Monte Carlo Tree Search

CSE599G1: Deep Reinforcement Learning

Aravind Rajeswaran and Kendall Lowrey

April 25, 2018

Setting

- An MDP with known s'~*P(s, a)* -- *model based method*
- Terminal States (Finite Horizon Problem).
- Discrete State, Action spaces with fully observed (perfect information) states
- Other methods too limited:
  - High branching factor
  - Inability to evaluate a state
  - Long term time dependencies

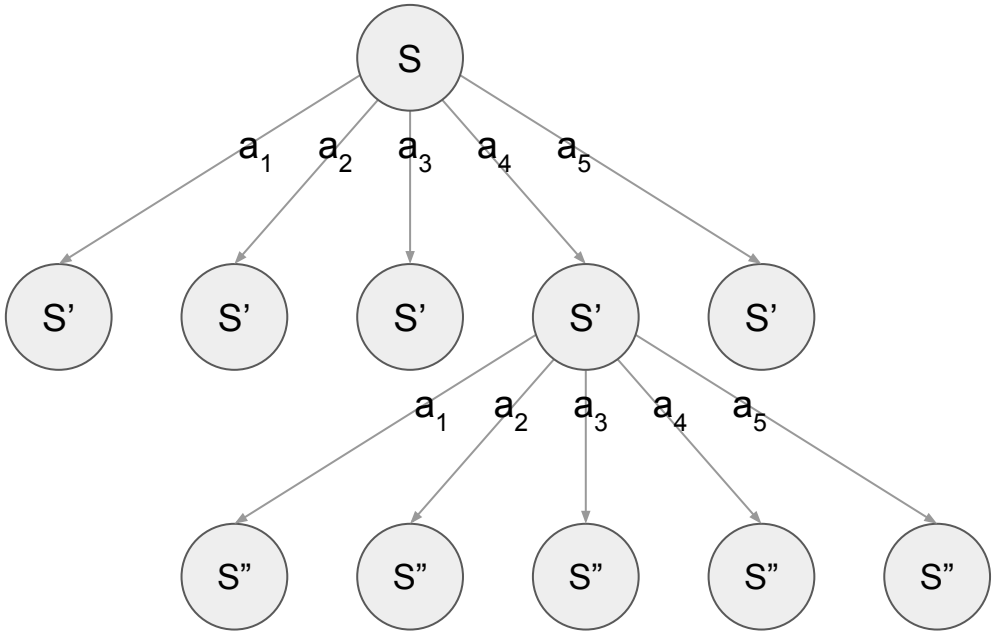
Setting

Knowing *s'~P(s, a)* can let us build a tree structure for mapping how states branch into next states through actions.

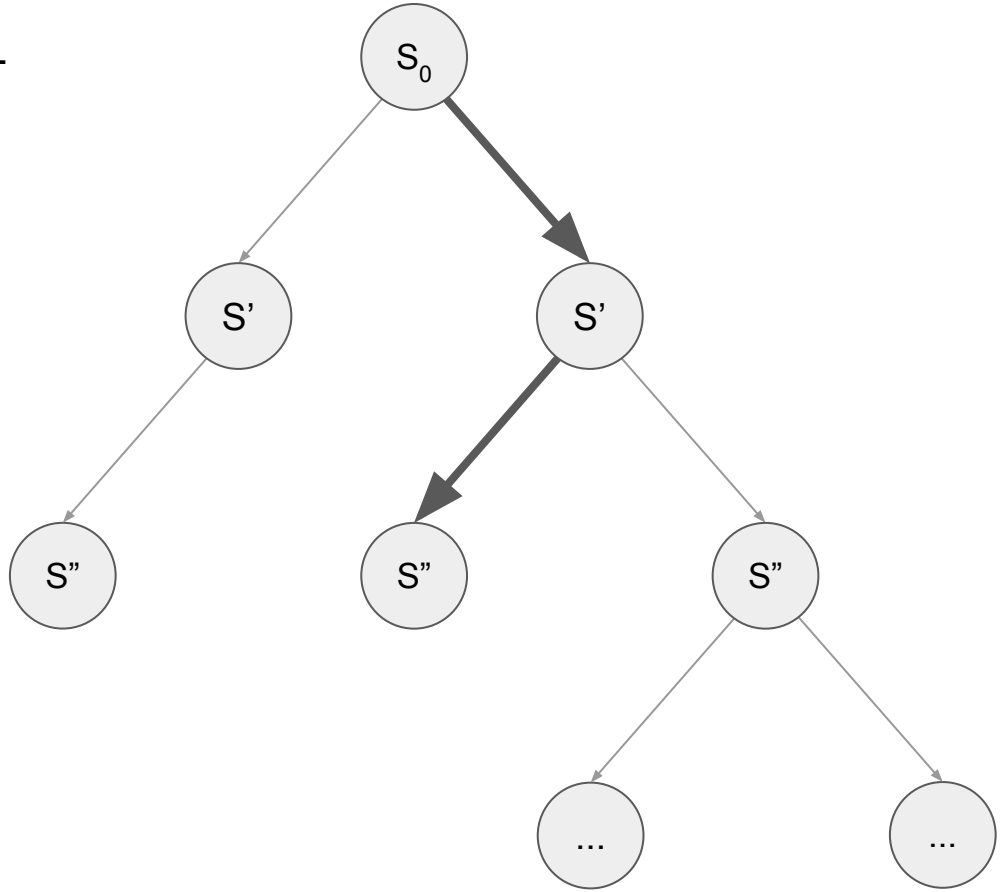
If the *branching factor* is too high, we can never fully build out the tree.

If we do not build out the whole tree, we cannot follow the branches that lead us to high reward (aka *winning*).

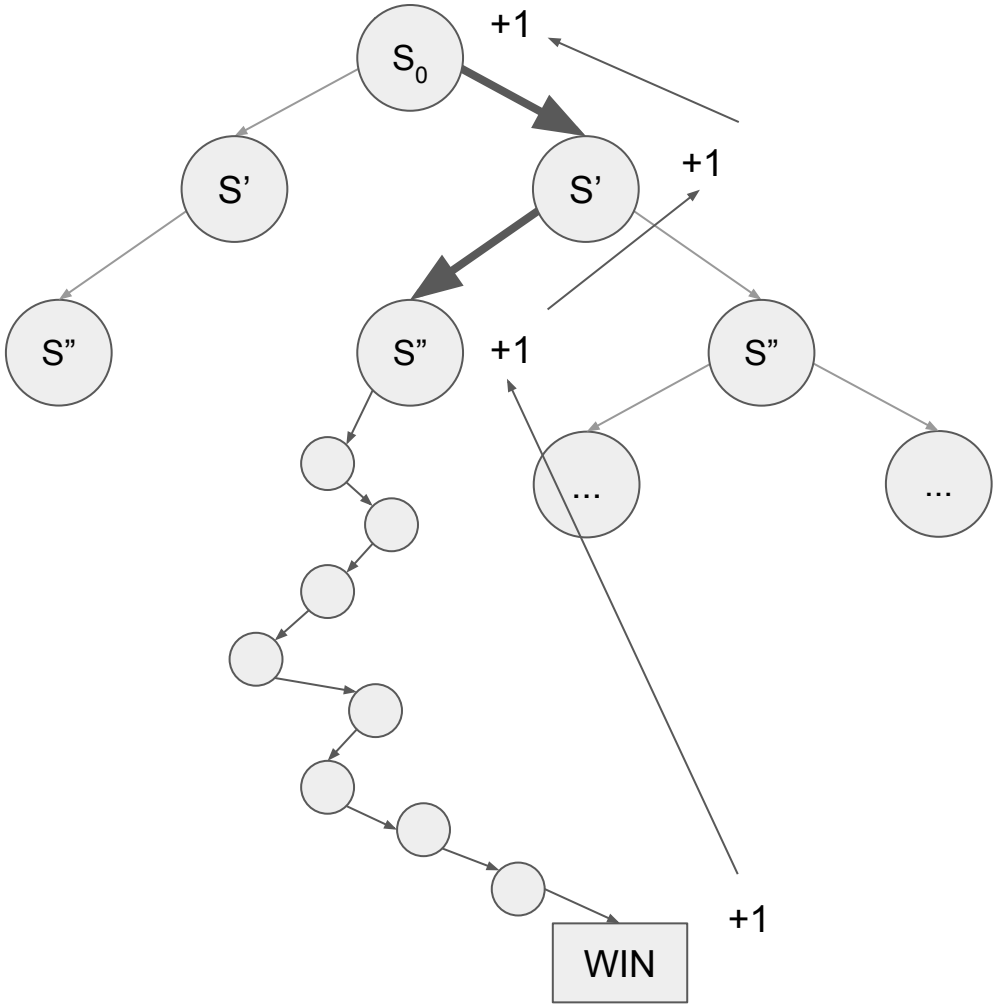
How do we know which branches to take?



TREE TRAVERSAL



SIMULATION
ROLLOUT



General MCTS Algorithm

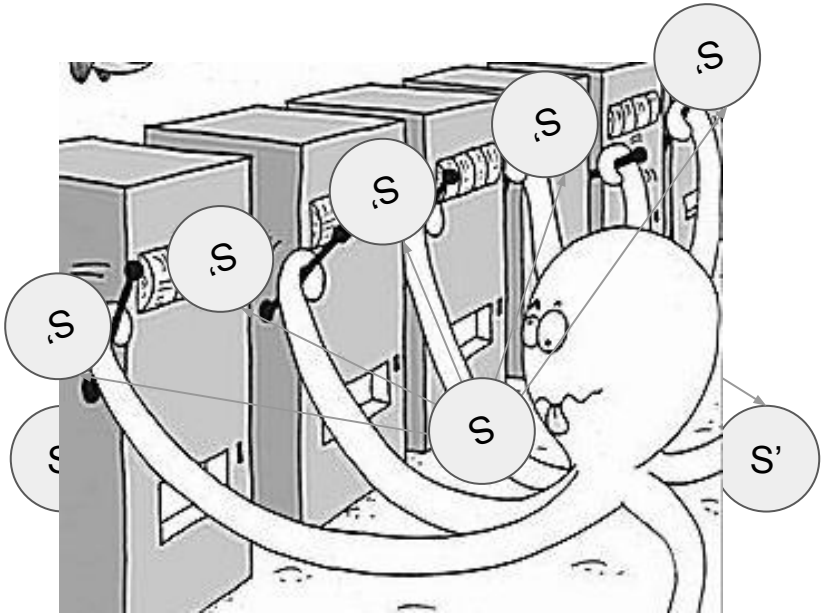
```
MCTSEARCH(S):
    create new tree T with state S as root
    LOOP:
        L ← TRAVERSE T until leaf node
        V ← SIMULATE MDP from L
        BACKUP value V through T
    return BESTCHILD(S)
```

General MCTS Algorithm

```
MCTSEARCH(S):
    create new tree T with state S as root
    LOOP:
        L ← TRAVERSE T until leaf node
        V ← SIMULATE MDP from L
        BACKUP value V through T
    return action of BESTCHILD(S)
```

No heuristics needed: we can default to random search
No minimum computation: we can query for action at anytime
No symmetries: the tree growth has no rules, and depends on how you traverse

How do we traverse our tree?



Each Layer of the tree is like a multi-armed bandits problem….

Upper Confidence Bound Strategy (Bandits)

$$argmax_j(\bar{X}_j + \alpha\sqrt{\frac{2lnN}{n_j}})$$

Where *j* is action index

X_j is the expected reward for action *j*

N is the total number of actions taken

n_j is number of times action *j* has been selected

α weights between exploiting the reward X_j or the exploration of action *j*

Bandit based Monte-Carlo Planning
http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.102.1296

UCB1 applied to Trees (UCBT)

From the general MCTS algorithm we have win/loss rates and visitation.

UCB gives us a way to balance exploitation vs exploration.

Combined:

$$UCT(S_j, S) = \frac{Q(S_j)}{N(S_j)} + \alpha\sqrt{\frac{2lnN(S)}{N(S_j)}}$$

Shown to converge to MINMAX given ‘enough’ simulation time.

L. Kocsis and C. Szepesvari, “Bandit based Monte-Carlo Planning,”
in Euro. Conf. Mach. Learn. Berlin, Germany: Springer, 2006, pp. 282–293.
L. Kocsis, C. Szepesvari, and J. Willemson, “Improved Monte-Carlo Search,”
Univ. Tartu, Estonia, Tech. Rep. 1, 2006.

UCT MCTS Algorithm

```
MCTSEARCH(S):
    create new tree T
    LOOP:
        L ← TRAVERSE T until leaf
        V ← SIMULATE MDP from L
        BACKUP value V through T
    return BESTCHILD(S)
```

```
BESTCHILD(S):
    return argmax UCT(S_j,S)

BACKUP(S,V):
    N(S) += 1
    Q(S) += V
    BACKUP(parent(S),V)
```

If we run this for some set time / computational limit, the hope is that the most explored branches are the same as the highest reward branches.

Case Study: AlphaGo Lee

Policy Network (12 layer CNN) from Supervision:

30M game positions from humans trained to maximize likelihood of actions.

RL Policy Network (12 layer CNN):

Maximize likelihood of game wins (Policy Gradient Method) wrt actions.

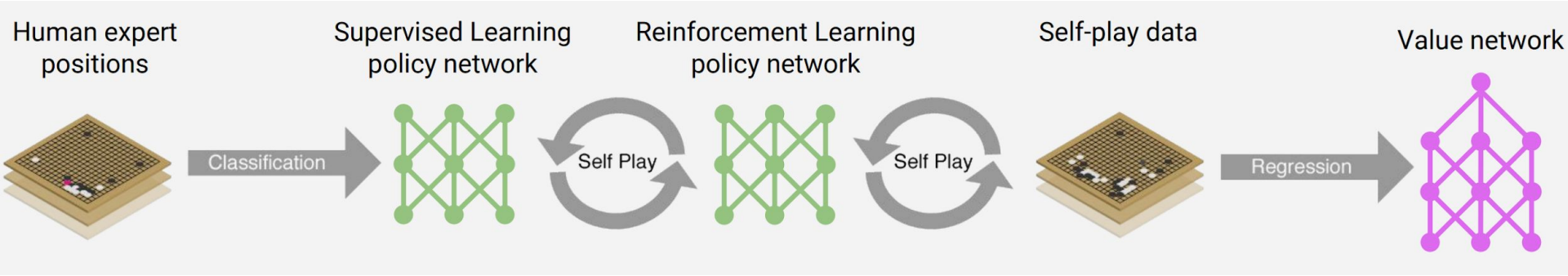
Value Network (12 layer CNN):

30M self play games, minimize MSE between game position win/loss.

Value Network becomes evaluation function.

Networks are learned “offline” -- after a game has been played, game data is used

Case Study: AlphaGo Lee



Policy Network augments UCT during 'Traverse' step

$$UCT(S_j, S) = \frac{Q(S_j)}{N(S_j)} + \alpha\sqrt{\frac{2lnN(S)}{N(S_j)}}$$

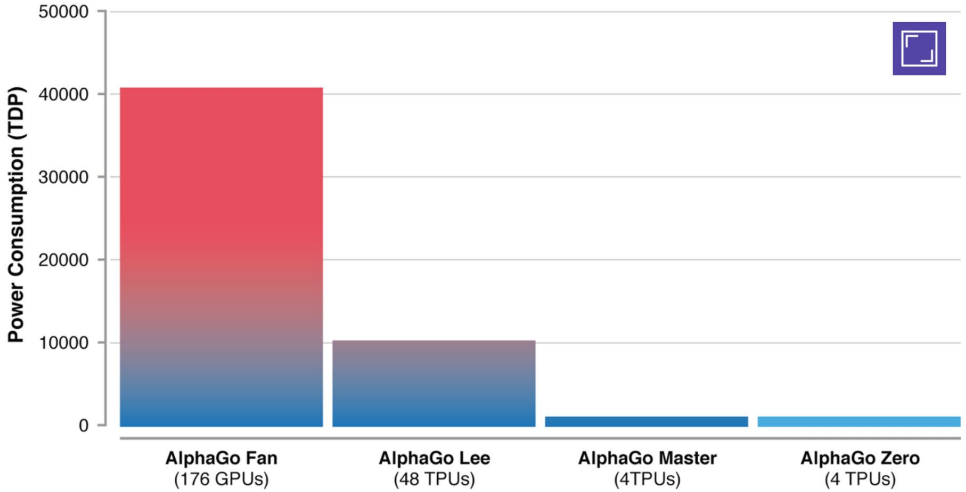
$$\alpha = cP(S, S_j)$$

Value Network is combined with fast rollouts (z) for 'Simulation':

$$Q(S_L) = (1-\alpha)v_0(S_L) + \alpha z_L$$

Case Study: AlphaGo Zero

No more supervised learning step: straight to RL Policy network starting from random play.
Policy Network and Value network are combined: more efficient training.
No ‘Simulation’ step: direct board evaluation with value function.



Case Study: AlphaZero

Same as AlphaGo Zero, but with Go-specifics removed:

- No data augmentation due to rotational invariance
- Allows for more than win/loss

For NN with parameters *theta*, final game state *z*, and action probabilities *pi*

$$(\mathbf{p}, v) = f_\theta(s), \qquad l = (z - v)^2 - \boldsymbol{\pi}^\top \log \mathbf{p} + c||\theta||^2$$

Looking Forward

MCTS as Tabular Representaiton?

MCTS as Trajectory Optimization?

AlphaZero as Guided Policy Search?

References

http://mcts.ai/pubs/mcts-survey-master.pdf

http://www0.cs.ucl.ac.uk/staff/d.silver/web/Resources.html