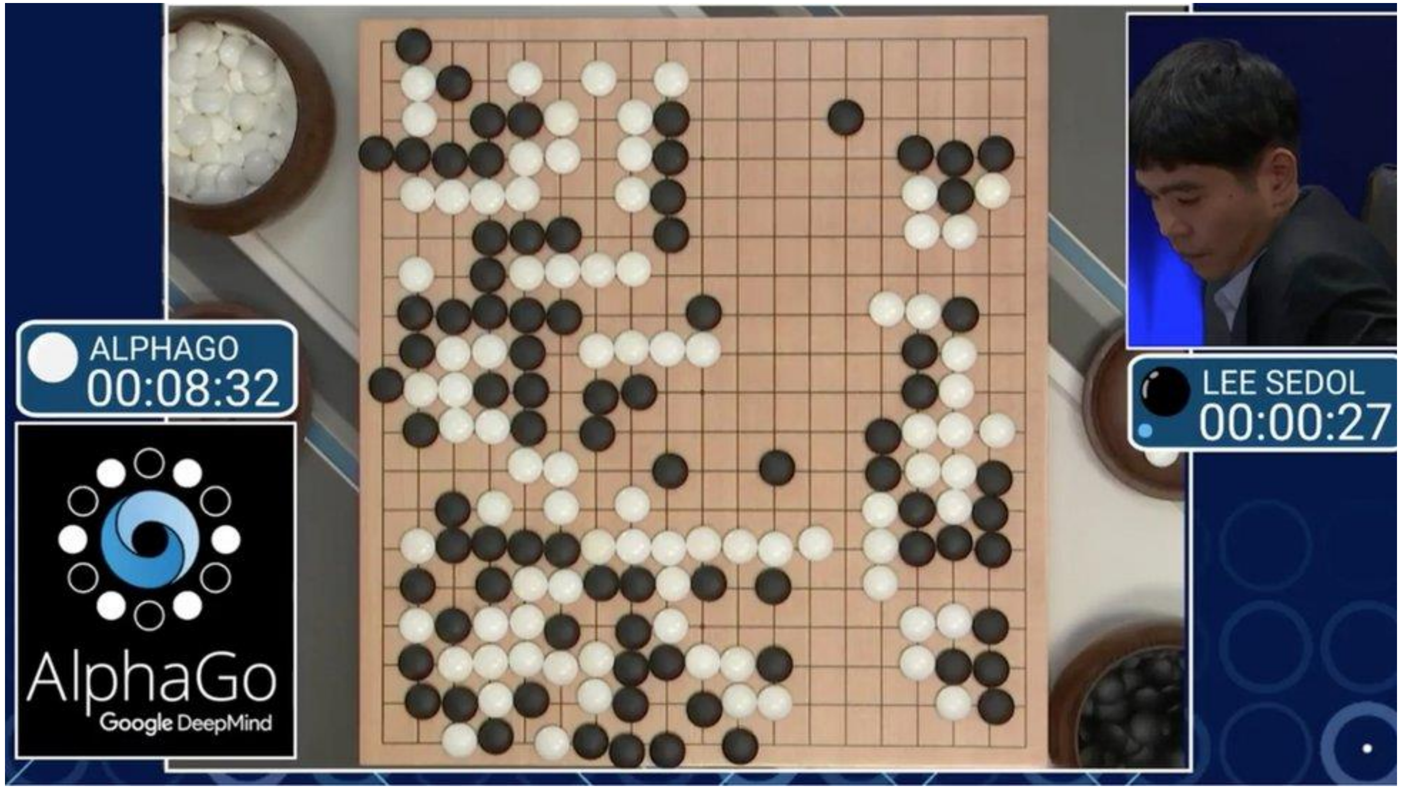
AlphaGo

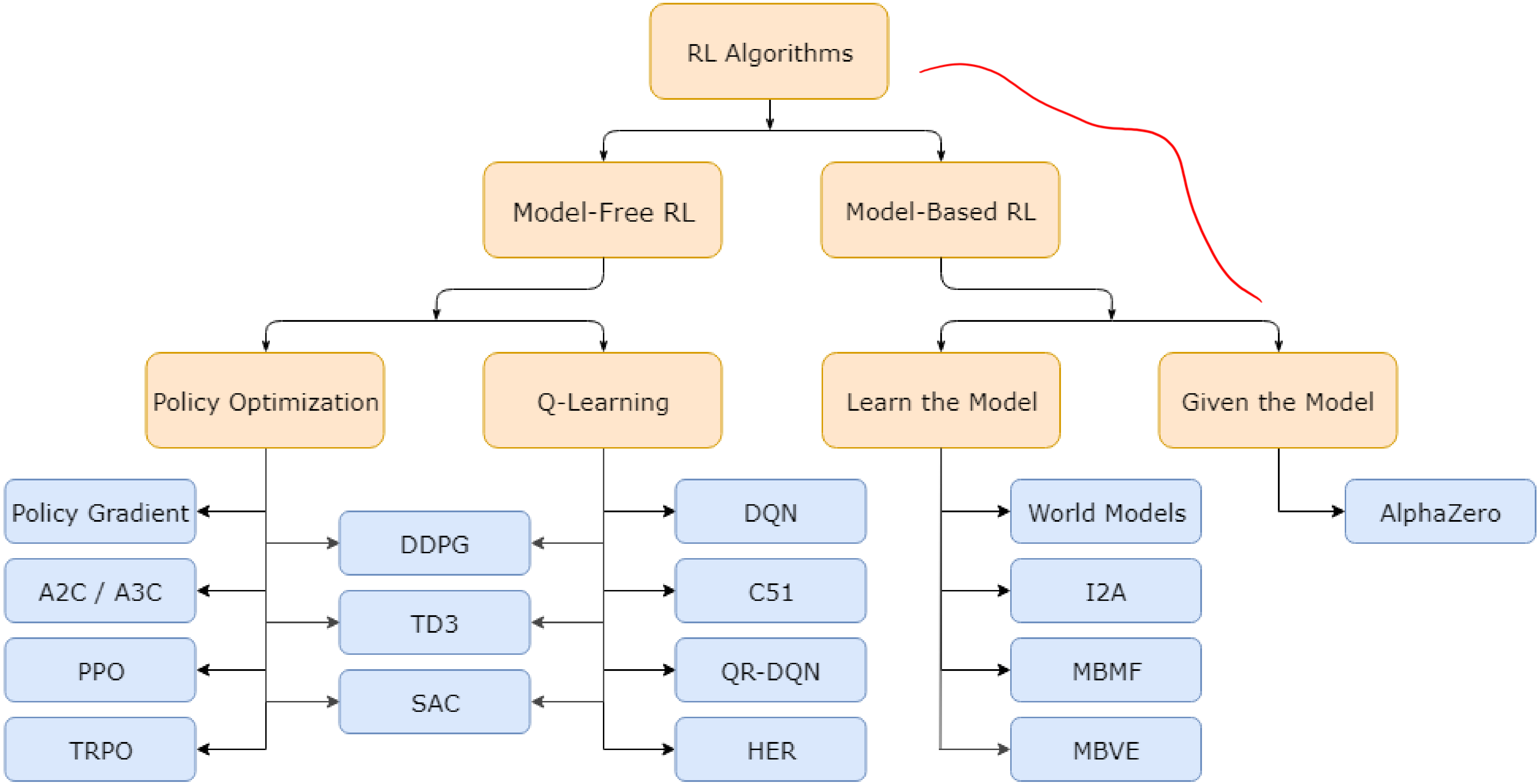


Instructor: Daniel Brown --- University of Utah

Announcements

- Mid-semester feedback is open! Due Feb 26th.
- No Class on Wednesday, but reading assignment.
- Final project groups due March 10
- Final project paragraph pitches due March 17th

Rough Taxonomy of RL Algorithms



Policy Gradient Recap

1. Start with random policy parameters θ_0
2. Run the policy in the environment to collect N rollouts (episodes) of length T and save returns of each trajectory.

$$a_t \sim \pi_\theta(\cdot\,|s_t) \Rightarrow (s_0, a_0, r_0, s_1, a_1, r_1, \dots, r_T, s_{T+1})$$

$$D = \{\tau_1, \dots \tau_N\}, \qquad R = \{R(\tau_1), \dots R(\tau_N)\}$$

3. Compute policy gradient

$$\nabla_\theta J(\pi_\theta) = E_{\tau \sim \pi_\theta}\left[\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t)\ R(\tau)\right]$$

4. Update policy parameters

$$\theta_{k+1} \leftarrow \theta_k + \alpha \nabla_\theta J(\pi_\theta)\Big|_{\theta_k}$$

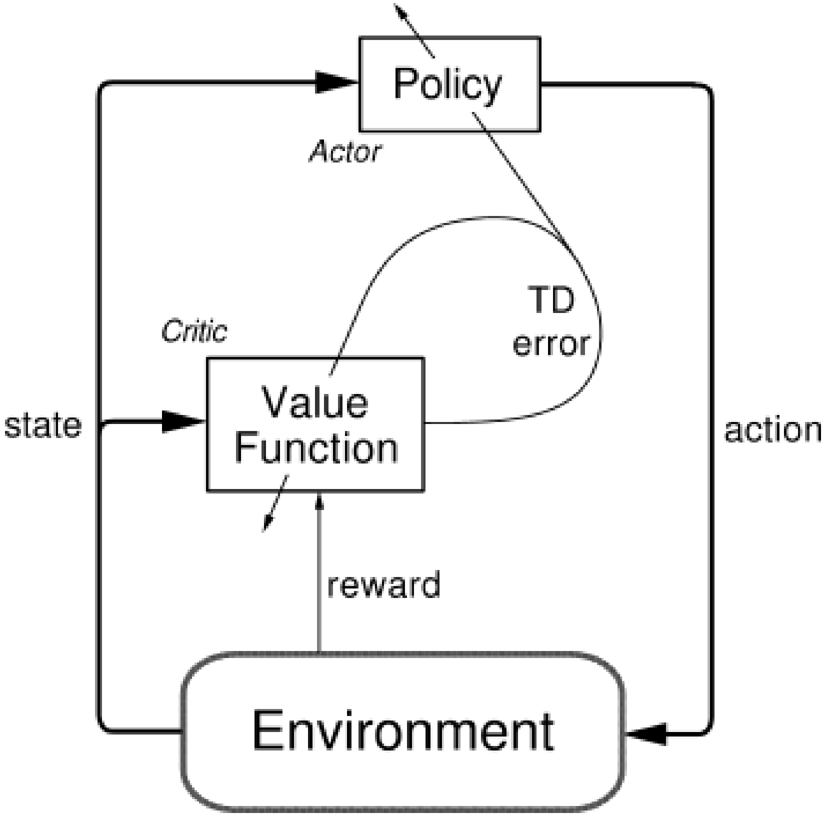
5. Repeat (Go to 2)

Policy Gradient RL Algorithms

- We can directly update the policy to achieve high reward.
- Pros:
  - Directly optimize what we care about: Utility!
  - More stable than Q-Learning methods like DQN and scales well to high-dimensional continuous control tasks.
- Cons:
  - On-Policy -> Sample-inefficient we need to collect a large set of new trajectories every time the policy parameters change.
  - Q-Learning methods are usually more data efficient since they can reuse data from any policy (Off-Policy)

Actor Critic Algorithms

- Combining value learning with direct policy learning
  - One example is policy gradient using the advantage function



How to get an AI to play Go

- Branching factor close to 250
- Depth close to 150
- O(250^150) ~= 5x10^350

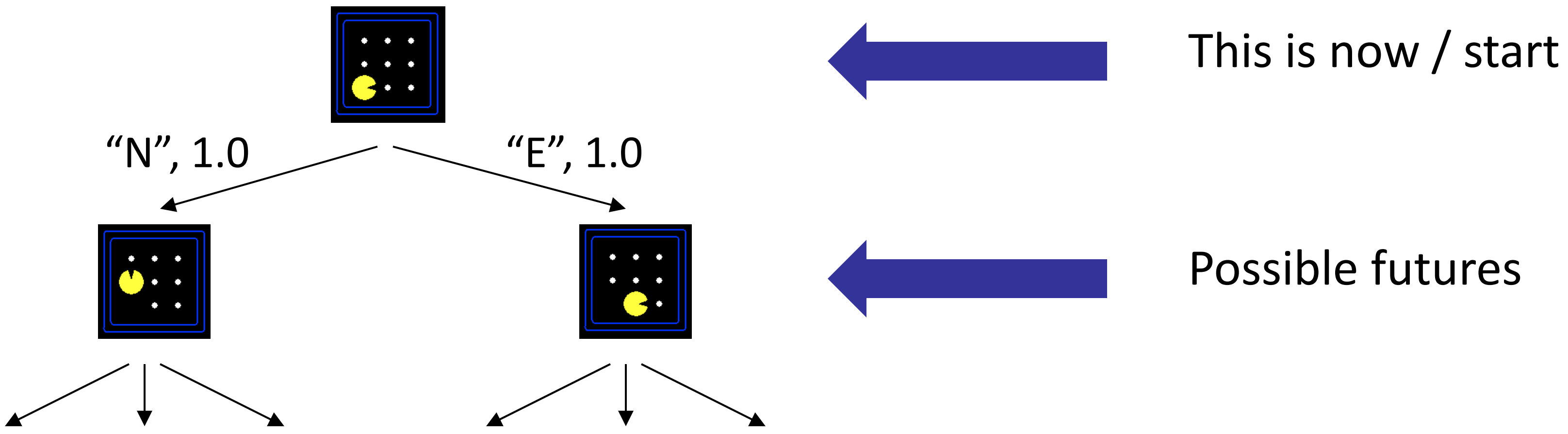

ALPHAGO

How AlphaGo works

- Monte Carlo Tree Search (MCTS)
  - How AI chooses next move
- Value Network
  - AI assess new positions using this network
- Reinforcement Learning
  - Trains the AI by using the current best agent to play against itself

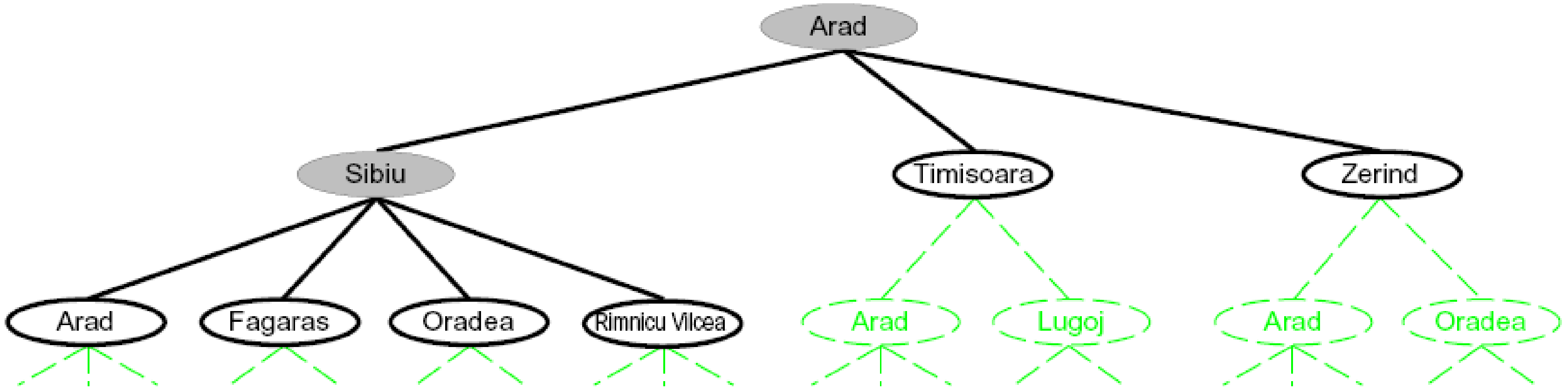
+ Imitation Learning

Search Trees



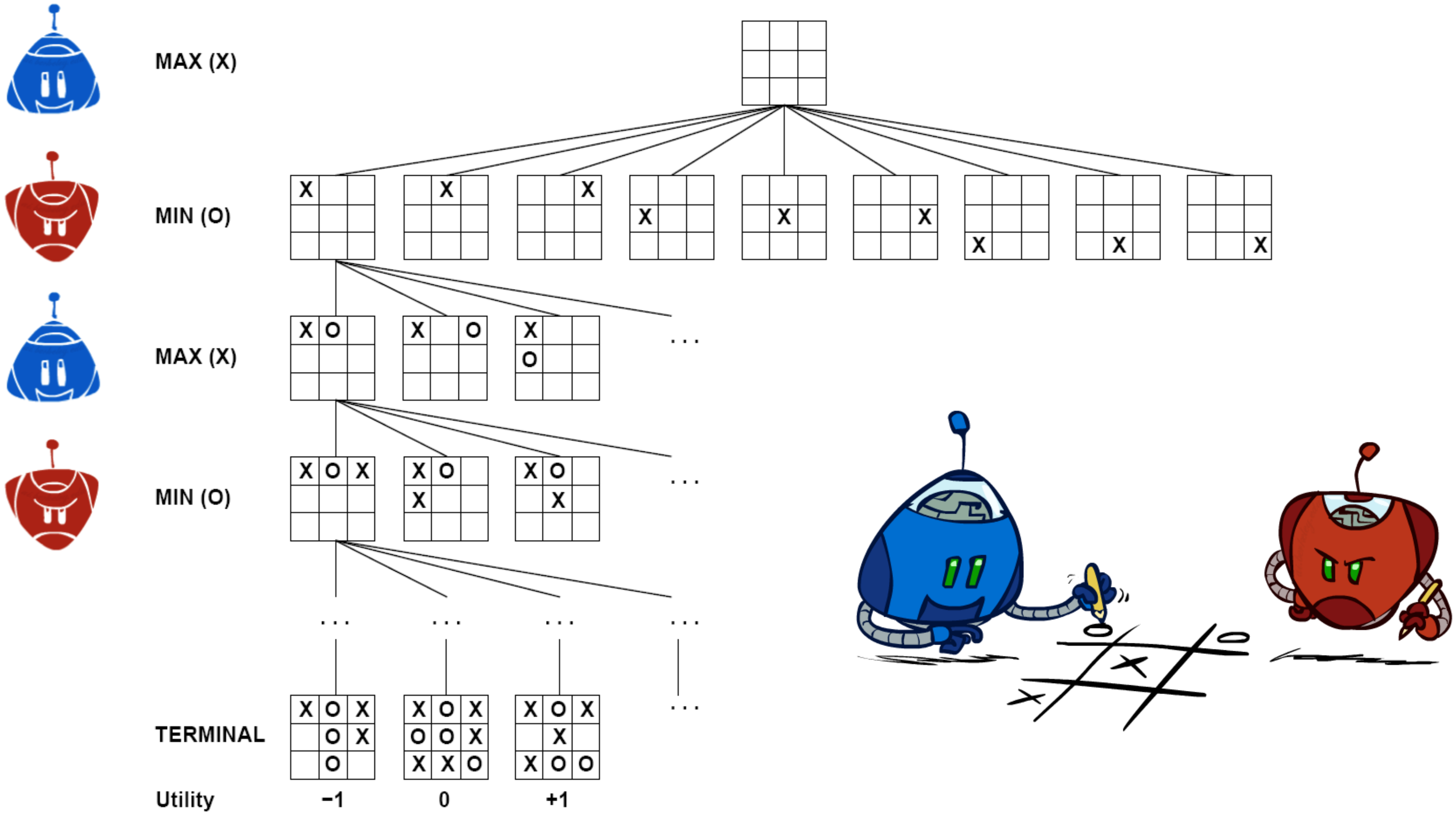
- A search tree:
  - A "what if" tree of plans and their outcomes
  - The start state is the root node
  - Children correspond to successors
  - Nodes show states, but correspond to PLANS that achieve those states
  - For most problems, we can never actually build the whole tree

Searching with a Search Tree



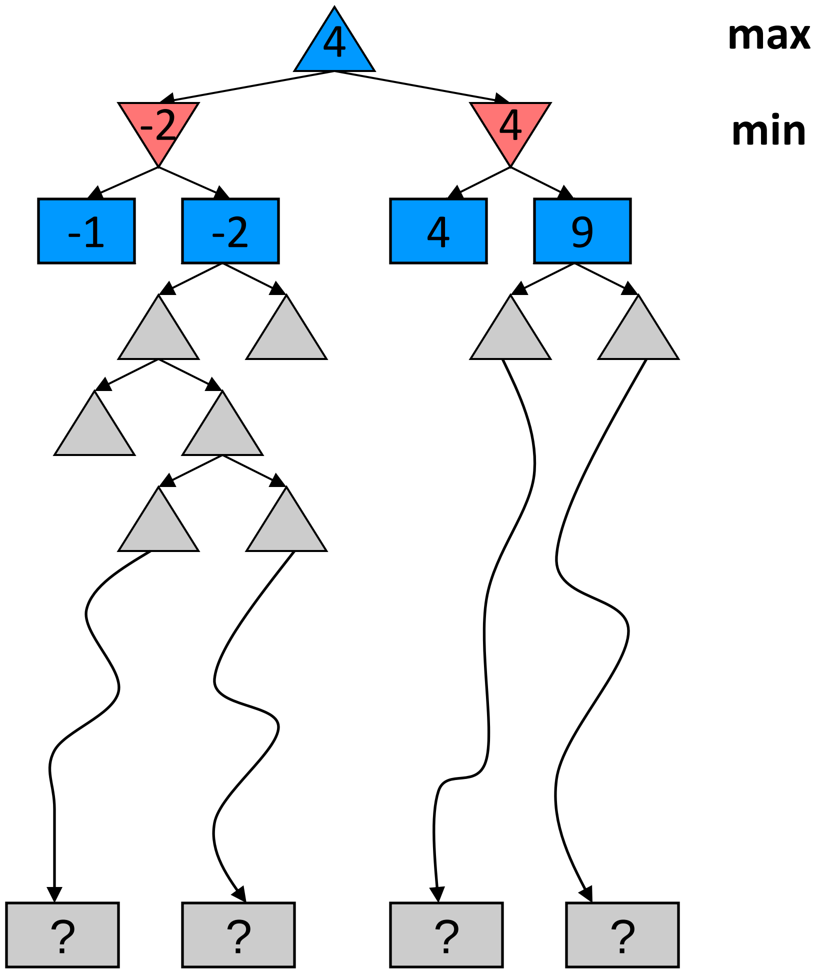
- Search:
  - Expand out potential plans (tree nodes)
  - Maintain a fringe of partial plans under consideration
  - Try to expand as few tree nodes as possible

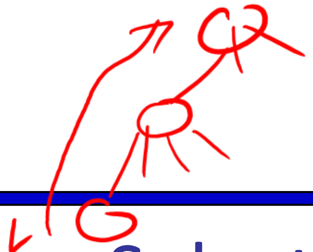
Min-Max Search Tree



Resource Limits

- Problem: In realistic games, cannot search to leaves!
- Solution: Depth-limited search
  - Instead, search only to a limited depth in the tree
  - Replace terminal utilities with an evaluation function for non-terminal positions
- Guarantee of optimal play is gone
- More plies makes a BIG difference





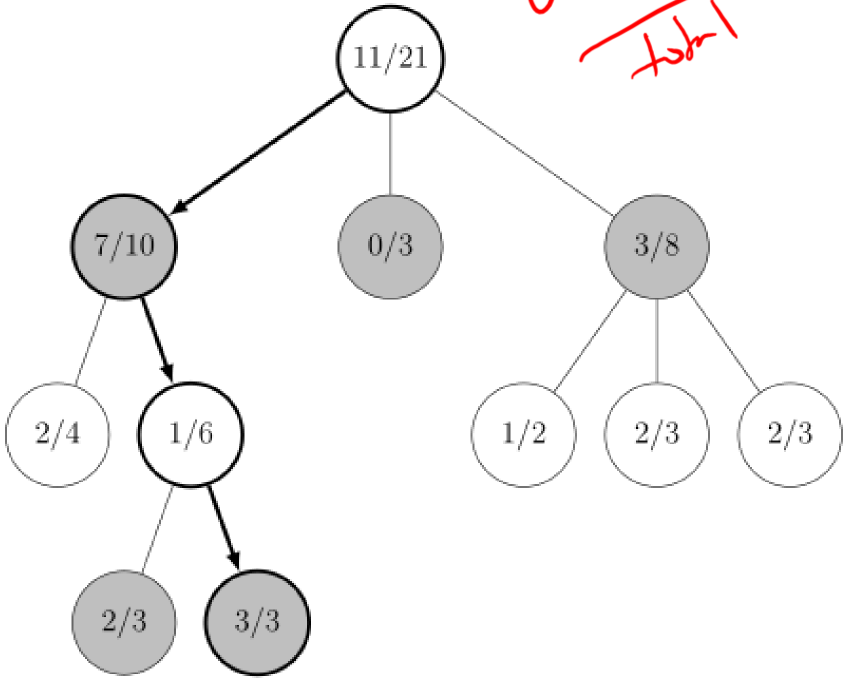
Monte Carlo Tree Search (MCTS)

- Selection
  - Starting at root node, select child nodes recursively in tree until a leaf node L (unexplored node in fringe) is reached
- Expansion
  - Chosen leaf node, L, is added to the search tree and children are added to fringe.
- Evaluation (simulation)
  - Run a simulated playout from L until you reach terminal state.
- Backup
  - Using simulation result, go back up the tree and update statistics (values and visit counts) of encountered nodes.

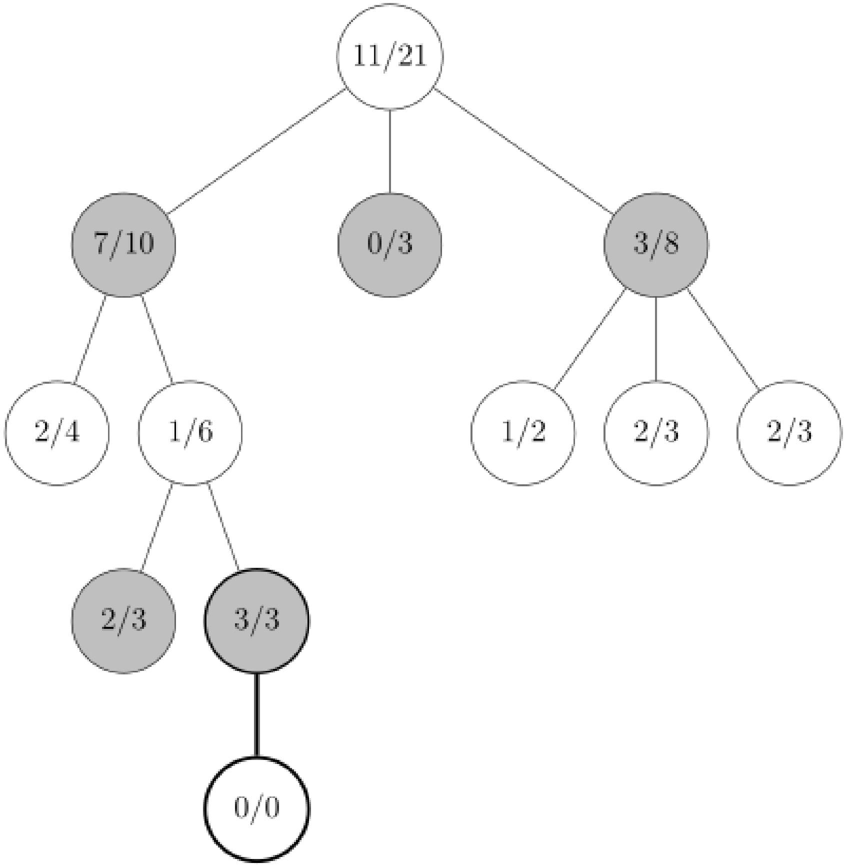
Example

Selection

$\frac{\text{win rate}}{\text{total}}$

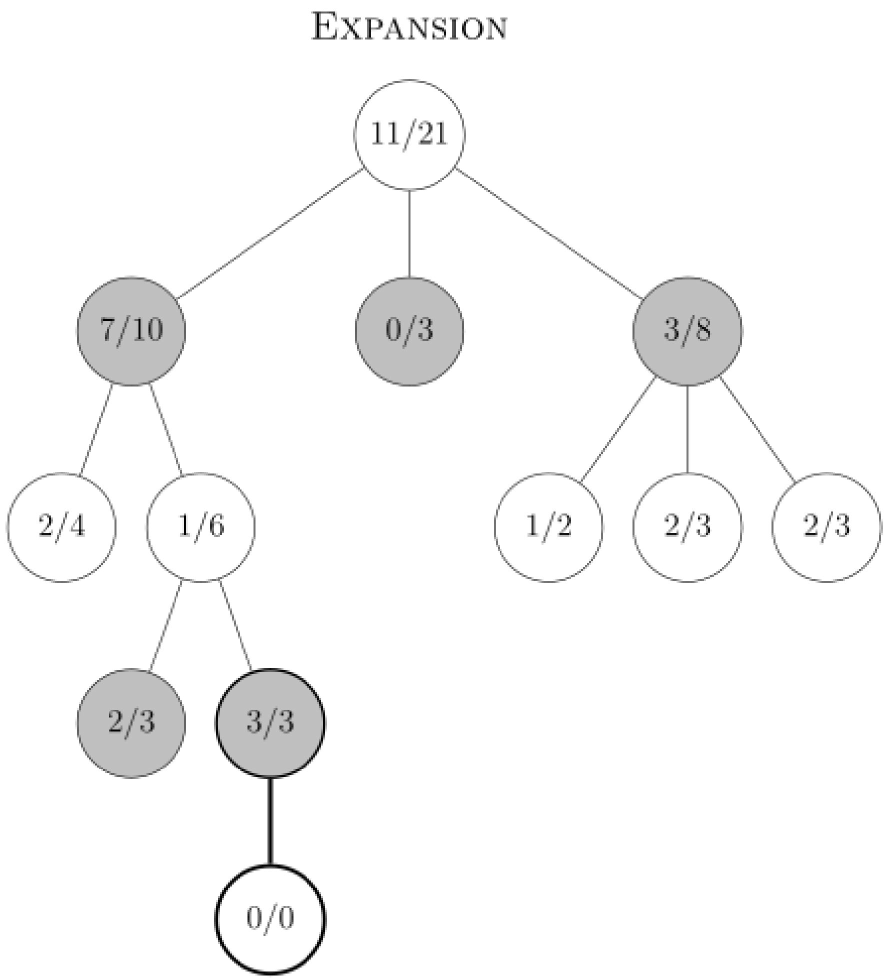


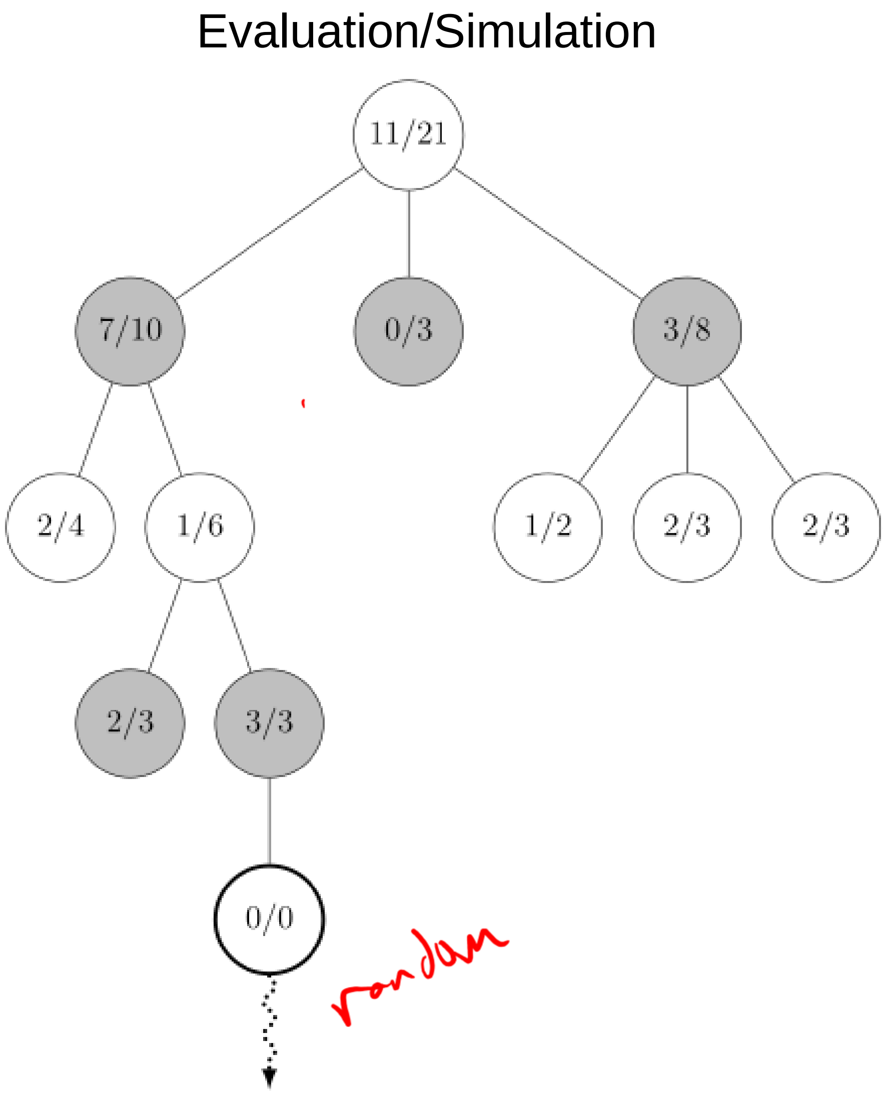
Expansion



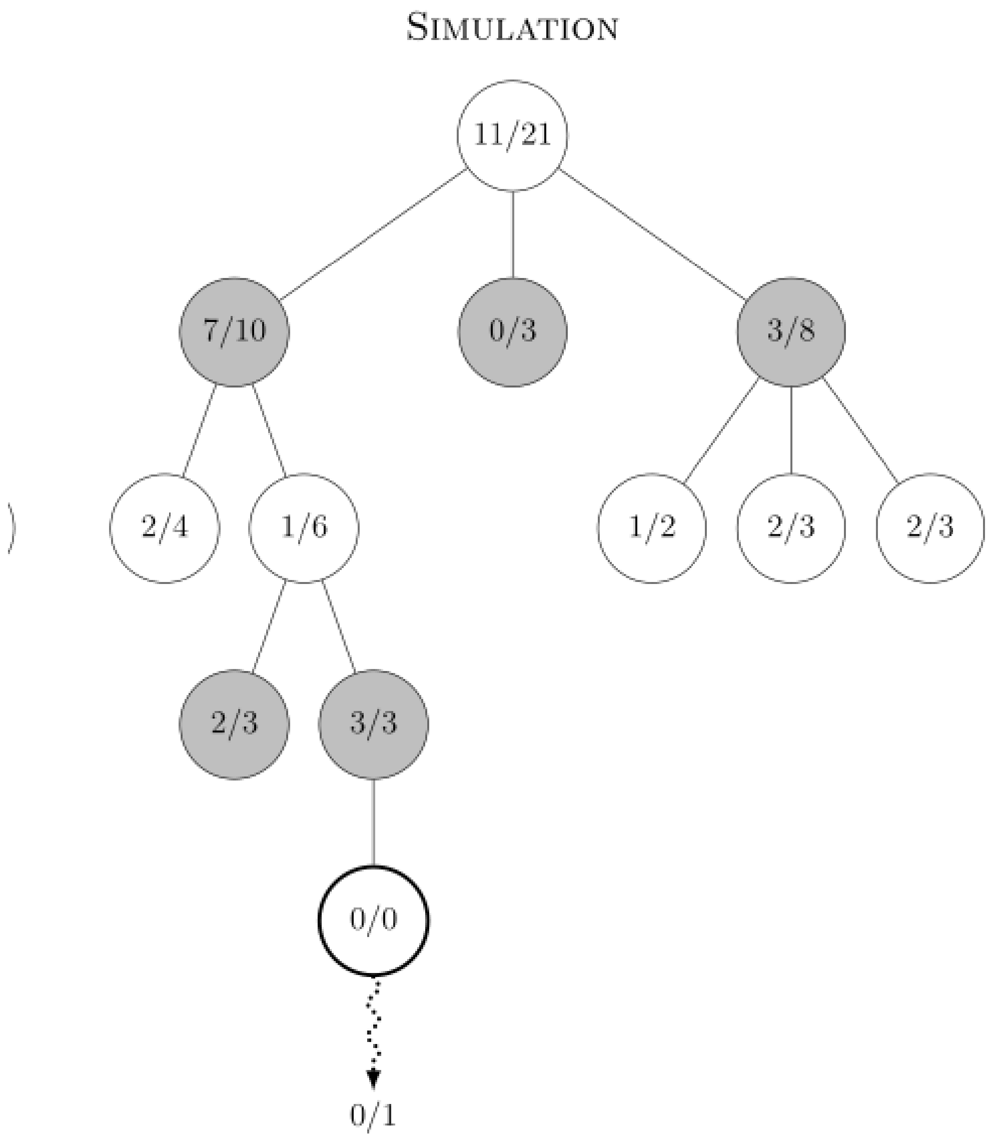
Uses some kind of exploration function to select.
Based on empirical value estimate + exploration bonus
based on visit counts (optimism in the face of uncertainty).

Example

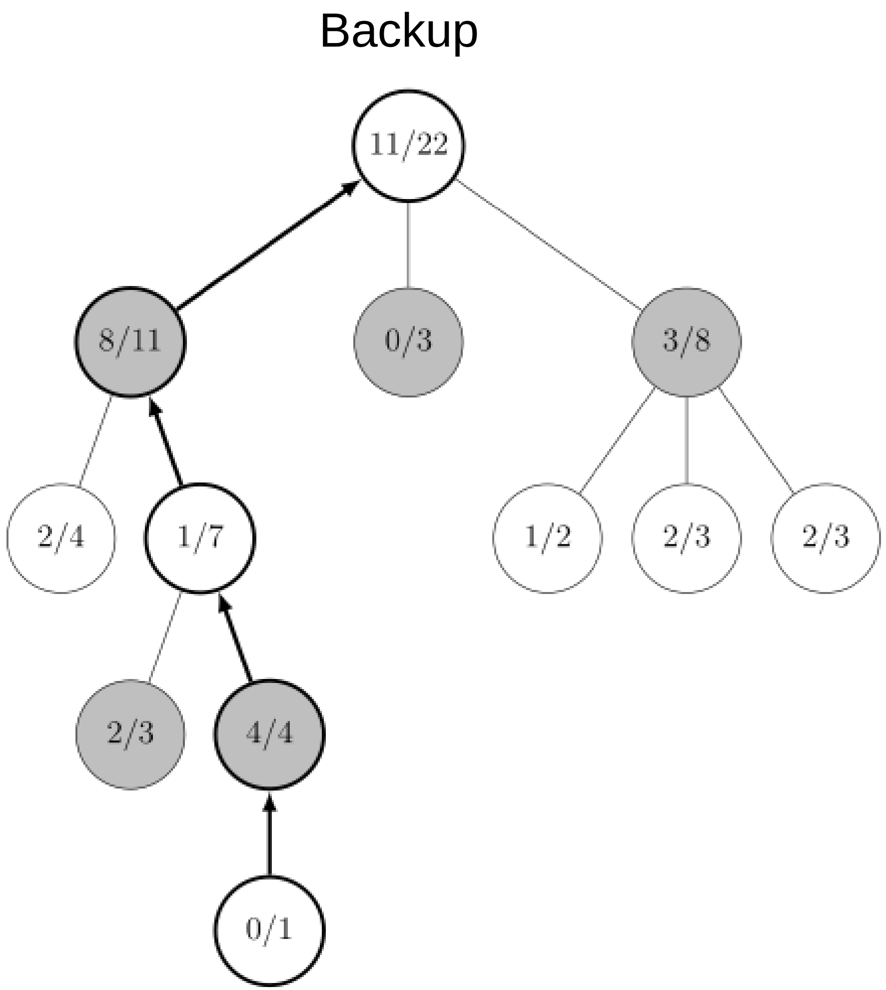

Expansion
11/21
7/10
0/3
3/8
2/4
1/6
1/2
2/3
2/3
2/3
3/3
0/0


Evaluation/Simulation
11/21
7/10
0/3
3/8
2/4
1/6
1/2
2/3
2/3
2/3
3/3
0/0
random

Example



Simulation
11/21
7/10
0/3
3/8
2/4
1/6
1/2
2/3
2/3
2/3
3/3
0/0
0/1



Backup
11/22
8/11
0/3
3/8
2/4
1/7
1/2
2/3
2/3
2/3
4/4
0/1

How to scale MCTS to Go?

- Standard MCTS achieved strong amateur play but was never able to beat a Go professional.

AlphaGo has several additional bells and whistles

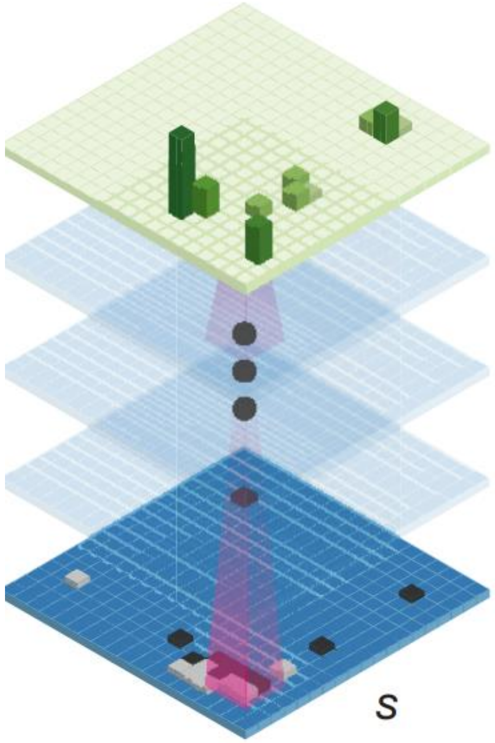
1. Imitation Learning policy learned from human gameplay
2. Fast rollout policy to sample actions in MCTS
3. RL policy that improves on Imitation Learning policy
4. Value function trained to predict value of RL policy during self-play


ALPHAGO

Supervised/Imitation Learning

- Maximize likelihood of human actions given game state

$$p_\sigma(a_h|s)$$



- Trained on 30 million Go games scraped from the internet.
- Network outputs a softmax distribution over all possible moves.
- Update σ to maximize $\log p_\sigma(a_h|s)$
- Standard classification problem

Feature Engineering

- Lots more than just where the black and white stones are:

Extended Data Table 2 | Input features for neural networks

Feature	# of planes	Description
Stone colour	3	Player stone / opponent stone / empty
Ones	1	A constant plane filled with 1
Turns since	8	How many turns since a move was played
Liberties	8	Number of liberties (empty adjacent points)
Capture size	8	How many opponent stones would be captured
Self-atari size	8	How many of own stones would be captured
Liberties after move	8	Number of liberties after this move is played
Ladder capture	1	Whether a move at this point is a successful ladder capture
Ladder escape	1	Whether a move at this point is a successful ladder escape
Sensibleness	1	Whether a move is legal and does not fill its own eyes
Zeros	1	A constant plane filled with 0
Player color	1	Whether current player is black

Feature planes used by the policy network (all but last feature) and value network (all features).

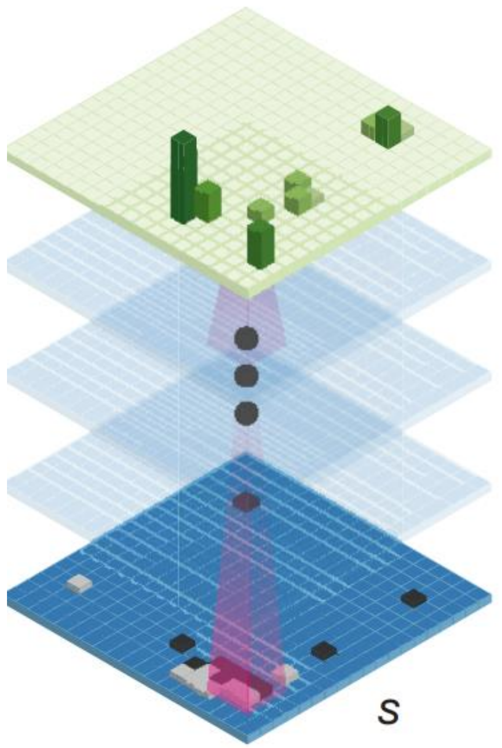
55.7% accuracy with just stone colors.

57% accuracy with all features. Leads to much stronger play.

Supervised/Imitation Learning

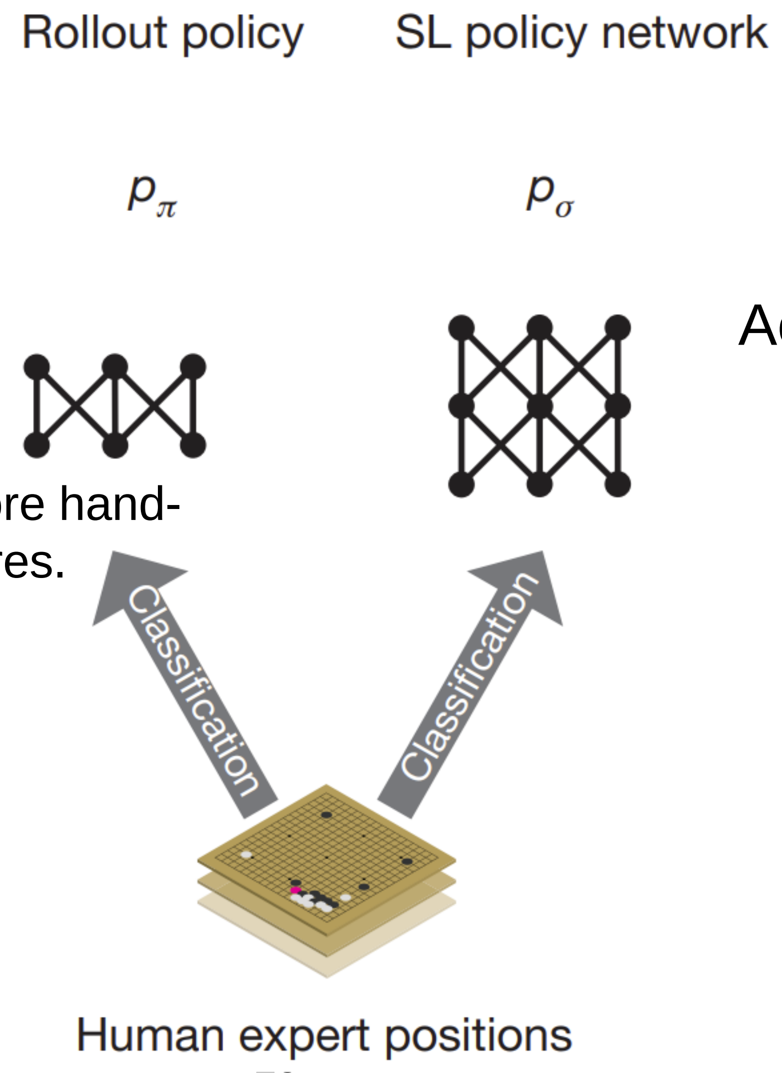
- Maximize likelihood of human actions given game state

$$p_\sigma(a_h|s)$$



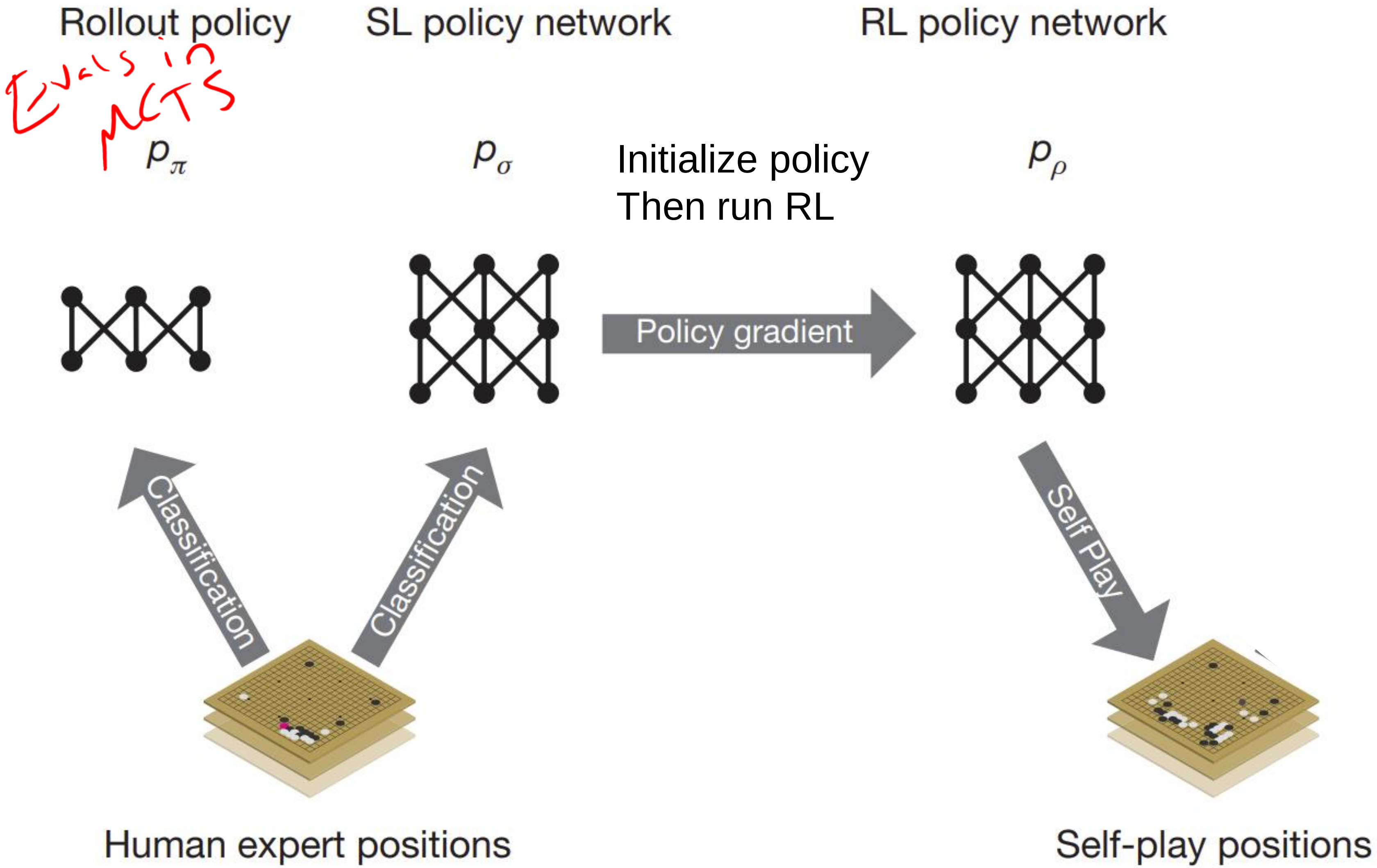
24%
Accuracy
$2\mu s$

Much smaller. More hand-engineered features.



57%
Accuracy
$3ms$

Policy Gradient Reinforcement Learning



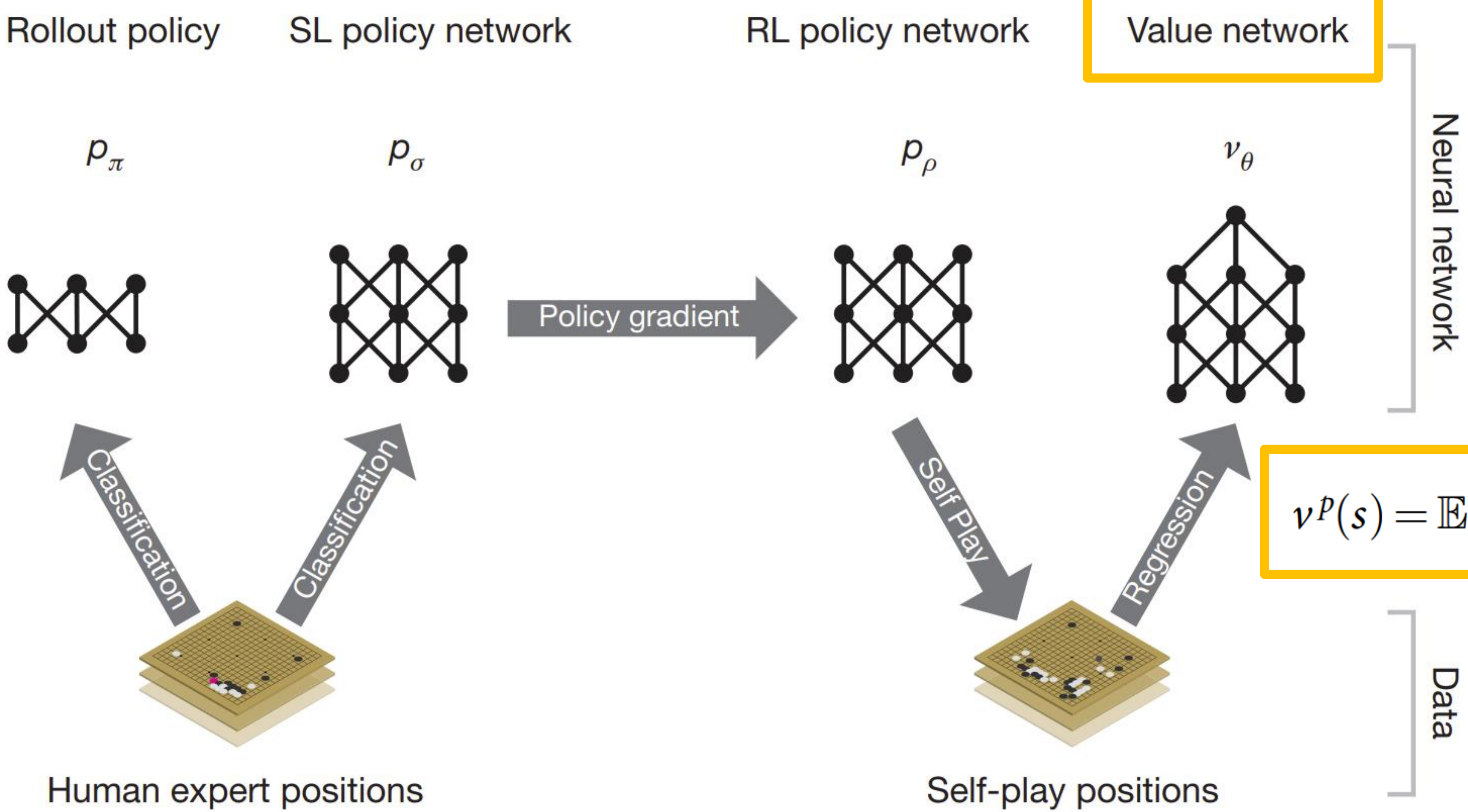
RL Policy Gradient Algorithm

- Start with pretrained imitation learning policy
- Pick random previous version of RL policy as opponent
- Run Policy Gradient RL with r_{end}^{i} =+1 if win, -1 if lose

$$\rho_{k+1} \leftarrow \rho_k + \alpha \frac{1}{n} \sum_{i=1}^{n} \sum_{t=1}^{T^i} \nabla_\rho \log p_\rho(a_t^i | s_t^i) \left(r_{end}^{i} - \underbrace{v(s_t^i)}_{\text{baseline}} \right)$$

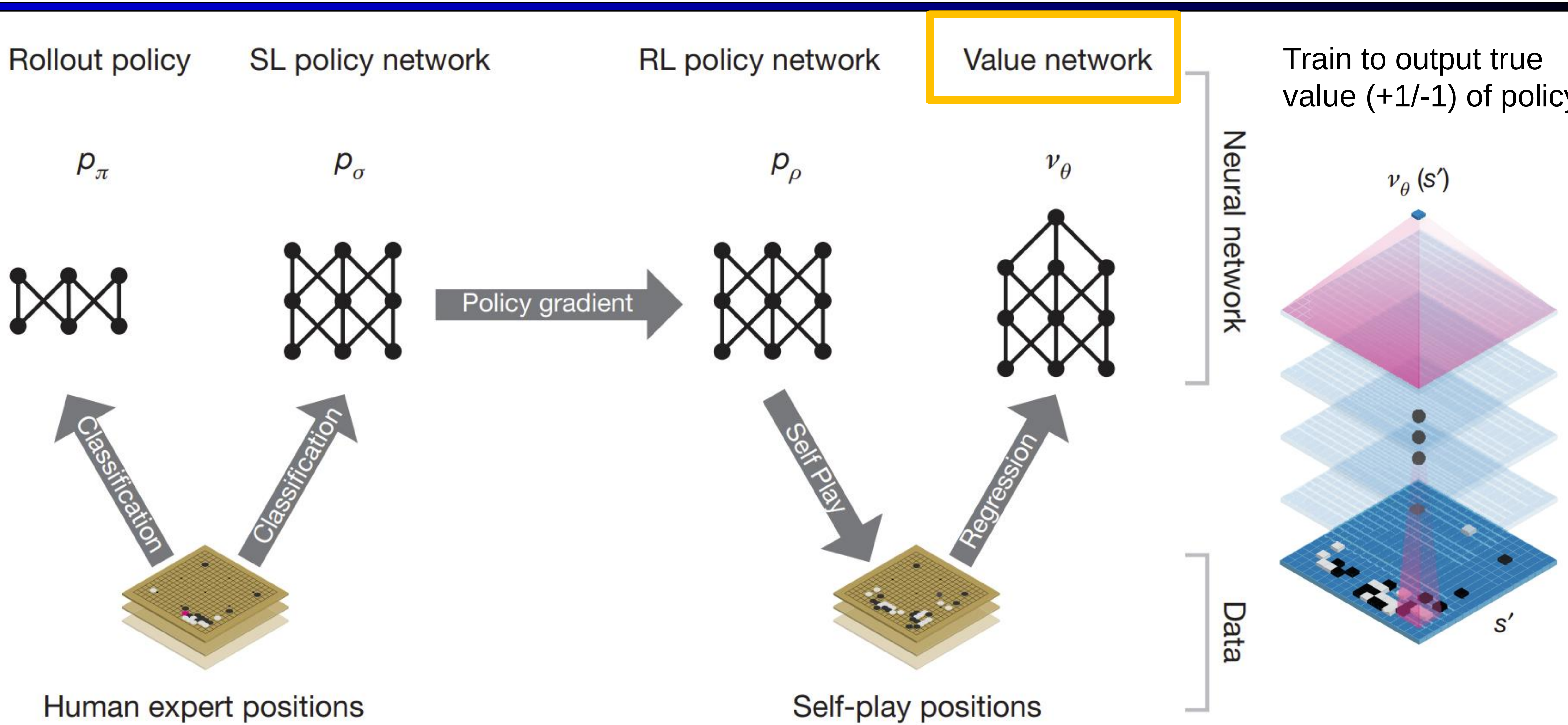
- Results:
  - 80% win rate against imitation policy
  - 85% win rate against best open source Go program (100,000 simulations per move)
  - Impressive since AlphaGo policy is not even using search!

Reinforcement Learning



$$v^p(s) = \mathbb{E}[z_t | s_t = s,\ a_{t\dots T} \sim p]$$

Reinforcement Learning



Train to output true value (+1/-1) of policy.

Direct Evaluation (Monte Carlo Rollouts)

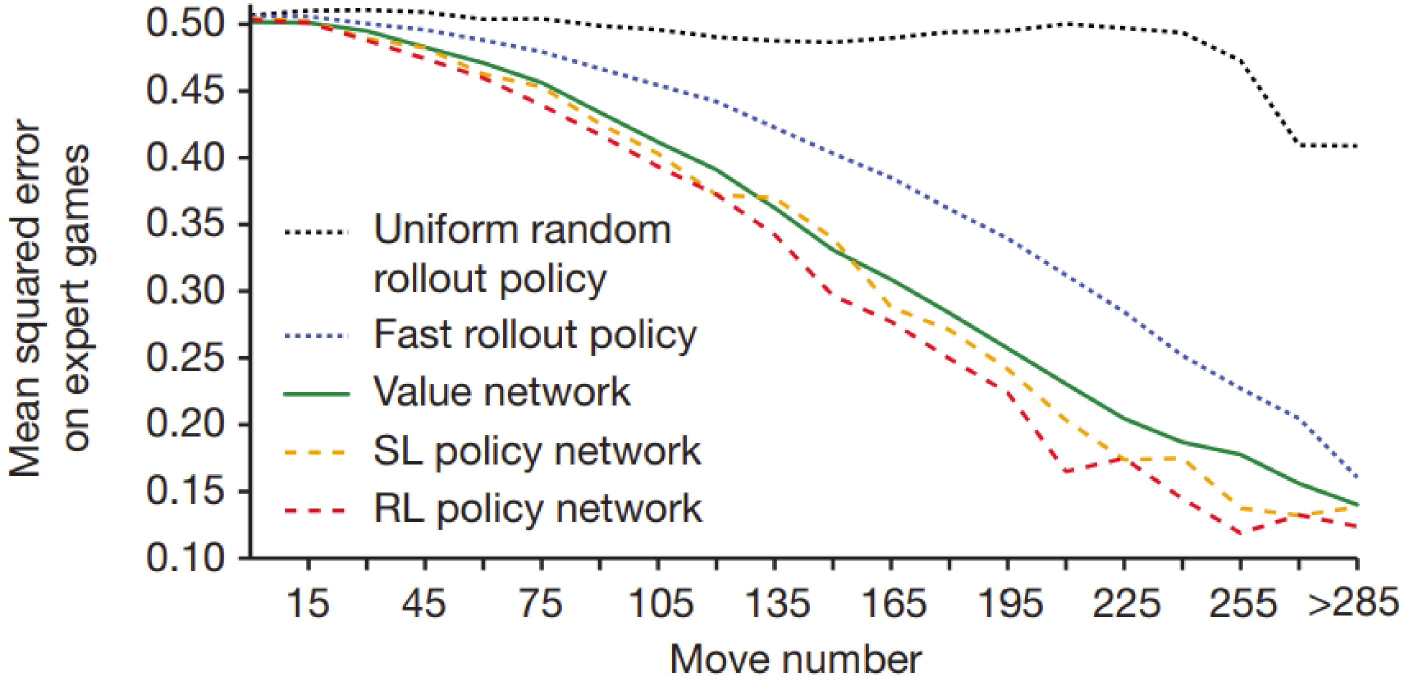
- Goal: Compute values for each state under π
- Idea: Average together observed sample values
  - Act according to π
  - Every time you visit a state, write down what the sum of discounted rewards turned out to be
  - Average those samples
- This is called direct evaluation or Monte Carlo evaluation

$$V^{\pi}(s) = E_{\pi}\left[\sum_{t=0}^{T} \gamma^t r_t\right] \approx \frac{1}{N}\sum_{i=1}^{N}\sum_{t=0}^{T} \gamma^t r_t$$

Learning a Value Network

- Supervised Learning
  - Given state *s*
  - Train V(s) to match true reward (+1/-1) at end of game (MSE loss).
  - Same target for all states in a game.
  - Uses self-play to generate tons of games and samples states to avoid overfitting by simply memorizing games.

Evaluation of board positions (predicting win/loss) using value function vs. Monte Carlo Rollouts with different policies.



Value network can evaluate board positions as well as running Monte Carlo rollouts using SL or RL policy but using 15,000 times less compute!

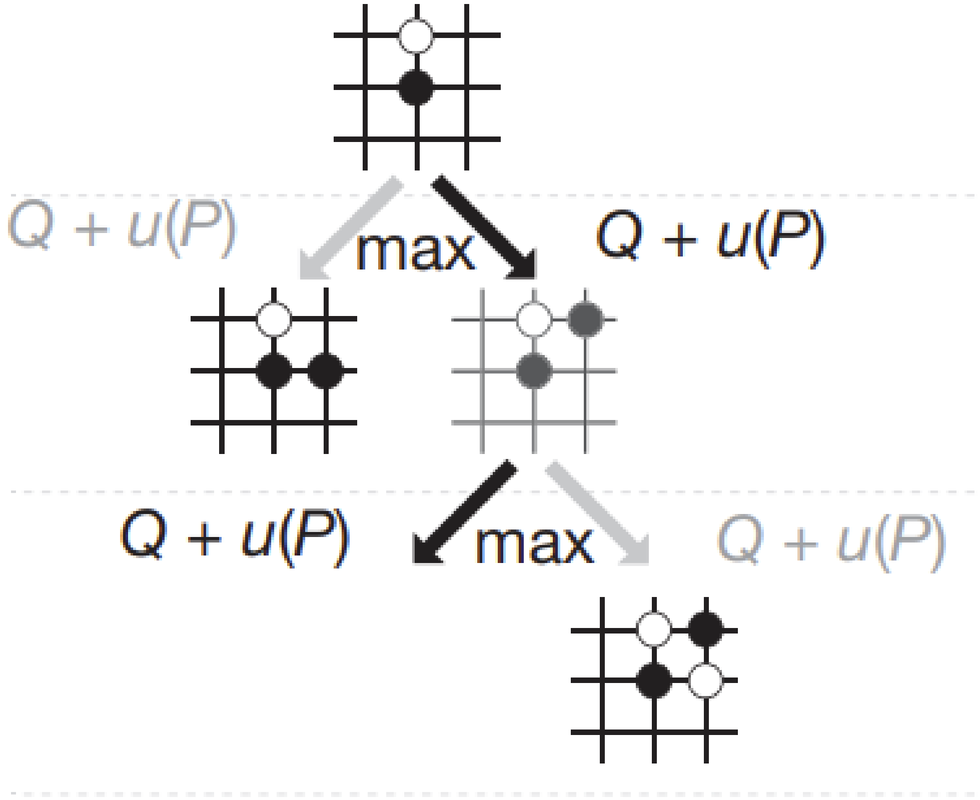
Fast Lookahead Search via MCTS

- Monte Carlo Tree Search to select actions via lookahead search
  - Supervised Learning (SL) policy predicts probability for each legal action
  - Value function is used to predict win/loss from any given state in tree
  - Fast rollout policy (baby version of SL policy) is used for fast random rollouts to get a second opinion of value of a state.

Fast Lookahead Search via MCTS

- Selection/Expansion
  - Each edge of search tree stores
    - Action value Q(s,a)
    - Visit count N(s,a)
    - Prior probability P(s,a)
  - Action selection based on value and exploration bonus

$$a_t = \underset{a}{\operatorname{argmax}}\left(Q(s_t, a) + u(s_t, a)\right)$$



$$u(s,a) \propto \frac{P(s,a)}{1+N(s,a)}$$

When expanding a leaf node, Supervised Learning (SL) policy predicts probability for each legal action and stores these as P(s,a)

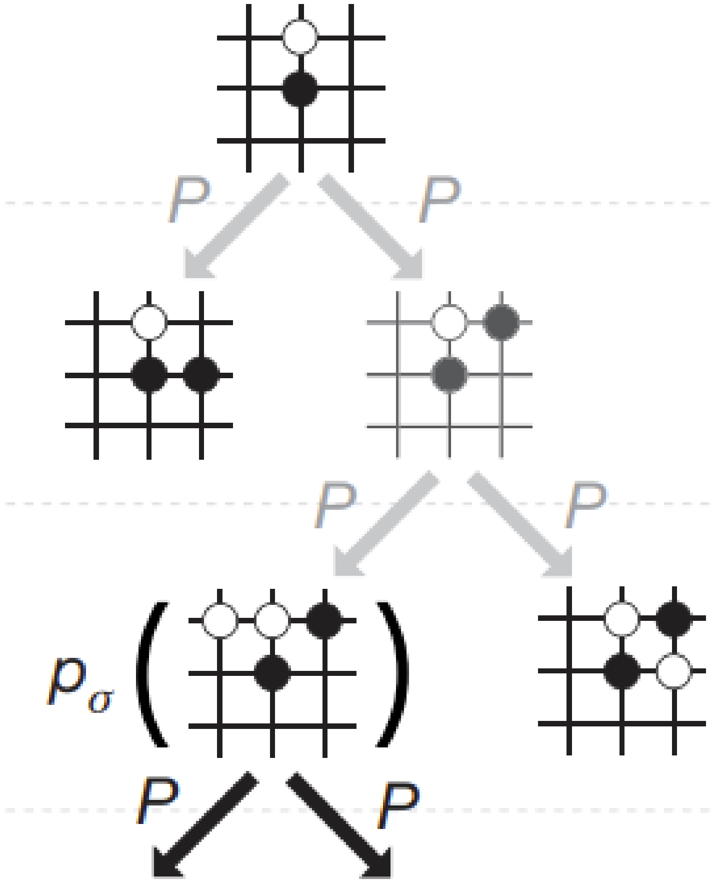
Fast Lookahead Search via MCTS

- Selection/Expansion
  - Each edge of search tree stores
    - Action value Q(s,a)
    - Visit count N(s,a)
    - Prior probability P(s,a)
  - Action selection based on value and exploration bonus

$$a_t = \underset{a}{\operatorname{argmax}}(Q(s_t, a) + u(s_t, a))$$

$$u(s, a) \propto \frac{P(s, a)}{1 + N(s, a)}$$

When expanding a leaf node, Supervised Learning (SL) policy predicts probability for each legal action and stores these as P(s,a)

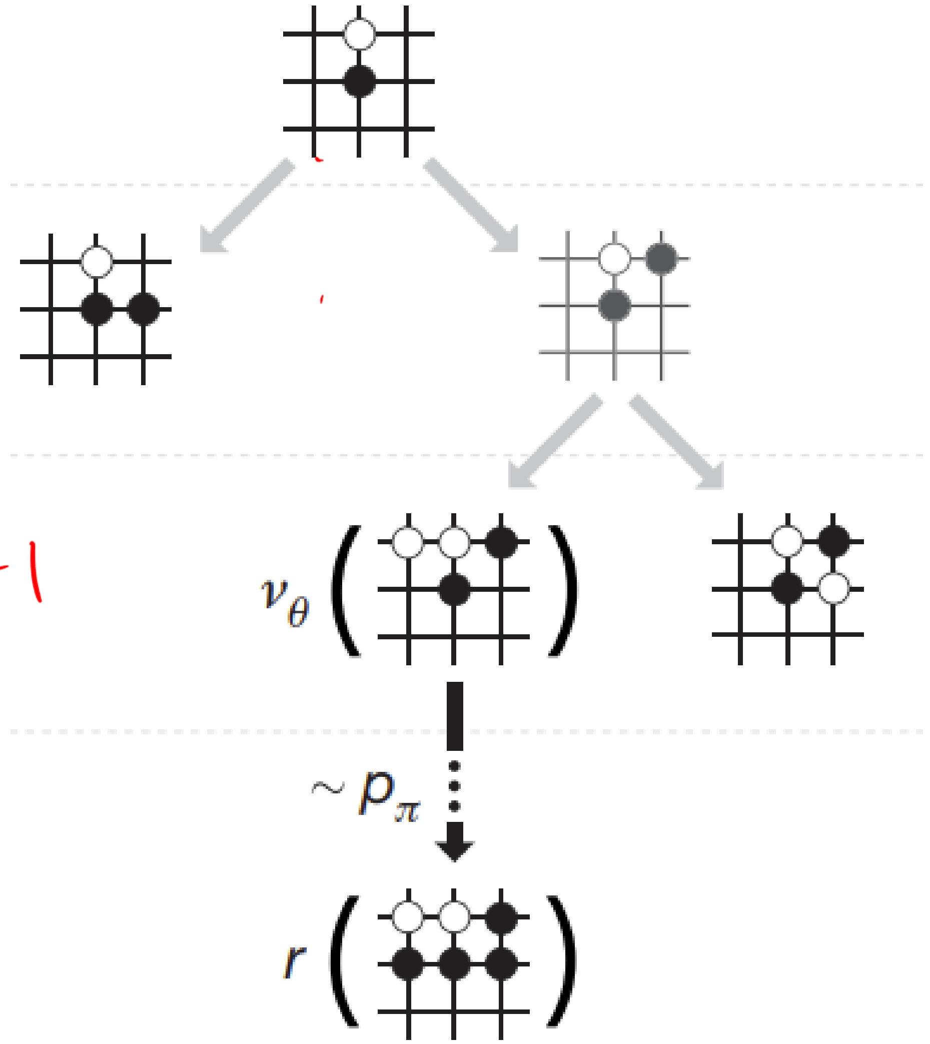


Fast Lookahead Search via MCTS

- Evaluation
  - After expanding a leaf node get two opinions on the value of the state
    - Evaluate with value function v_θ
      - Returns predicted probability of win
    - Evaluate with fast rollout policy p_π
      - Play against itself for one game
      - Super fast. Trained on human games.
    - Combine to estimate value

+1/-1

$$V(s_L) = (1-\lambda)v_\theta(s_L) + \lambda z_L$$



Fast Lookahead Search via MCTS

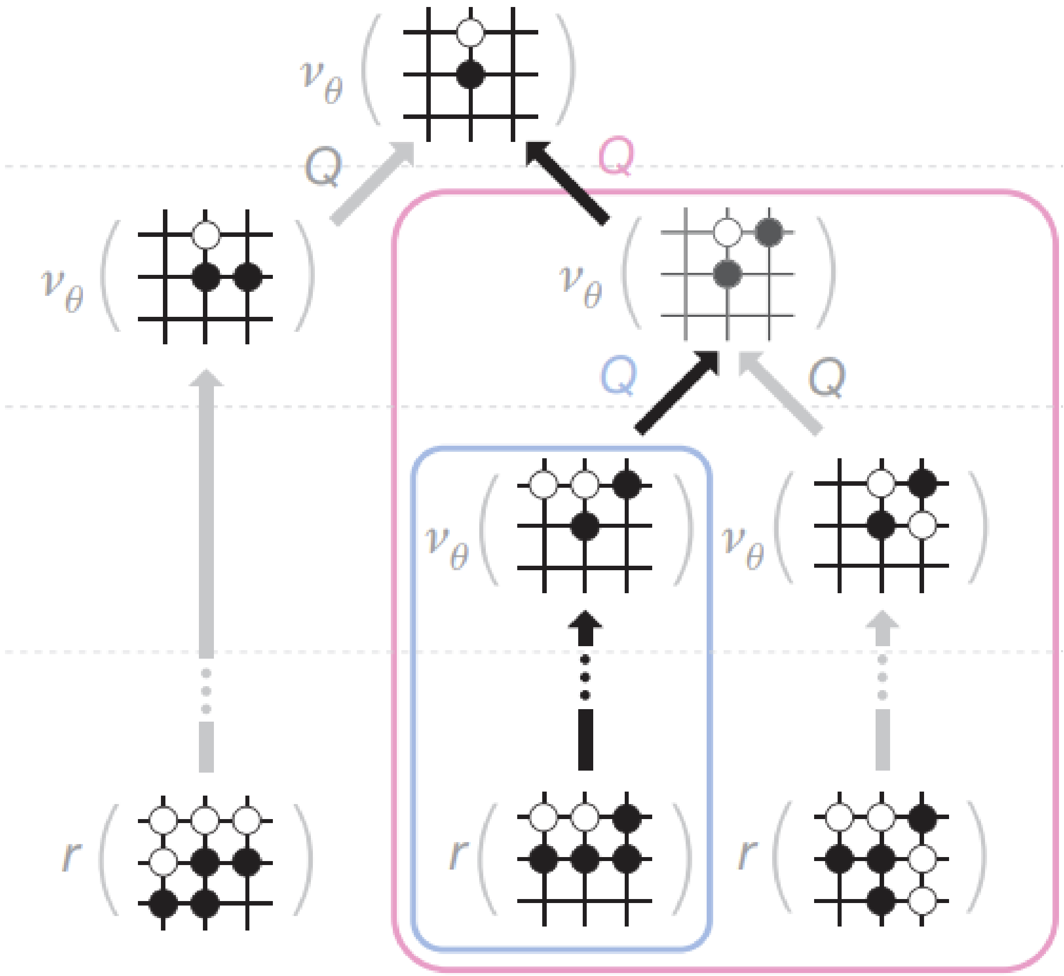
- Backup
  - Update action values and visit counts of all traversed edges.

$$N(s,a) = \sum_{i=1}^{n} 1(s,a,i)$$

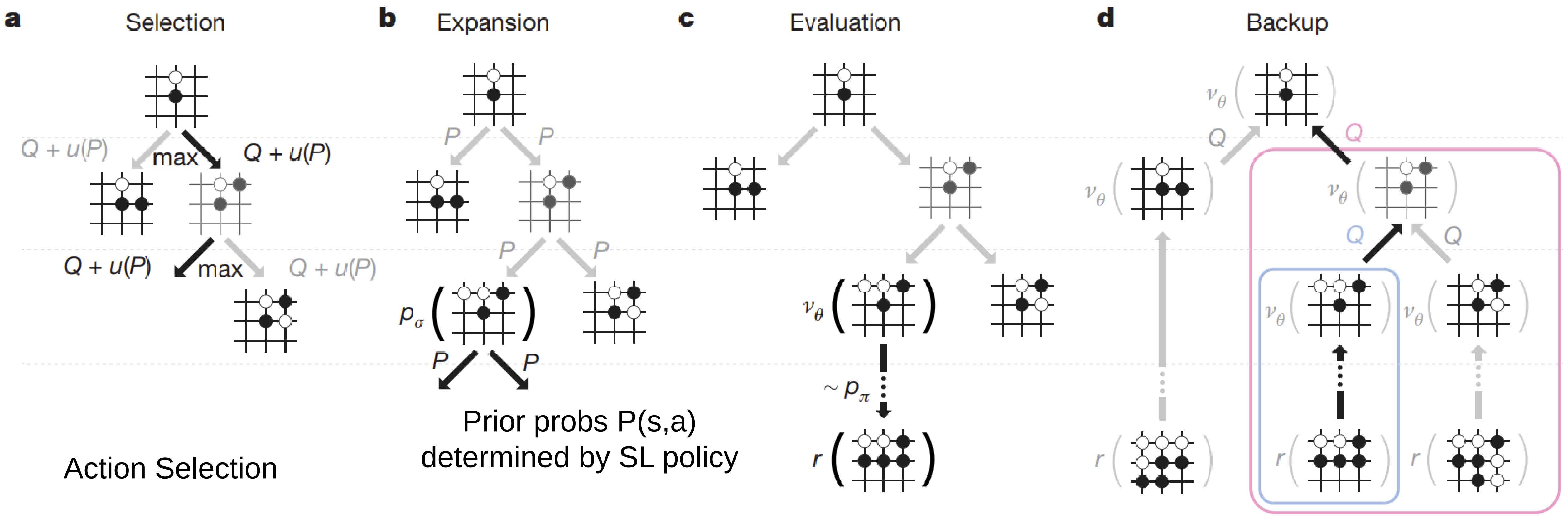
Number of times edge (s,a) was selected.

$$Q(s,a) = \frac{1}{N(s,a)} \sum_{i=1}^{n} 1(s,a,i) V(s_L^i)$$

Mean evaluation of all simulations passing though edge (s,a).



AlphaGo MCTS Overview



Action Selection

Prior probs P(s,a) determined by SL policy

$$a_t = \operatorname{argmax}(Q(s_t, a) + u(s_t, a))$$

$$u(s,a) \propto \frac{P(s,a)}{1+N(s,a)}$$

$$V(s_L) = (1-\lambda)v_\theta(s_L) + \lambda z_L$$

$$N(s,a) = \sum_{i=1}^{n} 1(s,a,i)$$

$$Q(s,a) = \frac{1}{N(s,a)} \sum_{i=1}^{n} 1(s,a,i)V(s_L^i)$$

Where is the RL policy??


ALPHAGO


ALPHAGO