

Flowchart:

```
A[Self-Play Game Generation] --> B[MCTS Search]
B --> C[Improved Policy  $\pi$ ]
C --> D[Select Move & Advance Game State]
D --> A
D --> E[Game Outcome  $z$  (win/loss)]
subgraph Data Collection
  F[(State  $s$ , Search Policy  $\pi$ , Outcome  $z$ )]
end
D --> F
F --> G[Training Data Pool]
G --> H[Neural Network Training]
H --> I[Updated Network  $\theta$ ]
I --> B
I --> A
```

Diagram Components:

Self-Play Game Generation (A):

- The process starts with the current best network playing games against itself.
- The game state (s) is initialized (typically as an empty board) and advanced move-by-move.

MCTS Search (B):

- For each state s encountered during self-play, a Monte Carlo Tree Search (MCTS) is executed.
- **Algorithm Details:**
 - **Selection:** Starting from the root, the search tree is traversed by selecting moves that maximize the PUCT formula:
$$Q(s, a) + c_{\text{uct}} \cdot P(s, a) \cdot [\sqrt{(\sum_b N(s, b)) (1 + N(s, a))}].$$
 - **Expansion and Evaluation:** When a leaf node is reached, it is expanded and evaluated once using the neural network f_{θ} . The network outputs both a move probability vector p and a value v .
 - **Backup:** The value v is backed up along the search path to update the action-value estimates (Q) and visit counts (N) for each edge.
- The outcome of MCTS is an improved policy (π), which reflects the distribution of visit counts (often after exponentiation and normalization).

Improved Policy π (C) & Move Selection (D):

- The search policy π is used to select the next move from the current state.
- A move is chosen (often via sampling or selecting the maximum visit count) to advance the game to a new state, which is then fed back into the self-play loop.

Game Outcome (z) (E):

- When the self-play game ends (by reaching a terminal state such as both players passing or resignation), the final game outcome z (a scalar win/loss reward) is determined.

Data Collection (F) and Training Data Pool (G):

- At each move, the tuple (state s , improved policy π , outcome z) is recorded.
- These tuples are stored in a training data pool for later use.

Neural Network Training (H):

- The neural network f_θ (which serves both as the policy and value network) is trained using the collected tuples.
- **Training Details:**
 - The loss function is a combination of the mean-squared error (between the predicted value v and the game outcome z) and a cross-entropy term (between the network's policy output p and the improved search policy π), plus regularization.
 - Gradient descent (with momentum and learning rate annealing) is applied to update the parameters θ .

Updated Network f_θ (I):

- The updated network is then used in subsequent self-play games and MCTS evaluations, closing the reinforcement learning loop.

Object Hierarchies and Data Structures:

Search Tree Nodes (in MCTS):

- **Node (s):** Represents a board position.
- **Edge (s, a):** For each legal action a from state s , the edge stores:
 - **N(s, a):** Visit count.
 - **W(s, a):** Total accumulated value from simulations.
 - **Q(s, a):** Mean action value (W divided by N).
 - **P(s, a):** Prior probability (from the network's output).

Neural Network Architecture:

- **Input:** A 19×19×17 stack (including current and past board positions, and a binary feature for the player to move).
- **Residual Tower:** A series of convolutional layers arranged in residual blocks to extract features.
- **Heads:**
 - **Policy Head:** Outputs move probabilities (p) for each board intersection plus the pass move.
 - **Value Head:** Outputs a scalar evaluation (v) in the range [-1, 1].

Training Data Tuple (s, π , z):

- **s:** The board state.
- **π :** The search-improved policy from MCTS.
- **z:** The final game outcome, interpreted from the perspective of the current player.

Monte Carlo Search Tree Details:

Overview:

Figure 2 zooms into the MCTS algorithm that is integral to selecting moves during self-play. The figure is broken into four stages:

(a) Selection Phase:

- **Traversal of the Search Tree:**
 - Starting from the root node (the current game state), the algorithm selects child nodes based on a combination of:
 - The **action value $Q(s,a)$** (the average evaluation from previous simulations).
 - An **exploration term $U(s,a)$** , which is proportional to the prior probability $P(s,a)$ (given by the network) and inversely related to the visit count $N(s,a)$.
 - The selection is governed by the PUCT (Predictor + Upper Confidence bounds for Trees) formula: $a = \operatorname{argmax}(Q(s, a) + c_{puct} \cdot \frac{P(s,a)\sqrt{\sum_b N(s,b)}}{1+N(s,a)})$
 - This balances exploitation (choosing moves with high estimated value) and exploration (trying less-visited moves).

(b) Expansion and Evaluation:

- **Leaf Node Expansion:**

- Once the traversal reaches a leaf node (a state that has not been fully explored), the node is expanded by generating all legal moves from that state.
- **Neural Network Evaluation:**
 - The leaf node's board state is passed to the neural network $f_{\theta}(\cdot)$, which outputs:
 - A set of prior probabilities $P(s, \cdot)$ for the new moves.
 - A value $V(s)$ that estimates the chance of winning from that state.
 - These priors are used to initialize the new edges in the tree.

(c) Backup (Backpropagation) Phase:

- **Propagating the Value:**
 - The evaluation $V(s)$ from the expanded node is backed up along the path taken during selection.
 - At each node along the path, the visit counts $N(s,a)$ are incremented, and the action values $Q(s,a)$ are updated to reflect the mean of the evaluations from simulations passing through that edge.
- **Virtual Loss:**
 - A mechanism (virtual loss) is used to prevent multiple search threads from exploring the same node simultaneously.

(d) Play Phase (Move Selection):

- **Determining the Move:**
 - After many simulations, the search yields visit counts for each move at the root.
 - The final move probabilities π are computed by normalizing these counts (often after applying a temperature parameter τ to adjust exploration).
 - A move is then selected based on these probabilities, and the search tree is updated for the next move.

Self Play Pipeline

Overview:

Figure 1 is divided into two panels that capture the two main phases of the system's learning loop.

Panel (a): Self-Play Generation

- **Process Flow:**
 - The current best network (denoted as f_{θ}) is used to play a full game against itself.

- At each time step s_t , the system invokes an MCTS search (detailed in Figure 2) that leverages the network's outputs to compute an improved move distribution π_t .
- Moves are then selected—either stochastically or deterministically—from these probabilities to advance the game state.
- **Outcome Recording:**
 - When a game terminates (due to both players passing, resignation, or reaching a maximum move limit), the final outcome z (win or loss from the perspective of the current player) is determined.
 - During the game, tuples $(\mathbf{s}, \boldsymbol{\pi}, \mathbf{z})$ are recorded for every move.

Panel (b): Neural Network Training

- **Input and Output:**
 - The neural network takes the raw board state s_t as input—a $19 \times 19 \times 17$ representation that encodes the current board position, previous moves, and the player to move.
 - It produces two outputs: a policy vector p_t (a probability distribution over all legal moves) and a scalar value v_t (an estimate of the win probability).
- **Training Objective:**
 - The network is trained to minimize a loss function that has two components:
 - A cross-entropy loss that aligns the network's policy p_t with the MCTS-derived policy π_t .
 - A mean-squared error loss that makes the predicted value v_t match the actual game outcome z .
 - The new parameters θ are then used in the next round of self-play, closing the feedback loop.

Note: also how they check for generalizability/model fidelity. Make sure model is not overfitting/underfitting(how do we check that out).