Monte Carlo Tree Search

Keith L. Downing

January 11, 2021

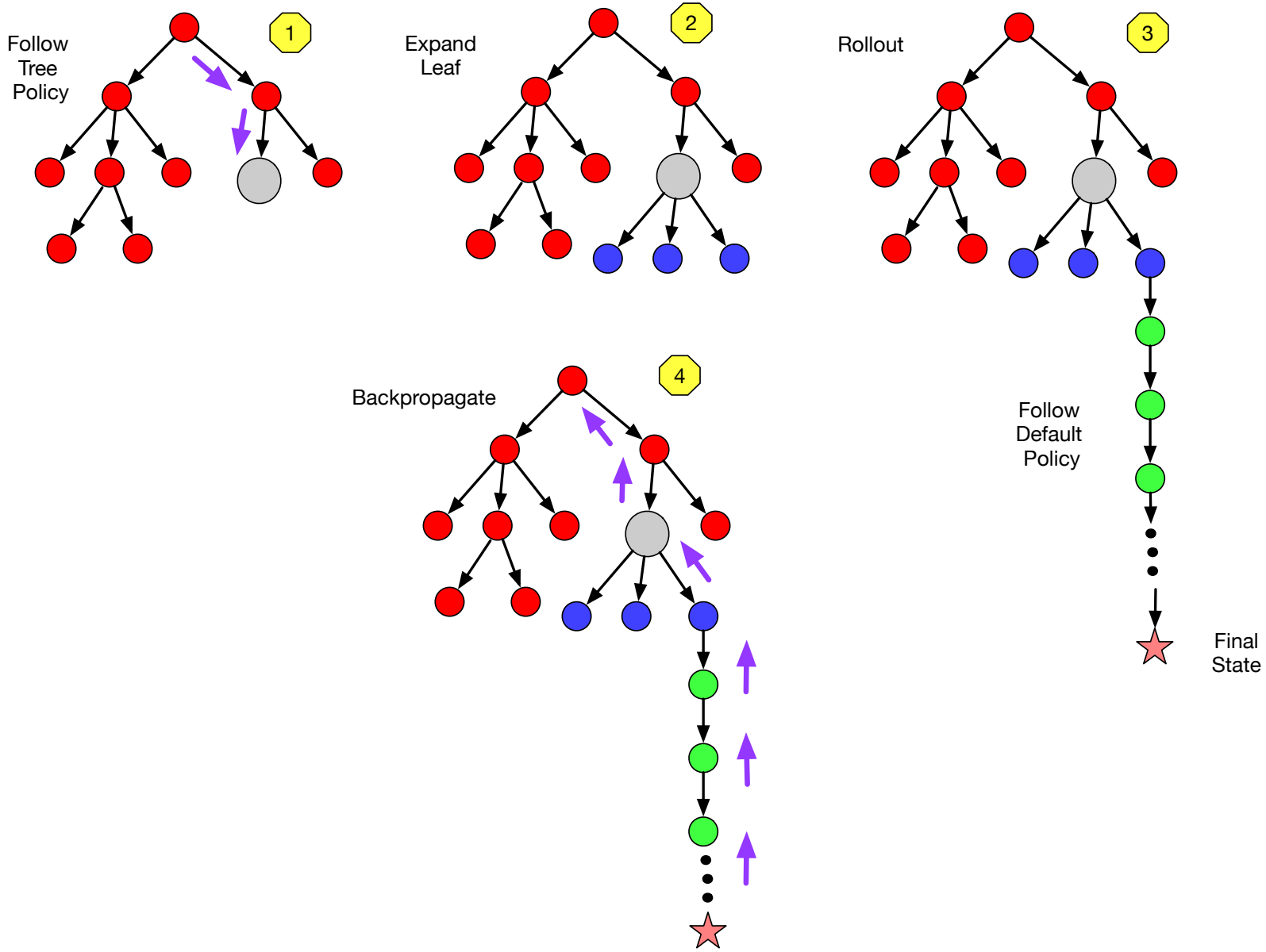
Monte Carlo (MC) Methods

- Methods that rely on random sampling to attain quantitative results and predictions.
- To determine whether or not a coin or die is fair, toss it 10,000 times and record the results.
- **Rollout Simulations** - In search, to estimate the value of state (S), instead of applying a heuristic directly, use a policy to play out (roll out) from S to a final state (F) . The evaluation of F (and possibly other information along the path from S to F) can then be used to update the value of S.
- If the policy (or the simulation environment) is non-deterministic, then do **many** (hundreds or thousands) of rollouts from S to get an accurate estimate of its value.

Monte Carlo Tree Search (MCTS)

- Similar to MiniMax: Generate a huge tree (but traverse it **many** times) as basis for selecting **one** move (from the root). Then retain the subtree rooted at the next state.
- Most AI GO players use MCTS.
- Originally designed for stochastic games: backgammon, poker and scrabble.
- Why use it for a deterministic game? The search space is so large, and most evaluation functions (i.e. heuristics) are so bad, that you need to try many alternate futures to assess the current state. For a deterministic game, the policy needs some element of stochasticity; otherwise, all rollouts would produce the same result.
- AlphaGo combines MCTS with RL actors and critics produced by Deep Learning.

MCTS Dynamics: Single Pass



MCTS Single Pass

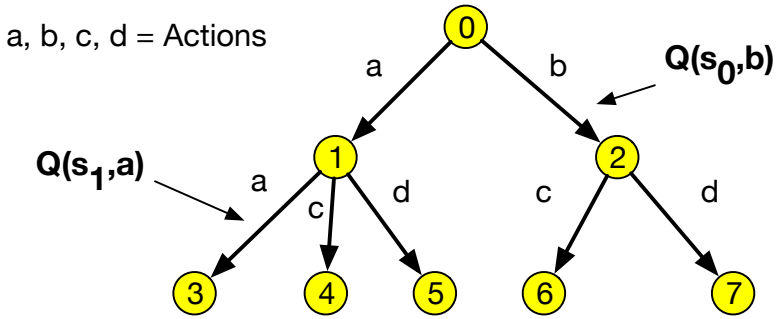
Each node in the tree represents a state in search space.

1. From the root state (R) use the tree policy to choose the next pre-existing child node.
2. When a leaf node (not necessarily a final state) is reached, apply domain-dependent operators to produce successor (child) states.
3. Pick one of the new successors (S) as a starting point and use the default policy to perform a single-track rollout simulation from S to a final state (F).
4. Propagate information about F (such as the winner), along the entire path from F back to S and then back to R. That info updates node statistics that influence the tree policy.

Many Variations

- Tree search may go directly from tree-policy moves to a rollout, without expanding the leaf node. Expansions may only occur intermittently, after several full passes.
- Nodes generated during the rollout may or may not be saved as part of the full tree.
- Tree policies vary considerably in the degree to which they explore (i.e., visit infrequently visited child nodes) and exploit (i.e. visit highly-rated child nodes).
- In many cases, the default policy is the one that our system is trying to improve (via learning). After many rounds of rollouts and information-backpropagation, the tree statistics are used to update that policy.

Q Values



- Ultimate goal of MCTS: **produce realistic Q(s,a) values**
- Q(s,a) = value (expected final result) of doing action a from state s.
- These appear on the **edges** between tree nodes.
- Over time (many expansions, rollouts and backpropagations), these Q values approach those found by Minimax.

The Tree Policy

- Choose branch with the highest combination of Q(s,a) + exploration bonus, u(s,a):

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

- In AlphaGo:

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

- with N(s,a) = number of times branch (s,a) has been traversed, and P(s,a) = prior probability p(a | s). In AlphaGo, these P(s,a) come from the Actor-SL neural network (trained on a database of expert moves).
- Note that this favors infrequently-visited branches: it promotes exploration.
- UCT (see next slide) is a common u(s,a) metric.

Upper Confidence Bound for Trees (UCT)

- Very popular basis for the exploration bonus, u(s,a), in MCTS.

$$u(s,a) = c \times \sqrt{\frac{log(N(s))}{1 + N(s,a)}}$$

- where c = a constant (often 1), N(s,a) = number of traversals of edge (s,a), N(s) = number of visits to node s.
- Encourages exploration: u(s,a) decreases as N(s,a) increases.

Bandit based Monte-Carlo Planning, Kocsis and Szepervari (2006)

Min-Max and Turn-Taking

- For two-person games (the standard type for MCTS), there is still a min-max component.
- Assume that player 1 is making a move **from** the state at the tree root and that players 1 and 2 alternate throughout the game.
- Assume that all evaluations (of final or intermediate states) are from the perspective of player 1. So if winning state for player 1 has a score of **+1**, then a winning state for player 2 should have a score of **-1**.
- These scores are backed up throughout the tree, affecting Q values on the branches.
- Thus, player 1 will normally choose branches with **high** Q values, while player 2 will choose those with **low** Q values.
- Taking the exploration bonus, u(s,a) , into account, the greedy best action choice for player 1 is:

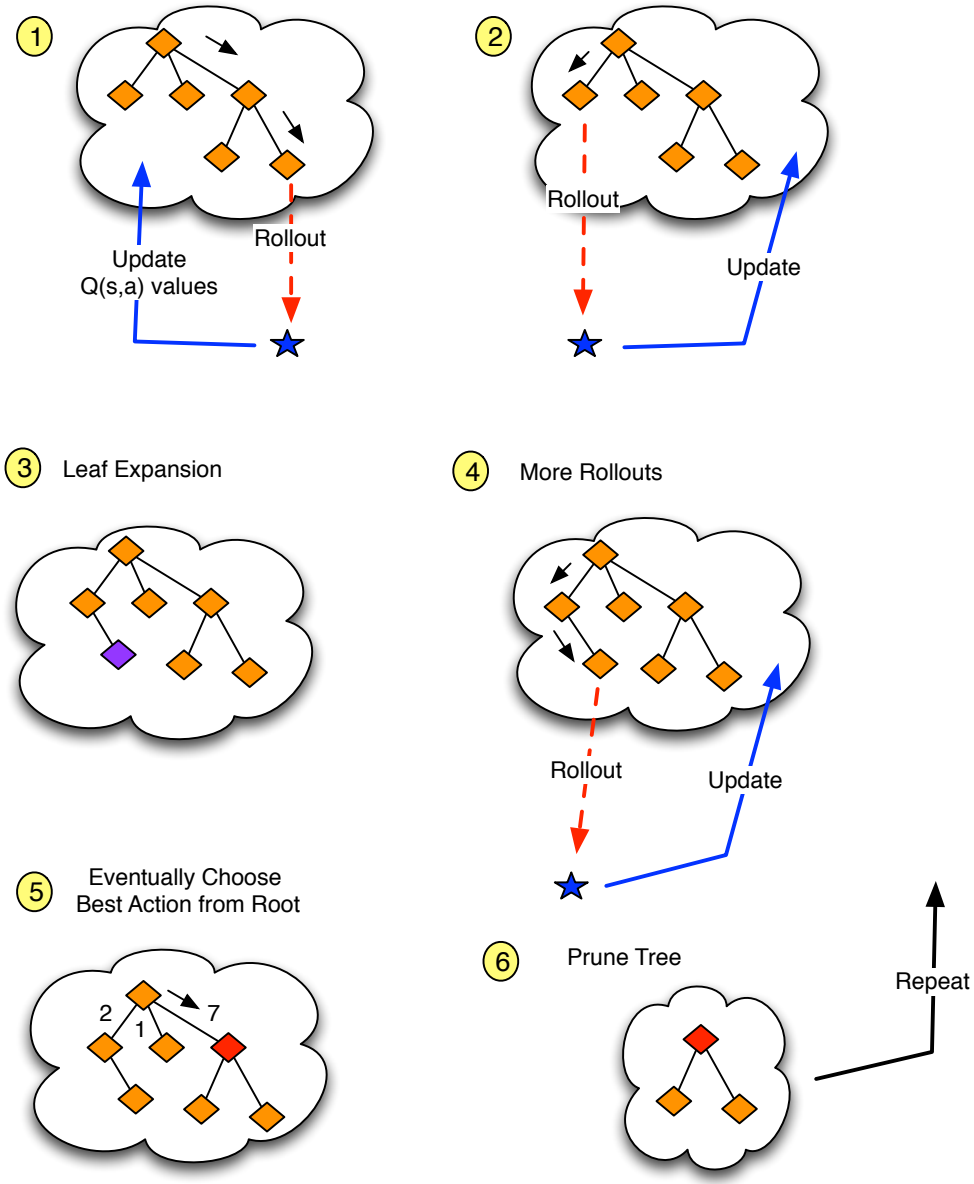
$$a_t = \underset{a}{\operatorname{argmax}}\, Q(s_t, a) + u(s_t, a)$$

- while the greedy choice for player 2 is:

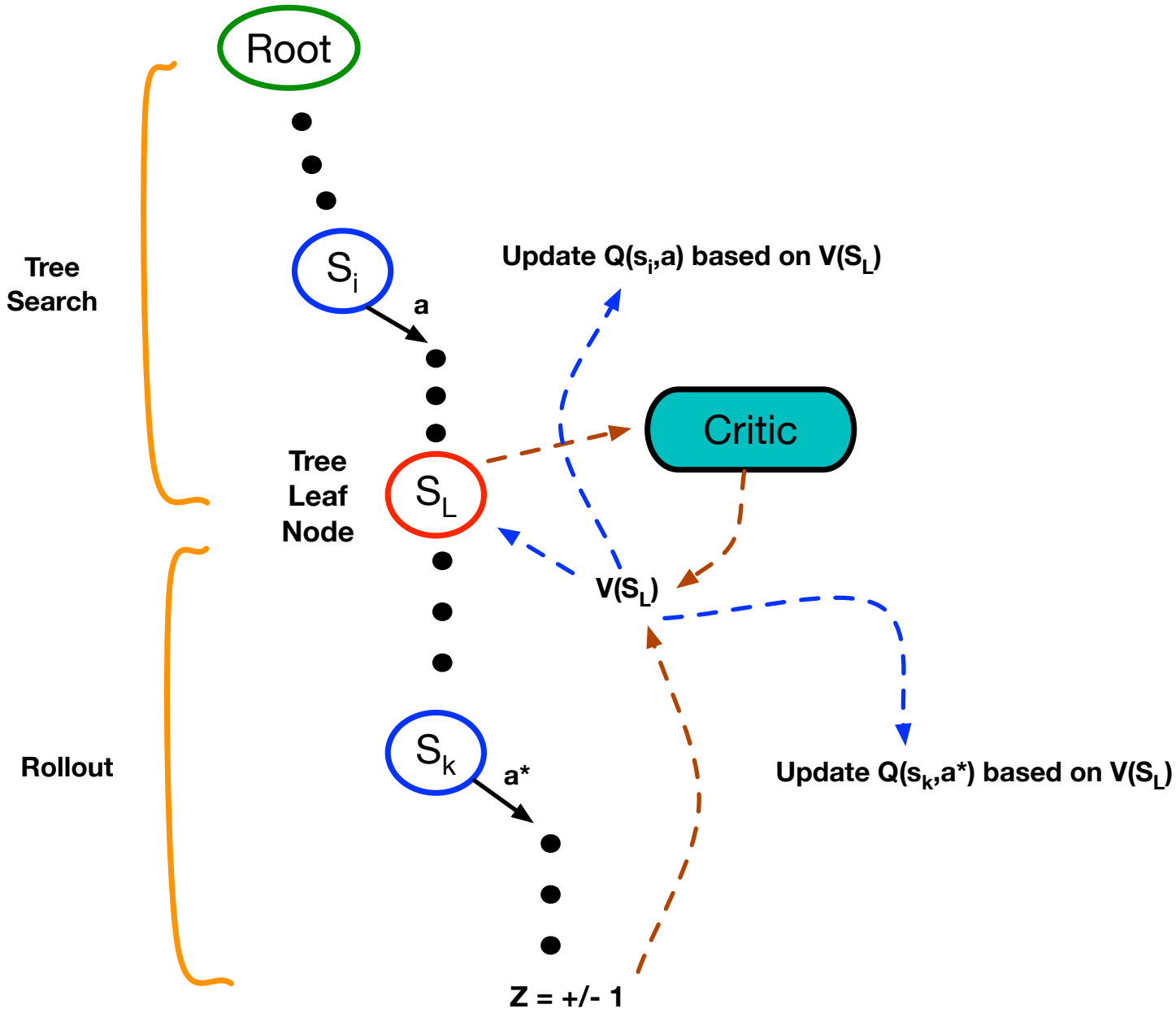
$$a_t = \underset{a}{\operatorname{argmin}}\, Q(s_t, a) - u(s_t, a)$$

- Each search state should include the current player moving **from** it.

MCTS Dynamics: Multiple Passes



Evaluating Leaves and Updating Q(s,a) in AlphaGo



Evaluating Leaves and Updating Q(s,a) in AlphaGo

1. Send the leaf state, s_L as input to the Critic-RL neural net to produce evaluation $v_\theta(s_L)$.
2. Get the final game result (z_L) produced by a rollout simulation from s_L, where $z_L \in \{-1, 0, +1\}$.
3. Combine the two using a mixing parameter, λ:

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

4. After each tree search + rollout, update counts and Q values so that:

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

where d(s,a,i) = 1 if branch (s,a) was traversed on the ith tree-search or ith rollout; otherwise, it is 0.

5. and:

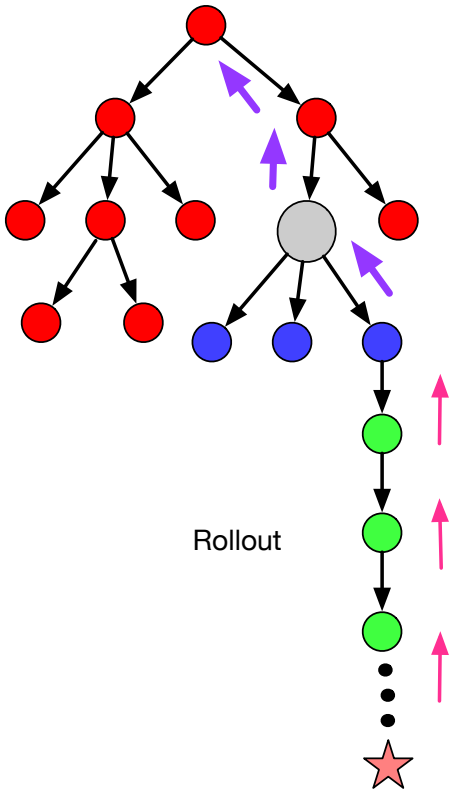
$$Q(s,a) = \frac{1}{N(s,a)} \sum_{i=1}^{n} d(s,a,i) V(s_L^i)$$

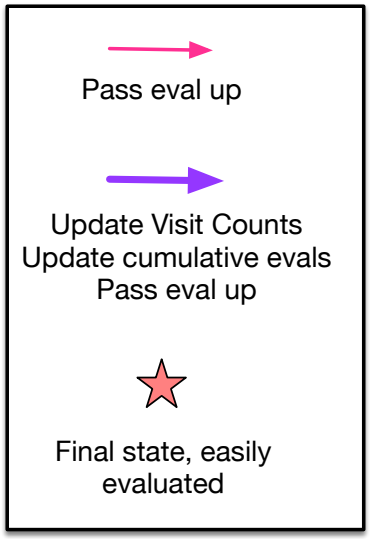
where s_L^i is the leaf node from the ith search.

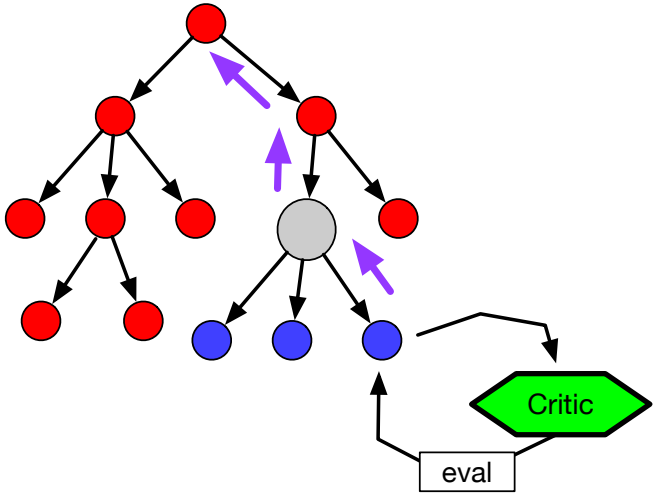
Alternative Leaf-Evaluation Protocols

- AlphaGo does both (rollout and critic evaluation) of each leaf node, and then combines the results.
- AlphaGo Zero does only the critic evaluation, so it never does rollouts. This saves resources, but early evals are poor.
- Alternatively, make a random (or stochastic, but biased by probability Ψ) choice of rollout versus critic-eval each time search reaches a leaf node.
- To account for improvements in the critic via learning, let Ψ change over time, gradually making rollouts less likely to occur.

Information Propagation in MCTS

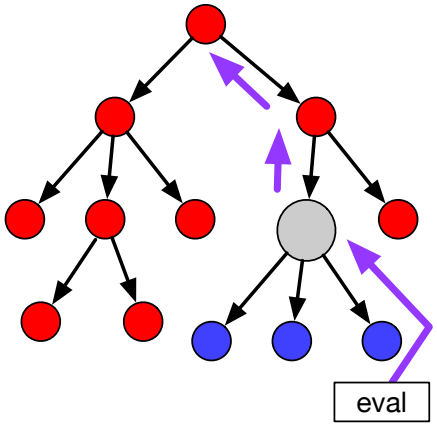


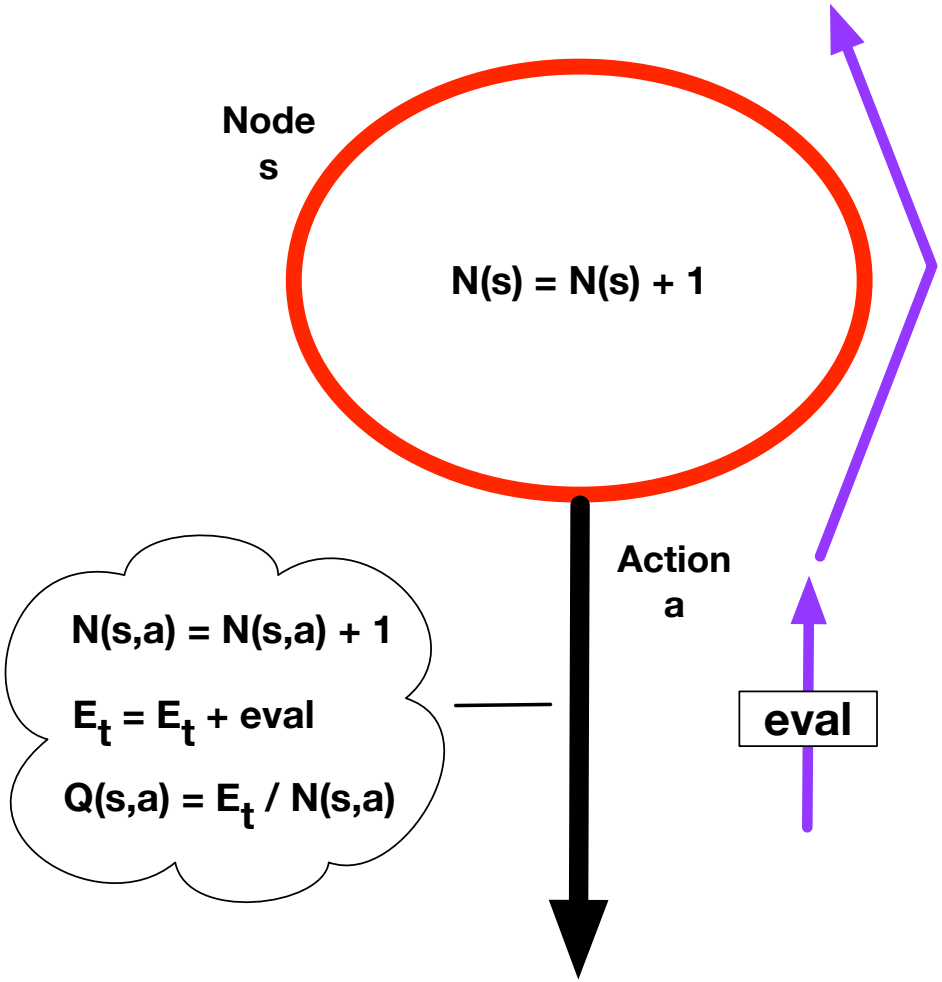




- Also called backpropagation. Don't confuse with neural net backprop!
- Pass back state evaluations, so that each node on the path can update visit counts – N(s) and N(s,a) – and rewards (basis for Q values).
- Rewards are either the final-state value (e.g. +1 or -1) found by a rollout, or the eval (e.g. +0.547) given by a critic.

Key Updates during MCTS Backpropagation





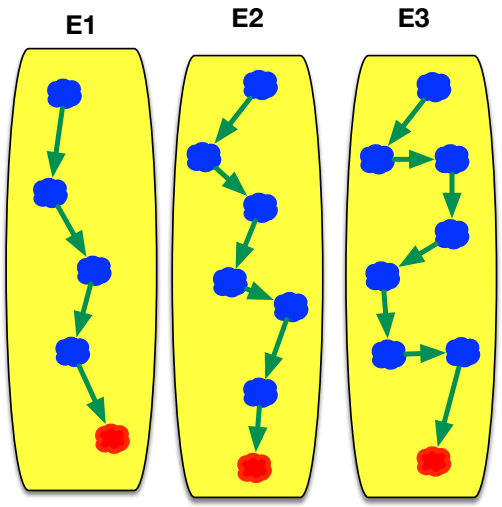
- This supports updates of u(s,a).
- Q(s,a) and u(s,a) then guide tree-policy search during the next descent (i.e. simulation)

MCTS Episodes and Actual Moves

- One MCTS episode = sequence of actual moves from start state to final state.
- Each actual move is based on many complete simulations, from tree root to a final node.
- Use the edge (from the root) with the highest visit count as the actual move.
- After many simulations, visit count is strongly correlated with Q(s,a) (and less with u(s,a)).
- The child end of that edge becomes the new tree root for the next round of simulations.

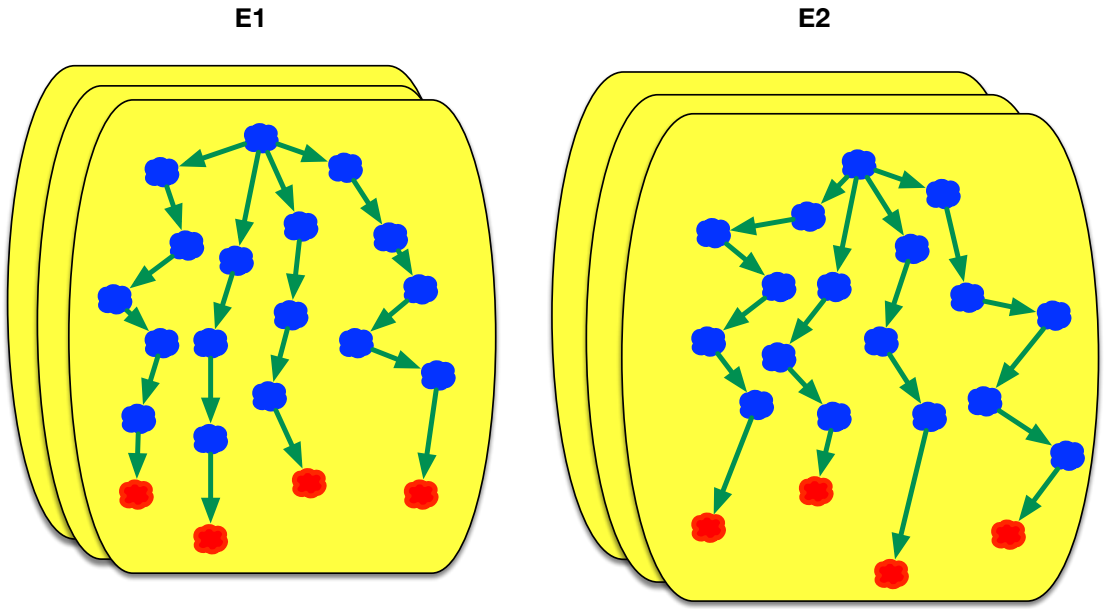
Standard MC RL versus MCTS

Monte Carlo RL



One Episode = One Simulation

Monte Carlo Tree Search



One Episode = Many Simulations, a Series of Trees

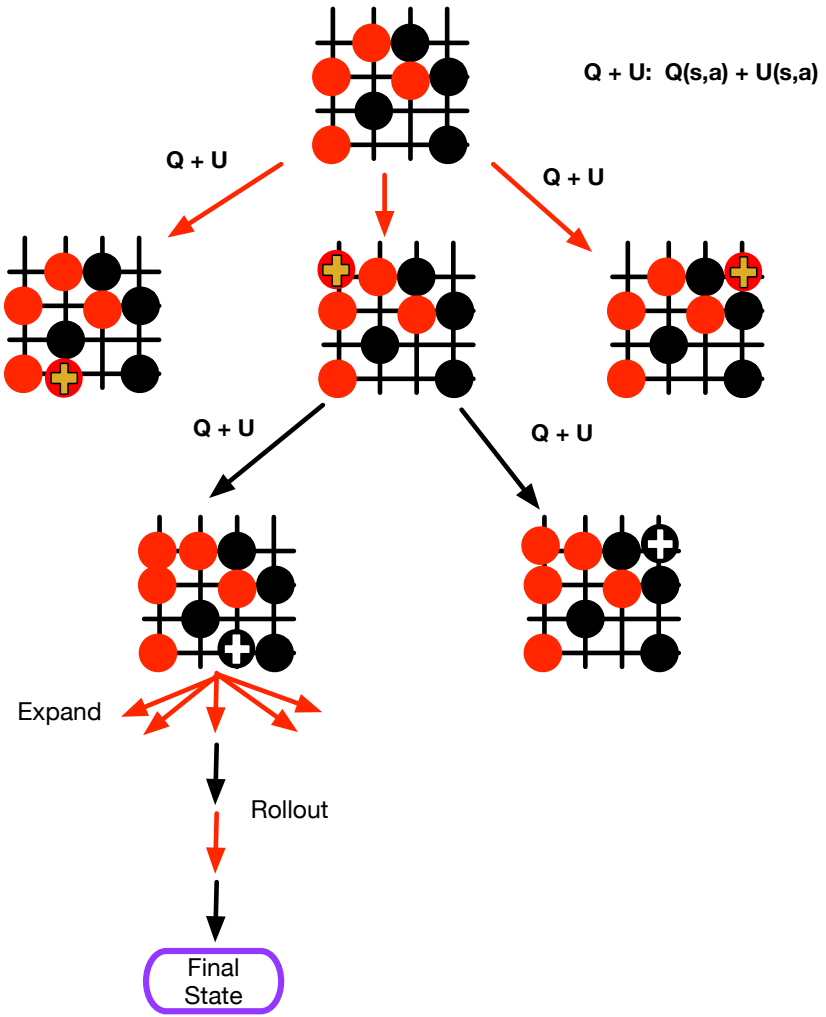
Policies in MCTS

- Tree policy - for moving from tree root to a leaf.
- Default policy - for rollout from leaf to final state. Same as behavior policy in standard RL terminology.
- Target policy (optional) - strategy being improved (learned) by repeated runs of MCTS.
- On-policy MCTS: Default policy = Target Policy
- Off-policy MCTS: Default and Target policies are different. Default is often more explorative, while Target is more greedy.

Learning in MCTS

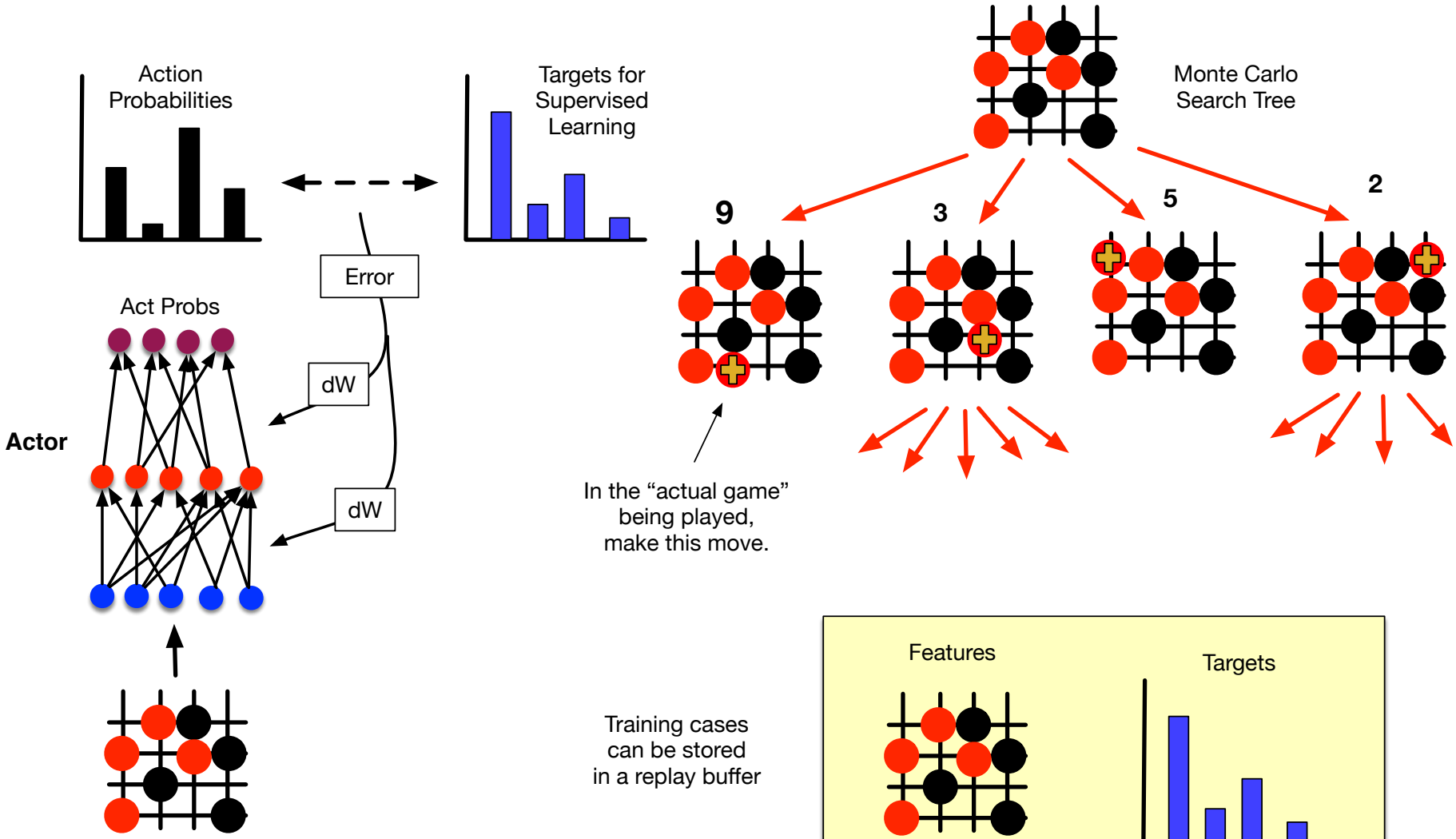
- On each simulation, MCTS updates Q(s,a) and u(s,a) values.
- Thus, tree policy learns $\rightarrow$ altered search behavior on next simulation.
- After many simulations, actual moves (based on visit counts) should be fairly intelligent ($\approx$ Minimax-based moves)
- However, when the episode (e.g. actual game) ends, all Q(s,a) and u(s,a) information is lost.
- So next episode begins from scratch; each episode is independent.
- Unless...the default (a.k.a. behavior) policy (b) changes.
- In on-policy RL, b = target policy (Π), and Π learns.
- What about off-policy MCTS?
  - Π learns, but b may not.
  - No inter-episode improvement, but Π improves.
  - MCTS explores, while Π exploits using MCTS results.

MCTS in AlphaGo

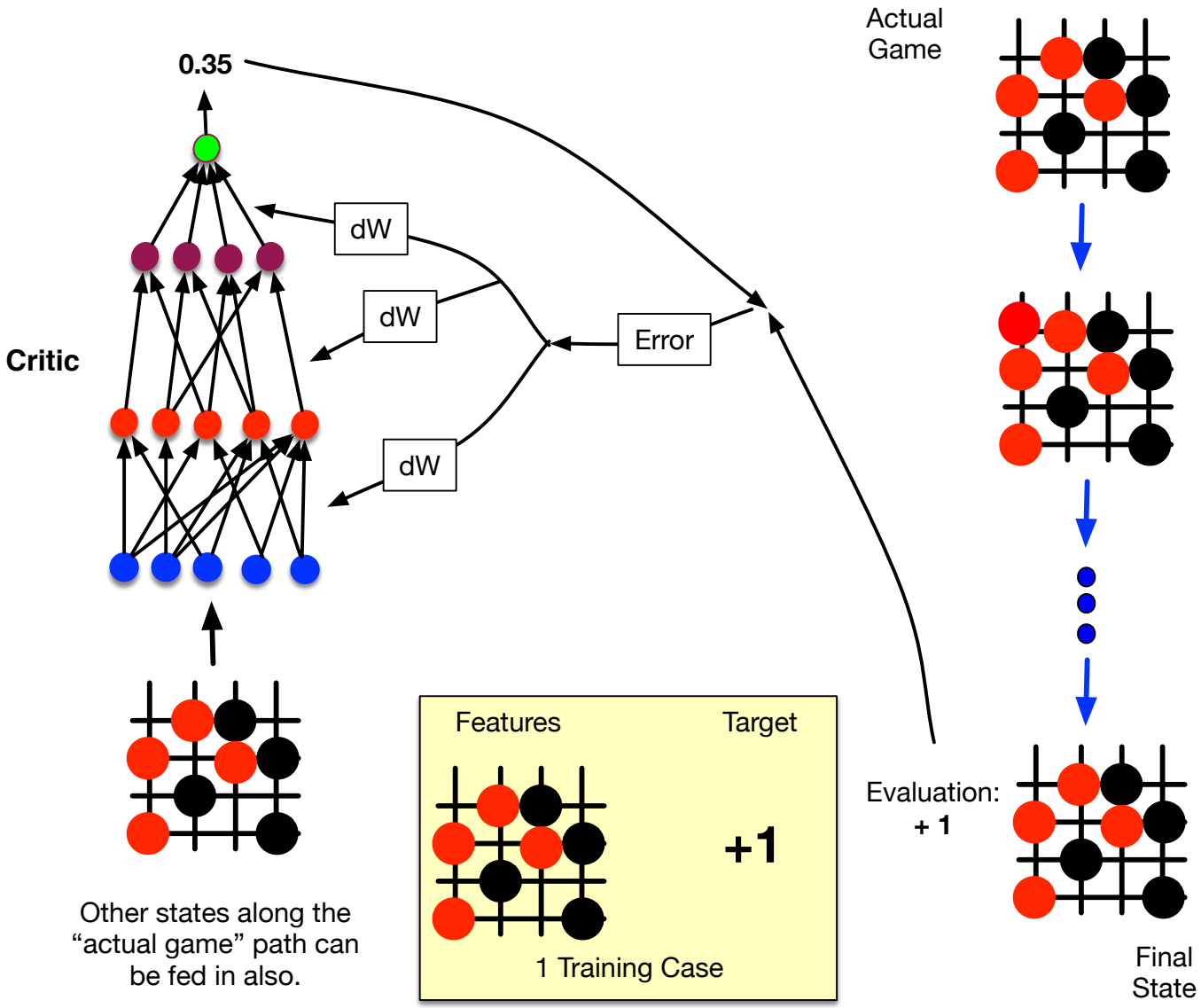


- Tree provides targets for training NN actor and critic.
- *Monte-Carlo tree search and rapid action value estimation in computer GO*, Gelly and Silver, AI Journal, 2011.

Training Actors with the MC Tree

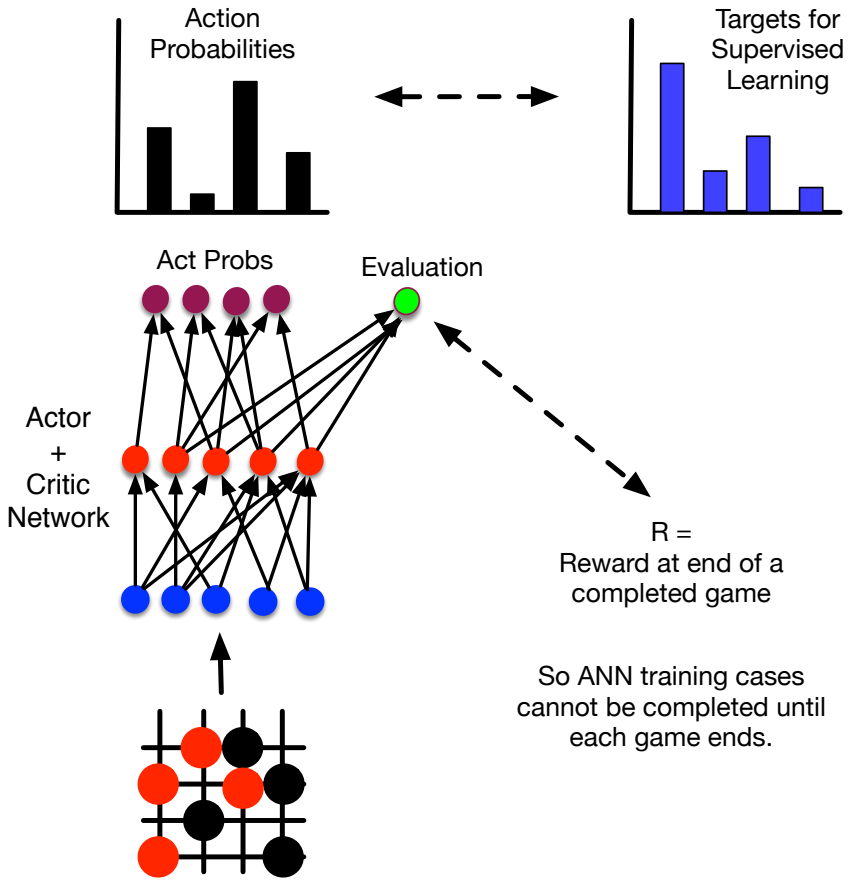


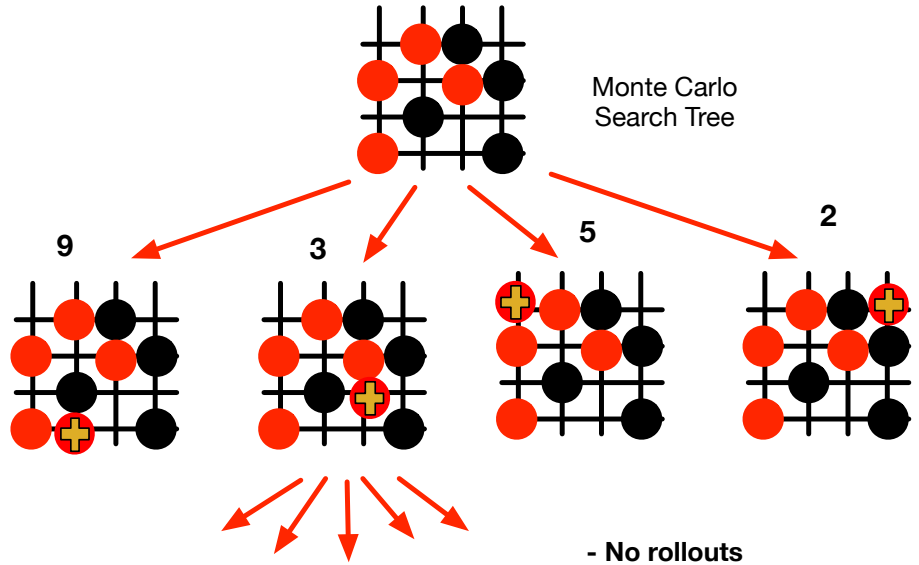
Training Critics with Actual Game Results



Actual game moves stem from MCTS.

Single Actor-Critic Network in AlphaGo Zero





- No rollouts
- Leaf nodes in MCTS are evaluated using the dual-purpose ANN to yield z values, which are then backed up.

Important Design Decisions in MCTS

1. Are leaf nodes to be evaluated by rollouts, a critic, or both. If both, will one or the other be applied at a leaf, or will both be applied and their results combined?
2. If both an actor and critic are used, will there be a separate neural network for each?
3. Will the actor's policy change over time via learning?
4. Will the critic adapt via learning?
5. Which version of the exploration bias, u(s,a)?
6. Will leaf nodes be expanded fully, or partially but once per simulation (or once every k simulations)?
7. Will states produced by rollouts be immediately added to the MC tree?

Important Parameters in MCTS

1. Control of exploration -vs- exploitation via the weighting factor (c) applied to u(s,a).
2. Number of **simulations** per actual move. Simulation = descent + backpropagation
3. Bias of rollout versus critic evaluation via Ψ. Is it static or dynamic?
4. During rollouts, the default policy may have a probability (ε) of choosing a random move rather than the best (so far) move. Exploration is enhanced when $\varepsilon > 0$.

As usual, many of these decisions are interrelated.