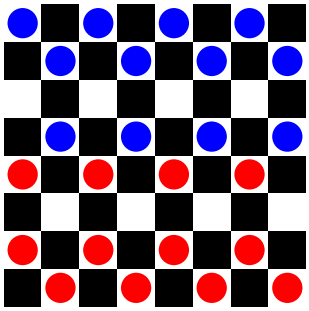
Checkers Game in Brief

1. **Objective:** Capture all opponent pieces or block their moves.
2. **Board:** Played on an 8x8 board using only the dark squares.
3. **Pieces:**
   - Each player starts with 12 pieces.
   - Pieces move diagonally forward, one square at a time (unless "kinged").
4. **Rules:**
   - Pieces move diagonally forward.
   - Capture by jumping over an opponent's piece to an empty square.
   - Reach the opponent's back row to get "kinged," allowing forward and backward movement.
5. **Winning:** Win by capturing all opponent pieces or blocking their moves.

Below is an example of a checkers board:



Designing a Learning System for the Checkers Game

The process of designing a learning system for Checkers, based involves the following steps:

1. Choosing the Training Experience

The training experience determines how a learning system gains knowledge and influences its ability to improve performance. Below are some examples of training experiences commonly used in game-based learning systems:

a. Self-Play

- The program plays games against itself.
- This approach generates a large amount of training data autonomously.
- **Advantages:**
  - Unlimited data generation without external intervention.
  - Encourages exploration of a wide range of board states.
- **Disadvantages:**
  - May not encounter strategies used by expert players unless explicitly modeled.

b. Human Games

- The program observes games played by humans to learn strategies and patterns.
- Training examples include board states and moves made by human players.
- **Advantages:**
  - Provides exposure to realistic and expert-level strategies.
  - Reduces the risk of overfitting to suboptimal self-play strategies.
- **Disadvantages:**
  - Limited by the availability of human game data.
  - Requires preprocessing to extract meaningful features.

c. Expert Feedback

- The program receives direct feedback from experts on the quality of each move.
- Feedback may include suggested corrections or ratings for the current board state.
- **Advantages:**
  - Accelerates learning by leveraging domain expertise.
  - Reduces the need for extensive trial-and-error exploration.
- **Disadvantages:**
  - Relies heavily on the availability and expertise of human trainers.
  - May introduce bias based on the specific expert's playstyle.
- **Experience** E: The system learns by playing games against itself.
- **Outcome**: Generates sequences of board states and final outcomes (win, lose, or draw).
- **Advantages**: No external trainer required; unlimited training data can be generated.

2. Choosing the Target Function

The target function represents the knowledge the system is learning:

- **Target Function** $V(b)$: Evaluates the quality of a board state b.
- **Definition of** $V(b)$**:**

$$V(b) = \begin{cases} +100 & \text{if } b \text{ is a winning board state,} \\ -100 & \text{if } b \text{ is a losing board state,} \\ 0 & \text{if } b \text{ is a drawn board state,} \\ V(b') & \text{for intermediate states, where } b' \text{ is the best achievable state.} \end{cases}$$

3. Choosing a Representation for the Target Function

The target function $V(b)$ must be represented in a computable and learnable form:

- **Linear Representation:**

$$V(b) = w_0 + w_1 \cdot x_1 + w_2 \cdot x_2 + \cdots + w_n \cdot x_n$$

- **Features:**
  - x_1: Number of black pieces,
  - x_2: Number of red pieces,

- x_3: Number of black kings,
- x_4: Number of red kings,
- x_5: Number of black pieces threatened by red,
- x_6: Number of red pieces threatened by black.

Consider a board state b with the following features:

- $x_1 = 8$: Number of black pieces,
- $x_2 = 7$: Number of red pieces,
- $x_3 = 2$: Number of black kings,
- $x_4 = 1$: Number of red kings.

The evaluation function $V(b)$ is represented as a linear combination of these features:

$$V(b) = w_0 + w_1 \cdot x_1 + w_2 \cdot x_2 + w_3 \cdot x_3 + w_4 \cdot x_4$$

Substituting the feature values:

$$V(b) = w_0 + w_1 \cdot 8 + w_2 \cdot 7 + w_3 \cdot 2 + w_4 \cdot 1$$

Where:

- w_0: Bias term,
- w_1, w_2, w_3, w_4: Weights representing the importance of each feature.

4. Choosing a Learning Algorithm

The learning algorithm updates $V(b)$ based on training data:

- **Training Examples:** Pairs $(b, V_{\text{train}}(b))$, where:

$$V_{\text{train}}(b) = \hat{V}(\text{Successor}(b)).$$

- **Weight Update Rule (LMS Algorithm):**

$$w_i \leftarrow w_i + \eta \cdot \left(V_{\text{train}}(b) - \hat{V}(b)\right) \cdot x_i$$

where:

- η: Learning rate,
- $V_{\text{train}}(b) - \hat{V}(b)$: Prediction error.

Squared Error Formula

The squared error is a measure used to quantify the difference between the predicted value and the actual value for a given dataset. It is defined as:

$$E = \frac{1}{2} \sum_{i=1}^{n} \left(V_{\text{train}}(b_i) - \hat{V}(b_i)\right)^2$$

Where:

- E: Squared error.
- n: Number of training examples.
- $V_{\text{train}}(b_i)$: Target value (actual value) for the i-th training example.
- $\hat{V}(b_i)$: Predicted value for the i-th training example.

5. Final System Design

The Checkers learning system comprises four modules:

1. **Performance System:** Plays Checkers using $\hat{V}(b)$ and generates a sequence of moves.
2. **Critic:** Evaluates game outcomes and assigns training values $V_{\text{train}}(b)$.
3. **Generalizer:** Updates weights of $V(b)$ using the LMS algorithm.
4. **Experiment Generator:** Generates new games by having the program play against itself.

Flowchart for the Learning System

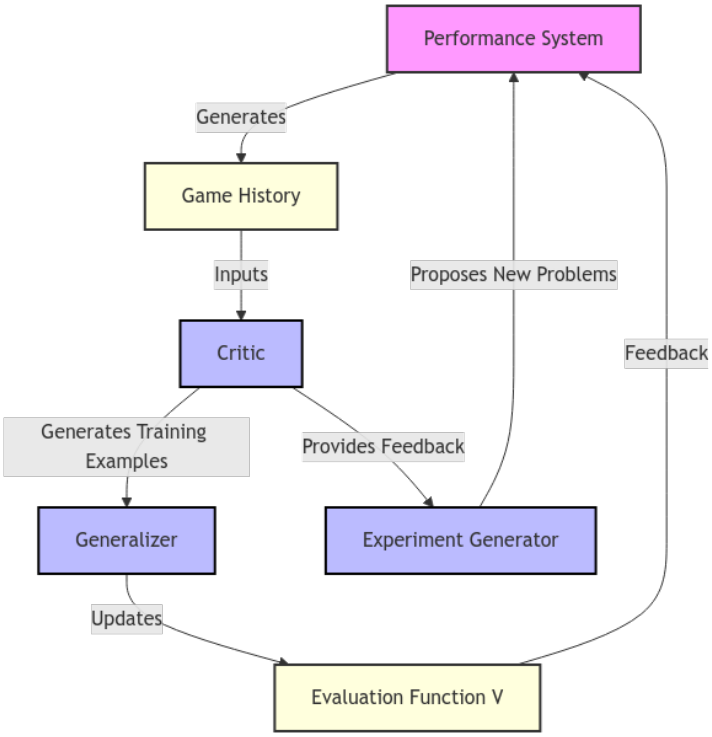


Figure 1: Flowchart for Checkers Learning System

Scenario: Checkers Game - Training via Self-Play

Step 1: Initial Board State

The system initializes with the following board state b_0:

- **Black Pieces (Player's side):** 8 pieces (including 2 kings).
- **Red Pieces (Opponent's side):** 7 pieces (including 1 king).
- **Features Extracted:**
  - $x_1 = 8$: Number of black pieces.
  - $x_2 = 7$: Number of red pieces.
  - $x_3 = 2$: Number of black kings.
  - $x_4 = 1$: Number of red kings.
  - $x_5 = 1$: Number of black pieces threatened by red.

– $x_6 = 2$: Number of red pieces threatened by black.

The evaluation function $V(b)$ is initialized as:

$$V(b) = w_0 + w_1 \cdot x_1 + w_2 \cdot x_2 + w_3 \cdot x_3 + w_4 \cdot x_4 + w_5 \cdot x_5 + w_6 \cdot x_6$$

Initial weights:

$$w_0 = 0.5,\ w_1 = 1.0,\ w_2 = -1.0,\ w_3 = 2.0,\ w_4 = -0.5,\ w_5 = -1.0,\ w_6 = 1.5$$

Step 2: Move Played

The system selects the **best move** based on $V(b_0)$, leading to a successor state b_1. The new features are:

- $x_1 = 7$: One black piece is captured.
- $x_2 = 6$: One red piece is captured.
- $x_3 = 2$: Black kings remain the same.
- $x_4 = 1$: Red kings remain the same.
- $x_5 = 0$: No black pieces are threatened.
- $x_6 = 3$: Number of red pieces threatened increases.

Step 3: Training Value Computation

The training value $V_{\text{train}}(b_0)$ is computed from the evaluation function of b_1:

$$V_{\text{train}}(b_0) = \hat{V}(b_1) = w_0 + w_1 \cdot x_1 + w_2 \cdot x_2 + w_3 \cdot x_3 + w_4 \cdot x_4 + w_5 \cdot x_5 + w_6 \cdot x_6$$

Substitute the feature values for b_1:

$$V_{\text{train}}(b_0) = 0.5 + (1.0 \cdot 7) + (-1.0 \cdot 6) + (2.0 \cdot 2) + (-0.5 \cdot 1) + (-1.0 \cdot 0) + (1.5 \cdot 3)$$

$$V_{\text{train}}(b_0) = 0.5 + 7 - 6 + 4 - 0.5 + 0 + 4.5 = 9.5$$

Step 4: Weight Update

Using the LMS algorithm, the weights are updated:

$$w_i \leftarrow w_i + \eta \cdot (V_{\text{train}}(b_0) - \hat{V}(b_0)) \cdot x_i$$

Assume:

- $\eta = 0.1$: Learning rate.
- $\hat{V}(b_0) = 8.0$: Predicted value for b_0.

Error:

$$\Delta V = V_{\text{train}}(b_0) - \hat{V}(b_0) = 9.5 - 8.0 = 1.5$$

Update the weight for x_1 (number of black pieces):

$$w_1 \leftarrow w_1 + \eta \cdot \Delta V \cdot x_1 = 1.0 + 0.1 \cdot 1.5 \cdot 8 = 1.0 + 1.2 = 2.2$$

Repeat this process for all features x_2, x_3, x_4, x_5, x_6.

Step 5: Iteration

The updated weights are used for subsequent moves. Over time, the system improves its evaluation function $V(b)$, leading to better gameplay strategies.