Mathematical Foundations and Machine Learning for Chess Position Evaluation

MAT399 - UG SEMINAR , LAKSHYA MISHRA

June 5, 2025

Formalizing Legal Moves in Chess using Logic and Set Theory

Introduction

This document provides a complete formalization of chess move legality using predicate logic and set theory. We cover both the abstract rules and concrete examples to demonstrate how these principles apply in practice.

1 Basic Definitions

Board Representation

Let the chessboard be modeled as:

$$P = \{♙, ♖, ♘, ♝, ♛, ♔, \}$$

- $S = \{a1, a2, ..., h8\}$: Set of all 64 squares
- $C = \{\text{White}, \text{Black}\}$: Player colors
- $\text{Piece}(p, c)$: Predicate meaning piece p belongs to color c
- $\text{On}(p, s)$: Piece p is on square s

Board State Function

Formally, the board is a function:

$$B : \{0, 1, \ldots, 7\} \times \{0, 1, \ldots, 7\} \to \mathcal{P} \tag{1}$$

where $\mathcal{P}$ is defined as:

$$\mathcal{P} = \{\text{empty}\} \cup \{\text{white/black} \times \text{piece type}\} \tag{2}$$

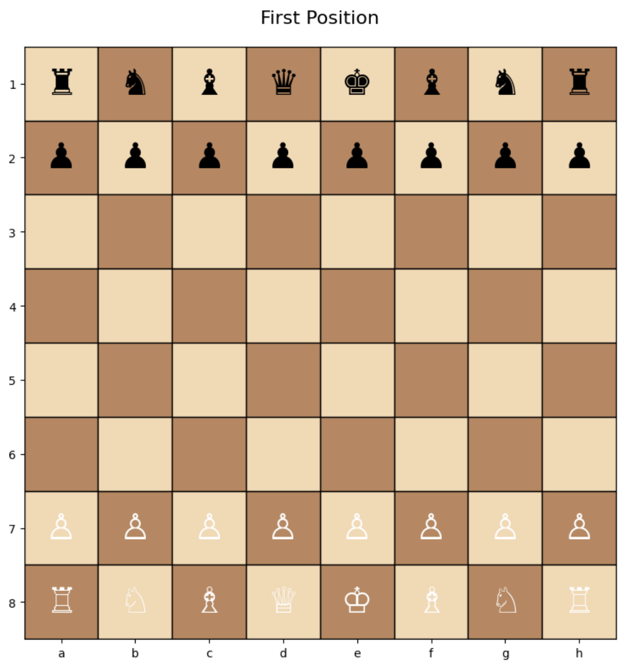

Figure 1: CHESS BOARD

2 Legal Move Formalization

Core Predicates

- $\text{Move}(p, s_1, s_2)$: Piece p moves from s_1 to s_2
- $\text{Legal}(p, s_1, s_2)$: The move is legal
- $\text{CanMoveTo}(p, s_1, s_2)$: Movement adheres to piece rules
- $\text{Occupied}(s)$: Square is occupied
- $\text{Enemy}(s, c)$: Square contains opponent's piece

Legal Move Definition

$$\text{Legal}(p, s_1, s_2) \iff \begin{cases} \text{On}(p, s_1) \\ \land\ \text{CanMoveTo}(p, s_1, s_2) \\ \land\ (\neg\text{Occupied}(s_2) \lor \text{Enemy}(s_2, \text{Color}(p))) \\ \land\ \neg\text{CheckAfter}(p, s_1, s_2) \end{cases} \tag{3}$$

3 Piece-Specific Movement Rules

1. Rook Movement

$$M_{\text{rook}}(x, y) = \{(i, y) : i \in [0, 7], i \neq x\} \cup \{(x, j) : j \in [0, 7], j \neq y\} \tag{4}$$

Obstruction condition:

$$\begin{aligned}
\text{obstructed row}((x,y),(i,y)) &\iff \exists k \in (\min(x,i)+1, \max(x,i)-1) : B(k,y) \neq \text{empty} \\
\text{obstructed col}((x,y),(x,j)) &\iff \exists k \in (\min(y,j)+1, \max(y,j)-1) : B(x,k) \neq \text{empty}
\end{aligned}$$

2. Knight Movement

$$M_{\text{knight}}(x,y) = \{(x \pm 1, y \pm 2), (x \pm 2, y \pm 1)\} \tag{5}$$

3. Bishop Movement

$$M_{\text{bishop}}(x,y) = \{(x \pm d, y \pm d) \mid d \in \mathbb{Z}, 1 \leq d \leq 7\}$$

4. Pawn Movement

Special rules include:

- Initial two-square move
- Diagonal captures
- En passant
- Promotion

4 Special Rules

Castling

$$\text{LegalCastle}(c, \text{side}) \iff \begin{cases} \neg\text{HasMoved}(k_c) \\ \wedge\ \neg\text{HasMoved}(r_c) \\ \wedge\ \text{EmptyBetween}(k_c, r_c) \\ \wedge\ \text{SafePath}(k_c) \end{cases} \tag{6}$$

En Passant

$$\text{EnPassant}(p, s_1, s_2) \iff \text{Pawn}(p) \wedge \text{EnPassantTarget}(s_2)$$

Promotion

$$\text{Promotion}(p, s_1, s_2) \iff \text{Pawn}(p) \wedge \text{Rank}(s_2) \in \{1, 8\}$$

5 Concrete Examples

Example 1: Black Pawn d7 → d5

$$\begin{aligned} \text{CanMoveTo}(\text{p}, d7, d5) &= \text{True} \quad \text{(Path clear)} \\ \neg\text{CheckAfter}(\text{p}, d7, d5) &= \text{True} \\ \Rightarrow \boxed{\text{Legal}(\text{p}, d7, d5) = \text{True}} \end{aligned}$$

Example 2: Illegal Knight Move b8 → b6

$$\boxed{\text{Legal}(\text{n}, b8, b6) = \text{False}} \quad \text{(Invalid L-shape)}$$

Move	**Valid?**	**Reason**
...d7→d5	Yes	Legal pawn double move
...b8→b6	No	Invalid knight move
Castling	Depends	King/rook conditions

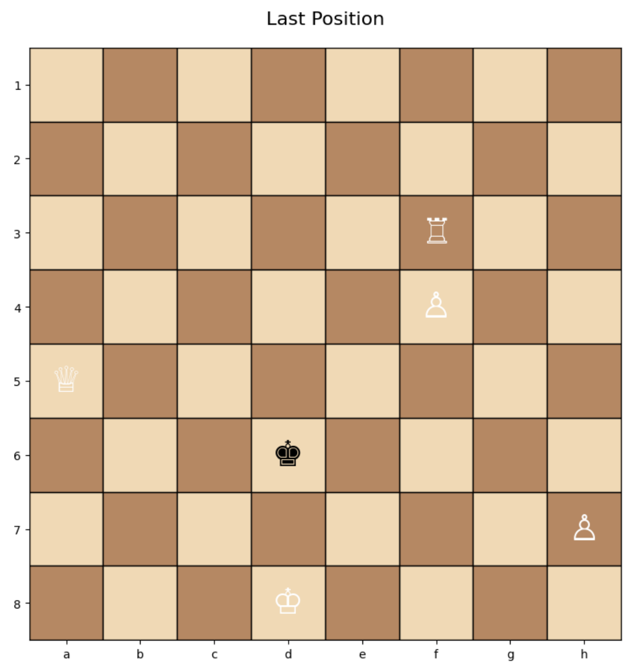

Figure 2: after game

6 Symmetry in Chess: A Group Theoretic Approach

6.1 Introduction

In this section, we explore the symmetries of the chessboard and chess positions using concepts from group theory. Symmetry plays a critical role in pattern recognition, data augmentation, and state-space reduction in our chess-based machine learning models.

6.2 The Symmetry Group of the Chessboard

The standard 8×8 chessboard possesses spatial symmetries similar to those of a square. The set of all such symmetries forms the dihedral group D_4 of order 8.

6.2.1 Elements of D_4

Let the elements of D_4 be:

- e: identity
- r_{90}, r_{180}, r_{270}: rotations by 90, 180, and 270 degrees
- h: reflection about the horizontal axis
- v: reflection about the vertical axis
- d: reflection about the main diagonal
- a: reflection about the anti-diagonal

6.2.2 Example: Board Invariance

If we define a function f : Board $\rightarrow$ Features that is symmetric under r_{180}, i.e.

$$f(B) = f(r_{180}(B)) \tag{7}$$

Then the board position and its 180-degree rotation are treated identically by the model. This can help in training efficiency:

$$\text{augmented dataset size} = \text{original} \times |G| \tag{8}$$

$$G \leq D_4 \tag{9}$$

6.3 Symmetry of Piece Movement

6.3.1 Example 1: Knight Moves

A knight move from (x, y) to $(x \pm 1, y \pm 2)$ and $(x \pm 2, y \pm 1)$ is invariant under 90-degree rotation. Thus,

$$M_{\text{knight}} = \text{orbit of } (\pm1, \pm2) \cup (\pm2, \pm1) \text{ under } D_4 \tag{10}$$

This symmetry can be encoded as a set under the group action of D_4.

6.3.2 Example 2: Rook and Bishop

- **Rook**: symmetric under horizontal/vertical reflections and 180-degree rotation.
- **Bishop**: symmetric under diagonal/anti-diagonal reflections.

Thus, piece movements can be grouped into equivalence classes under D_4 actions.

6.4 Application in ML Models

By using these symmetries:

- We reduce redundant representations of board states.
- Improve generalization by treating symmetric positions equivalently.
- Enable data augmentation by applying group actions to generate new samples.

6.5 Conclusion

Group theory offers a powerful abstraction for understanding and exploiting symmetry in chess. Incorporating D_4 actions into data preprocessing and model design contributes to both interpretability and performance in chess-based ML applications.

7 Optimization and Calculus in Machine Learning for Chess Position Evaluation

7.1 Abstract

This document outlines the mathematical foundations of optimization and calculus in training machine learning models for chess position evaluation. We focus on gradient-based optimization, loss functions, derivatives, and how these are applied in real training pipelines. Numerous examples, Python code, and diagrams are included to illustrate concepts clearly.

7.2 Introduction

In machine learning, particularly in neural networks used for evaluating chess positions, optimization and calculus are foundational tools. They guide how models learn from data, update their internal parameters, and minimize errors.

7.3 Objective Function and Loss

Let be our ML model evaluating a board position , where represents parameters (weights). Given true evaluation , the loss is:

$$L(\theta) = \mathbb{E}_{(X,y)\sim\mathcal{D}}\left[(f_\theta(X) - y)^2\right] \tag{11}$$

This is the **Mean Squared Error (MSE)**, a common choice for regression tasks.

7.4 Example

Suppose the evaluation for position is +0.5, and the model outputs . Then:

$$L(\theta) = (0.2 - 0.5)^2 = 0.09 \tag{12}$$

7.5 Gradient Descent

We update weights by moving in the negative direction of the gradient of the loss:

$$\theta \leftarrow \theta - \eta \nabla_\theta L(\theta) \tag{13}$$

Where η is the learning rate.

7.6 Example

Let

$$\theta = 1, \eta\nabla_\theta L(\theta) = 0.3, \eta = 0.1 \tag{14}$$

$$\theta \leftarrow 1 - 0.1 * 0.3 = 0.97 \tag{15}$$

7.7 Backpropagation and Chain Rule

To compute gradients, we use the chain rule. For a neural network:

$$L = (f_\theta(X) - y)^2 \tag{16}$$

If,

$$(f_\theta(X) = W_2(\sigma(W_1X)) \tag{17}$$

where is an activation function:

$$\delta(L)/\delta W_2 = 2 * (f_\theta(X) - y) * \sigma(W_1X) \tag{18}$$

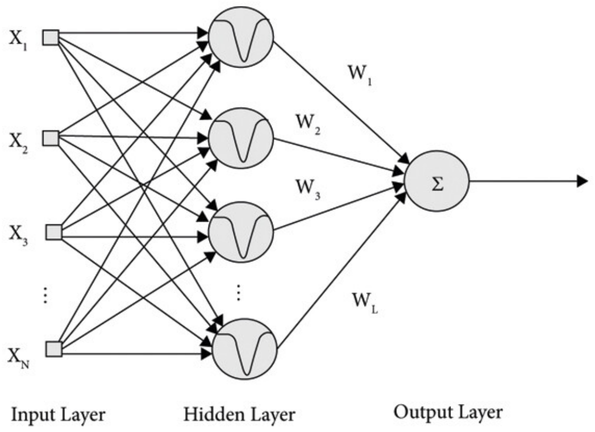

Figure 3: Simple Feedforward Neural Network for Evaluation

7.8 Regularization

To prevent overfitting, we use regularization:

- **L2 Regularization**: Encourages smaller weights.
- **Dropout**: Randomly removes neurons during training.

7.9 Convexity and Convergence

In linear models, the loss landscape is convex, ensuring convergence to global minima. However, deep models may get stuck in local minima or saddle points.

7.10 Example Training in PyTorch

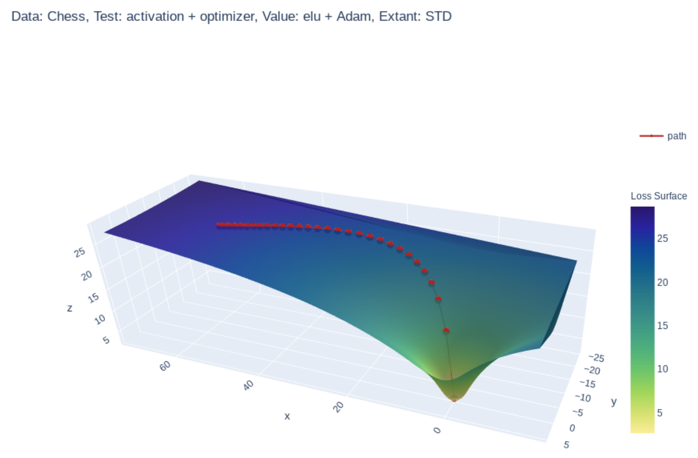

Figure 4: Loss landscape for shallow vs deep models

Listing 1: PyTorch Training Loop

```
import torch
import torch.nn as nn

class LSTM(nn.Module):
    def __init__(self, input_size, hidden_layer_size, output_size):
        ...
loss_fn = nn.MSELoss()
optimizer = torch.optim.Adam(model.parameters(), lr=0.001)

for i in range(epochs):
    for seq, labels in train_inout_seq:
        optimizer.zero_grad()
        y_pred = model(seq)
        single_loss = loss_function(y_pred, labels)
        single_loss.backward()
        optimizer.step()
```

7.11 Application to Chess

Each input is a vector representation (e.g., one-hot, FEN embedding) of a board. The output is a scalar evaluation, mimicking Stockfish or human-labeled centipawn scores.

Training Loop

- Forward pass: compute

$$f_\theta(X) \tag{19}$$

- Compute loss:

$$L(\theta) \tag{20}$$

- Backward pass:

$$\nabla_\theta L(\theta) \tag{21}$$

- Update

$$\theta \tag{22}$$

7.12 Conclusion

Optimization and calculus provide the mathematical scaffolding for learning in ML models. With these tools, we can teach models to approximate evaluation functions critical for chess engines and analysis.

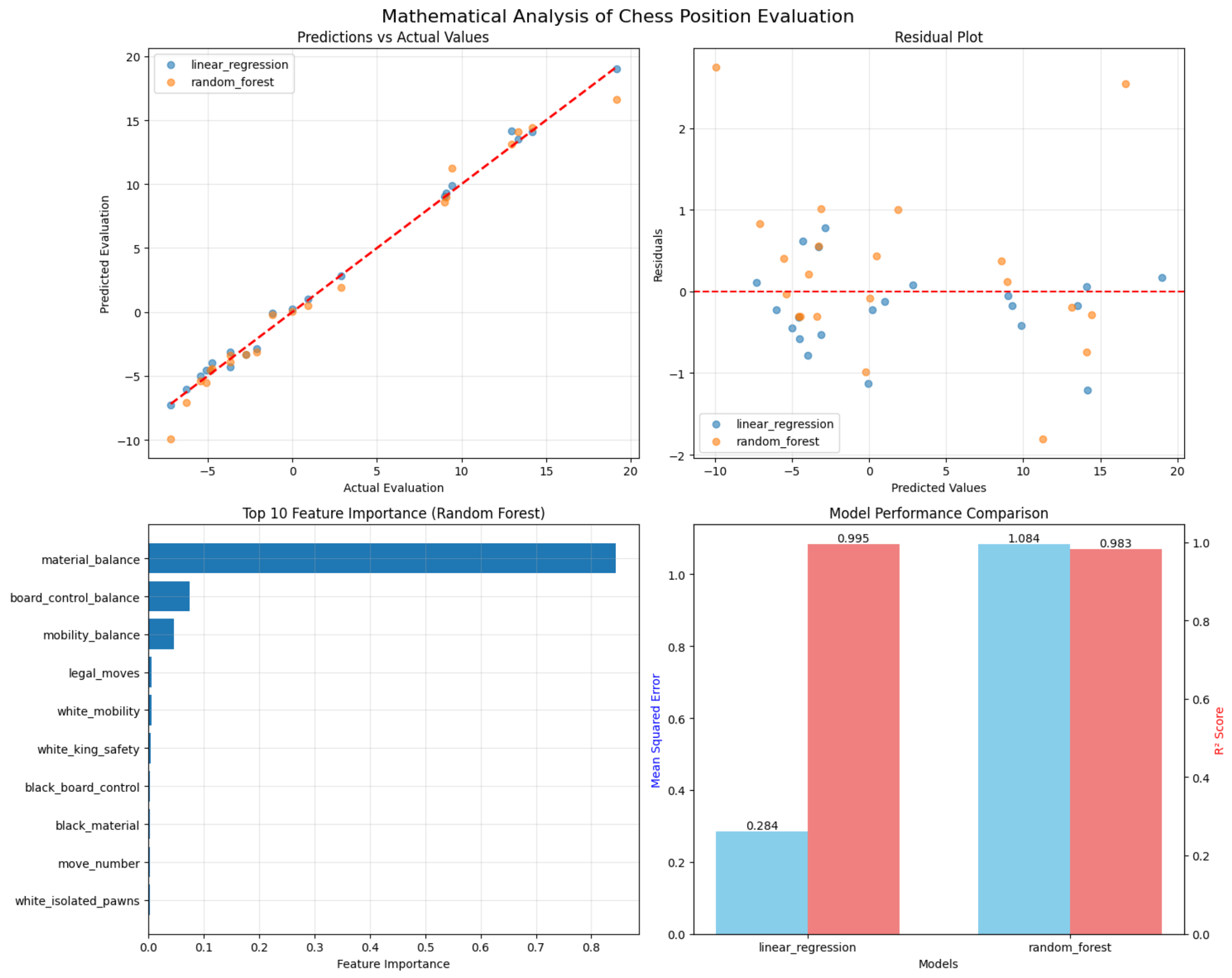

Figure 5: Analysis