

About This Project

This project was supervised by Andrew Shulman and Evangelos Kobotis. There were weekly zoom meetings in which we went over theoretical points, set goals and discussed our results. Our main goal was to understand how we can teach computers to play chess and consider a variety of methods that culminate with the more modern Machine Learning approach.

The Game of Chess

Chess is a traditional board game that has been enjoyed by everyone from kings and queens to schoolchildren and hobbyists. It is frequently referred to as a game of strategy, skill, and patience. In the sixth century AD, the game of chess was invented in India and swiftly moved to Persia and eventually to the Islamic world. From there, it traveled to Europe in the ninth century, where the nobility took a liking to it. Chess is now played across the world by players of all ages and socioeconomic backgrounds. On a square board with 64 squares organized in an 8x8 grid, chess is played. A king, a queen, two rooks, two knights, two bishops, and eight pawns are given to each player at the start of the game. The goal of the game is to checkmate your opponent's king, which means trapping it so that it cannot escape capture.

Chess and Computers

Looking back at the history of the game, chess computation is something relatively new, dating back about 70 years ago. The first chess program was created by Alan Turing and David Champernowne in 1948 and since then, we have developed chess programs that have become stronger as time has passed.



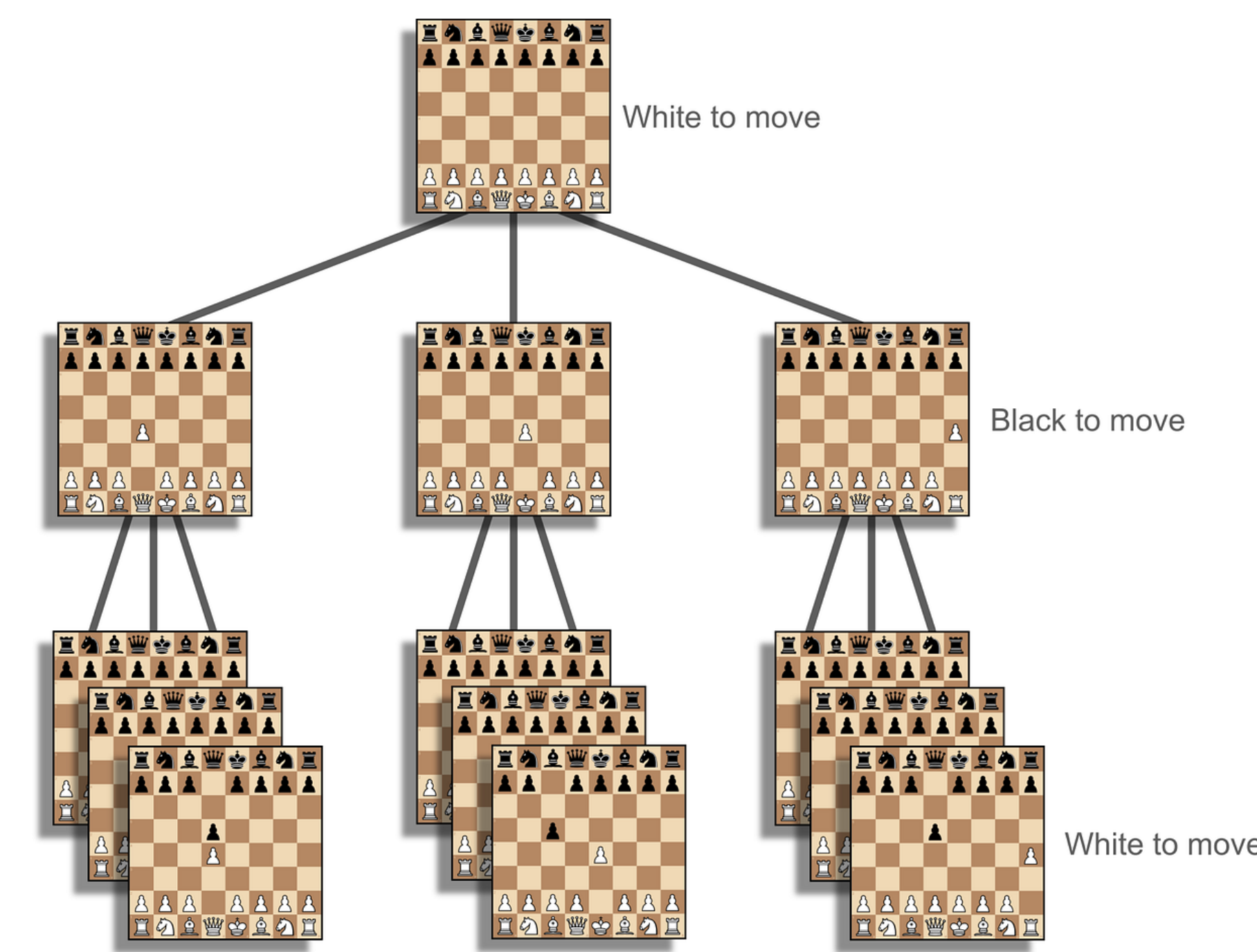
In 1997, Deep Blue, a chess-playing supercomputer, defeated Grandmaster Garry Kasparov and becoming the first computer to beat a world-champion chess player in a classic match format. Chess program has never stopped developing and continues to improve, surpassing humans by compute billions of scenarios within a second. The highest-rated computer chess program, Stockfish, has more than 3500 ELO now.

Chess Parametrization

In our project, we have created a chess board using a Python nested array. This involved generating eight arrays that corresponded to the eight columns of a chess board, with each array containing elements that represented individual squares on the board. The chess pieces were represented by integer values. We have implemented a 6-digit Operation Code that allows users to access and move the chess pieces. The first two digits indicate the type of move (non-capture/capture, promotion, or castling), while the next two digits indicate the starting square of the piece that the player wants to move. Finally, the last two digits specify the destination square of the piece after the move is made. By utilizing this Operation Code, users are able to input all possible moves in a game of chess using only six digits.

The Minimax Algorithm

The minimax algorithm is a decision-making method used in chess programming and other two-player games. It finds the optimal move for one player by assuming their opponent will always make the move that is best for them. The algorithm gets its name from maximizing the current player's score while minimizing the opponent's score.



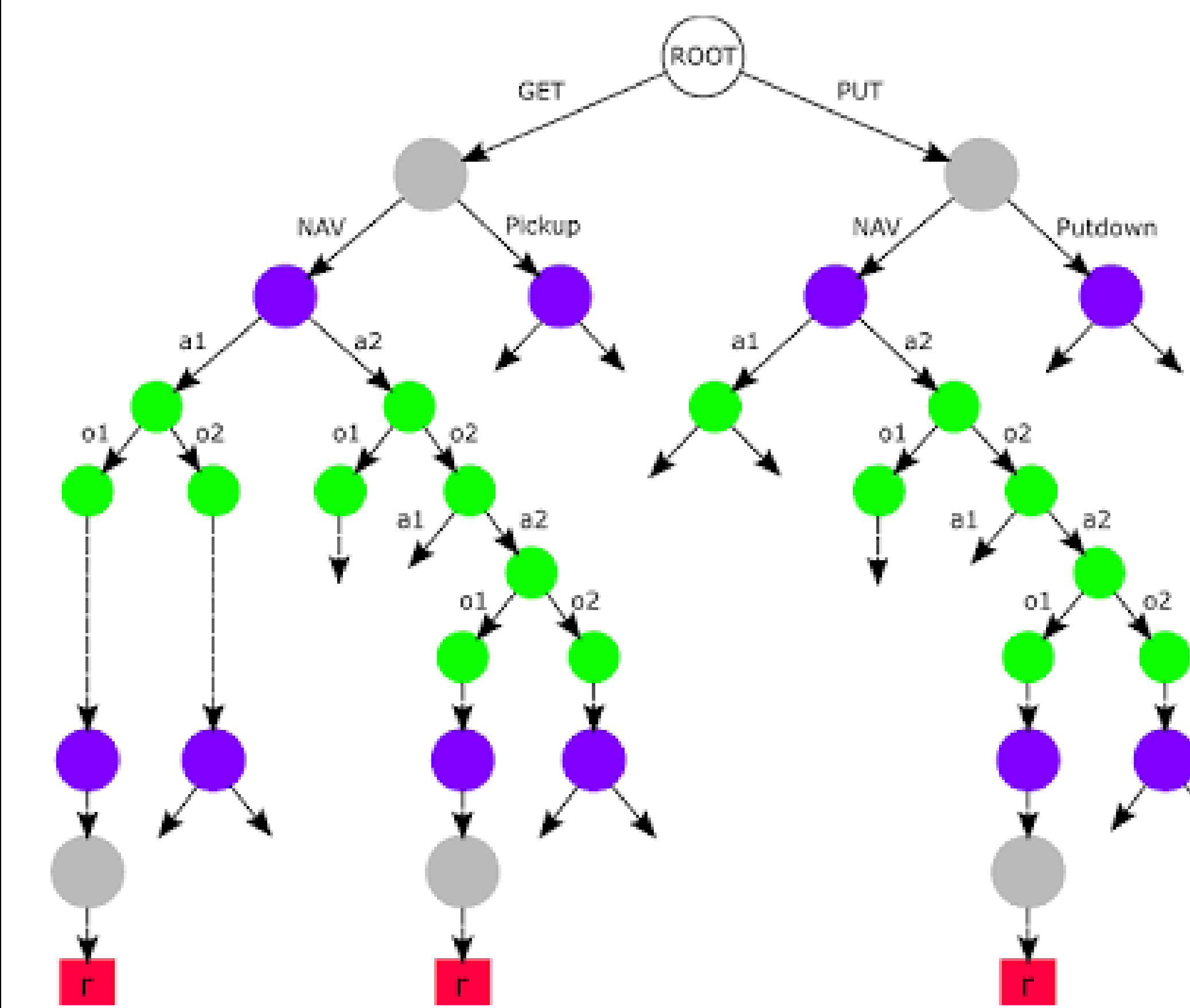
The algorithm explores the *game tree*, where the current board position is the root, and each branch represents a possible sequence of moves. Since chess has too many possible moves for a computer to analyze every possible game outcome to checkmate, the minimax algorithm relies on an evaluation function. This function assigns a numerical score to a position based on factors like material advantage, pawn structure, king safety, and piece mobility.

The player whose turn it is (the maximizer) selects the move that leads to the highest possible score. The algorithm then assumes the opponent (the minimizer) will respond by selecting the move that leads to the lowest possible score for the first player. The process is recursive, with scores being *backtracked* up the tree. The algorithm chooses the path that minimizes the maximum potential loss, assuming the opponent will play perfectly.

The Monte Carlo Tree Search

The Monte Carlo tree search (MCTS) is a heuristic search algorithm used for decision-making processes. It is used a lot in deciding the best move in a game. It tries to explore the most promising parts of a game tree by combining random sampling with a tree-based search. It consists of four main steps:

- **Selection:** The algorithm starts at the current game state and traverses the search tree by choosing the most promising child nodes. It balances exploitation (exploring nodes with a high win rate) with exploration (exploring nodes that have not been visited many times).
- **Expansion:** If a selected node is not a terminal game state, the algorithm expands the tree by adding one or more new child nodes representing possible next moves.
- **Simulation:** A simulated game is played out from the newly expanded node to a terminal state (win, lose, or draw) by making random moves. The outcome is recorded.
- **Back propagation:** The simulation result is used to update the statistics (win/loss ratio and number of visits) of all the nodes along the path from the new node back to the root of the tree.

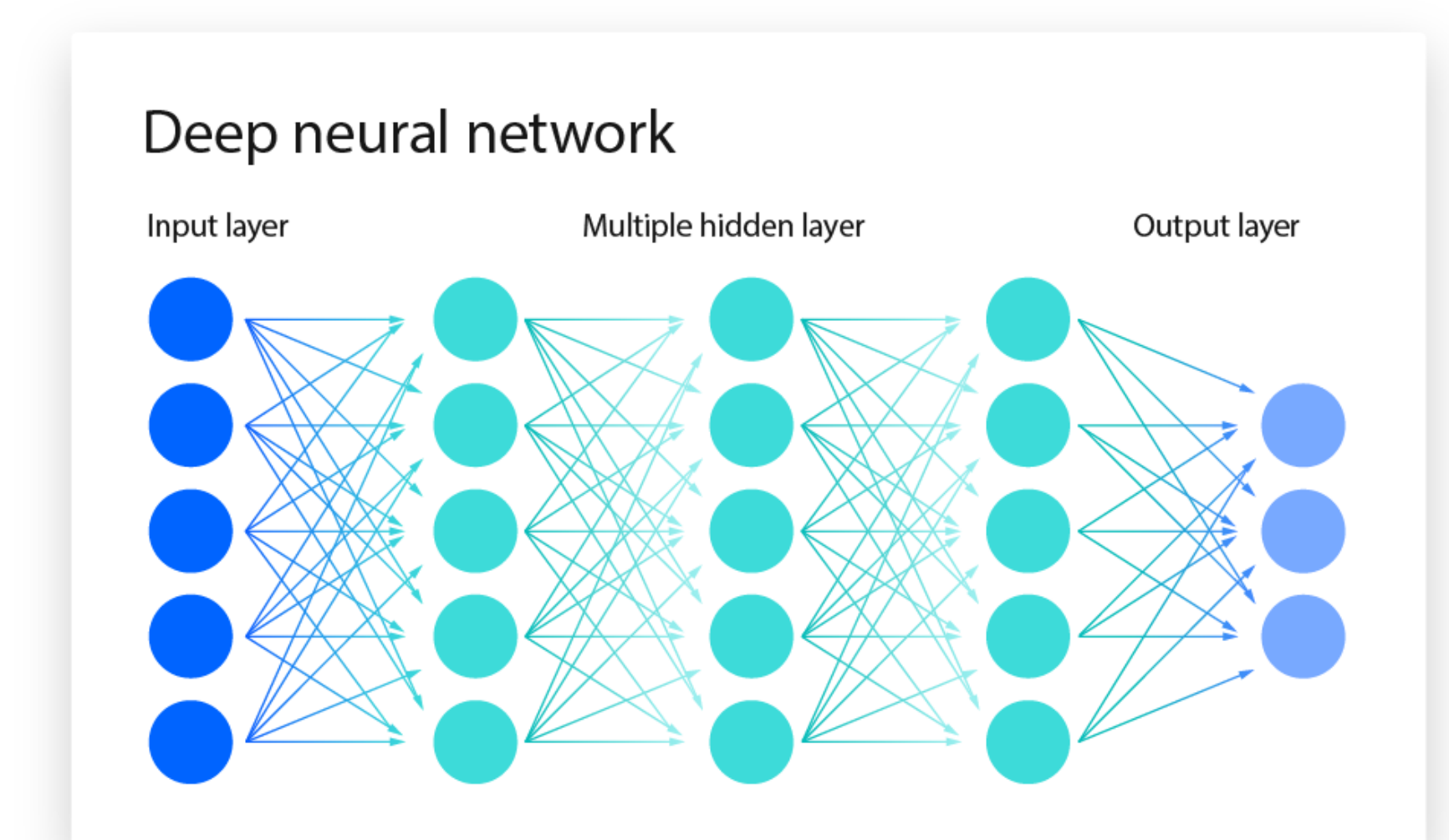


In the case of chess, instead of using alpha-beta pruning and heuristic evaluation functions, pure MCTS relies on random play-outs to evaluate positions, though modern engines like AlphaZero integrate it with neural networks for greater strength.

A hybrid Monte Carlo Minimax for chess combines the tactical precision of minimax search with the strategic, best-first approach of Monte Carlo Tree Search (MCTS) to create a more robust and efficient algorithm. This approach aims to leverage the strengths of both methods while mitigating their individual weaknesses in the game of chess.

Neural Networks

A neural network consists of connected units or nodes called neurons, which model the neurons in the brain. Artificial neuron models that mimic biological neurons more closely have also been recently investigated and shown to significantly improve performance. These are connected by edges, which model the synapses in the brain. Each artificial neuron receives signals from connected neurons, then processes them and sends a signal to other connected neurons. The *signal* is a real number, and the output of each neuron is computed by some non-linear function of the totality of its inputs, called the activation function. The strength of the signal at each connection is determined by a weight, which adjusts during the learning process.



Typically, neurons are organized into layers. Different layers may perform different transformations on their inputs. Signals travel from the first layer (the input layer) to the last layer (the output layer), possibly passing through multiple intermediate layers (hidden layers). A network is typically called a deep neural network if it has at least two hidden layers.

After implementing MCTS, creators of AlphaZero removed rollout() functionality altogether and replaced it with Current Neural Networks. This means instead of randomly picking out moves from leaf node, they actually used probability distribution of each child node to select the best node.

After each move, Monte Carlo tree was re-built and previous tree was thrown away. But with replacement of rollout function with Neural Network made decision process even better because this network is retained throughout the training process even if we don't have the previous tree.

Starting with only the basic rules of chess, AlphaZero played millions of games against itself, continuously learning and improving its strategies. This self-play training, based on reinforcement learning, allowed it to develop an understanding of the game uninfluenced by human knowledge or biases.

Neural network evaluation: Instead of evaluating millions of positions per second like Stockfish, AlphaZero used a deep neural network to make highly advanced and more efficient positional evaluations. It searched 80,000 positions per second compared to Stockfish's 70 million, but its network allowed it to focus on more promising moves.