Fraunhofer IIS

FRIEDRICH-ALEXANDER UNIVERSITÄT ERLANGEN-NÜRNBERG
TECHNISCHE FAKULTÄT

Online Planning with Discrete Actions

Christopher Mutschler

Outline

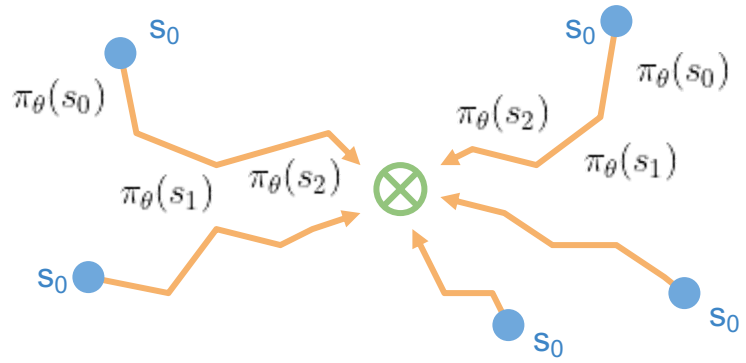
- Motivation: why model-based RL?
- What is a model? What are its inputs? What is a good model?
- **How can we use a model?**
  - Background Planning
    - Environment data augmentation / simulation
    - Sample efficient policy learning
  - **Online Planning**
    - Discrete Actions
    - Continuous Actions
  - Auxiliary tasks
- Real-world application

Online planning

- Background planning vs. Decision-Time planning

Background Planning

"Learn how to act for **any** situation"



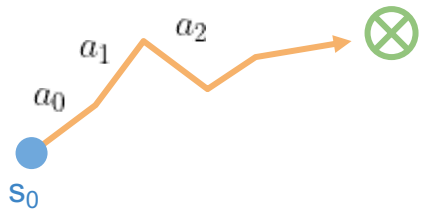
Optimization variables: shared parameters θ
(parameters of policy, or value, or …)

e.g.: $J(\theta) = \mathbb{E}_{s_0}[\sum_{t=0}^{H} \gamma^t r_t],\ a_t = \pi_\theta(s_t)$

Trained in expectation over all states

Decision-Time Planning

"Find best sequence of actions for my **current** situation"



Optimization variables: $a_0, a_1, \ldots, a_H$
(sequence of actions and/or states)

e.g.: $J(a_0, \ldots, a_H) = \sum_{t=0}^{H} \gamma^t r_t$

Optimizes only on a particular action sequence

https://sites.google.com/view/mbrl-tutorial
Sutton and Barto: Reinforcement Learning: An Introduction.

Online planning

- Background planning vs. Decision-Time planning

	Background Planning	Decision-Time Planning
Act on most recent state of the world	-	+
Act without any learning	-	+
Competent in unfamiliar situations	-	+
Independent of observation space	-	+
Partial observability	+	+ / -
Fast computation at deployment	+	-
Predictability and coherence	+	-
Same for discrete and continuous actions	+	-

Background

$$J(\theta) = \mathbb{E}_{s_0}[\sum_{t=0}^{H} \gamma^t r_t], \quad a_t = \pi_\theta(s_t)$$

joint angles? pixels? graphs?

Decision-Time

$$J(a_0, ..., a_H) = \sum_{t=0}^{H} \gamma^t r_t$$

https://sites.google.com/view/mbrl-tutorial

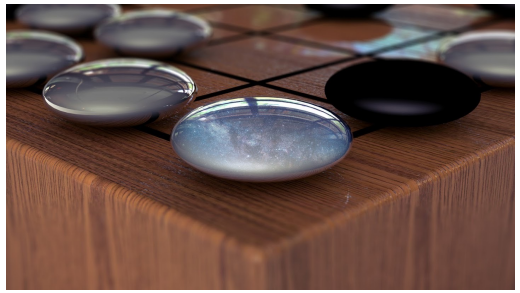
Outline

- Motivation: why model-based RL?
- What is a model? What are its inputs? What is a good model?
- **How can we use a model?**
  - Background Planning
    - Environment data augmentation / simulation
    - Sample efficient policy learning
  - **Online Planning**
    - **Discrete Actions**
    - Continuous Actions
  - Auxiliary tasks
- Real-world application

Discrete Actions

Environments with discrete actions

- Here we usually have problems with discrete actions and states (e.g., games, supply chains, production optimization, …)
- Their dimensionality is so large that it is impossible to enumerate the solutions

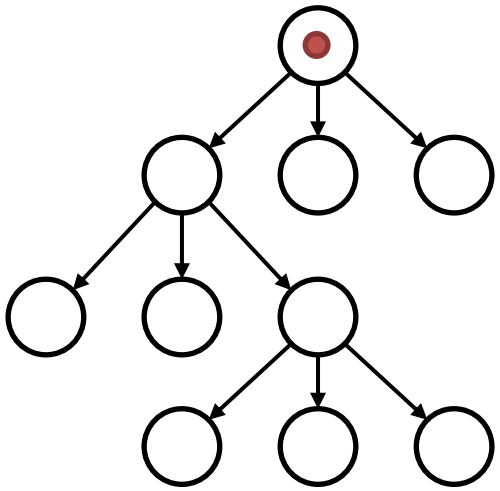


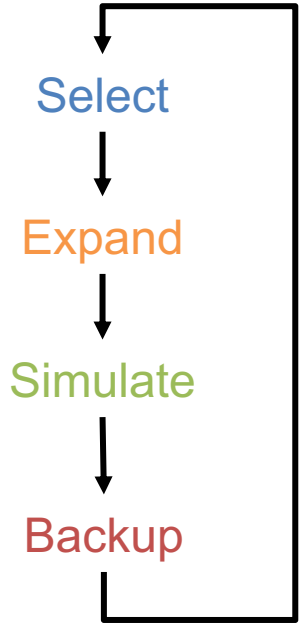
https://deepmind.com/research/case-studies/alphago-the-story-so-far

Discrete Actions

Monte Carlo Tree Search

- In the heart of modern online planning methods lies **Monte Carlo Tree Search (MTSC)**
- This is the work-horse behind Alpha Go and all its derivative work



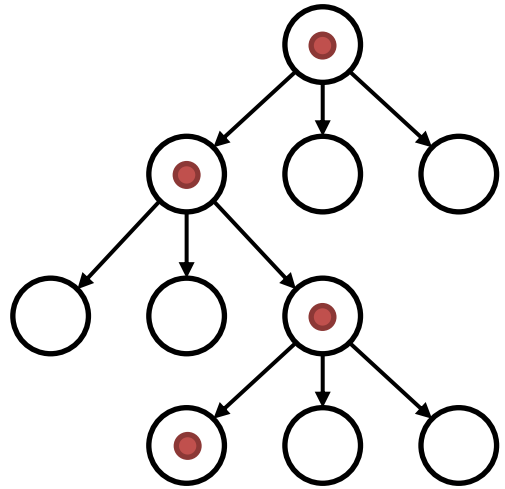


Discrete Actions: Monte Carlo Tree Search

1. Selection
   - How do we do selection in an MDP?
     → We follow a policy π_s
   - In nodes we have seen before, select action according to UCB (Bandits):

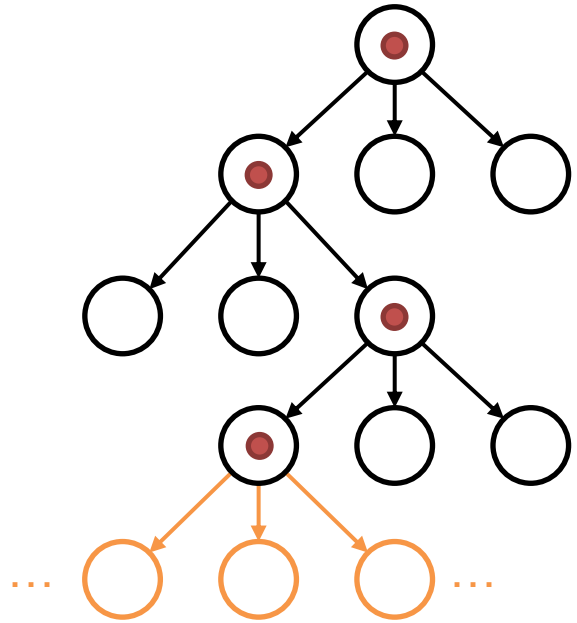
$$a \sim V_i + C\sqrt{\frac{\ln(N)}{n_i}}$$

     - V_i is value estimate at node i
     - C is tunable exploration parameter
     - N is number of visits of parent node
     - n_i is number of visits at node i
   - We do this until we reach a leaf node (where we don't know what to do next)



Discrete Actions: Monte Carlo Tree Search

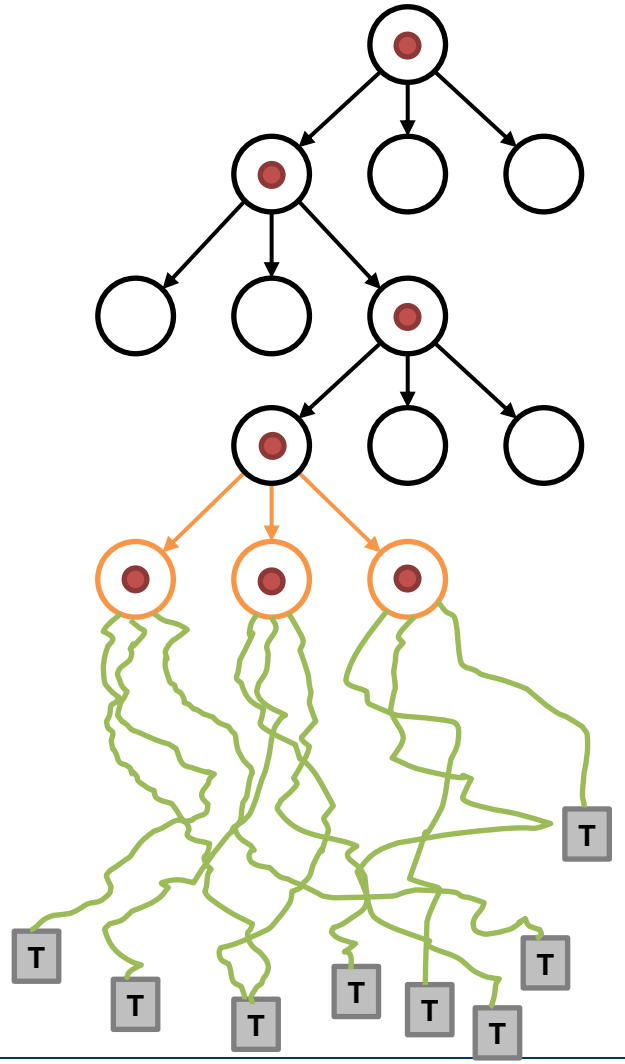
1. Selection
2. Expansion
   - We know the (transition) model so we know where actions lets us end up in (or we do a bunch of one-step simulations to know successor states)



Discrete Actions: Monte Carlo Tree Search

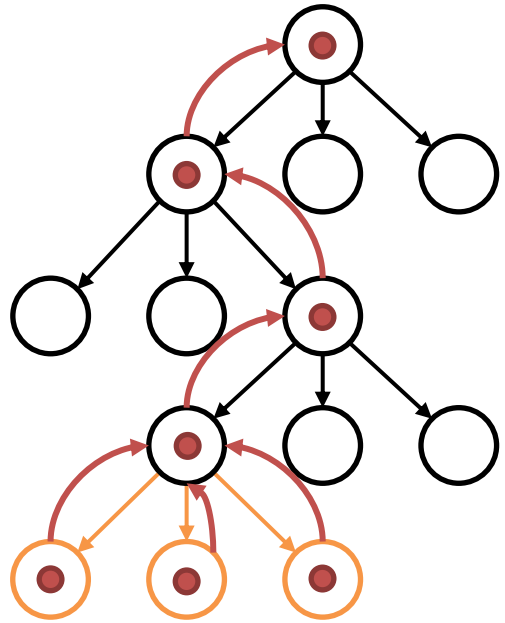
1. Selection
2. Expansion
3. Simulation
   - Randomly **choose** a new child node
   - Play **randomly** (i.e., with a random rollout policy π_r) until game finishes

Doing all these Monte-Carlo rollouts gives us $\hat{Q}(s, a)$



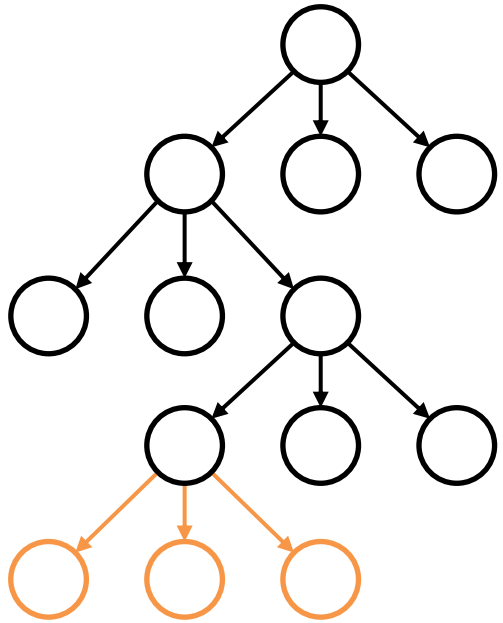
Discrete Actions: Monte Carlo Tree Search

1. Selection
2. Expansion
3. Simulation
4. Backup
   - We now have estimates of $\hat{Q}(s, a)$ of the leaf nodes
   - Backup Q-values and number of visits all the way up to the root of the tree



Discrete Actions: Monte Carlo Tree Search

1. Selection
2. Expansion
3. Simulation
4. Backup
   - We now have estimates of $\hat{Q}(s, a)$ of the leaf nodes
   - Backup Q-values and number of visits all the way up to the root of the tree
   - In fact, we now know a lot more about the tree!



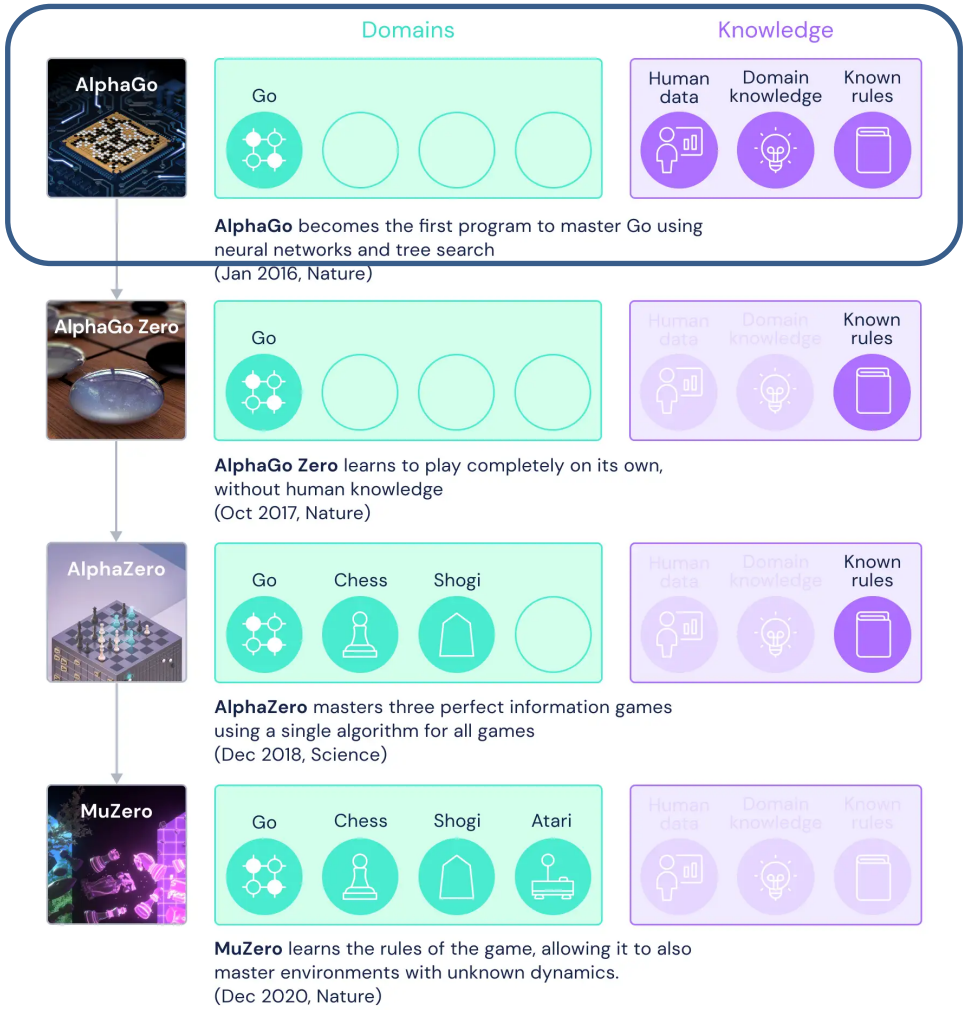
We so far used two policies:

- A selection policy π_s (that uses Q-values where we already have knowledge about)
- A random simulation policy π_r (for leaves onwards where we don't know anything)
- We can update our selection policy π_s now!
- π_s is getting deeper throughout the iterations of the algorithm and more accurate with backing up the information of leave nodes over time

Discrete Actions: Monte Carlo Tree Search

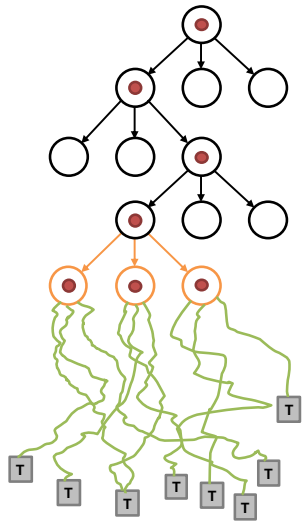
- Monte Carlo Tree Search is a planning algorithm, and we either
  - need a transition model, or we
  - need a way to do the simulation
- **But should we stop?**
  - In principle, we could run MCTS forever
  - There is a bunch of ways how to integrate planning and executing:
    1. Run MCTS until convergence starting from current real-world state
    2. take best action in real world
    3. go back to 1.
  - We could specify a maximal computation time per MCTS run
  - or: we solve the whole MDP and then run the tree greedy in real-world…
- **One more thing about the rollout policy π_r:**
  - π_s is the Q-value of being greedy in knowledgeable states under the assumption that we behave randomly in simulation (we underestimate Q)
  - Encode goal/success conditions and fail-by-termination (i.e., constraints)
    Example from Pac-Man: eating dots vs. avoiding ghosts

MCTS in practice: AlphaGo & Derivatives



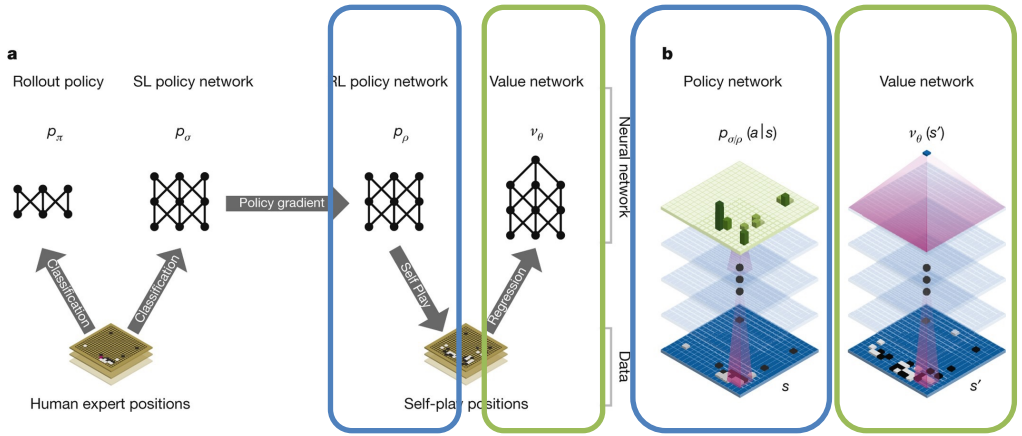
https://deepmind.com/blog/article/muzero-mastering-go-chess-shogi-and-atari-without-rules

MCTS in practice: AlphaGo



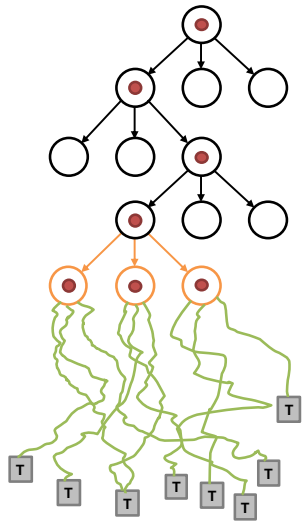
AlphaGo

- Large breadth and depth ($b \approx 250, d \approx 150$)
- Main idea: enhance MCTS with a learning component
- RL Policy network: selects actions
- RL Value network: predicts win/lose for each position



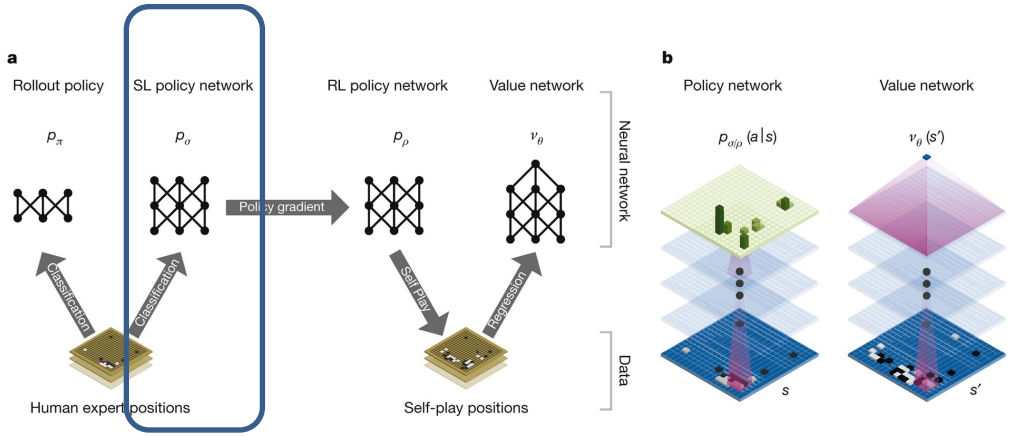
Silver, David, et al. "Mastering the game of Go with deep neural networks and tree search." nature 529.7587 (2016): 484-489.

MCTS in practice: AlphaGo



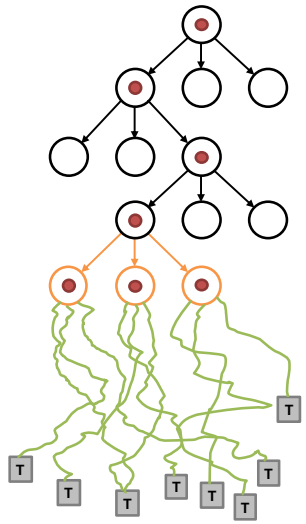
AlphaGo

- SL Policy network p_σ
  - Goal: predict good actions in states
  - Trained with randomly sampled state-action pairs from human experts (30M positions from KGS Go Server)
  - 13-layer neural networks with conv-layers and ReLUs
  - Accuracy of 57.0%



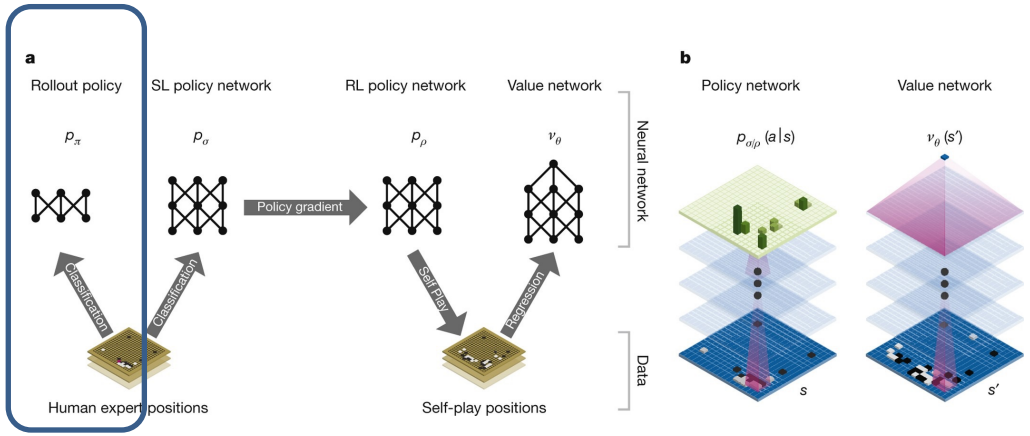
Silver, David, et al. "Mastering the game of Go with deep neural networks and tree search." nature 529.7587 (2016): 484-489.

MCTS in practice: AlphaGo



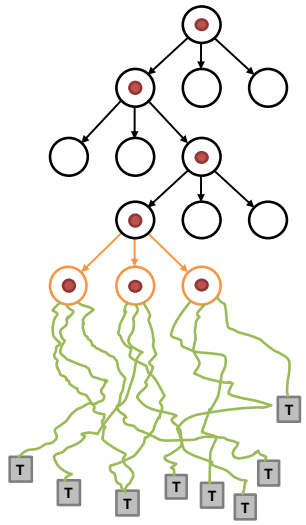
AlphaGo

- Rollout policy network p_π
  - same initialization and training but faster and less accurate
  - network only uses linear softmax of small (hand-crafted) input features
  - Accuracy of 24.2%
  - Forward pass only takes $2\mu s$ (instead of 3ms for p_σ)



Silver, David, et al. "Mastering the game of Go with deep neural networks and tree search." nature 529.7587 (2016): 484-489.

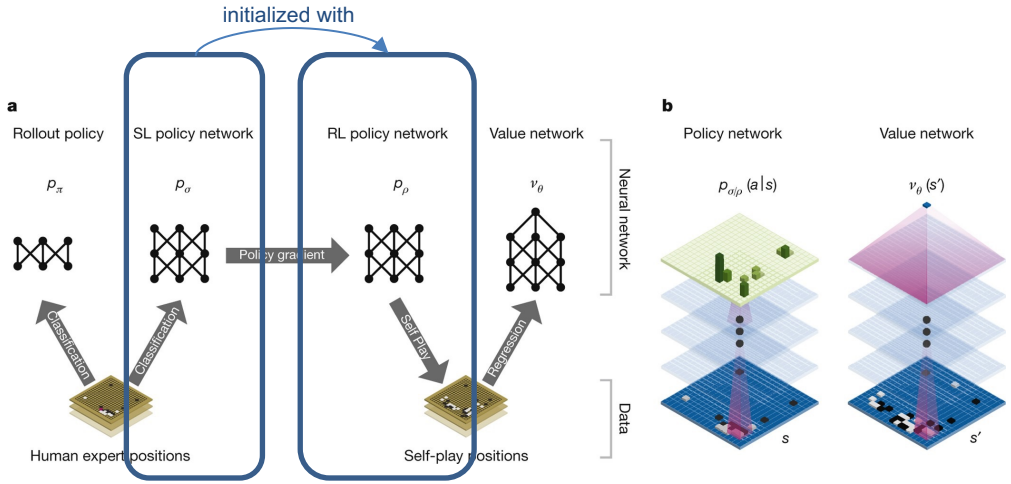
MCTS in practice: AlphaGo



AlphaGo

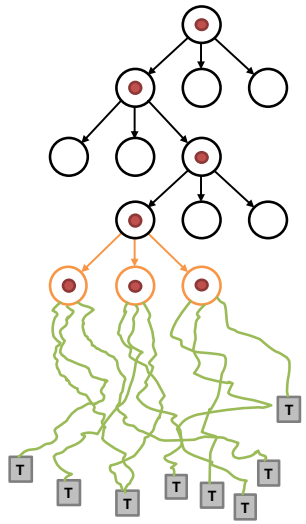
- Learning policy network p_ρ
  - Play self-games with p_ρ and a random ancestor of p_ρ
  - Reward is 0 everywhere and $z = \pm 1$ at the end depending on outcome
  - Policy network is improved with policy gradients
  - This already yields a very strong player: 80% winning probability against p_σ, 85% against Piachi[1] (compared to 11% of CNN SotAs)

[1] *as of 2016 the strongest open-source Go program a sophisticated Monte Carlo search program, ranked at 2 amateur dan on KGS*



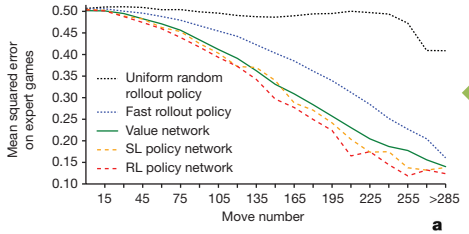
Silver, David, et al. "Mastering the game of Go with deep neural networks and tree search." nature 529.7587 (2016): 484-489.

MCTS in practice: AlphaGo

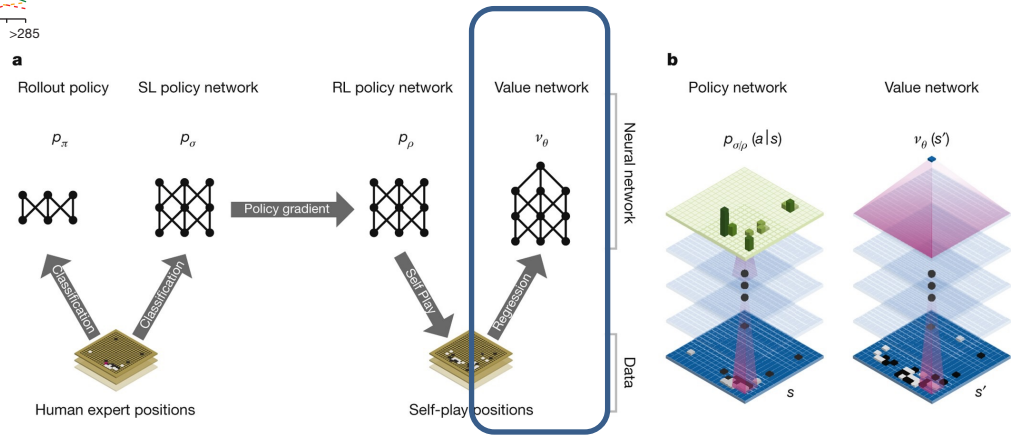


AlphaGo

- Learning value network v_θ
  - Used to approximate the value function under p_ρ
  - Neural network architecture like that of p_ρ but outputs single value
  - Use state-outcome pairs (s, z) and train using SGD and MSE

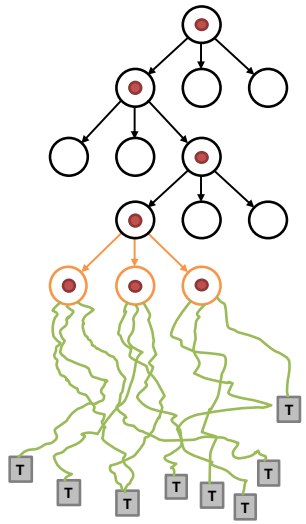


Accuracy of the position evaluation of value function vs. MC rollouts



Silver, David, et al. "Mastering the game of Go with deep neural networks and tree search." nature 529.7587 (2016): 484-489.

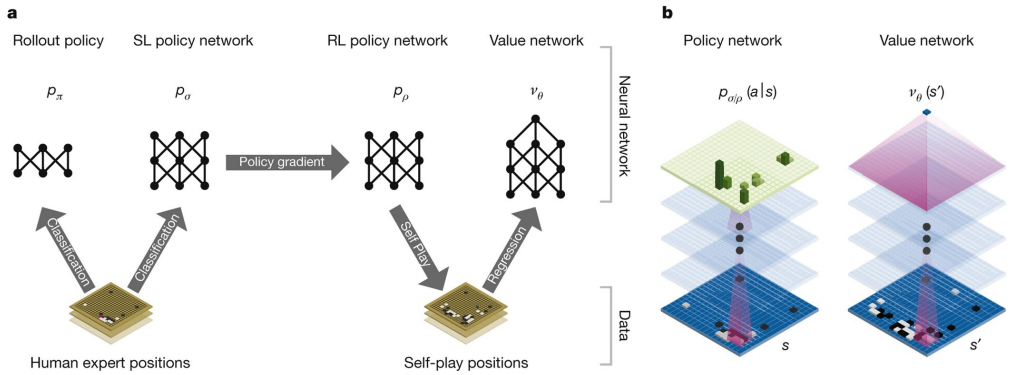
MCTS in practice: AlphaGo



AlphaGo: use MCTS as a work-horse

1. Selection
   - In nodes we have visited before, we select which action to take based on the RL policy network action probabilities and the Q-approximation in the tree nodes, in an UCB-like process[1]

[1] *UCB = upper confidence bounds, a theoretical framework to tackle exploration (see lecture on exploration strategies). We will not further go into details here. Important here is only the fact that we not simply take the argmax-action but we add an exploration bonus on top that we calculate through a state-visitation counter.*

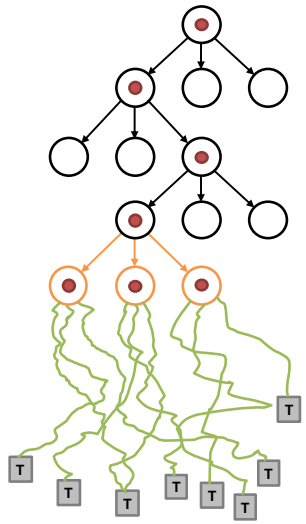


Silver, David, et al. "Mastering the game of Go with deep neural networks and tree search." nature 529.7587 (2016): 484-489.

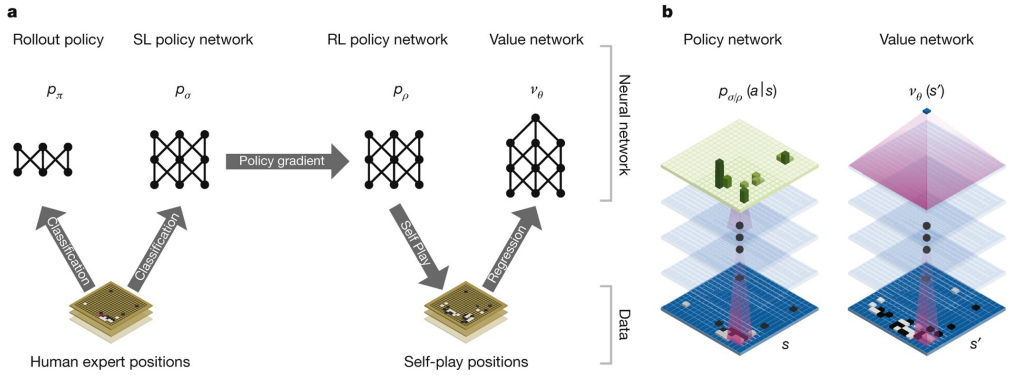
MCTS in practice: AlphaGo

AlphaGo: use MCTS as a work-horse

1. Selection
2. Expansion
   - When we reach a leaf node, the SL Policy network is used to assign a prior action probability. The tree is expanded according to the most probable action of the RL Policy network



It is worth noting that the SL policy network p_σ performed better in AlphaGo than the stronger RL policy network p_ρ, presumably because humans select a diverse beam of promising moves, whereas RL optimizes for the single best move. However, the value function

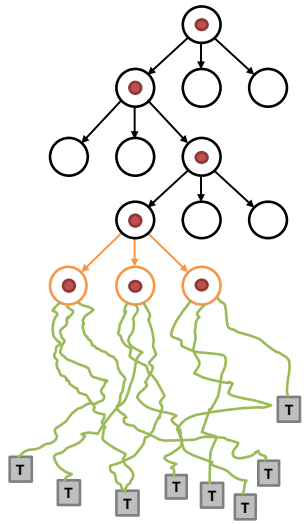


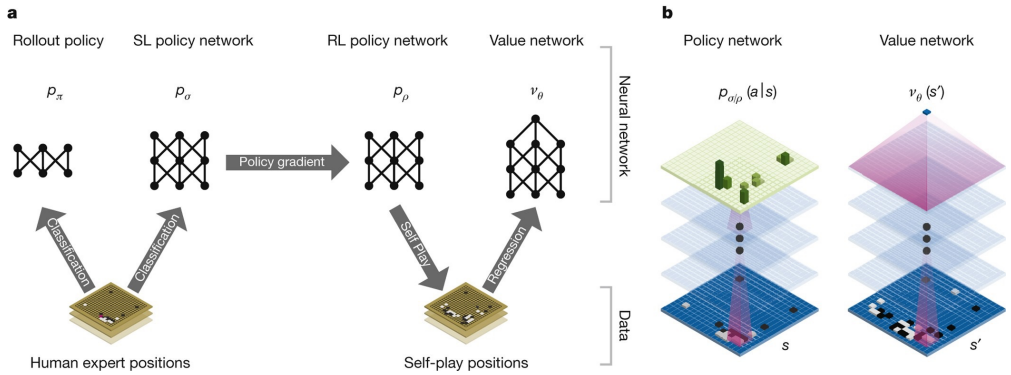
Silver, David, et al. "Mastering the game of Go with deep neural networks and tree search." nature 529.7587 (2016): 484-489.

MCTS in practice: AlphaGo

AlphaGo: use MCTS as a work-horse

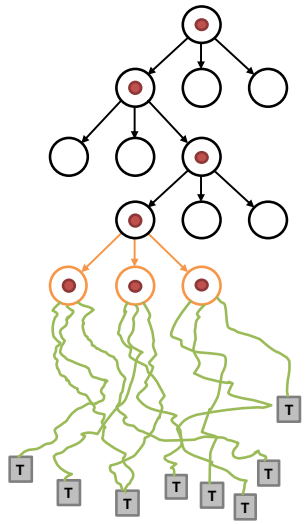
1. Selection
2. Expansion
3. Simulation
   - After the expansion, multiple simulations in parallel are performed
   - The “fast” rollout policy is used to play the game until it finishes.





Silver, David, et al. "Mastering the game of Go with deep neural networks and tree search." nature 529.7587 (2016): 484-489.

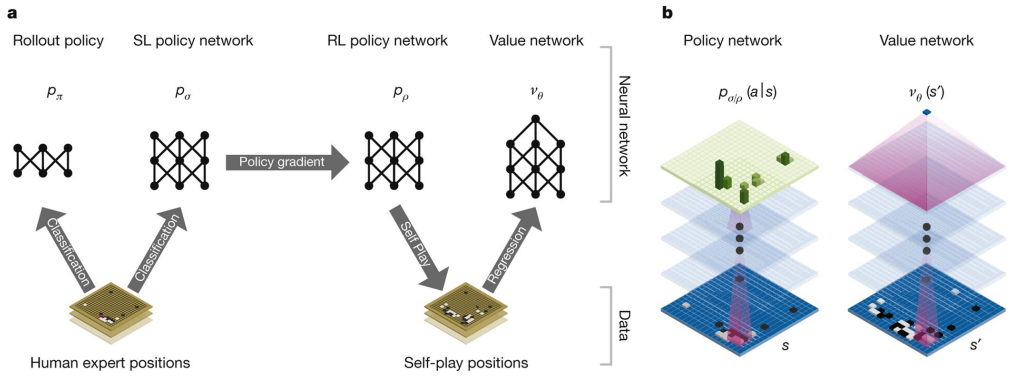
MCTS in practice: AlphaGo



AlphaGo: use MCTS as a work-horse

4. RL Networks training:
   - The value of the previous leaf node that was expanded is updated combining the RL value network and the final average reward from the rollouts
   - The Q values and visitations of the tree nodes are updated as in the classical MCTS algorithm.

   - **All games are played against a previous RL Policy as opponent (self-play)**



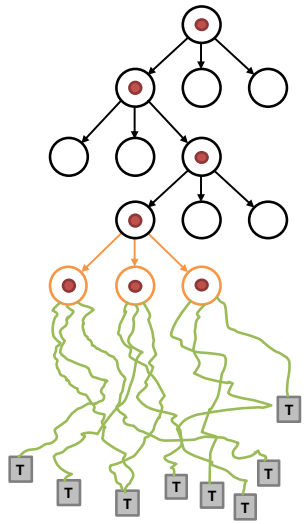
Silver, David, et al. "Mastering the game of Go with deep neural networks and tree search." nature 529.7587 (2016): 484-489.

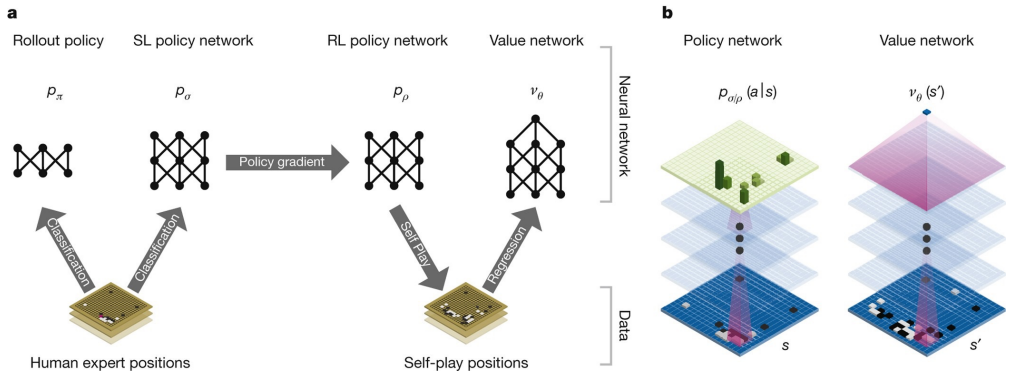
MCTS in practice: AlphaGo

AlphaGo: use MCTS as a work-horse

Deployment:

- Policy: Online MCTS!!!





Silver, David, et al. "Mastering the game of Go with deep neural networks and tree search." nature 529.7587 (2016): 484-489.

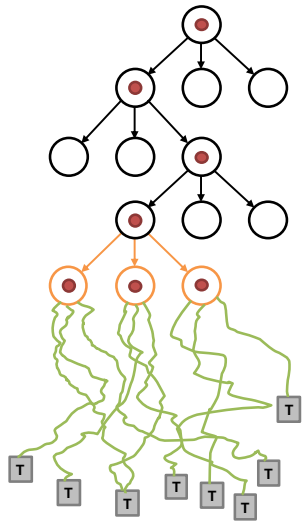
MCTS in practice: AlphaGo & Derivatives

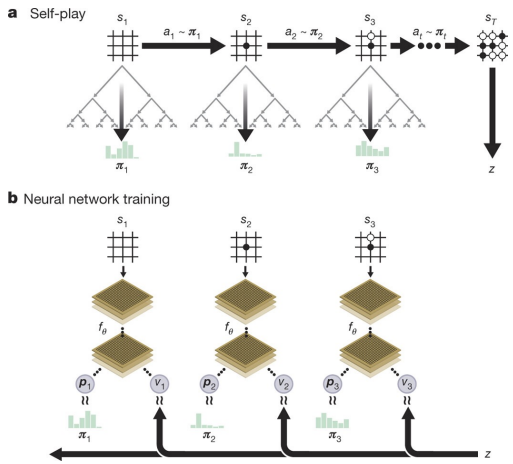
https://deepmind.com/blog/article/muzero-mastering-go-chess-shogi-and-atari-without-rules

MCTS in practice: AlphaGo Zero

AlphaGo Zero

- Main differences to AlphaGo:
  1. No supervised initialization from human games
  2. Only black & white stones (no hand-crafted features)
  3. Joint RL Policy and Value network
  4. No simulation rollouts during deployment – rely only on networks
- Self-play against strongest version of the Policy/Value network
- Use MCTS also in training:
  - Train the RL Policy head of the network to mimic the powerful MCTS policy during training
  - Train the RL Value head of the network to predict the outcome of the game (as in Alpha Go)
  - Think of it as trying to distill the power of MCTS search during training to the Policy/Value network





Silver, David, et al. "Mastering the game of go without human knowledge." nature 550.7676 (2017): 354-359.

ALPHAGO ZERO CHEAT SHEET

The training pipeline for AlphaGo Zero consists of three stages, executed in parallel

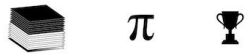
SELF PLAY

Create a 'training set'

The best current player plays 25,000 games against itself

See MCTS section to understand how AlphaGo Zero selects each move

At each move, the following information is stored



The game state (see 'What is a Game State' section)

The search probabilities (From the MCTS)

The winner (+1 if this player won, -1 if this player lost - added once the game has finished)

RETRAIN NETWORK

Optimise the network weights

A TRAINING LOOP

Sample a mini-batch of 2048 positions from the last 500,000 games

Retrain the current neural network on these positions

- The game states are the input (see 'Deep Neural Network Architecture')

Loss Function

Compares predictions from the neural network with the search probabilities and actual winner

PREDICTIONS: p, v — Cross-entropy + Mean-squared error + Regularisation — π, ACTUAL

After every 1,000 training loops, evaluate the network

EVALUATE NETWORK

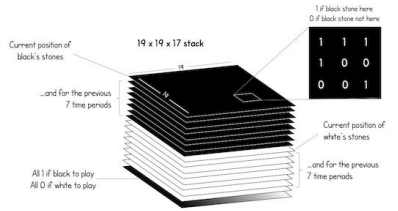
Test to see if the new network is stronger

Play 400 games between the latest neural network and the current best neural network

Both players use MCTS to select their moves, with their respective neural networks to evaluate leaf nodes

Latest player must win 55% of games to be declared the new best player

WHAT IS A 'GAME STATE'



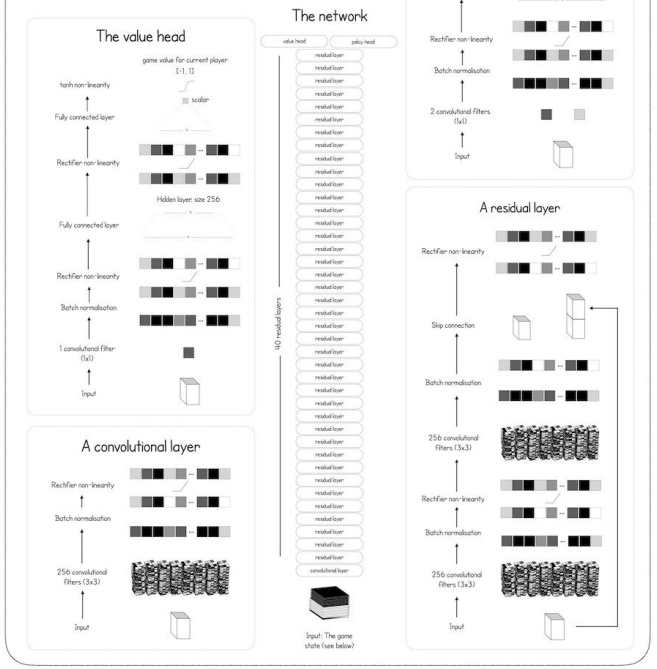
This stack is the input to the deep neural network

THE DEEP NEURAL NETWORK ARCHITECTURE

How AlphaGo Zero assesses new positions

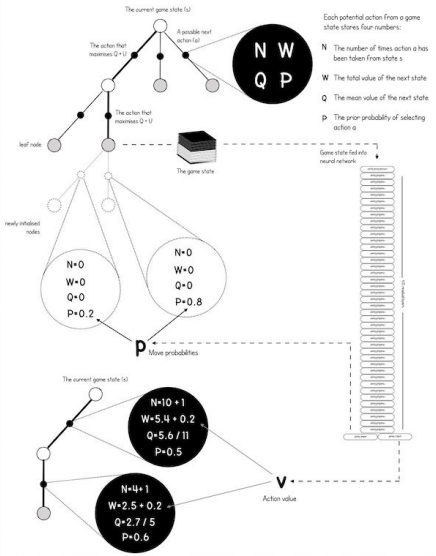
The network learns 'tabula rasa' (from a blank slate)

At no point is the network trained using human knowledge or expert moves



MONTE CARLO TREE SEARCH (MCTS)

How AlphaGo Zero chooses its next move



First, run the following simulation 1,600 times...

Start at the root node of the tree (the current game state)

1. Choose the action that maximises...

$Q + U$

- Q: The mean value of the next state
- U: A function of P and N that increases if an action hasn't been explored much, relative to the other actions, or if the prior probability of the action is high

Early on in the simulation, U dominates (more exploration), but later, Q is more important (less exploration)

2. Continue until a leaf node is reached

The game state of the leaf node is passed into the neural network, which outputs predictions about two things:

- **p**: Move probabilities
- **v**: Value of the state (for the current player)

The move probabilities p are attached to the new feasible actions from the leaf node

3. Backup previous edges

Each edge that was traversed to get to the leaf node is updated as follows:

$N \rightarrow N + 1$

$W \rightarrow W + v$

$Q = W / N$

...then select a move

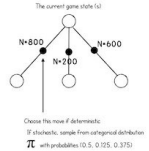
After 1,600 simulations, the move can either be chosen:

Deterministically (for competitive play)
Choose the action from the current state with greatest N

Stochastically (for exploratory play)
Choose the action from the current state from the distribution

$\pi \sim N^{1/\tau}$

where τ is a temperature parameter controlling exploration



Other points

- The sub-tree from the chosen move is retained for calculating subsequent moves
- The rest of the tree is discarded

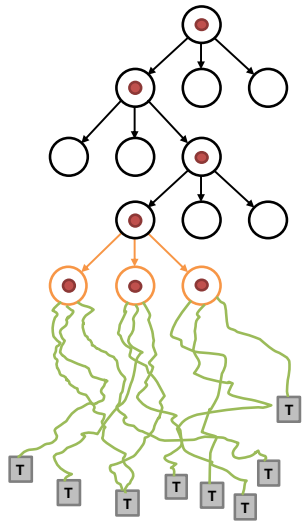
https://medium.com/applied-data-science/alphago-zero-explained-in-one-diagram-365f5abf67e0

MCTS in practice: AlphaGo & Derivatives

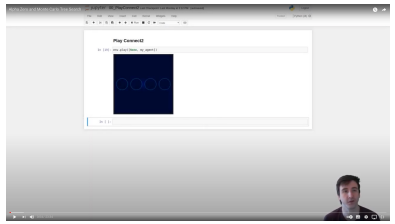
https://deepmind.com/blog/article/muzero-mastering-go-chess-shogi-and-atari-without-rules

MCTS in practice: AlphaZero

- *The ”smallest” technical jump between versions*
- Main differences to AlphaGo Zero:
  - Expand AlphaGo Zero ideas to other games
  - RL Policy and Value networks in AlphaGo & AlphaGo Zero exploited Go rules
    - Augmented data since Go board is invariant to rotation and translation
    - Go games can end with win or lose (no draw like chess). Training of AlphaGo (Zero) exploited this to estimate the probability of winning
  - Self-play is now against the current version of the networks and not against the strongest player so far.



- See also “Alpha Zero and Monte Carlo Tree Seach”: https://www.youtube.com/watch?v=62nq4Zsn8vc



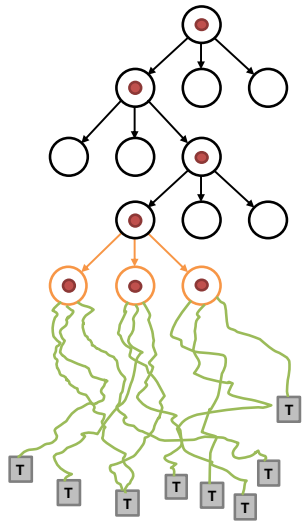
Silver, David, et al. "A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play." Science 362.6419 (2018): 1140-1144.

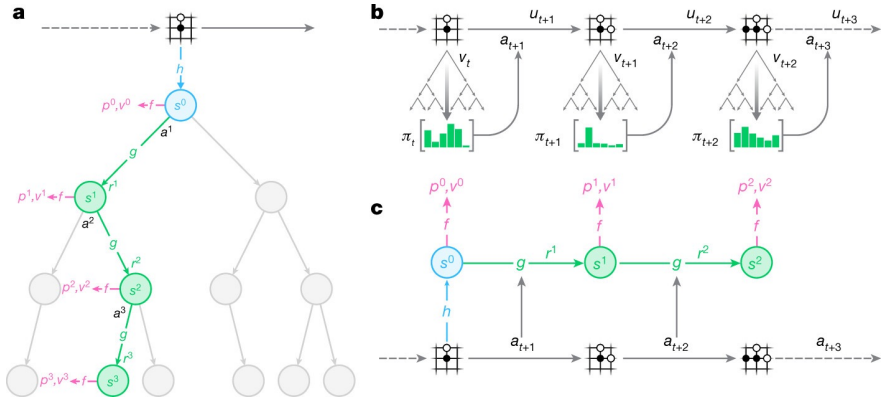
MCTS in practice: AlphaGo & Derivatives

https://deepmind.com/blog/article/muzero-mastering-go-chess-shogi-and-atari-without-rules

MCTS in practice: MuZero

- Does not assume zero-sum games (win/lose) but general RL structure (rewards)
- Policy is still MCTS, but uses **learned model** instead of game rules
- Learns three components:
  - **h**: function that transforms observations to latent states
  - **g**: model that predicts next latent state and reward, based on current latent state and action
  - **f**: function that predicts current policy (from MCTS), and Value of latent state
- All components are learned end-to-end using real game data stored in a replay buffer.





Schrittwieser, Julian, et al. "Mastering atari, go, chess and shogi by planning with a learned model." Nature 588.7839 (2020): 604-609.