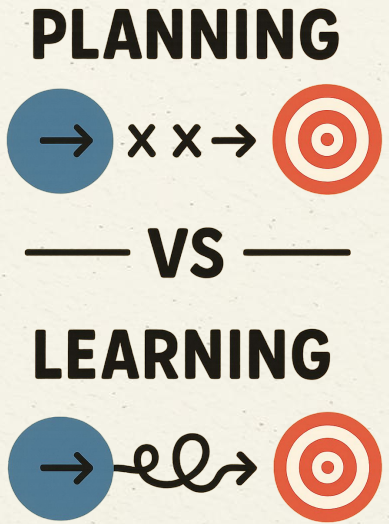
Planning vs. Learning: Monte Carlo Tree Search Meets Deep Reinforcement Learning

Planning v Learning

What’s the difference in AI?



Planning

Decision-Time Search

- Uses a model of the environment to simulate outcomes and plan the best action sequence *before* acting.
    - Example: Monte Carlo Tree Search (MCTS) builds a search tree using the known rules of a game to decide moves.

Learning

Experience-Driven Optimization

- learns a good policy or value function *from trial-and-error experience*, without needing a provided model.
    - Example: Deep Reinforcement Learning (DRL) uses deep neural networks to approximate the optimal policy or value by learning from reward feedback

(Part 1: Key Concepts in RL — Spinning Up documentation)

Key Distinction & Trade-offs

Model-based

- Planning relies on model-based simulations

trade-offs

- Can find optimal decisions on the fly given a perfect simulator
  - may be slow per decision.

Experience-Driven Optimization

- learning can be model-free (no prior model given)

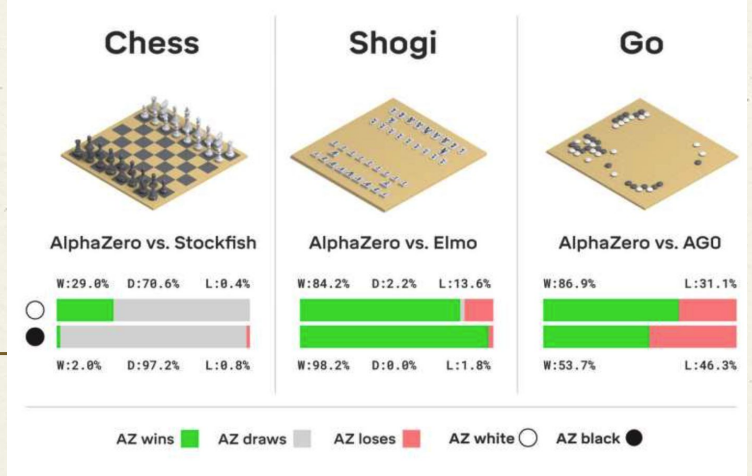
trade-offs

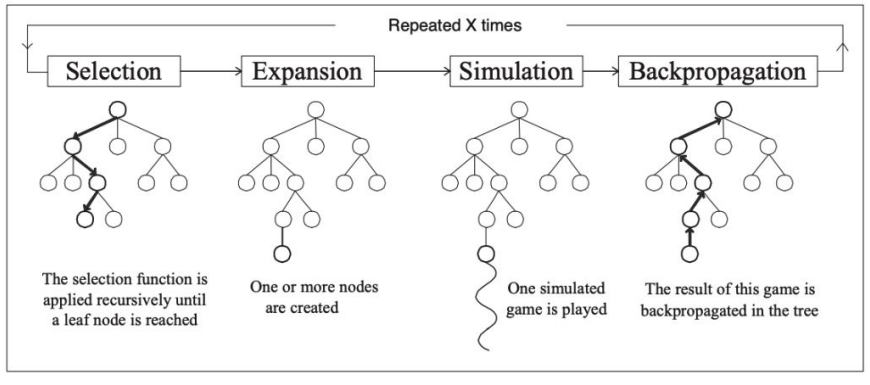
- expends effort upfront to generalize from many episodes, enabling fast decisions later

(Part 1: Key Concepts in RL — Spinning Up documentation)

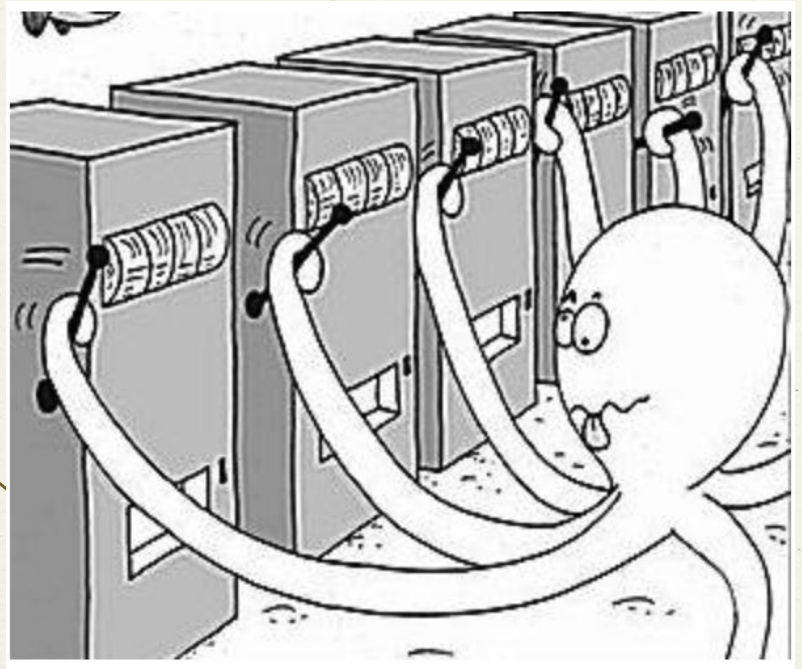
Context

The synergy of planning and learning enabled breakthroughs like AlphaZero, which combined MCTS with deep neural networks to achieve superhuman performance in Go, chess, and shogi





Monte-Carlo Tree Search



Generally speaking

MCTS acts as a multi-armed bandit

source

What does that entail?

A big advantage

MCTS is an “anytime” algorithm – it can return an action at any time, and more computation (more simulations) improves the decision quality. This makes it flexible under different time constraints.

1

Requires a forward model

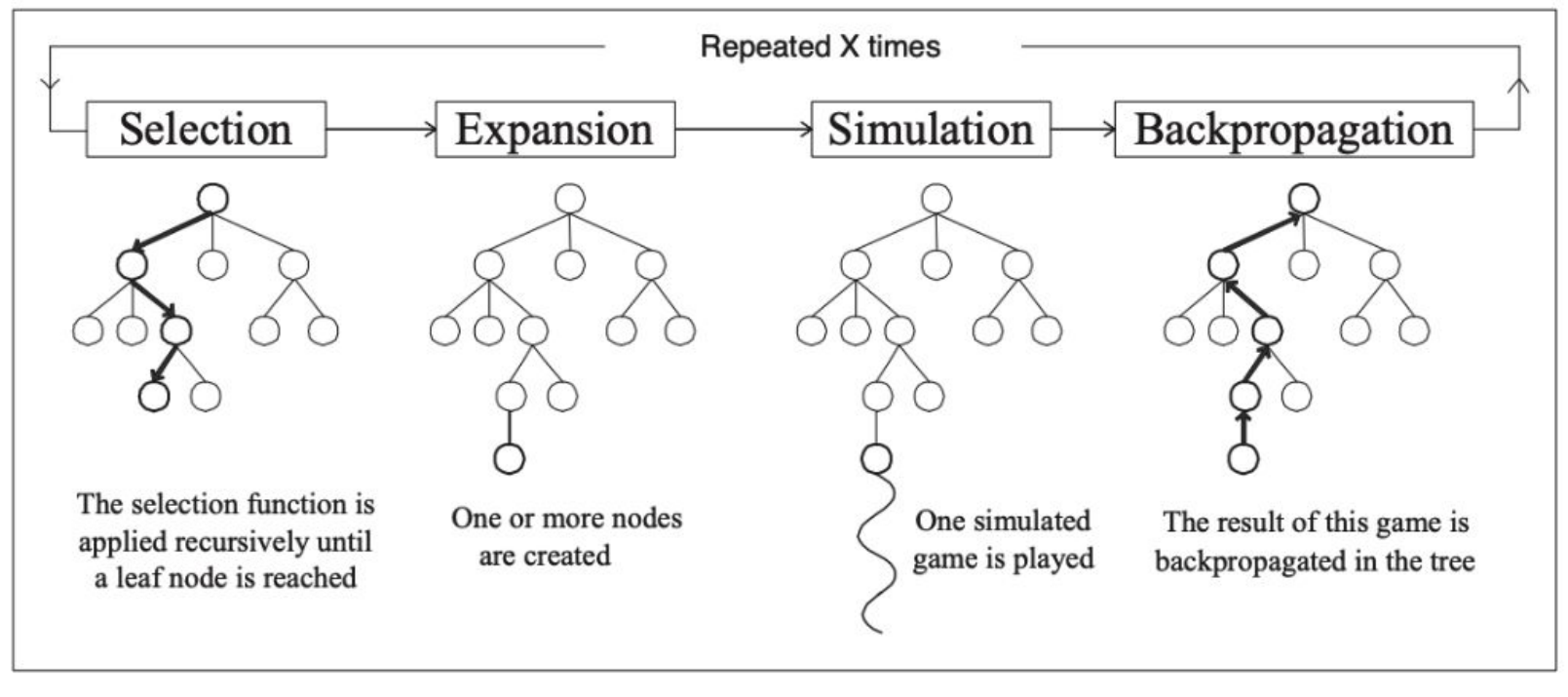
doesn’t learn general knowledge across games
Performs a fresh search for each decision from the current state.

2

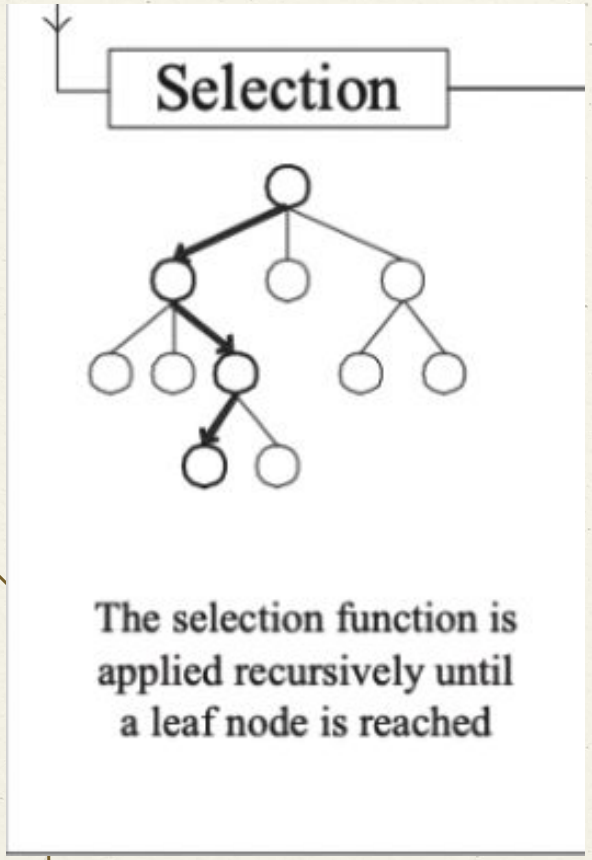
Is A Heuristic

it can work without domain-specific heuristics- great for for games like Go where designing an evaluation function was hard.

Back to this image….



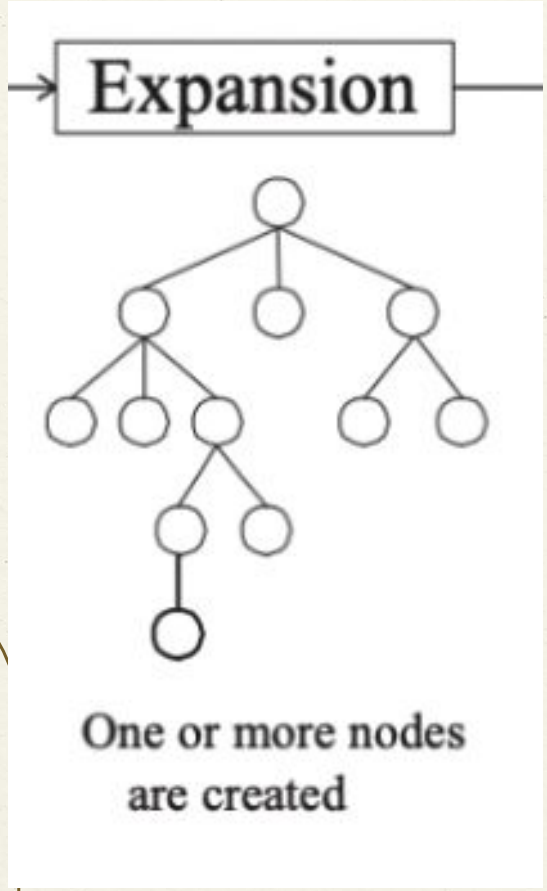
adapted from Chaslot 2006



Selection

Traverse the current tree from the root state, at each step choosing the child node with the best *potential* (according to a selection policy) until a leaf node is reached.
Positives: balance exploitation of known good moves and exploration of less-visited moves.

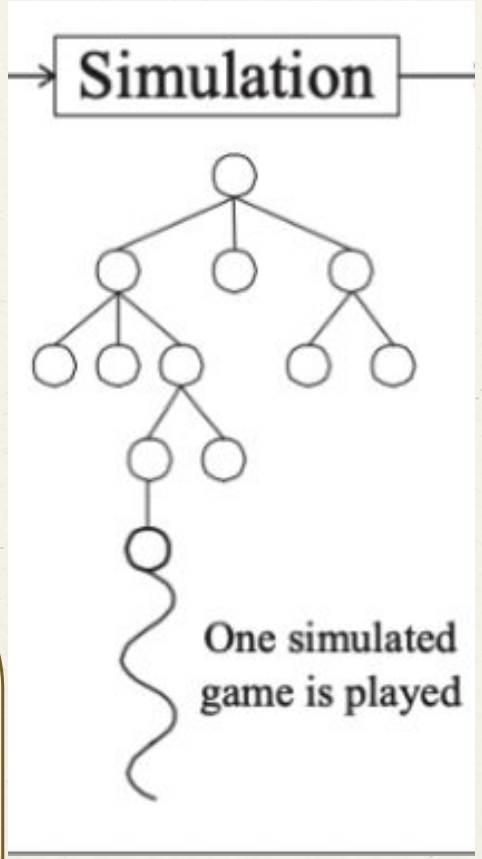
source



Expansion

If the leaf node is non-terminal (game not over), add one or more child nodes (unexplored moves) to the tree
Choose one newly expanded node to investigate.

source

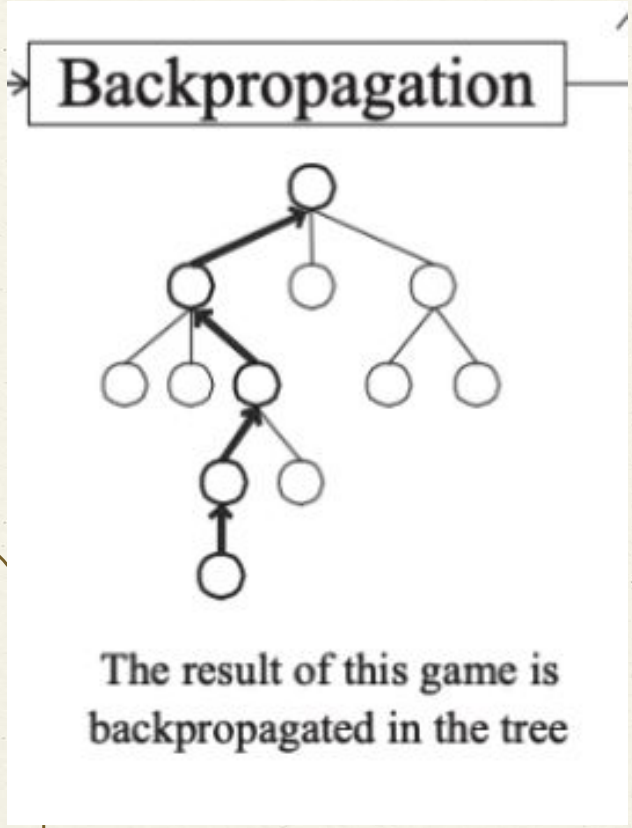


Simulation

From the new node, simulate a random (or guided) play-out to the end of the game (or for a certain depth) to get an outcome, e.g. win or loss

Provides a Monte Carlo estimate of the value of that node.

source



BackProp

Propagate the simulation result backward up the tree: update each node along the path (increment visit counts, and update value estimates).

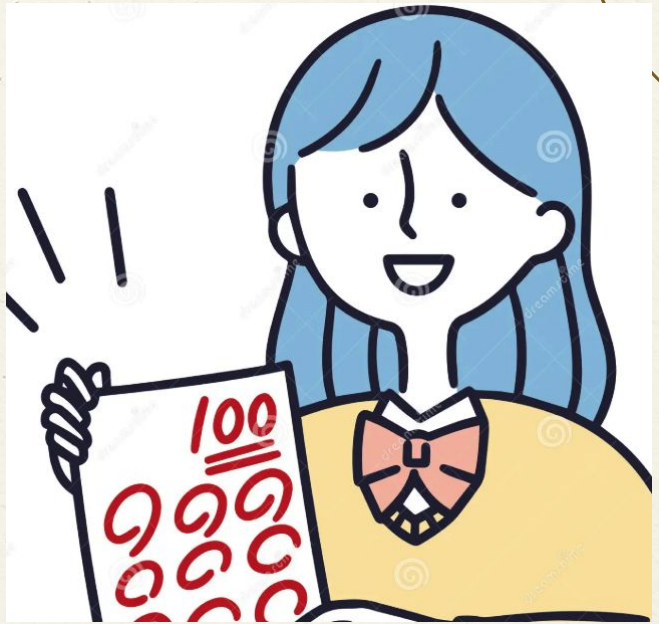
For example: in a game setting, if the simulation resulted in a win for the current player, increment the win count for the nodes that led to that outcome.

source

What's being updated though?

Each node stores two things :

position



estimated value

- Given a model M_v and a simulation policy π
- For each action a ∈ A
  - Simulate K episodes from current (real) state s_t

$$\{s_t, a, R^k_{t+1}, ..., S^k_T\}^K_{k=1} \sim \mathcal{M}_v, \pi$$

  - Evaluate actions by mean return (Monte-Carlo evaluation)

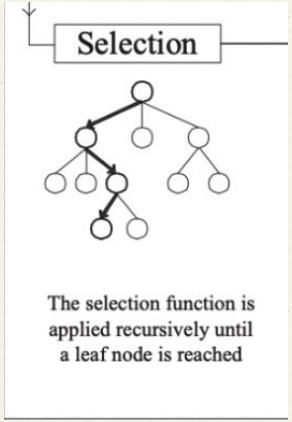
$$Q(s_t, a) = \frac{1}{K} \sum_{k=1}^{K} G_t \xrightarrow{P} q_\pi(s_t, a)$$

- Select current (real) action with maximum value

$$a_t = \underset{a \in A}{\text{argmax}}\, Q(s_t, a)$$

- This is essentially doing 1 step of policy improvement

Stanford Slides



The Crux of MCTS

Choose nodes that potentially lead to high rewards (exploitation) while occasionally trying lesser-known nodes to gather information (exploration)

Upper Bound Confidence Formula:

$$\frac{w_i}{s_i} + c\sqrt{\frac{\ln s_p}{s_i}}$$

- w_i : this node's number of simulations that resulted in a win
- s_i : this node's total number of simulations
- s_p : parent node's total number of simulations
- c : exploration parameter

Try moves with good results so far, but every now and then try moves you haven't tried much (or at all)

source

Strengths & Weaknesses

Domain General

Works on **any** environment where simulations are possible

Efficiency

MCTS concentrates search on more promising moves- useful in time-constrained scenarios

Performance

MCTS with enough simulations can achieve strong play, **but** MCTS alone still falls short when num simulations is simply too much

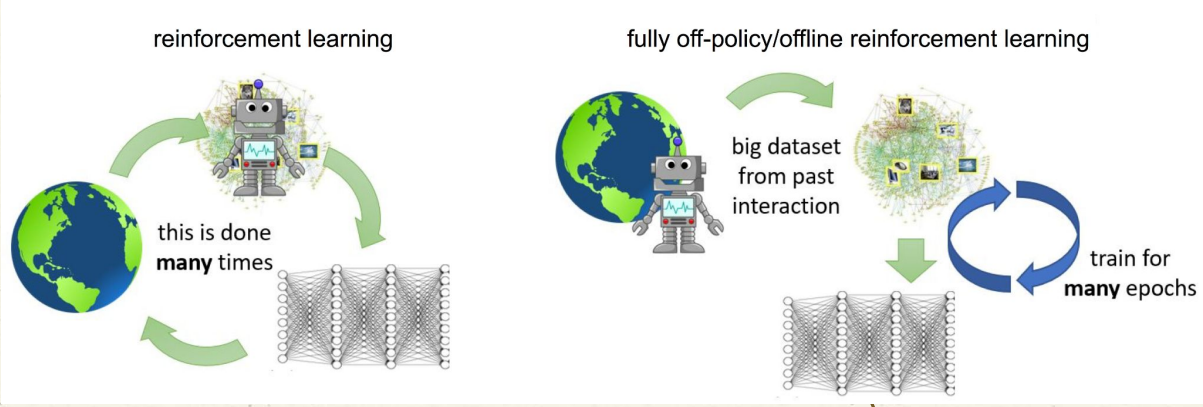
No Long-Term Learning

MCTS doesn't learn a the value function for reuse - in contrast, learning-based methods accumulate knowledge

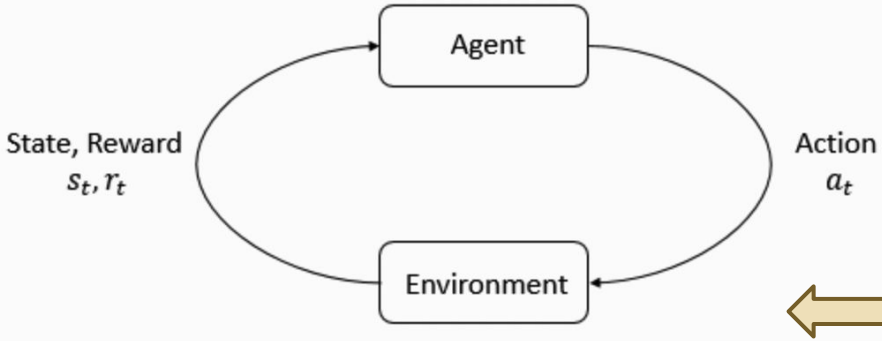
DEMO!!

MCTS on Tic-Tac-Toe

Deep Reinforcement Learning



RL Recap



Agent-environment interaction loop.

An agent interacts with an environment, receiving a state from which it determines an action. The environment then doles out feedback in the form of rewards.

Goal: learn a **policy** that maximizes cumulative reward

Key Difference between planning systems :

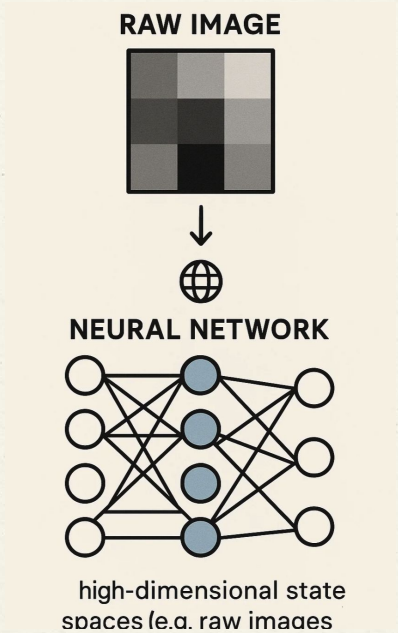
- RL typically does *not* assume a known model of the environment. Instead, the agent learns from direct experience (trial and error)
  - By accumulating these experiences, the agent can evaluate which actions are good in which states.

Central Trade-off: *exploration vs. exploitation*

source

The “Deep” Part of DeepRL

- These systems use neural nets to handle complex inputs or generalize across similar states
  - i.e : we use the neural nets to approximate the **policy/value function**
  - This allows systems to generalize problems with multidimensional data - like pixel states in a game screen or from cameras



Value Functions & QLearning

Value functions estimate how good a state is **in terms of future rewards**

<u>One Common Value-Based Approach:</u>

- Learns the **Q-value** (the expected value of taking a given action, and following through with optimal policy after)

$$Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma \max_{a'} Q(s',a') - Q(s,a)]$$

Initialize Q(s,a) arbitrarily (or zeros). Then, as the agent experiences transitions, update the Q-value for (state, action) towards the observed reward plus the estimated value of the next state.

Initialize $Q(s,a)$ arbitrarily
Repeat (for each episode):
  Initialize s
  Repeat (for each step of episode):
    Choose a from s using policy derived from Q
    Take action a, observe r, s'
    Update
      $Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma \max_{a'} Q(s',a') - Q(s,a)]$
    $s \leftarrow s'$;
  Until s is terminal

<u>image credits</u>

More on QLearning

1 Convergence

Over many updates, Q(s,a) converges to the true optimal Q-values (given sufficient exploration). The agent can then pick the action with highest estimated value in each state.

2 Completely Model Free

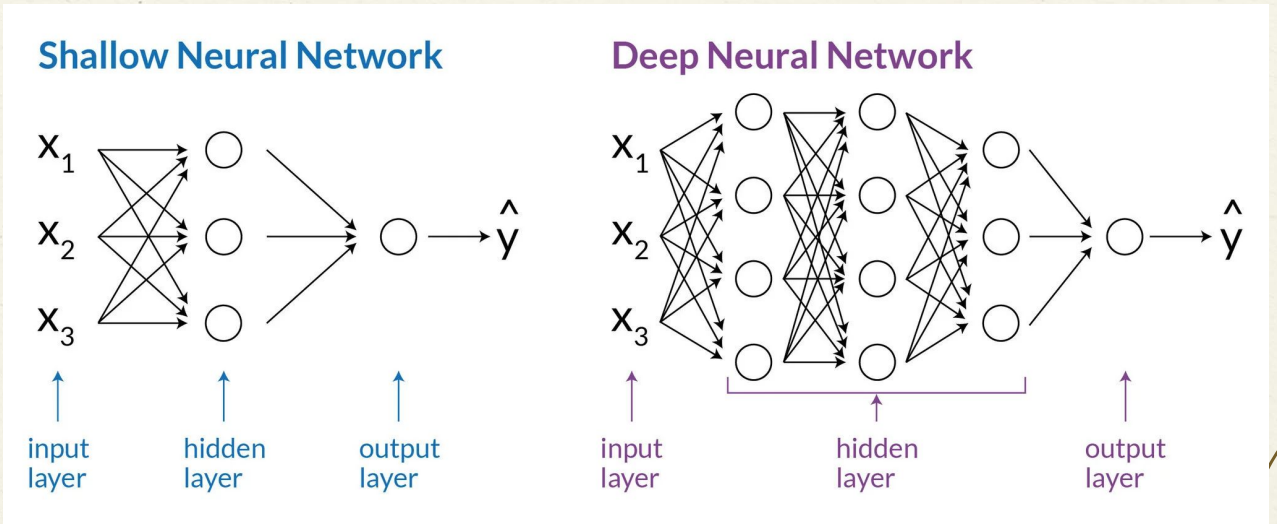
Because it's an off-policy algorithm, the learning of Q assumes following the optimal policy even while you might be exploring with a different policy.

3 Exploration Challenge...

if the agent always greedily takes the current best Q action, it might miss out on trying actions that could actually be better.

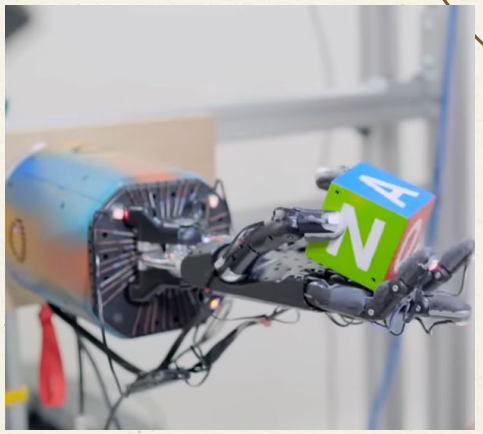
From the RL lecture, do you remember how this is often solved?

However, for large or continuous state spaces, we need function approximation…



which is where deep neural nets come in!

Success Stories with DeepRL



… And Challenges

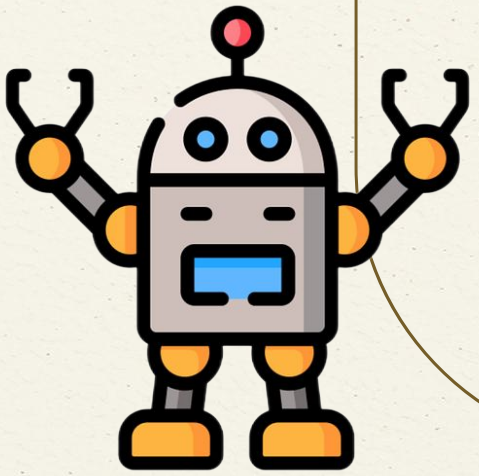
Efficiency

Agents often require an enormous number of interactions (millions/ billions of frames for Atari) – far more than a human would need – highlighting a sample inefficiency issue in pure model-free learning.

Brittleness

Training is unstable at times – things like choosing hyperparameters, dealing with exploration, and credit assignment over long timescales can be tricky.

- Techniques like reward shaping, or algorithms like policy gradients vs Q-learning, each have their own stability concerns.



DEMO!!

QLearning on GridWorld

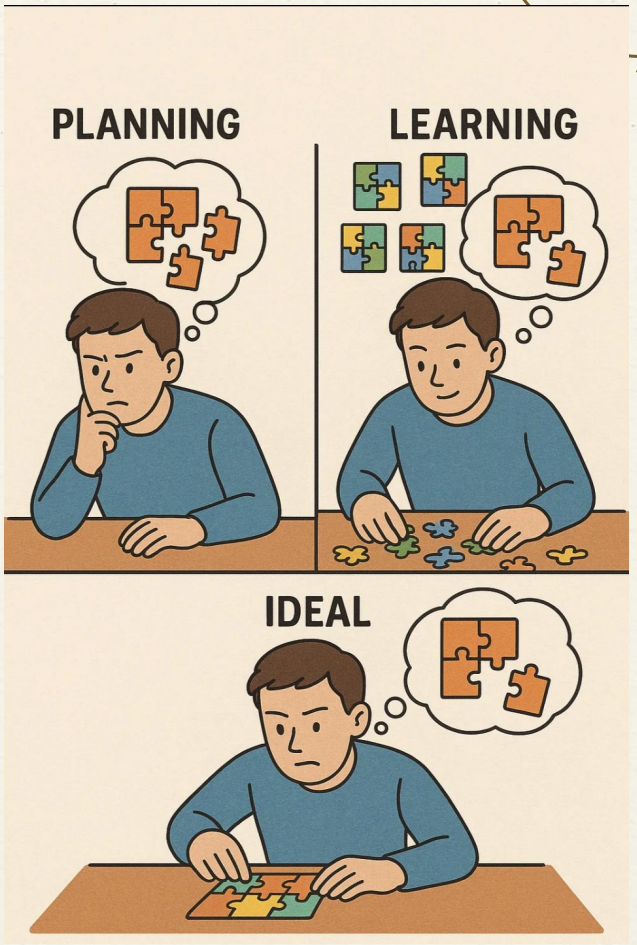
A Deeper Comparison

MCTS

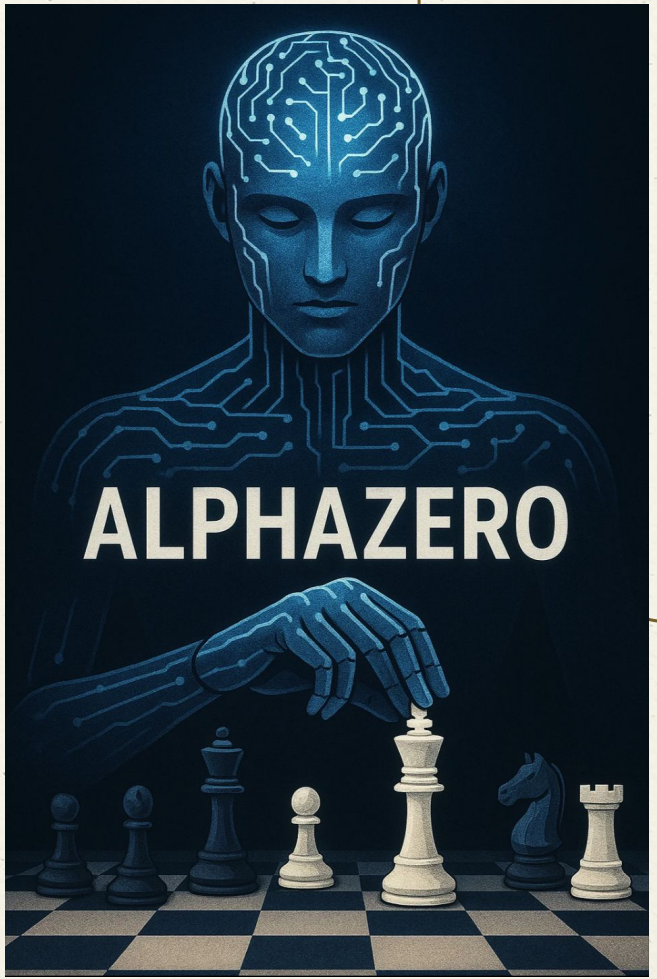
- Require a model. They need the rules of the game or a simulator to roll out outcomes
- Planning expends most of its computation at decision time
- Limited generalization & improves with deeper search
- Excels with perfect-simulators & deterministic problems

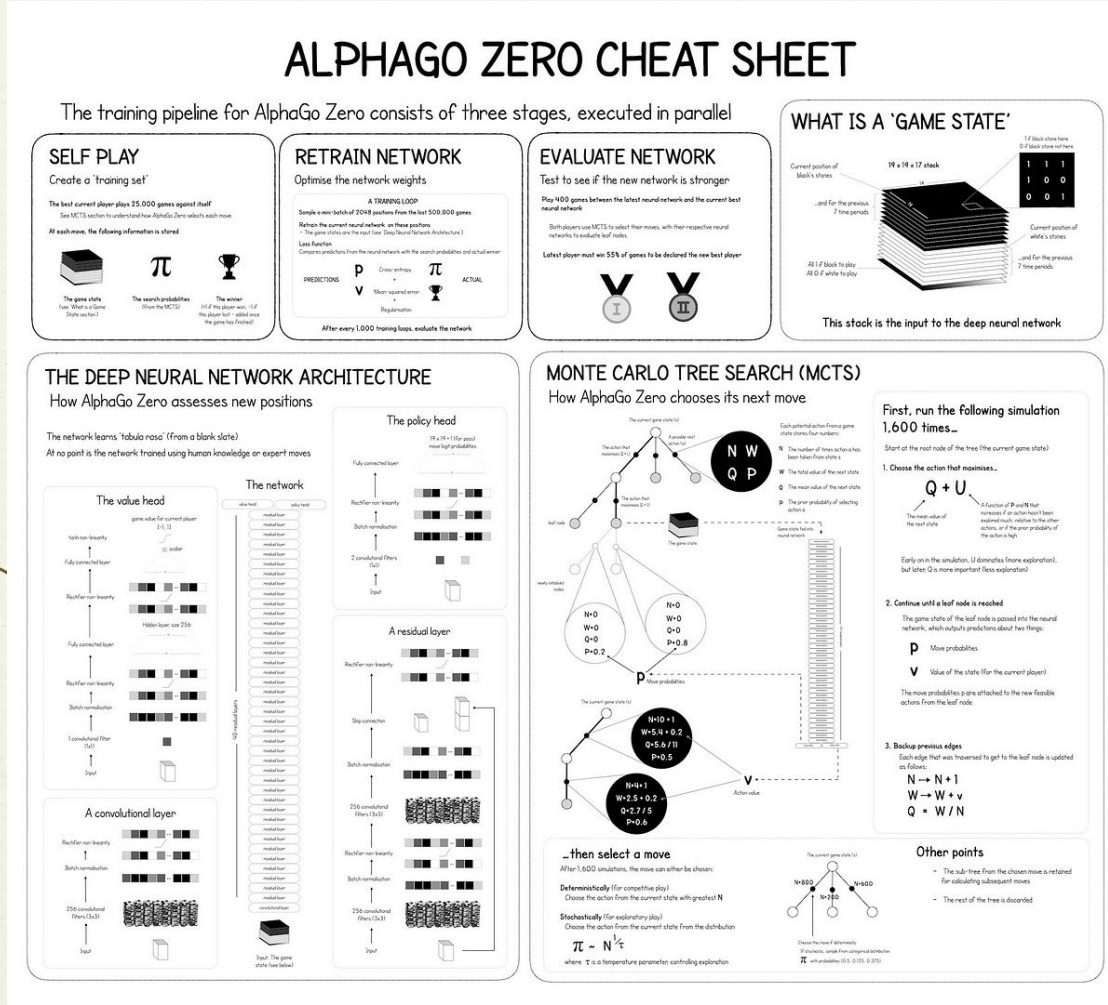
DeepRL

- Model-free, learning what to do from scratch via data
- Expends computation at training time- agent can make a quick decision by just evaluating its learned policy or value function
- good generalization & improves with experience (data points)
- Excels at long-horizon tasks and situations with no reliable model



Alpha Zero

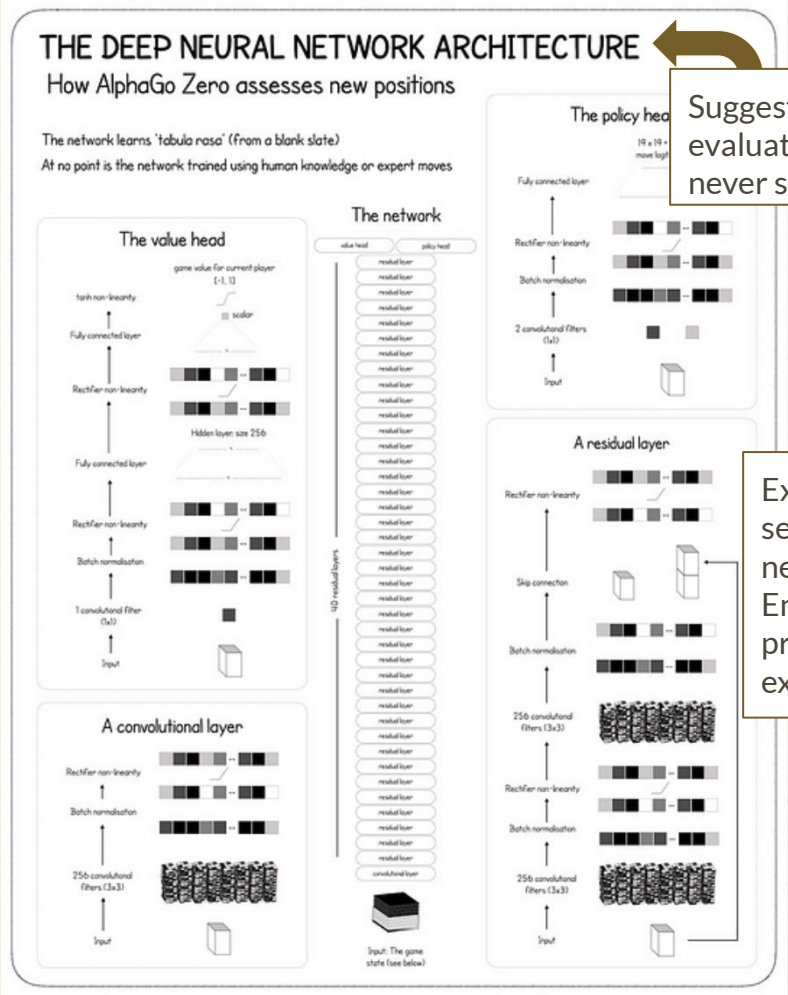




Original models used human expert games as training data, **AlphaGo Zero removed this need**

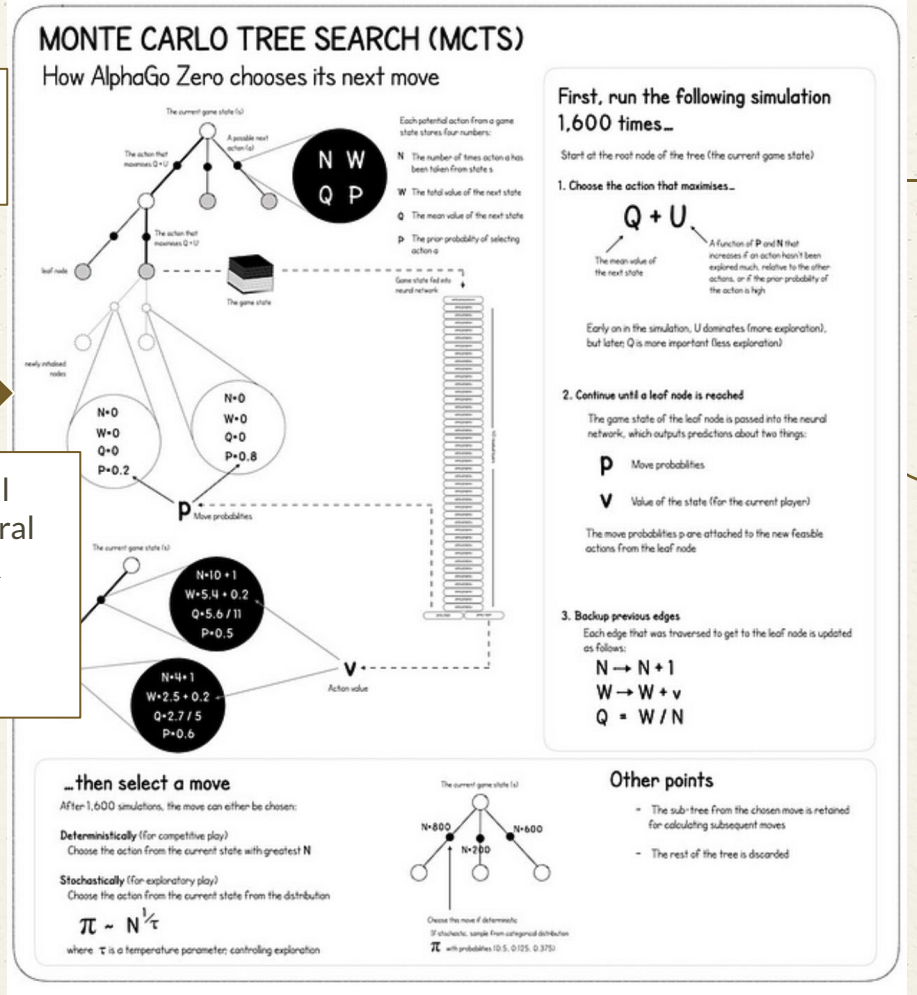
Alpha Zero went a **step further** & generalized to many other games

source

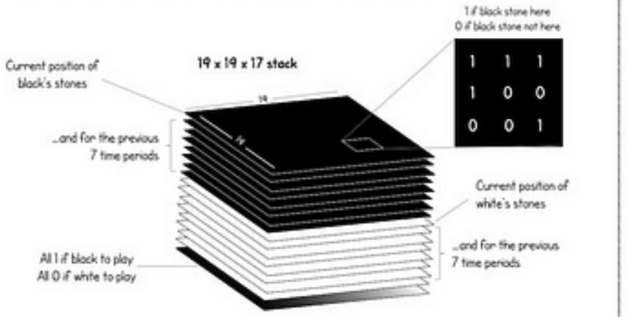


Suggest moves and evaluate positions never seen before

Explore tactical sequences neural net may miss & Encourages probabilistic exploration



WHAT IS A 'GAME STATE'



This stack is the input to the deep neural network

Keeps track of both players' previous 7 moves as well as all current states

Self-improvement via MCTS + Neural Nets

Tends deterministic

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')

π

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 — ACTUAL: $(\pi$, winner$)$

After every 1,000 training loops, evaluate the network

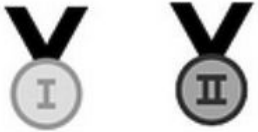
EVALUATE NETWORK

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



Why This Combination?

Sample Efficiency

MCTS essentially focuses the learning on the most critical mistakes- addressed the credit assignment problem by forcing near-optimal play trajectories during training

Strong, Quick Policies

AlphaZero searches far fewer positions than traditional engines because the learned policy prior steers it toward promising moves - the game tree explores far less

Generalization to New Positions

Once trained, the neural net can evaluate and play positions it never exactly saw. This means AlphaZero can play instantly with its neural network alone if needed, although weaker

Beyond AlphaZero

In 2019, DeepMind introduced MuZero, which basically says: we don't even need to hardcode the simulator; we can learn a model that's good enough to plan with. MuZero is "learning to plan" in some sense

Final DEMO!!

Playing Against "AlphaZero"

Quick Quiz !!

1. What is the fundamental difference between a planning algorithm like MCTS and a learning algorithm like Q-learning or policy gradient in how they improve decision-making?

2. MCTS uses four phases: Selection, Expansion, Simulation, Backpropagation. What is the purpose of the Simulation phase in MCTS, and why is it important for evaluating a state?

3. In reinforcement learning, what does the agent aim to maximize, and how is this different from what a planning algorithm maximizes during its search?

4. How does AlphaZero use Monte Carlo Tree Search during self-play training, and what role does the neural network play in the search?

References

- Dimitri Bertsekas. *Reinforcement Learning and Optimal Control* – Lecture Slides, MIT. (Definition of planning vs. learning) (Reinforcement Learning and Optimal ControlA Selective Overview).
- Csaba Szepesvári, Levente Kocsis (2006). **“UCT”**. (Introduced the UCT algorithm combining MCTS with UCB1, proved convergence) (UCT - Chessprogramming wiki).
- Emma Brunskill, Stanford CS234 Reinforcement Learning (2024). *Lecture 14: Monte Carlo Tree Search.* (Discussion of AlphaZero and MCTS achievements) (Lecture 14: Monte Carlo Tree Search).
- Swarthmore CS63 Course Reading. *Monte Carlo Tree Search - About* (MCTS steps and properties) (Monte Carlo Tree Search - About) (Monte Carlo Tree Search - About).
- OpenAI Spinning Up (2018). *Key Concepts in RL*. (Clear introduction to RL terms and success examples) (Part 1: Key Concepts in RL — Spinning Up documentation) (Part 1: Key Concepts in RL — Spinning Up documentation).
- DeepMind (Silver et al.). *AlphaGo Zero* Nature 2017 & *AlphaZero* Science 2018. (AlphaZero methodology and results) (AlphaZero AI system able to teach itself how to play games, play at highest levels) (AlphaZero AI system able to teach itself how to play games, play at highest levels).
- Jonathan Hui (2018). “MCTS in AlphaGo Zero” – Medium blog. (Describes how AlphaGo Zero’s MCTS uses neural network outputs) (Monte Carlo Tree Search (MCTS) in AlphaGo Zero | by Jonathan Hui | Medium) (Monte Carlo Tree Search (MCTS) in AlphaGo Zero | by Jonathan Hui | Medium).
- Surag Nair (2017). *AlphaGo Zero Tutorial*. (Explanation of AlphaGo Zero training loop and loss functions) (Simple Alpha Zero) (Simple Alpha Zero).
- Hacker News (Eric Jang, 2018). *Discussion on MCTS vs Policy Gradients*. (Intuition on when planning helps, e.g., perfect simulator scenario) (Can someone share some intuition of the tradeoffs between monte-carlo tree searc... | Hacker News).
- DeepMind Blog (2019). *MuZero: Mastering Games Without Rules.* (Combining planning with learned models) (MuZero - Wikipedia).