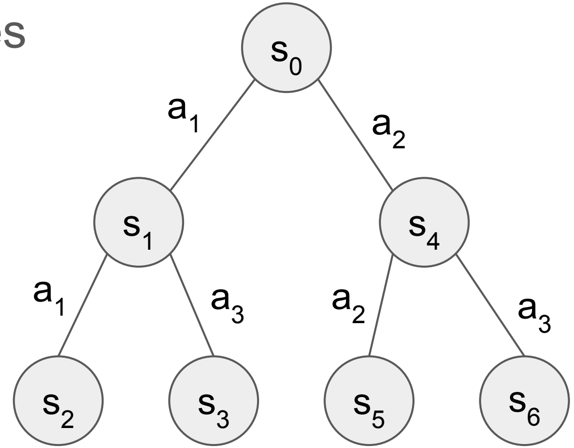
Monte Carlo tree search

and the AlphaZero algorithm

Roland Bernard

Introduction - Search

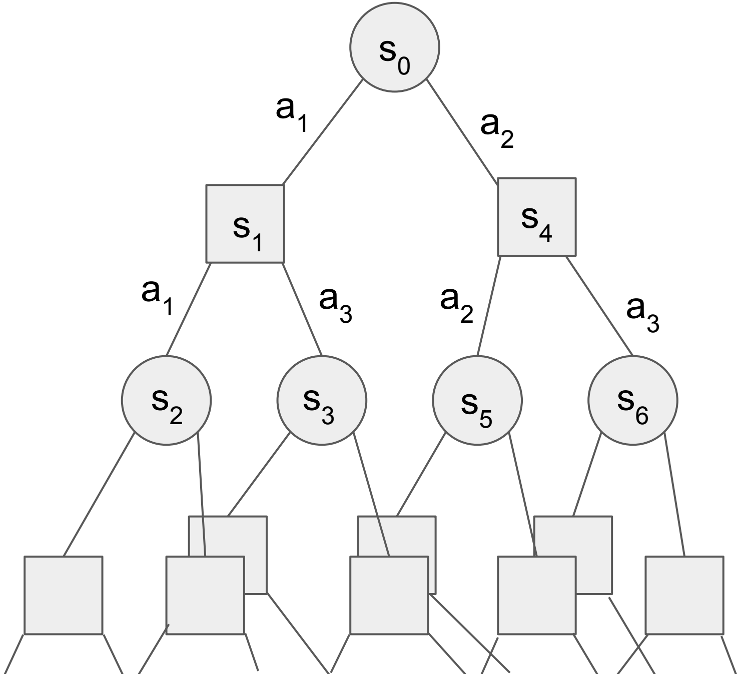
- Can be used to solve many problems
- Often the search space is too large
- Monte Carlo tree search algorithm is a heuristic search algorithm
- Allows making decisions by observing only a small part of the search space
- Often MCTS used in software playing board games
- So I will focus mostly on board games



Introduction - Board games

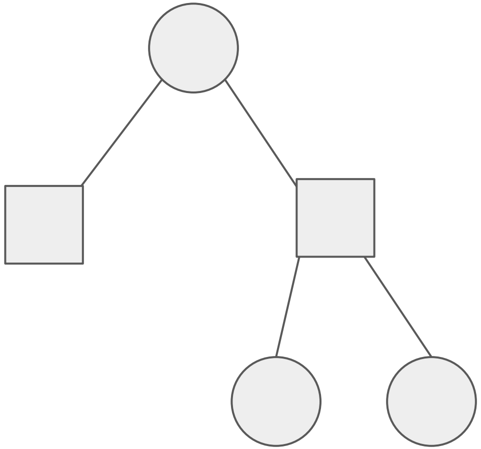
- Zero-sum games
- Deterministic transition function
- Alternating players
- E.g. Chess or Go

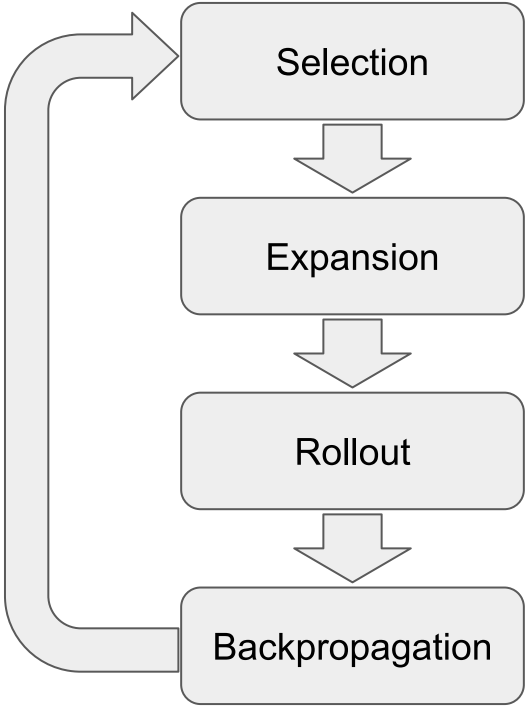
$$v^*(s) = \begin{cases} r(s), & \text{if } s \text{ is terminal} \\ \max_{a \in A_s} -v^*(f(s,a)), & \text{otherwise} \end{cases}$$



Pure MCTS - Overview

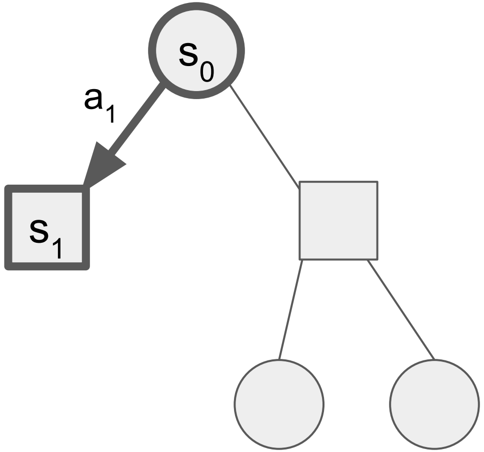
- Operates on a search tree
- Statistics for each edge: Q(s, a), N(s, a)

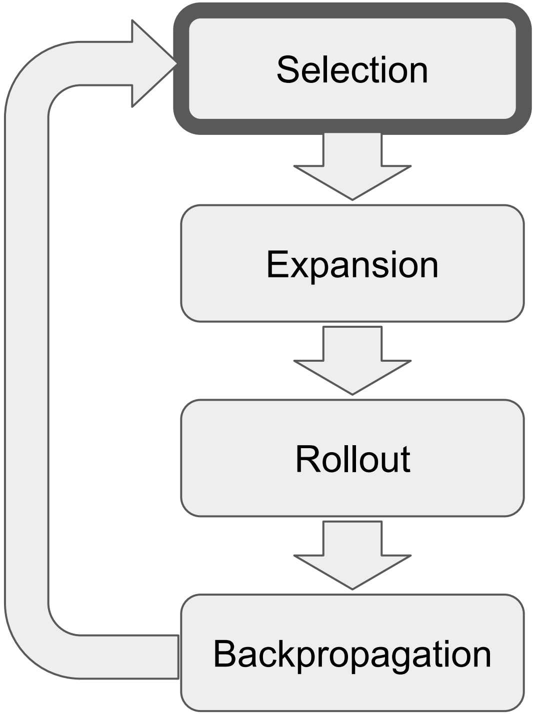




Pure MCTS - Overview

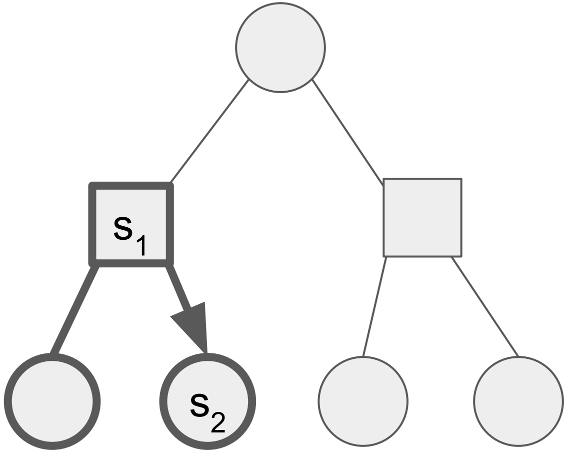
- Operates on a search tree
- Statistics for each edge: Q(s, a), N(s, a)

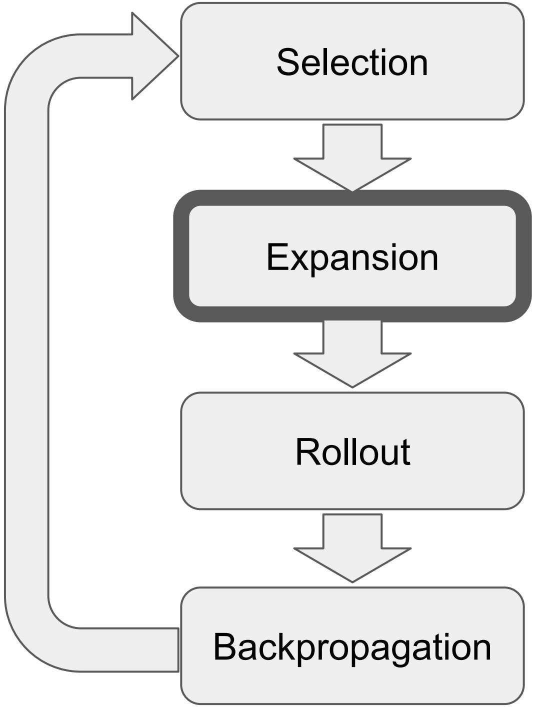




Pure MCTS - Overview

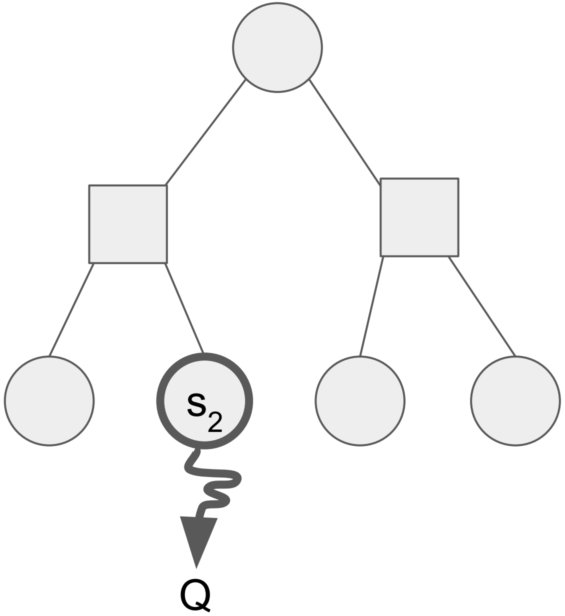
- Operates on a search tree
- Statistics for each edge: Q(s, a), N(s, a)

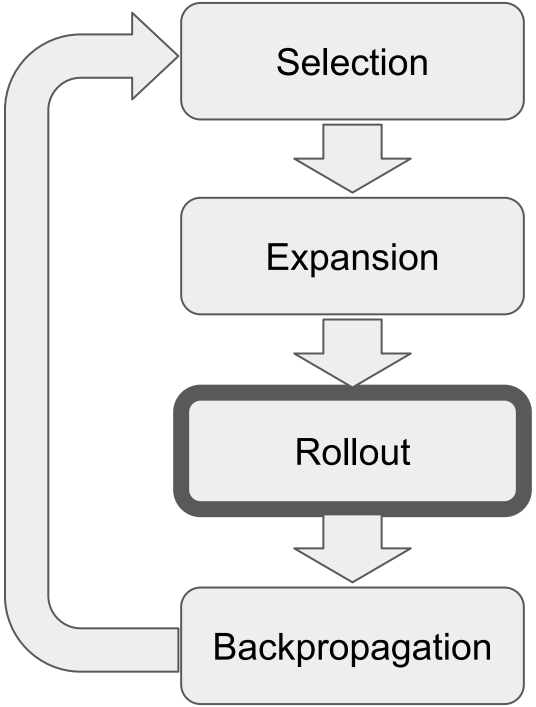




Pure MCTS - Overview

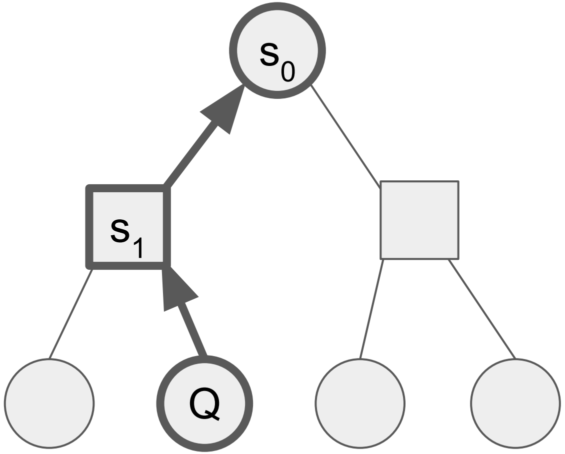
- Operates on a search tree
- Statistics for each edge: Q(s, a), N(s, a)

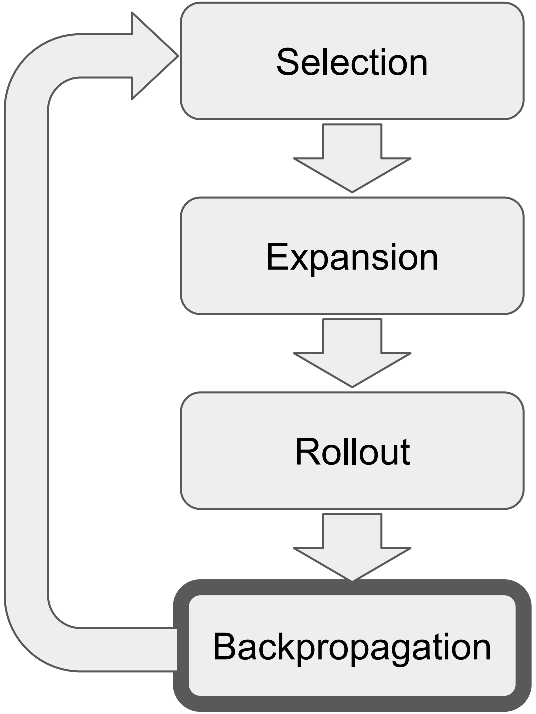




Pure MCTS - Overview

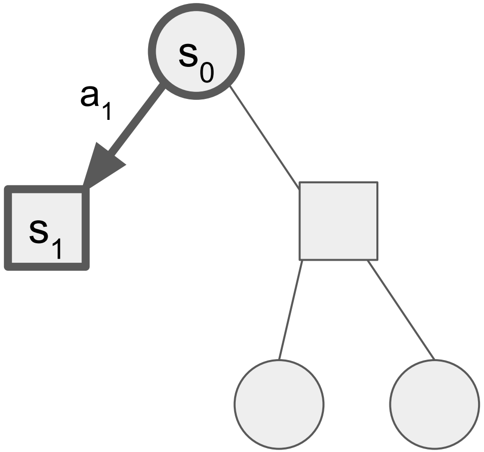
- Operates on a search tree
- Statistics for each edge: Q(s, a), N(s, a)





Pure MCTS - Selection

- Start at root node
- Select one child node until leaf node is reached
- Prefer more promising actions



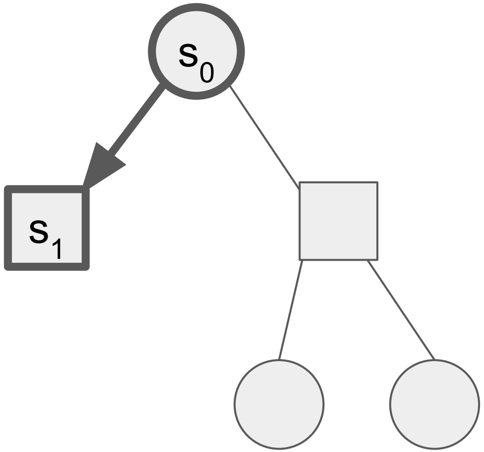
Pure MCTS - Selection - UCT

- Upper Confidence Bound 1 for Trees
- The constant controls the balances between the exploitation and exploration

$$a = \arg\max_{a \in A_s} \left(Q(s,a) + c_{\mathrm{utc}} \sqrt{\frac{\ln N(s)}{N(s,a)}} \right)$$

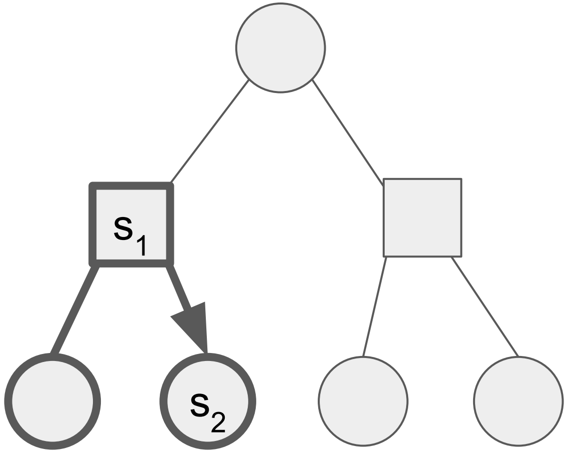
$$N(s) = \textstyle\sum_{a \in A_s} N(s,a)$$

$$c_{\mathrm{utc}} = \sqrt{2}$$



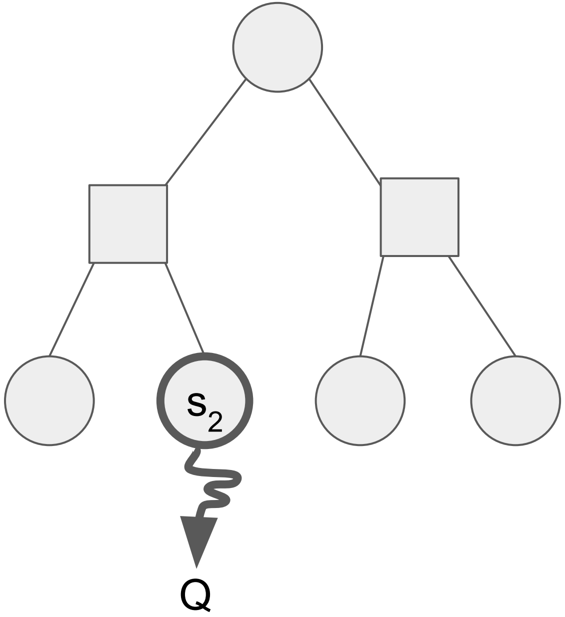
Pure MCTS - Node expansion

- Add child nodes for the legal actions
- After every round or after the number of rollouts
- Statistics for new edges are initialized to zero



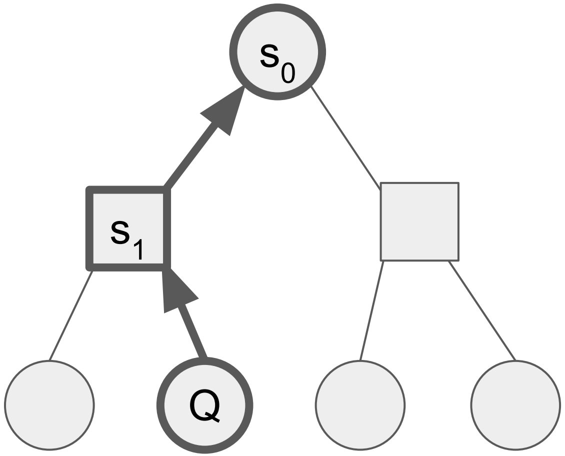
Pure MCTS - Rollout/Simulation

- Play until the end of the game
- Choose random actions
- Observe final value of the game



Pure MCTS - Backpropagation

- Update statistics for all edges on the search path
- Increase visit count N(s, a)
- Update expected value Q(s, a)



Pure MCTS - Finishing the search

- Stop after a fixed number of rounds, or until time runs out
- Make decision based on statistics from root node
- Maximum visit count or maximum estimated value
- Using visit counts is more stable

$$a = \arg\max_{a \in A_{s_0}} N(s_0, a)$$

MCTS - Advantages

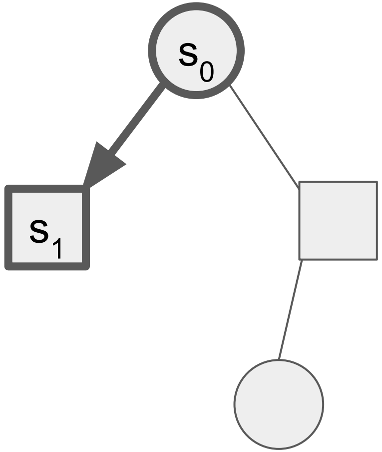
- Prunes large parts of the search space
- Useful for large search spaces
- Focusing on most promising actions
- Pure MCTS requires only the game mechanics

MCTS - Disadvantages

- Prunes large parts of the search space
- Might miss subtle strategy due to pruning
- Performance depends on the problem domain

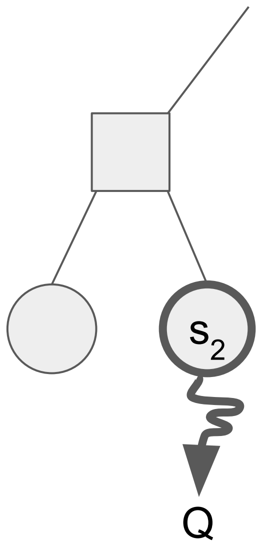
MCTS improvements - Selection with predictor

- Predict value of each actions to expand
- Use predicted value to influence selection
- MCTS can still correct if predictions are wrong
- Improves efficiency by focusing on better actions
- PUCT (Predictor + UCT) algorithms
- Predictor can use handcrafted heuristics or machine learning



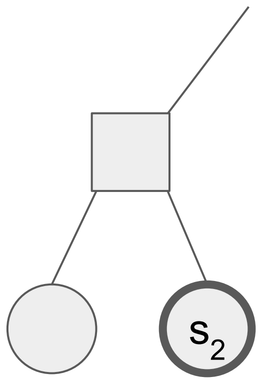
MCTS improvements - Stronger rollout policy

- MCTS performance depends on rollout results
- Better policy during rollout can lead to better overall estimates
- Rollout policy should be fast
- Can use handcrafted heuristics or machine learning



MCTS improvements - Evaluation function

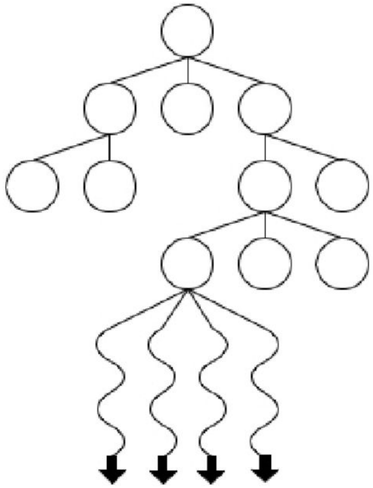
- Replace rollouts with an evaluation function
- Predicts the outcome of the game without rollout
- Can be more efficient if games are long
- Can use handcrafted heuristics or machine learning
- Can be combined with random playouts



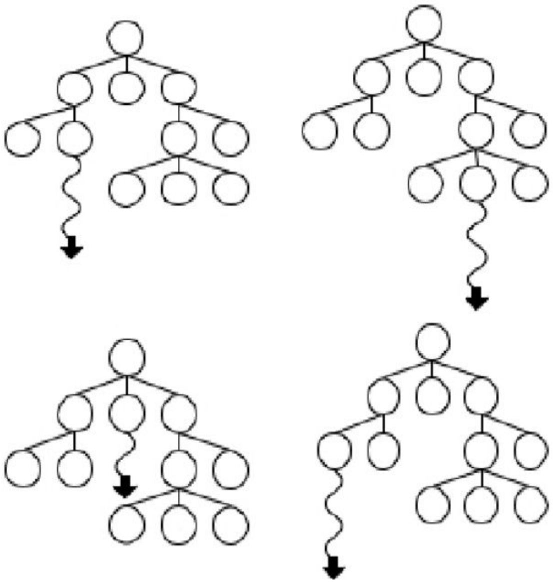
$Q = f(s_2)$

MCTS improvements - Concurrent execution

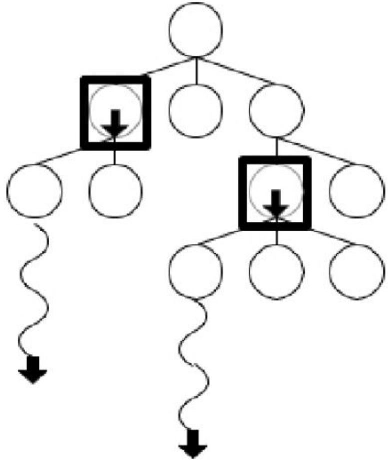
Leaf Parallelization



Root Parallelization

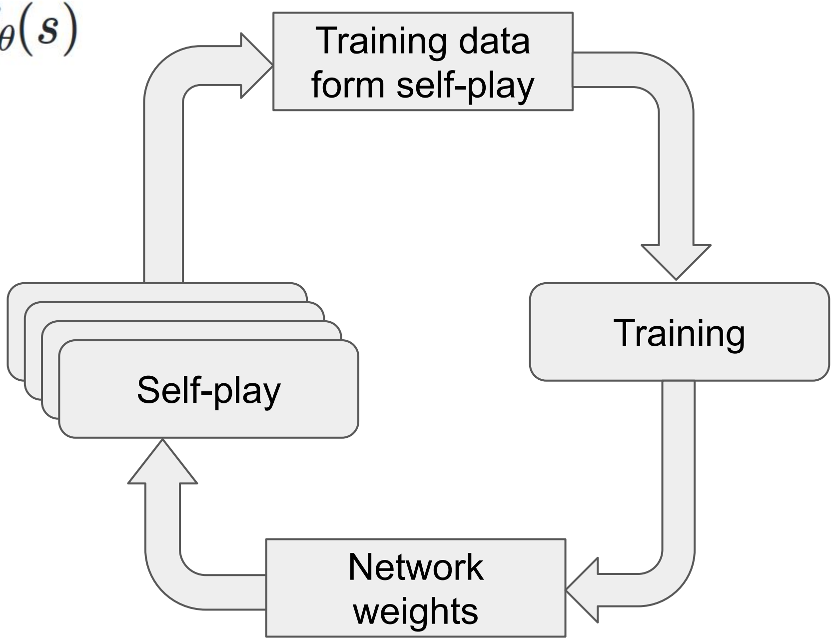


Tree Parallelization

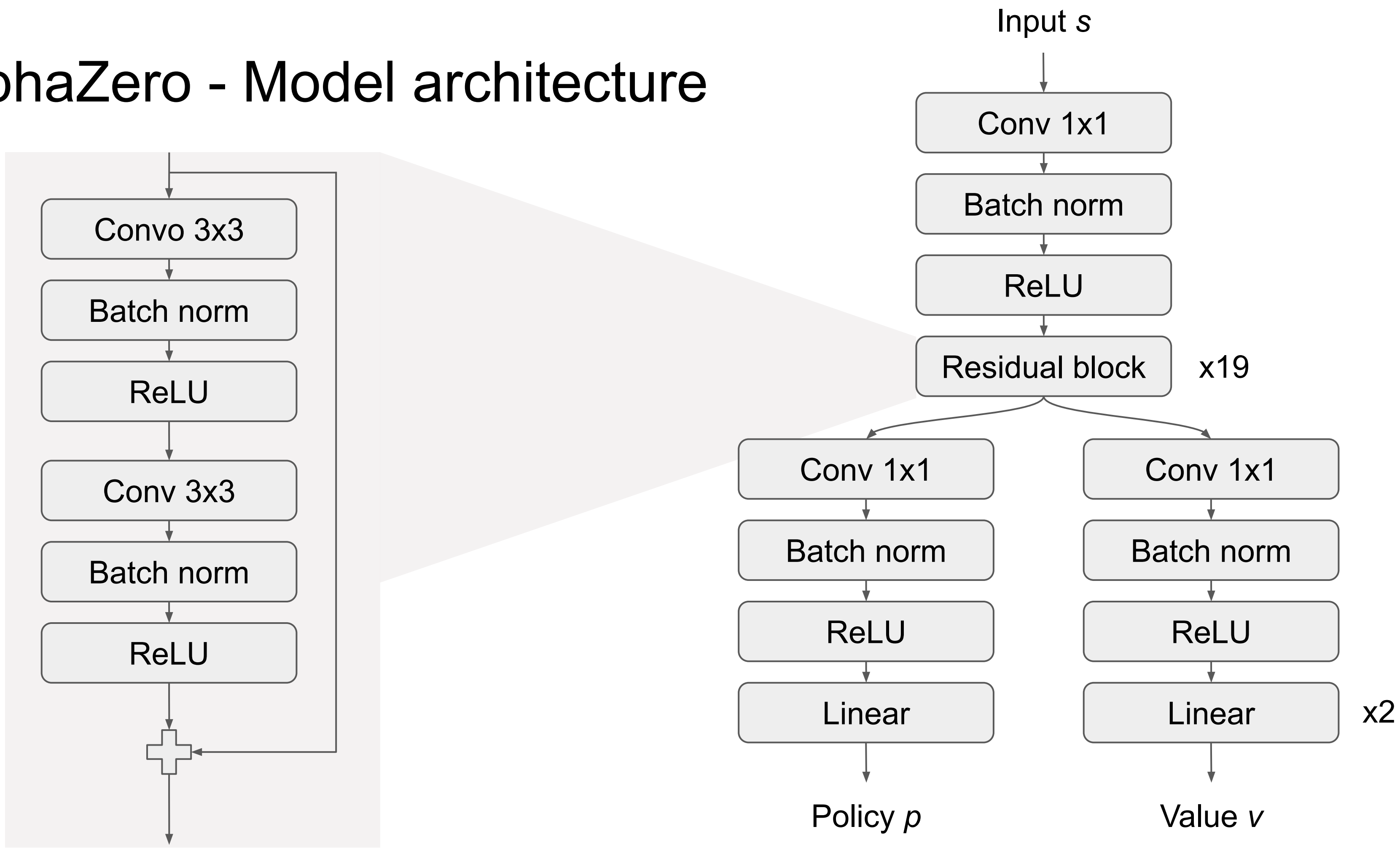


AlphaZero - Overview

- Reinforcement learning algorithm
- Uses a deep neural network $(p, v) = f_\theta(s)$
- Uses MCTS during play

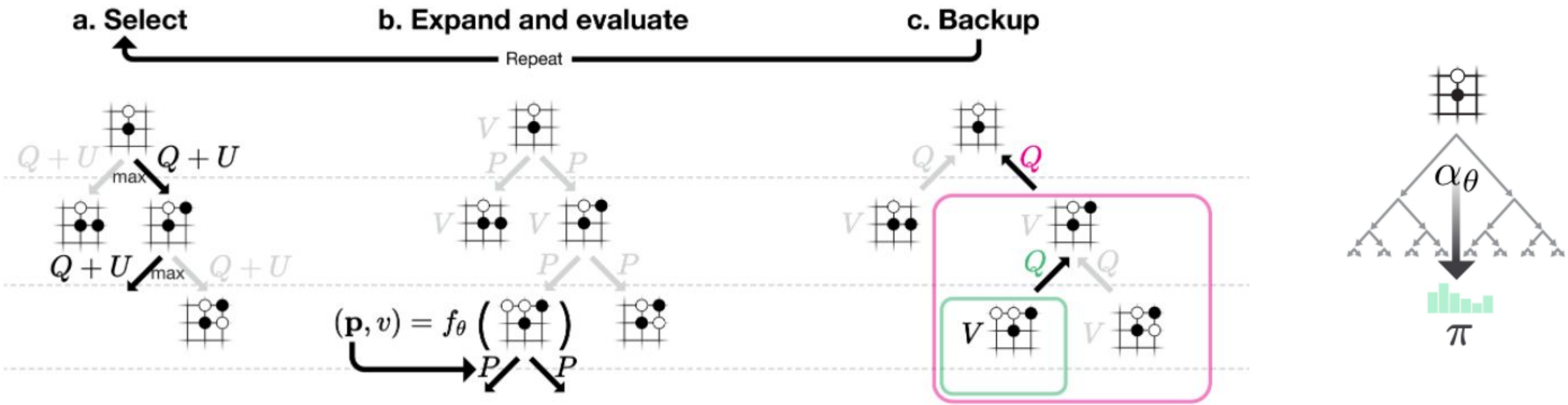


AlphaZero - Model architecture



AlphaZero - Search algorithm

- Keep for each edge: Q(s, a), N(s, a), P(s, a)
- Use policy output of network to set R(s, a) during expansion
- Use value output of network for backpropagation



AlphaZero - Search algorithm - PUCT

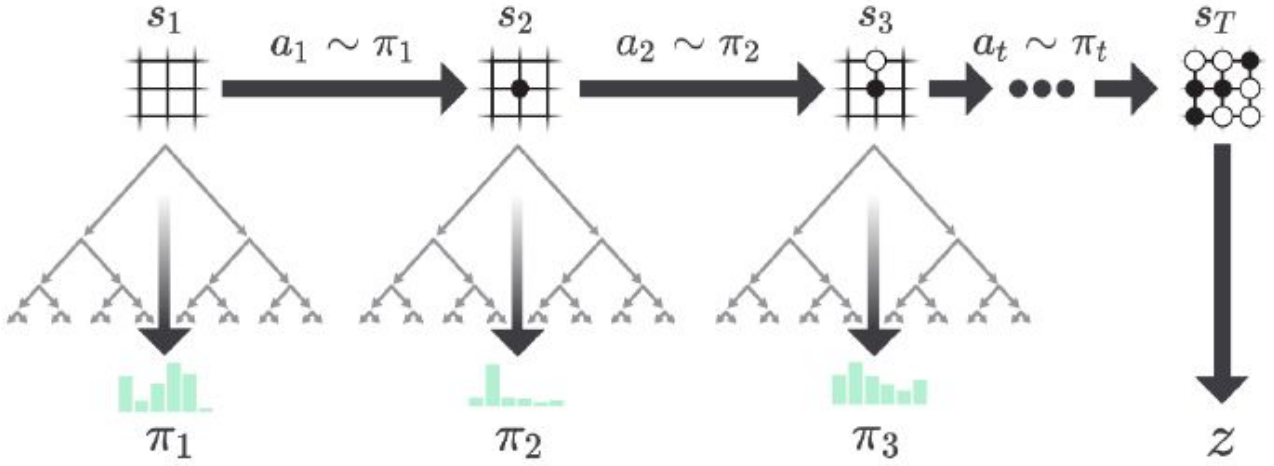
- Selection using a different formula
- Selection incorporates P(s, a) values
- Prefer exploring actions with high P(s, a)

$$a = \arg\max_{a \in A_s} \left(Q(s,a) + c_{\mathrm{putc}} P(s,a) \frac{\sqrt{N(s)}}{1 + N(s,a)} \right)$$

$$N(s) = \textstyle\sum_{a \in A_s} N(s,a)$$

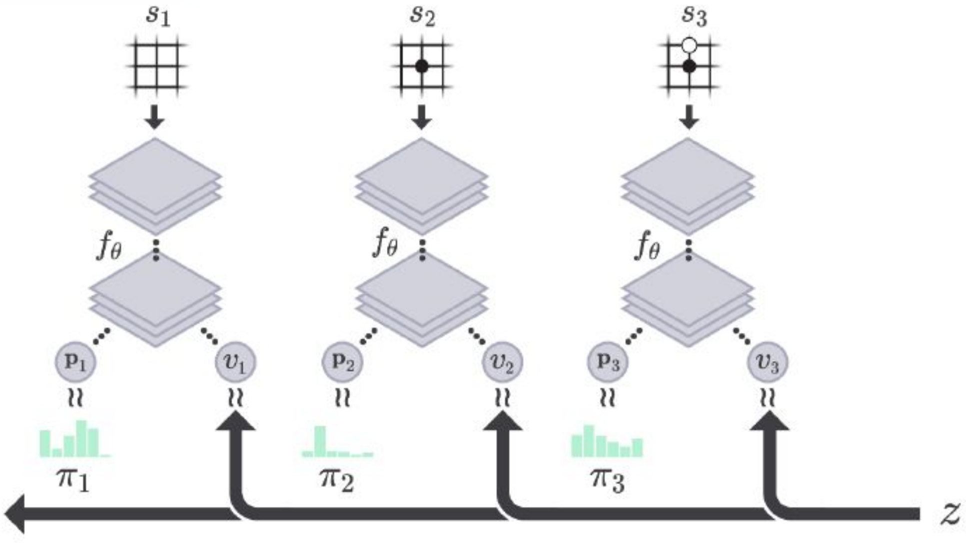
AlphaZero - Self-Play

- Games are played using the latest network parameters
- Each action is selected by using MCTS to obtain action probabilities *π*
- *π* is stronger than the raw network output *p*
- Game outcome *z* is determined
- *(s, π, z)* saved for each step in the game



AlphaZero - Training

- Uses data generated by the self-play process
- Sample from the most recent games *(s, π, z)*
- Train network parameter *Θ* so that *(p, v)* = f_{Θ}*(s)* matches *(π, z)* more closely
- Weights are saved every few steps to use them in self-play

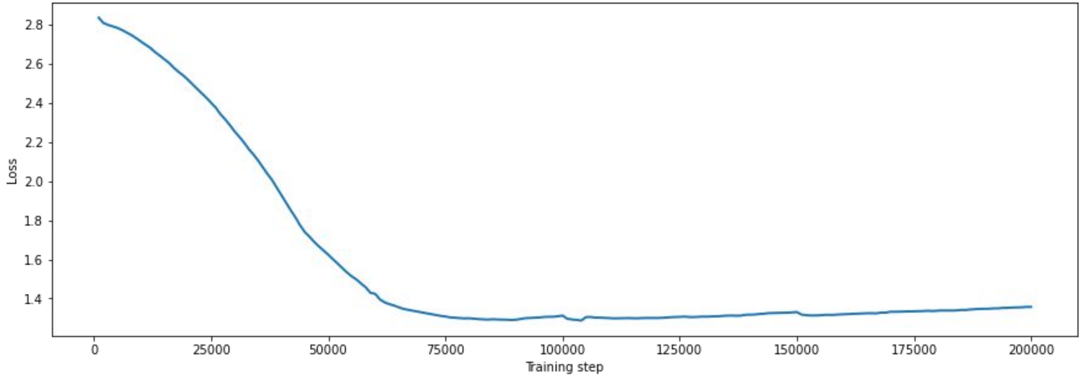


Demo - What I tested

I Implemented different algorithms for solving the Connect 4 game

- Simple Minimax
- Pure Monte Carlo tree search
- AlphaZero

I trained an AlphaZero deep neural network for 200000 iterations:



Demo - Results