đích xây dựng một mô hình học sâu với đầu vào là là một hình cờ, và đầu ra là kết quả đánh giá hình cờ đó dưới dạng một số nguyên.

#### 2.4.2.2. Xử lý dữ liệu

Vì dữ liệu của cột Evaluation đánh giá không đồng nhất, cần xử lý dữ liệu chuyển chúng về toàn bộ dạng số để thuận tiện cho việc huấn luyện. Quá trình xử lý dữ liệu của em gồm các bước như sau:

1. Chuyển đổi cột Evaluation thành các số nguyên, với các giá trị âm, dương ta giữ nguyên, nhưng với giá trị checkmate, do dữ liệu số lớn nhất nằm trong khoảng 8000, ta sẽ quy đổi chiếu hết là +10000 cho bên trắng và -10000 cho bên đen.
2. Lấy mẫu dữ liệu: Do tài nguyên trên máy tính cá nhân có hạn, em đã sử dụng nguồn tài nguyên miễn phí trên Kaggle để có thể huấn luyện mô hình, tuy nhiên rất khó để có thể huấn luyện trên cả 12 triệu data, với mức độ mô hình của mình, em quyết định sẽ lấy khoảng 5 triệu dữ liệu ngẫu nhiên cho cả tập train và tập test.
3. Do giá trị output nằm trong đoạn [-10000, 10000] là một khoảng rất lớn, điều này có thể khiến mô hình xảy ra hiện tượng exploring gradient nên em đã thêm một bước chuyển đổi dữ liệu từ Centipawn sang tỉ lệ Win-Draw-Lose (WDL) thông qua hàm Sigmoid(x/410). Cách này được lấy ý tưởng từ Stockfish NNUE [12].



###### **Hình 2.26.** Chuyển đổi giá trị từ Centipawn sang tỉ lệ WDL [12]

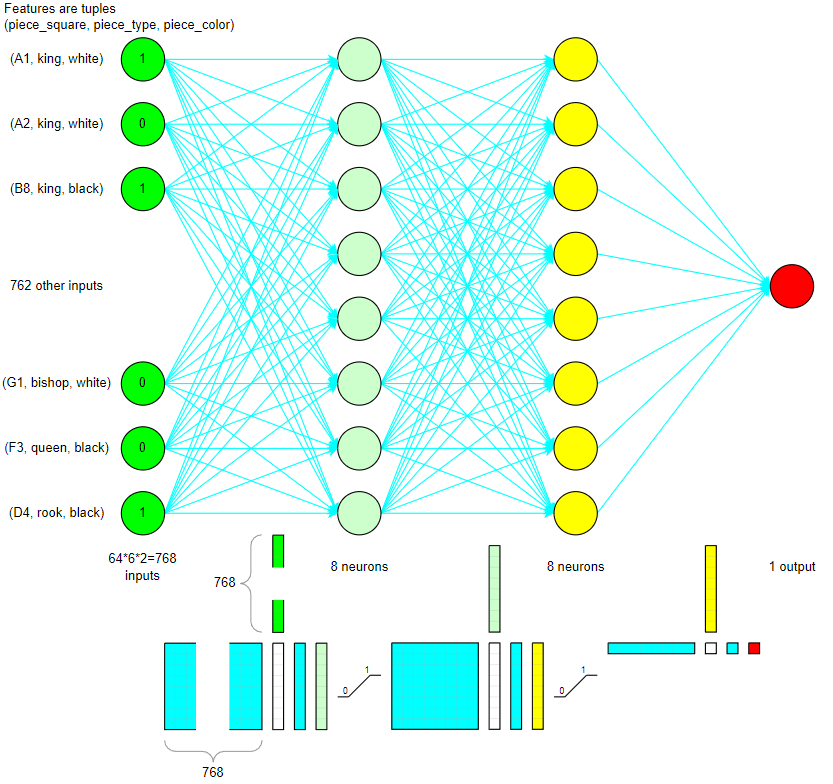
Ở đây, tỉ lệ WDL này sẽ được nhìn dưới góc nhìn của bên trắng, ta có thể nhận thấy rằng với giá trị centipawn lớn hơn 1000, trắng đã có hơn 90% tỉ lệ thắng. Điều này là hoàn toàn hợp lý trong cờ vua, khi mà 1000 Centipawn tương đương với 1 quân hậu và 1 quân tốt. Và khi giá trị gốc càng nhỏ, có nghĩa là tỉ lệ của thắng của bên trắng sẽ thấp hơn. Khi tỉ lệ càng gần về 1 càng có lợi cho trắng, càng về 0 thì càng có lợi cho đen, còn khi tỉ lệ bằng 0.5 thì có nghĩa là hai bên đang cân bằng.

1. Sau khi xử lý dữ liệu cho cột Evaluation, nhiệm vụ hiện tại của ta chỉ còn từ 1 FEN, phân tích trích xuất các đặc trưng cần thiết tùy theo mô hình. Ở đây các đặc trưng cần thiết sẽ được em chuyển thành các numpy array và sau đó lưu lại dưới dạng đuôi “hd5” để có thể tải lên Kaggle để huấn luyện mô hình.
2. 5 triệu dữ liệu sẽ được chia theo tỉ lệ 0.8/0.2 cho tập training và tập validation.

#### 2.4.2.3. Cấu trúc của mô hình

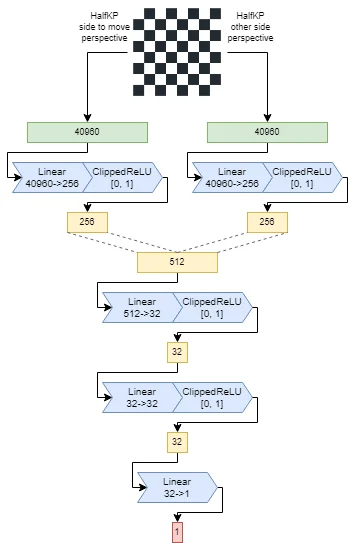
Ban đầu, ý tưởng của em là xây dựng một mô hình phức tạp với nhiều lớp, nhiều đặc trưng để có thể phân tích bàn cờ một cách chính xác nhất. Tuy nhiên, sau khi hoàn thiện huấn luyện model theo mô hình CNN, tích hợp vào cây tìm kiếm, em đã nhanh chóng nhận ra rằng một mô hình phức tạp chưa chắc đã là một mô hình có thể đưa ra những đánh giá chuẩn xác, nhưng chắc chắn rằng phức tạp cũng đồng nghĩa với việc với từng hình cờ một, mô hình sẽ tốn rất nhiều thời gian để có thể đưa ra đánh giá. Vì vậy một mô hình ít phức tạp sẽ là lựa chọn hợp lý hơn.

Lựa chọn mô hình và đặc trưng:



###### **Hình 2.27.** Mô hình NNUE đơn giản [12].

Stockfish đã giới thiệu cho người dùng một mô hình đơn giản gồm 3 lớp Linear từ 768 -> 8, 8->8, 8-> 1. Với đặc trưng đầu vào sẽ là ô của quân cờ hiện tại, loại quân cờ và màu của quân cờ. Vì trên bàn cờ có 6 loại quân cờ, bàn cờ tổng cộng có 64 ô, và có 2 phe đen trắng, nên đặc trưng đầu vào sẽ có 6 \* 64 \* 2 = 768 ô. Kết quả của output sẽ là 1 giá trị duy nhất thể hiện, kết quả đánh giá bàn cờ. Tuy nhiên sau khi thử huấn luyện trên mô hình này, cũng như thử tăng số lớp, tăng kích thước của lớp, kết quả của MSE loss thấp nhất dao động trong khoảng 0.016 đến 0.017 trên tập validation, nhưng R2 chỉ ở mức 0.5, mô hình này mặc dù có tốc độ tính toán nhanh nhưng dự đoán không chính xác. Vì vậy vẫn cần có một mô hình phức tạp hơn.



###### **Hình 2.28.** Mô hình HalfKP đơn giản [12]

Đây là cấu trúc đầu tiên được sử dụng với phần mềm Stockfish, với đầu vào là 40960 \* 2 tương ứng với góc nhìn của 2 bên là bên trắng và bên đen. Hàm Activation là ClipReLU, với . Điều này giới hạn output đầu ra với x đầu vào luôn là đoạn [0; 1]. Vì đầu vào đều là các số 0, 1 và đầu ra cũng bị giới hạn như vậy, sử dụng ClipReLU sẽ giúp giảm overfitting, khiến cho mô hình nhanh hội tụ hơn và giảm thiểu nhiễu, giúp cho mô hình luôn ổn định trong phạm vị dự đoán được. Đầu ra vẫn sẽ là 1 output duy nhất đại diện cho giá trị của bàn cờ.

Đặc trưng đầu vào của mô hình này sẽ gồm các tuples, mỗi tuples sẽ gồm 4 phần (vị trí của quân vua, vị trí của ô, loại quân cờ, màu quân cờ) và loại quân ở đây sẽ khác quân vua. Tức là ta đang quan sát dưới góc nhìn của quân vua và tất cả các quân khác ở trên bàn cờ. Ta có tối đa 64 vị trí quân vua, và với 64 ô ở trên bàn cờ, ta có thể chọn 5 loại quân bất kì gồm tốt, mã, tượng, xe, hậu để xếp lên, đồng thời mỗi bên có 2 quân. Như vậy ta sẽ cần 64 \* 64 \* 5 \* 2 = 40960 đặc trưng đầu vào dưới góc nhìn của bên trắng. Bên đen cũng tương tự như vậy, tuy nhiên vị trí các quân đứng sẽ phải nhìn dưới góc nhìn của bên đen, ví dụ trong trạng thái khởi đầu, vua trắng sẽ thấy bản thân ở ô e1, thì vua đen cũng sẽ thấy bản thân ở ô e1. Các vị trí index tương ứng sẽ được gán = 1.

Input của mô hình này bao gồm 2 góc nhìn 40960 \* 2, cho bên trắng và bên đen. Mỗi đầu vào đi qua một lớp tuyến tính, chuyển đổi từ 40960 chiều xuống còn 256 chiều. Sau đó sử dụng hàm kích hoạt ReLu giới hạn đầu ra trong đoạn [0; 1]. Kế đến hai đầu ra từ các lớp ClippedReLU được kết hợp lại, ở đây khi concat cần lưu ý đến thêm 1 vấn đề, đó là góc nhìn quân bên hiện tại thực hiện nước đi.

accumulator = (stm \* torch.cat([w, b], dim=1)) + ((1 - stm) \* torch.cat([b, w], dim=1))

“stm” là biến để chỉ bên đi hiện tại, nếu là bên trắng thì stm = 1, nếu không thì bằng 0. Đầu ra kết hợp này đi qua một lớp tuyến tính khác nên kích thước giảm từ 512 -> 32, sử dụng ClipReLU để giới hạn. Đầu ra lại đi qua một lớp tuyến tính, giữ nguyên kích tước 32 và vẫn sử dụng ClipReLU để giới hạn đầu ra. Sau cùng đầu ra đi qua 1 lớp tuyến tính để giảm xuống còn 1 chiều tạo ra giá trị cuối cùng.

#### 2.4.2.4. Huấn luyện và đánh giá

Quá trình huấn luyện và đánh giá mô hình là một bước quan trọng trong việc phát triển tối ưu hóa các mô hình học sâu. Mục tiêu chính là tối ưu hóa trọng số của mô hình sao cho nó có thể dự đoán chính xác các đầu ra từ đầu vào đã cho. Trong phần này em sẽ trình bày cách em huấn luyện mô hình đã nêu ở trước.

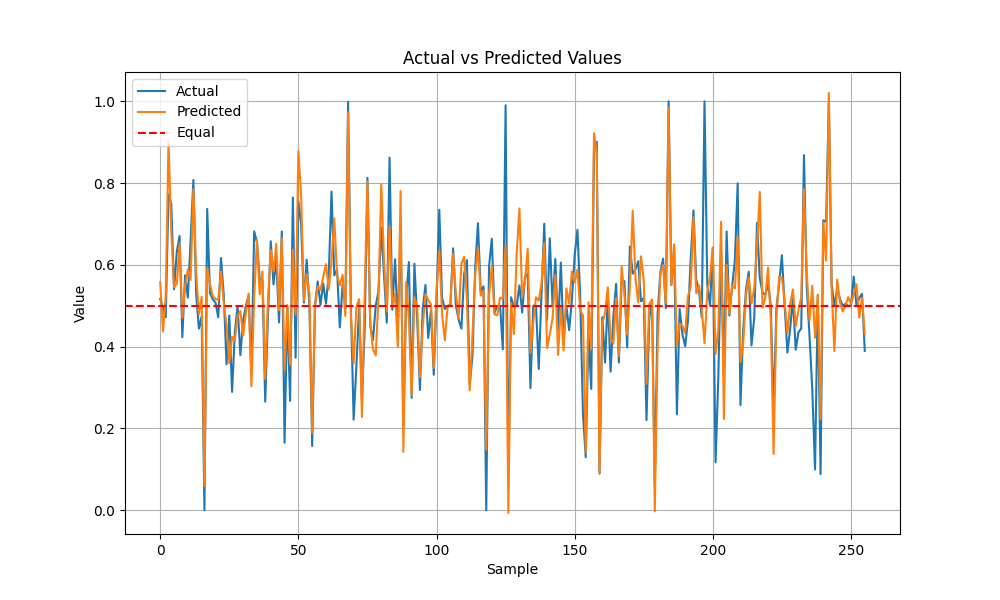
Kỹ thuật và tham số cuối cùng sử dụng:

* Optimizer: Sử dụng AdamW để mô hình nhanh chóng hội tụ.
* Learning Rate: 0.00003
* Weight\_decay: 0.1
* Hàm mất mát: Mean Squared Error (MSE)
* Epochs: Do 1 session trên Kaggle có giới hạn thời gian nên em chỉ train khoảng 10 epochs, mỗi epochs train mất khoảng 20 - 30 phút.
* Dữ liệu dữ đã được chuẩn bị từ trước: train 4 triệu, validation: 1 triệu.
* Batch Size: 64
* Sử dụng Early Stopping để tránh overfitting và lưu model tốt nhất khi loss trên tập validation giảm.
* Kết hợp cross-validation để tránh overfitting.

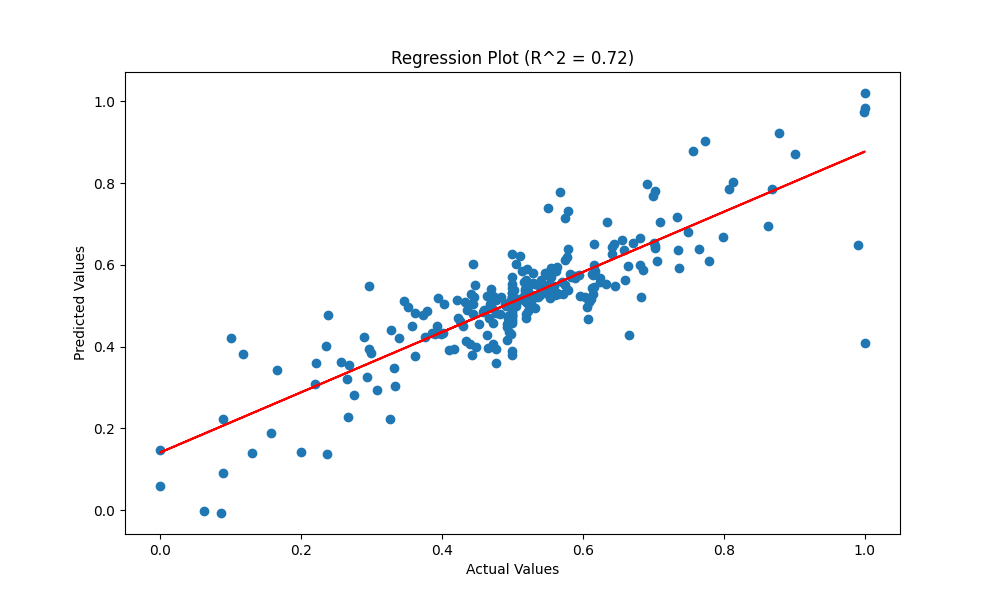
- Sử dụng GPU-100 trên Kaggle.com

**Kết quả huấn luyện**:

###### **Hình 2.29.** Training Loss và Validation Loss



###### **Hình 2.30.** Dự đoán và thực tế trên 150 mẫu ngẫu nhiên từ tập valiation



###### **Hình 2.31.** Biểu đồ hồi quy mô hình trên 150 mẫu ngẫu nhiên

* Dựa vào biểu đồ training loss và validation loss, ta thấy sự giảm dần của cả training loss và validation loss theo từng epoch, cho thấy mô hình đang hội tụ tốt.
* Sự chênh lệch giữa train loss và validation loss không quá lớn, tuy nhiên với learning rate thấp và weight decay lớn, điều này cho thấy model có xu hướng overfitting, cần tăng số lượng dữ liệu đầu vào để tổng quát hóa. Tuy nhiên vì nguồn tài nguyên có hạn cũng như kĩ thuật lập trình còn yếu nên em chưa thể thực hiện được điều này.
* Biểu đồ “Actual vs Predicted Values” cho ta thấy cái nhìn tổng quan về cách mà mô hình dự đoán so với thực tế. Mô hình dựa đoán bám tương đối sát so với thực tế, tuy nhiên, có rất nhiều giá trị lệch hoàn toàn, điều này có thể dẫn đến sai số lớn khi chuyển về Centipawn.
* Hệ số R ^ 2 = 0.72 cho thấy mô hình có khả năng giải thích khá tốt dữ liệu, nhưng có rất nhiều điểm cách xa đường hồi quy, cho thấy mô hình có độ chính xác không cao và cần cải thiện.

## 2.5. Kết luận chương

Trong chương 2 em đã trình bày các kiến thức liên quan đến việc tìm kiếm nước đi tốt nhất trên bàn cờ vua. Để xử lý vấn đề này em đã chia nhỏ bài toán ra làm ba phần chính, bao gồm: bộ sinh nước đi nhanh, một cây tìm kiếm hoàn chỉnh, và một hàm lượng giá bàn cờ đánh giá một cách đầy đủ nhất. Ở phần bộ sinh nước đi, em đã liệt kê cách biểu diễn bàn cờ, cách biểu diễn nước đi, thực hiện các nước đi bằng bitboard và các phép bitwise. Với cây tìm kiếm em đã tập trung trình bày về cây tìm kiếm Negamax và các thuật toán xung quanh nó như: cắt tỉa alpha-beta, quiescence search, null-move pruning, Zobrist Hashing và Transposition Table, PVS, Iterative Deepening và Aspiration Windows cũng như các hàm sắp xếp nước đi theo các ưu tiên khác nhau, và cuối cùng là Late move redutions. Sau cùng em đã trình bày hai cách để có thể xây dựng ra hàm lượng giá bàn cờ, xây dựng hàm lượng giá tĩnh bao gồm: giai đoạn cờ, quân và vị trí quân, khả năng di chuyển, độ an toàn của vua, cấu trúc tốt, cột mở và nửa mở. Đối với hàm lượng giá dựa trên mô hình học sâu, em đã cố gắng tìm hiểu những kiến thức liên quan đến học sâu cơ bản cũng như trình bày cách em đã xây dựng một mô hình để có thể đánh giá bàn cờ và trả ra kết quả. Tuy nhiên vì kiến thức của bản thân còn hạn chế, em chưa thể xây dựng được một mô hình học sâu đủ nhanh và hiệu quả. Mô hình đánh giá bàn cờ không chính xác và chậm hơn rất nhiều so với hàm lượng giá tĩnh, một phần lo do mô hình học sâu phức tạp, phần khác là do mô hình học sâu không thể biên dịch ra mã máy như những hàm lượng giá tĩnh. Mô hình học sâu không học được đâu là quân cần phải được ưu tiên hơn

Dưới đây là bảng so sánh giữa cây tìm kiếm sử dụng hàm lượng giá tĩnh và mô hình học sâu. G1 kí hiệu cho lượng giá tĩnh và G2 kí hiệu cho mô hình học sâu. Ta so sánh với độ sâu 4, thời gian, số lượng nút duyệt, và nước đi tốt nhất mà hai hàm lượng giá mang lại. Cột cuối cùng là nước đi tốt hơn so sánh theo Stockfish 16 NNUE trên lichess.org. Fen của các thế cờ lần lượt là:

P1: "r3k2r/p1ppqpb1/bn2pnp1/3PN3/1p2P3/2N2Q1p/PPPBBPPP/R3K2R w KQkq - 0 1"

P2: "r2q1rk1/ppp2ppp/2n1bn2/2b1p3/3pP3/3P1NPP/PPP1NPB1/R1BQ1RK1 b - - 0 9"

P3: "rnbqkb1r/pp1p1pPp/8/2p1pP2/1P1P4/3P3P/P1P1P3/RNBQKBNR w KQkq e6 0 1"

P4: "2k5/5R2/3K4/8/8/8/8/8 w - - 12 7"

**Bảng 2.2.** So sánh tốc độ, tổng số nút duyệt, và nước đi tốt nhất của mô hình học sâu và hàm lượng giá tĩnh

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| FEN | G1time | G1nodes | G1move | G2time | G2nodes | G2move | Better move |
| P1 | 7.335s | 18890 | d5e6 | 42.77s | 11606 | e2a6 | e2a6 |
| P2 | 0.16s | 20141 | a7a6 | 16.31s | 5427 | f6e4 | a7a6 |
| P3 | 0.03s | 5323 | g7h8r | 12.83s | 4200 | g7h8b | g7h8r |
| P4 | 0.004s | 855 | f7d7 | 1.89s | 761 | d6d5 | f7d7 |

Như vậy ta có thể thấy đôi khi, mô hình học sâu vẫn có thể dự đoán chính xác hơn nếu cùng ở một độ sâu, tuy nhiên với thời gian tăng lên gấp vài chục thậm chí hàng trăm lần, hàm lượng giá tĩnh hoàn toàn có thể lấy ưu thế về tốc độ của mình để tìm kiếm sâu hơn, đồng thời hàm lượng giá tĩnh cũng cho kết quả tốt hơn so với mô hình học sâu.

# XÂY DỰNG CHESS BOT

## 3.1. Website Lichess.org

### 3.1.1. Giới thiệu

Lichess.org là một nền tảng cờ vua trực tuyến miễn phí, mã nguồn mở. Với mục tiêu phổ biến và nâng cao khả năng chơi cờ của cộng đồng toàn câu, Lichess cung cấp nhiều tính năng và dịch vụ đa dạng cho người chơi ở mọi trình độ.



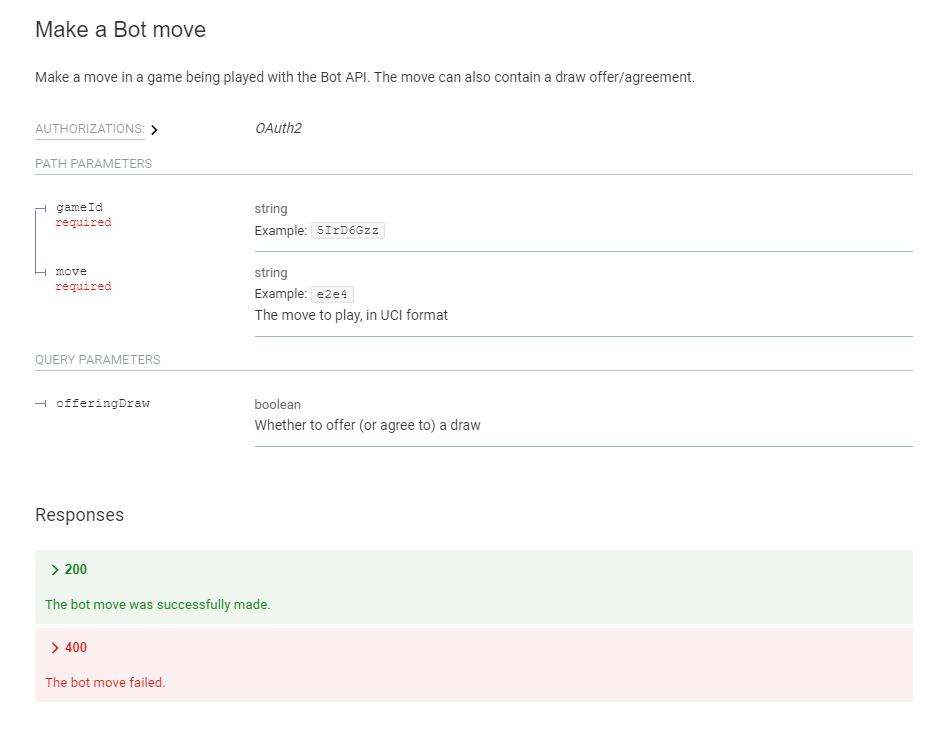
###### **Hình 3.1.** Chơi cờ trên website Lichess.org

Các tính năng chính của Lichess.org bao gồm:

* **Chơi cờ trực tuyến**: Người dùng có thể tham gia các ván cờ trực tiếp với những người chơi khác từ khắp nơi trên thế giới hoặc đấu với máy tính với nhiều cấp độ khó khác nhau.
* **Giải đấu**: Lichess tổ chức các giải đấu cờ vua theo nhiều hình thức khác nhau ở mọi thời điểm.
* **Phân tích ván cờ**: Công cụ phân tích mạnh mẽ giúp người chơi xem lại các ván đấu, tìm hiểu sai lầm và nước đi tốt nhất thông qua phân tích của các engine cờ vua hàng đầu như Stockfish.
* **Học cờ**: Cung cấp rất nhiều thế cờ, bài học, video từ cơ bản đến nâng cao
* Giao diện và thiết kế đơn giản thân thiện, dễ sử dụng.

### 3.1.2. Lichess API

Lichess API [13] là một tập hợp các API cung cấp khả năng truy cập và tương tác với các dịch vụ và dữ liệu của Lichess. API này là nền tảng cho việc phát triển các ứng dụng Chess Bot có thể kết nối đến người duf9 ng, cho phép các lập trình viên tích hợp và mở rộng các tính năng của Lichess.



###### **Hình 3.2.** API thực hiện một nước đi cho Bot

Các tính năng chính của Lichess API bao gồm:

* **Truy xuất thông tin:** API cung cấp khả năng truy cập thông tin về người chơi, giải đấu, ván cờ, và các câu đố cờ vua. Người dùng có thể lấy dữ liệu về lịch sử ván đấu, xếp hạng, và thống kê của các người chơi.
* **Tương tác trực tiếp:** Các lập trình viên có thể sử dụng API để tạo và quản lý các ván đấu, gửi nước đi, và theo dõi tiến trình của các trận đấu trực tiếp.
* **Phân tích và báo cáo:** API cung cấp các công cụ để phân tích ván cờ, bao gồm việc lấy thông tin phân tích từ các engine cờ vua và tạo báo cáo chi tiết về các ván đấu
* **Quản lý giải đấu:** API hỗ trợ việc tạo và quản lý các giải đấu, cho phép tự động hóa quá trình tổ chức và điều hành các sự kiện cờ vua trực tuyến.

### 3.1.3. Lichess Bot

Lichess Bot là một phần mở rộng của nền tảng Lichess, cho phép người dụng tạo và quản lý các tài khoản bot để thi đấu cờ vua tự động.

Bot của em cũng sử dụng Lichess API với một số chức năng sau:

* **Nhận và chấp nhận thách đấu**: Bot sử dụng API để nhận thông báo về các lời thách đấu từ người chơi và tự động chấp nhận các lời thách đấu này
* **Quản lý ván đấu**: Bot sử dụng API để gửi nước đi và theo dõi tiến trình của các trận đấu trực tiếp, đảm bảo rằng các nước đi thực hiện liên tục và chính xác.

## 3.2. Kết nối Chess bot với website Lichess.org

### 3.2.1. Thư viện Berserk

Thư viện Berserk [11] là một thư viện Python mã nguồn mở dành cho API của Lichess. Thư viện này cung cấp một cách tương tác với API của Lichess tiện lợi và thực hiện các tác vụ khác nhau. Chẳng hạn như:

* Xử lý các định dạng dữ liệu JSON và PGN
* Cung cấp xác thực dựa trên token cho tài khoản Lichess.
* Tích hợp với giao thức xcas thực OAuth2.
* Tự động chuyển đổi các giá trị thời gian thành đối tượng datetime.

Một số tính năng chính của thư viện berserk bao gồm:

* Tích hợp dễ dàng với bất kỳ đối tượng nào thực hiện mô hình Request – Session, bao gồm cả đối tương từ thư việt requests\_oauth.
* Truy cập vào nhiều endpoint của Lichess API, như quản lý tài khoản, phân tích trò chơi, các hoạt động trên bàn cờ, tương tác với bot, phát sóng, thách đấu, câu đố…
* Khả năng phát trực tiếp các sự kiện đến, trạng thái trò chơi và các dữ liệu thời gian thực khác từ API của Lichess.
* Hỗ trợ từ Python 3.8 và các phiên bản sau đó.

Thư viện Berserk cung cấp một cách tiện lợi và phong phú để các lập trình viên Python tương tác với API của Lichess và xây dựng các ứng dụng hoặc tích hợp xung quanh nền tảng cờ vua trực tuyến phổ biến này.

### 3.2.2. UCI Protocols

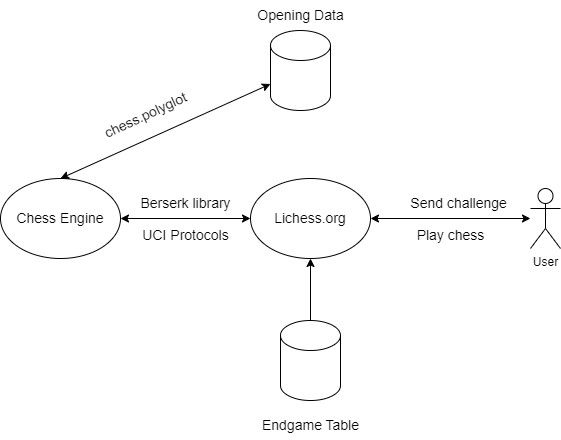
Universal Chess Interface (UCI) là một giao diện tiêu chuẩn để kết nối phần mềm cờ vua với các chương trình cờ máy tính. UCI được phát triển để tạo ra một giao diện chuẩn và đơn giản cho việc giao tiếp giữa các phần mềm cờ vua. Trong giao thức trên, việc truyền nước đi giữa các engine và giao diện người dung được thực hiện thông qua chuẩn algebraic notation.

Chuẩn algebraic notation đặc trưng cho các ô trên bảng cờ bằng các cặp kí tự. Cụ thể: Cột được ký hiệu bằng các chữ cái từ “a” đến “h”, bắt đầu từ cột bên trái của bàn cờ. Hàng được ký hiệu bằng các số từ “1” đến “8” bắt đầu từ hàng dưới cùng của bàn cờ. Ví dụ, vị trí của quân vua trắng ở trạng thái khởi đầu là e1.

Các nước đi được biểu diễn bằng cách sử dụng định dạng algebraric notation như sau:

* Một nước đi được biểu diễn bằng vị trí của quân cờ trước và sau khi di chuyển. Ví dụ e2e4.
* Nếu quân tốt được đẩy đến ô phong cấp, nó sẽ cần cung cấp thông tin quân được phong, ví dụ như phong quân hậu, ta có e7e8q.
* Các kí hiệu luôn được viết thường vì ta luôn biết lượt đi trên bàn cờ là của bên nào.

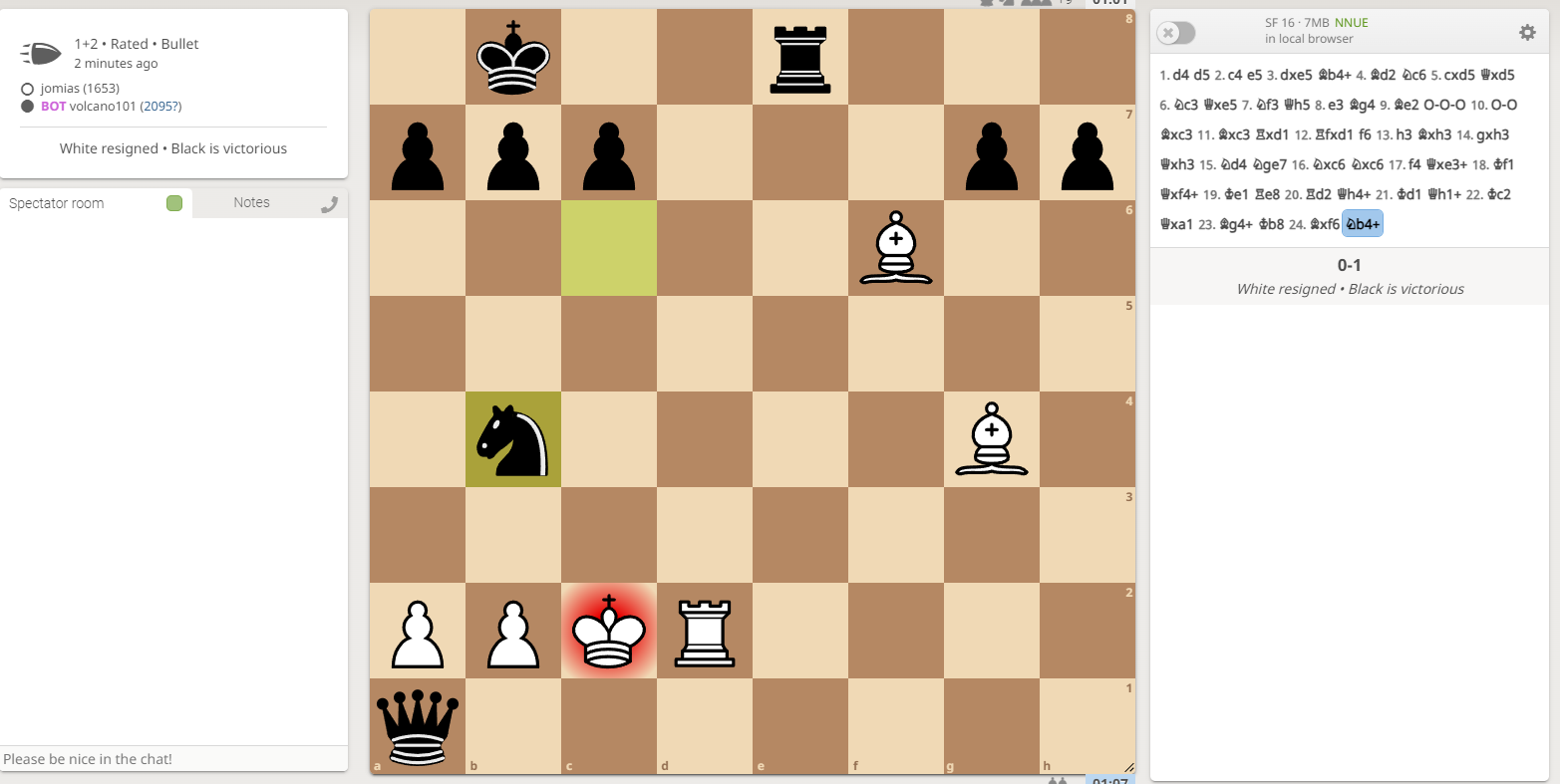
### 3.2.3. Cấu trúc của chương trình



###### **Hình 3.3.** Cấu trúc của chương trình

Chess Bot được xây dựng dựa trên các thuật toán của chương 2 sẽ được mở rộng và hoàn thiện để có thể trở thành một Lichess Bot.

* Các nước đi khai cuộc sẽ được lưu trong những file .bin sau đó, Chess Engine sẽ có thể lấy được các nước đi này thông qua thư viện chess.polyglot [14]
* Chess Bot có thể “giao tiếp” với Lichess.org thông qua Lichess API, tuy nhiên việc sử dụng thuần túy API sẽ mất rất nhiều thời gian để xử lý cũng như rất khó tổ chức mã nguồn, thay vì vậy ta sẽ sử dụng thư viện Berserk [15] và giao thức UCI.
* Chess Bot gửi request đến Lichess.org thông qua Lichess API để lấy các nước đi tàn cuộc.
* Người chơi sẽ gửi lời thách đấu đến Chess Bot thông qua website Lichess.org, với mỗi session, khi Chess Bot được kết nối, nó sẽ chỉ nhận lời thách đấu có ảnh hưởng đến rating, đồng thời bắt đầu trận đấu. Hiện tại thì Chess Bot chỉ có thể chơi một ván đấu ở một thời điểm.
* Người chơi thông qua giao diện thực hiện các nước đi trên bàn cờ.
* Mỗi khi đổi lượt đi, Chess Bot sẽ cập nhật bàn cờ lại một lần theo FEN mà Lichess phản hồi, sau đó thực hiện tìm kiếm nước đi tốt nhất lại từ đầu, trả về nước đi dưới format UCI để Lichess thực hiện nước đi và cập nhật bàn cờ.
* Chess Bot sẽ ưu tiên thực hiện tìm kiếm nước đi khai cục trong dữ liệu có trước, nếu không còn nước đi trong đó thì mới thực hiện tìm kiếm trên cây tìm kiếm. Ta cũng có thể quyết định dễ dàng việc có sử dụng dữ liệu liên quan đến bảng tàn cục hay không. Hiện này, tàn cục với 7 quân trở xuống đã được giải.
* Hiện tại Chess Bot chưa có một chiến lược xử lý thời gian hiệu quả, em cho Chess Bot sử dụng thời gian cho mỗi nước đi là thời gian còn lại / 40. Bởi vì trung bình một trận đấu sẽ có 40 nước đi và sẽ có độ trễ khi thực hiện nước đi bằng request đến website.



###### **Hình 3.4.** Chess Bot xây dựng đánh bại bản thân

# KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN

1. **Thuận lợi và khó khăn**

Về thuận lợi: Trong khoảng thời gian 3 tháng, dưới sự hướng dẫn nhiệt tình từ giảng viên hướng dẫn: TS. Cao Thị Luyên, sự giúp đỡ từ bạn bè và vốn kiến thức sẵn có từ ngôn ngữ lập trình python, niềm yêu thích với thuật toán, em đã có đủ thời gian chuẩn bị, thu thập thông tin liên quan đến tài mình đã chọn. Các nguồn tài liệu phong phú và sự hỗ trợ từ cộng đồng lập trình cờ vua cũng đã giúp em hoàn thành đề tài này.

Về khó khăn: Trong khi hoàn thành đồ án, do thiếu kinh nghiệm trong lĩnh vực học sâu, và với nguồn tài nguyên giới hạn, em đã không thể phát triển mô hình học sâu đến mức độ mà bản thân kì vọng. Lựa chọn sử dụng ngôn ngữ lập trình python gây ra nhiều vấn đề về tốc độ tìm kiếm.

1. **Kết quả đạt được**

* Tìm hiểu và áp dụng Bitboard, một kĩ thuật tối ưu việc biểu diễn và sinh nước đi.
* Tìm hiểu, áp dụng và kết hợp được các thuật toán tìm kiếm khác nhau trong cờ vua để xây dựng một cây tìm kiếm hoàn chỉnh và đầy đủ.
* Có thêm những kiến thức cơ bản, kĩ thuật liên quan đến học sâu và huấn luyện mô hình học sâu.
* Xây dựng được Chess Bot AI với khả năng tìm kiếm nước đi và chơi cờ ở mức độ khá có thể nhận thách đấu và chơi trên nền tảng của website Lichess.org.

1. **Những vấn đề còn tồn tại**

* Tốc độ tìm kiếm khi sử dụng mô hình học sâu còn rất chậm và mô hình học sâu đánh giá thiếu ổn định.
* Chess Bot chưa có chiến lược sử dụng thời gian tìm kiếm một cách hợp lý.
* Chess Bot chưa thể hoạt động mọi lúc và nhận thách đấu từ nhiều người cùng lúc.

1. **Hướng phát triển**

* Xây dựng lại Chess Bot bằng ngôn ngữ C, đồng thời mở rộng các loại thách đấu.
* Cải thiện mô hình học sâu để tăng thêm tính ổn định và sự chính xác.
* Tìm hiểu các thuật toán liên quan khác như Lazy SMP và tìm kiếm song song để cải thiện cây tìm kiếm hơn nữa.

# TÀI LIỆU THAM KHẢO

|  |  |
| --- | --- |
| [1] | https://www.chess.com/. |
| [2] | M. Malpe and M. S. Bhaila, “Analysis of Game Tree Search Algorithms Using Minimax Algorithm and Alpha-Beta Pruning,” in *International Conference on Artificial Intelligence (ICAI)*, New York, 2023. |
| [3] | D. Klein, Neural Networks For Chess, Cornell University, 2023. |
| [4] | P. Bijl and A. P. Tiet, *Exploring modern chess engine architectures,* Amsterdam: https://www.cs.vu.nl/~wanf/theses/bijl-tiet-bscthesis.pdf, 2021. |
| [5] | https://www.chessprogramming.org. |
| [6] | M. P. Schadd and M. H. Winands, “Quiescence Search for Stratego,” in *The 21st Benelux Conference on Artificial IntelligenceAt*, : Eindhoven, 2009. |
| [7] | O. David-Tabibi and N. Netanyahu, “Verified Null-Move Pruning,” *ICGA Journal,* vol. 25, 2002. |
| [8] | D. Breuker, J. V. D. Herik and J. Uiterwijk, “Replacement Schemes for Transposition Tables,” 1970. |
| [9] | P. Kannan, *http://pradu.us/,* http://pradu.us/, 2007. |
| [10] | J. Schaeffer, “The History Heuristic,” 1983. |
| [11] | https://github.com/maksimKorzh/bbc. |
| [12] | https://github.com/official-stockfish/nnue-pytorch/blob/master/docs/nnue.md. |
| [13] | https://lichess.org/api. |
| [14] | https://python-chess.readthedocs.io/. |
| [15] | https://github.com/lichess-org/berserk. |
| [16] | C. M. King, *Bitboard Chess engine in C,* https://www.youtube.com/watch?v=QUNP-UjujBM&list=PLmN0neTso3Jxh8ZIylk74JpwfiWNI76Cs, 2020. |
| [17] | https://chessify.me/blog/top-chess-engines. |