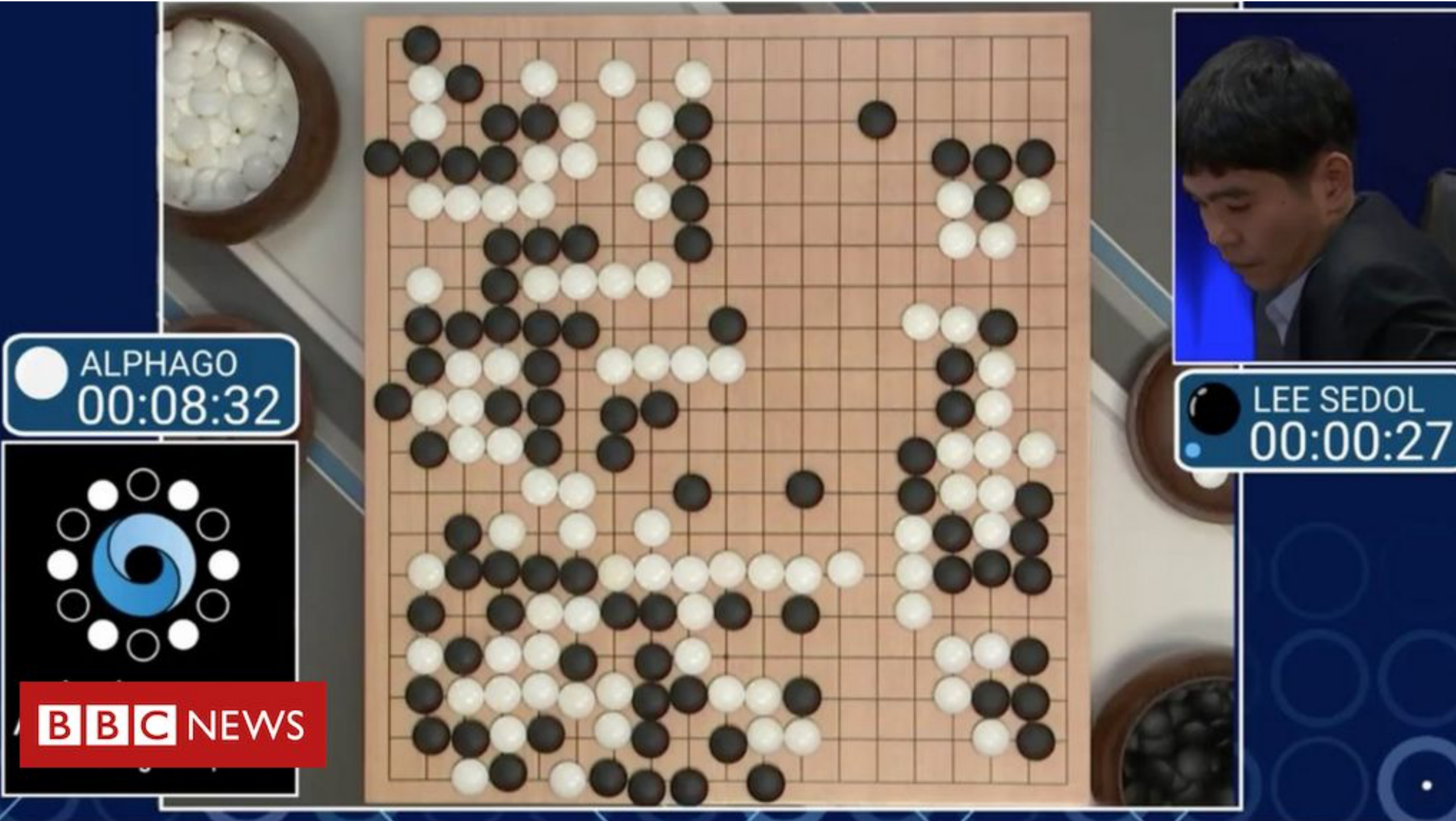
Case Study: AlphaGo


ALPHAGO
00:08:32
LEE SEDOL
00:00:27
BBC NEWS

Outline for Today:

1. Setting

2. The imitation learning component

3. The policy Gradient Component

4. The combination of policy, value, and tree search

Setting: Two player Markov Games:

$$\mathscr{M} = \{S, A, f, r, H, s_0\}$$

$s' = f(s, a)$

Setting: Two player Markov Games:

$$\mathscr{M} = \{S, A, f, r, H, s_0\}$$

We have two players π_1 and π_2, they take turn to play:

$$s_0, \quad a_0 \sim \pi_1(s_0), s_1 = f(s_0, a_0), \quad a_1 \sim \pi_2(s_1), s_2 = f(s_1, a_1), \ldots, s_H$$

player 1

player 2

Setting: Two player Markov Games:

$$\mathcal{M} = \{S, A, f, r, H, s_0\}$$

We have two players π_1 and π_2, they take turn to play:

$$s_0, \quad a_0 \sim \pi_1(s_0), s_1 = f(s_0, a_0), \quad a_1 \sim \pi_2(s_1), s_2 = f(s_1, a_1), \ldots, s_H$$

Sparse reward at the termination state: $r(s_H) = 1$ if wins, -1 otherwise

Setting: Two player Markov Games:

$$\mathcal{M} = \{S, A, f, r, H, s_0\}$$

We have two players π_1 and π_2, they take turn to play:

$$s_0, \quad a_0 \sim \pi_1(s_0), s_1 = f(s_0, a_0), \quad a_1 \sim \pi_2(s_1), s_2 = f(s_1, a_1), \ldots, s_H$$

Sparse reward at the termination state: $r(s_H) = 1$ if wins, -1 otherwise

Min-max formulation:

$$\max_{\pi_1} \min_{\pi_2} \mathbb{E}\left[r(s_H) \mid \pi_1, \pi_2\right]$$

Setting: Two player Markov Games:

Denote optimal value function $V^\star$ as:

$$V^\star(s) = \max_{\pi_1} \min_{\pi_2} \mathbb{E}[r(s_H) \mid s_0 = s, \pi_1, \pi_2]$$

Setting: Two player Markov Games:

Denote optimal value function $V^\star$ as:

$$V^\star(s) = \max_{\pi_1} \min_{\pi_2} \mathbb{E}[r(s_H) \,|\, s_0 = s, \pi_1, \pi_2]$$

The optimal game value if we start at s, and both player plays optimally…

Setting: Two player Markov Games:

Denote optimal value function $V^\star$ as:

$$V^\star(s) = \max_{\pi_1} \min_{\pi_2} \mathbb{E}[r(s_H) \mid s_0 = s, \pi_1, \pi_2]$$

The optimal game value if we start at s, and both player plays optimally…

It’s a zero-sum game, i.e., they cannot both win or both lose…

Setting: Two player Markov Games:

Denote optimal value function $V^\star$ as:

$$V^\star(s) = \max_{\pi_1} \min_{\pi_2} \mathbb{E}[r(s_H) \mid s_0 = s, \pi_1, \pi_2]$$

The optimal game value if we start at s, and both player plays optimally…

It's a zero-sum game, i.e., they cannot both win or both lose…

Player 2 tries to minimize the expected win rate of player 1,
which is equivalent to maximizes its own win rate

Setting: Two player Markov Games:

Min-max formulation:

$$\max_{\pi_1} \min_{\pi_2} \mathbb{E}\left[r(s_H) \mid \pi_1, \pi_2 \right]$$

Setting: Two player Markov Games:

Min-max formulation:

$$\max_{\pi_1} \min_{\pi_2} \mathbb{E}\left[r(s_H) \,|\, \pi_1, \pi_2 \right]$$

Go has known and deterministic dynamic, i.e., $s' = f(s, a)$ is known and simple, in theory we can do **Dynamic Programming** to solve the max-min formulation..

Setting: Two player Markov Games:

Min-max formulation:

$$\max_{\pi_1} \min_{\pi_2} \mathbb{E}\left[r(s_H) \,|\, \pi_1, \pi_2\right]$$

Go has known and deterministic dynamic, i.e., $s' = f(s, a)$ is known and simple, in theory we can do **Dynamic Programming** to solve the max-min formulation..

But…

Setting: Two player Markov Games:

Min-max formulation:

$$\max_{\pi_1} \min_{\pi_2} \mathbb{E}\left[r(s_H) \mid \pi_1, \pi_2\right]$$

Go has known and deterministic dynamic, i.e., $s' = f(s, a)$ is known and simple, in theory we can do **Dynamic Programming** to solve the max-min formulation..

But…

For Go, $H \approx 150$, $|A| \approx 250$, and $|S| \approx |A|^H$

Setting: Two player Markov Games:

Min-max formulation:

$$\max_{\pi_1} \min_{\pi_2} \mathbb{E}\left[r(s_H) \mid \pi_1, \pi_2\right]$$

Go has known and deterministic dynamic, i.e., $s' = f(s, a)$ is known and simple, in theory we can do **Dynamic Programming** to solve the max-min formulation..

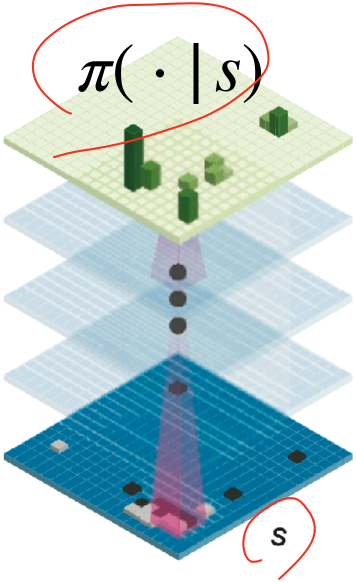
But…

For Go, $H \approx 150$, $|A| \approx 250$, and $|S| \approx |A|^H$

Thus, we cannot enumerate, we must **generalize via function approximation..**

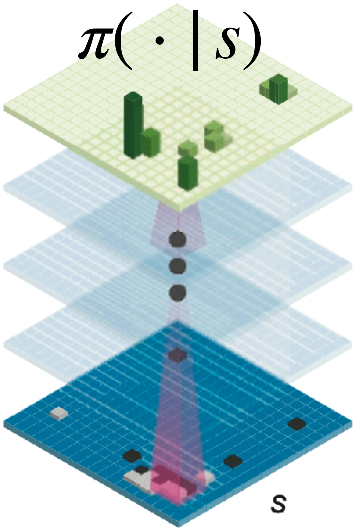
Setting: Function Approximation

1. Policy Network $\approx \pi^\star$

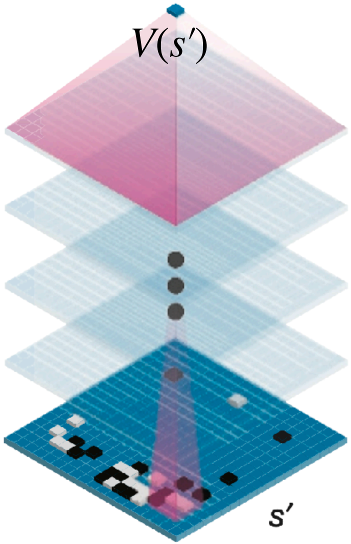


Setting: Function Approximation

1. Policy Network $\approx \pi^\star$



2. Value Network $\approx V^\star(s')$



Outline for Today:

1. Setting

2. The imitation learning component

3. The policy Gradient Component

4. The combination of policy, value, and tree search

Warm start our policy net via Imitation Learning

Warm start our policy net via Imitation Learning

1. Randomly sampled an expert dataset containing 30m (s, a) pairs from KGS Go Server…

Warm start our policy net via Imitation Learning

1. Randomly sampled an expert dataset containing 30m (s, a) pairs from KGS Go Server…

2. Form imitation learning loss function, e.g., Negative Log-likelihood

$$\min_{\pi} \sum_{s,a} - \ln \pi(\underbrace{a}_{\text{expert action}} \mid s)$$

Warm start our policy net via Imitation Learning

1. Randomly sampled an expert dataset containing
30m (s, a) pairs from KGS Go Server…

2. Form imitation learning loss function, e.g., Negative Log-likelihood

$$\min_{\pi} \sum_{s,a} - \ln \pi(a \mid s)$$

3. Optimize via Stochastic Gradient Descent:

$$\theta_{t+1} = \theta_t - \eta \sum_{(s,a)\in B} \nabla_\theta \left(-\ln \pi_{\theta_t}(a \mid s) \right) / |B|$$

unbiased est of Gradient

Warm start our policy net via Imitation Learning

1. Randomly sampled an expert dataset containing 30m (s, a) pairs from KGS Go Server…

2. Form imitation learning loss function, e.g., Negative Log-likelihood

$$\min_{\pi} \sum_{s,a} - \ln \pi(a \mid s)$$

3. Optimize via Stochastic Gradient Descent:

$$\theta_{t+1} = \theta_t - \eta \sum_{(s,a)\in B} \nabla_\theta \left(- \ln \pi_{\theta_t}(a \mid s) \right) / |B|$$

Behavior Cloning!

How well can it predict expert moves on a hold out test dataset?

It achieves 57% accuracy on expert test dataset

How well can it predict expert moves on a hold out test dataset?

It achieves 57% accuracy on expert test dataset

How well does this BC policy perform?

How well can it predict expert moves on a hold out test dataset?

It achieves 57% accuracy on expert test dataset

How well does this BC policy perform?

Test it against the open-source Go program: Pachi (ranked 2 amateur dan on KGS)

How well can it predict expert moves on a hold out test dataset?

It achieves 57% accuracy on expert test dataset

How well does this BC policy perform?

Test it against the open-source Go program: Pachi (ranked 2 amateur dan on KGS)

Win rate: 11%

Outline for Today:

1. Setting

2. The imitation learning component

3. The policy Gradient Component

4. The combination of policy, value, and tree search

Further Improving Policy via PG on Self-playing

Further Improving Policy via PG on Self-playing

1. We warm start our PG procedure using the BC policy…

Further Improving Policy via PG on Self-playing

1. We warm start our PG procedure using the BC policy…

2. We then iterate as follows:

Further Improving Policy via PG on Self-playing

1. We warm start our PG procedure using the BC policy…

2. We then iterate as follows:

$$\pi_{\theta_0} = \pi_{BC}$$

For $t = 0 \rightarrow T - 1$

Further Improving Policy via PG on Self-playing

1. We warm start our PG procedure using the BC policy…

2. We then iterate as follows:

$$\pi_{\theta_0} = \pi_{BC}$$

For $t = 0 \rightarrow T - 1$

Randomly select a previous policy π_{θ_τ}, $\tau < t$

Further Improving Policy via PG on Self-playing

1. We warm start our PG procedure using the BC policy…

2. We then iterate as follows:

$\pi_{\theta_0} = \pi_{BC}$

For $t = 0 \to T - 1$ (# fictitious play to avoid catastrophic forgetting..)

Randomly select a previous policy $\pi_{\theta_\tau}, \ \ \tau < t$

Further Improving Policy via PG on Self-playing

1. We warm start our PG procedure using the BC policy…

2. We then iterate as follows:

$\pi_{\theta_0} = \pi_{BC}$

For $t = 0 \to T - 1$ (# fictitious play to avoid catastrophic forgetting..)

Randomly select a previous policy π_{θ_τ}, $\tau < t$

Terminate state

Play $\underbrace{\pi_{\theta_t}}_{\text{player 1}}$ against $\underbrace{\pi_{\theta_\tau}}_{\text{player 2}}$, get a trajectory $(s_0, \underbrace{a_0}_{\pi_{\theta_t}}, s_1, \underbrace{a_1'}_{\pi_{\theta_\tau}}, s_2, \underbrace{a_2}_{\pi_{\theta_t}}, s_3, a_3' \ldots s_H)$

Further Improving Policy via PG on Self-playing

1. We warm start our PG procedure using the BC policy…

2. We then iterate as follows:

$\pi_{\theta_0} = \pi_{BC}$

For $t = 0 \rightarrow T - 1$ (# fictitious play to avoid catastrophic forgetting..)

Randomly select a previous policy $\pi_{\theta_\tau}, \quad \tau < t$

Play π_{θ_t} against π_{θ_τ}, get a trajectory $(s_0, a_0, s_1, a_1', s_2, a_2, s_3, a_3' \ldots s_H)$

PG update: $\theta_{t+1} = \theta_t + \eta \sum_{h: a_h \sim \pi_{\theta_t}} \nabla_\theta \ln \pi_{\theta_t}(a_h \mid s_h) r(s_H)$

Reinforce

How does the performance improved after PG optimization?

How does the performance improved after PG optimization?

Test it against the open-source Go program: Pachi (ranked 2 amateur dan on KGS)

RL policy has win rate 85%

(BC: 11%)

Outline for Today:

1. Setting

2. The imitation learning component

3. The policy Gradient Component

4. The combination of policy, value, and tree search

Final stage of training: Learn a value function $\hat{V}(s) \approx V^\star(s)$

Denote the PG policy as $\hat{\pi}$, we will approximate $V^{\hat{\pi}}$ instead:

$$V^{\hat{\pi}}(s) = \mathbb{E}\left[r(s_H) \,|\, s_0 = s, \hat{\pi}, \hat{\pi}\right]$$

Final stage of training: Learn a value function $\hat{V}(s) \approx V^{\star}$

Denote the PG policy as $\hat{\pi}$, we will approximate $V^{\hat{\pi}}$ instead:

$$V^{\hat{\pi}}(s) = \mathbb{E}\left[r(s_H) \,|\, s_0 = s, \hat{\pi}, \hat{\pi}\right]$$

i.e., the value of the game when both players play $\hat{\pi}$, starting at s

Final stage of training: Learn a value function $\hat{V}(s) \approx V^\star$

Denote the PG policy as $\hat{\pi}$, we will approximate $V^{\hat{\pi}}$ instead:

$$V^{\hat{\pi}}(s) = \mathbb{E}\left[r(s_H) \,|\, s_0 = s, \hat{\pi}, \hat{\pi}\right]$$

i.e., the value of the game when both players play $\hat{\pi}$, starting at s

We use simple least square regression here:

$$\min_{\beta} \sum_{s,z} (V_\beta(s) - z)^2$$

parameters of value network

state from the Game

value of the Game

Final stage of training: Learn a value function $\hat{V}(s) \approx V^\star$

Denote the PG policy as $\hat{\pi}$, we will approximate $V^{\hat{\pi}}$ instead:

$$V^{\hat{\pi}}(s) = \mathbb{E}\left[r(s_H) \mid s_0 = s, \hat{\pi}, \hat{\pi}\right]$$

i.e., the value of the game when both players play $\hat{\pi}$, starting at s

We use simple least square regression here:

$$\min_{\beta} \sum_{s,z} (V_\beta(s) - z)^2$$

Where s is a **random state in one game play**, and z is the outcome of the play..

Final stage of training: Learn a value function $\hat{V}(s) \approx V^\star$

Denote the PG policy as $\hat{\pi}$, we will approximate $V^{\hat{\pi}}$ instead:

$$V^{\hat{\pi}}(s) = \mathbb{E}\left[r(s_H) \,|\, s_0 = s, \hat{\pi}, \hat{\pi}\right]$$

i.e., the value of the game when both players play $\hat{\pi}$, starting at s

We use simple least square regression here:

$$\min_{\beta} \sum_{s,z} (V_\beta(s) - z)^2$$

Where s is a **random state in one game play**, and z is the outcome of the play..

(We only keep one sample per game play, i.e., we are really sampling $s \sim d^{\hat{\pi}}$ i.i.d)

Final stage of training: Learn a value function $V(s) \approx V^{\star}$

$(\hat{\pi}, \hat{\pi})$

Self-play 30m games, and get 30m (s, z) pairs

Final stage of training: Learn a value function $V(s) \approx V^\star$

Randomly sampled

Self-play 30m games, and get 30m (s, z) pairs

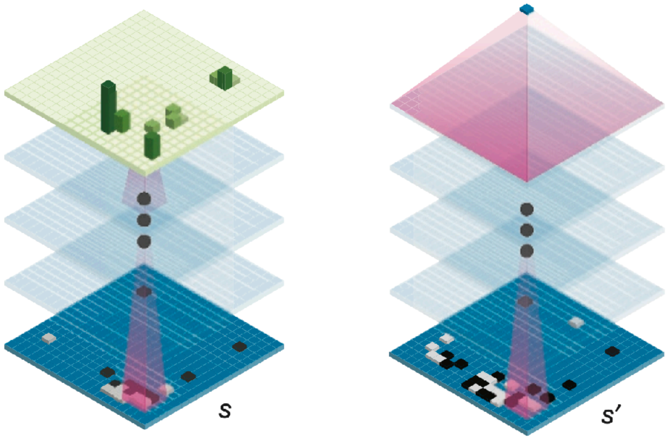
Optimize least square via SGD again:

$$\beta_{t+1} = \beta_t - \eta \sum_{(s,z)\in B} (V_{\beta_t}(s) - z)\,\nabla_\beta V_\beta(s)$$

$$\sum_{s.a} \left(V_\beta(s) - z \right)^2$$

Summary so far

We have learned a policy $\hat{\pi}$ (BC+PG) and $\hat{V} \approx V^{\hat{\pi}}$



To make the program even more powerful, we combine them with a **Search Tree**

Combine with Tree Search (a naive version)

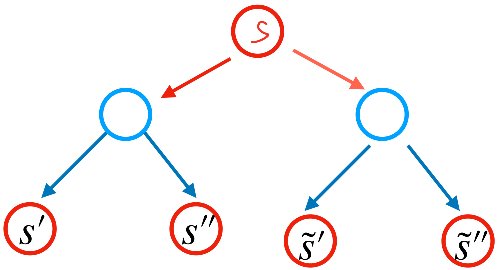
Imagine that we are at state s right now, let's simulate all possible moves into the future

Combine with Tree Search (a naive version)

Imagine that we are at state s right now, let's simulate all possible moves into the future

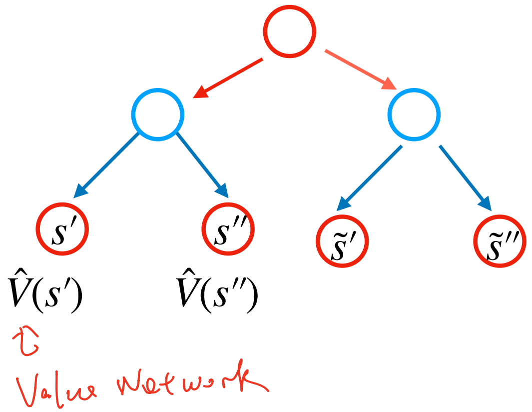
2 actions

$f(s, a) \leftarrow$ known



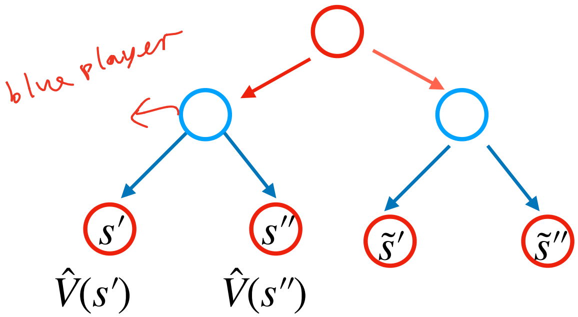
Combine with Tree Search (a naive version)

Imagine that we are at state s right now, let's simulate all possible moves into the future



Combine with Tree Search (a naive version)

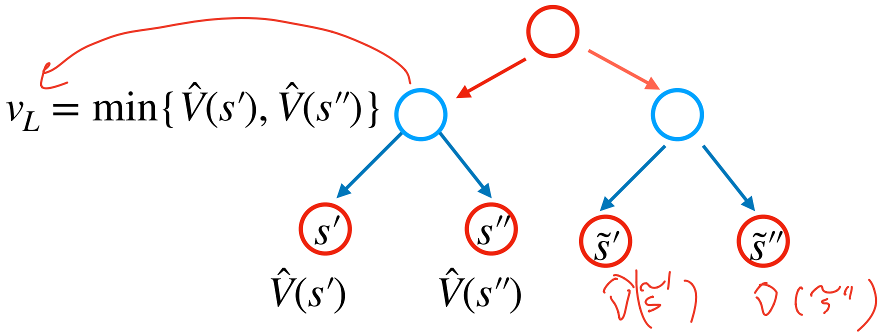
Imagine that we are at state s right now, let's simulate all possible moves into the future



$\hat{V}(s')$: win rate of red player starting at s'

Combine with Tree Search (a naive version)

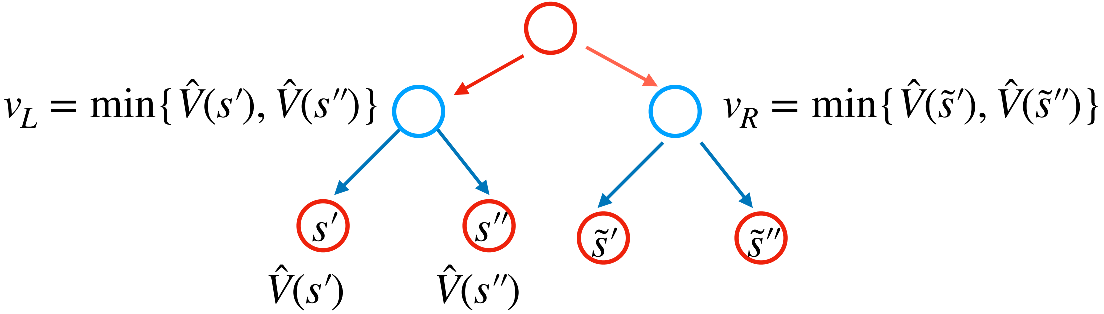
Imagine that we are at state s right now, let's simulate all possible moves into the future



$\hat{V}(s')$: win rate of red player starting at s'

Combine with Tree Search (a naive version)

Imagine that we are at state s right now, let's simulate all possible moves into the future



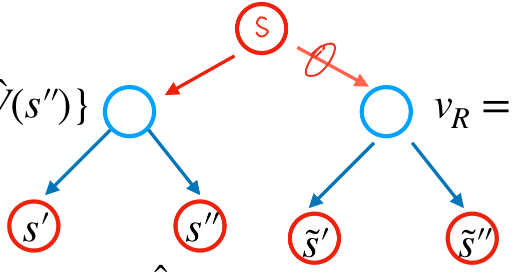
$\hat{V}(s')$: win rate of red player starting at s'

Combine with Tree Search (a naive version)

Imagine that we are at state s right now, let's simulate all possible moves into the future

$$v_{root} = \max\{v_L, v_R\}$$

$$v_L = \min\{\hat{V}(s'), \hat{V}(s'')\} \qquad v_R = \min\{\hat{V}(\tilde{s}'), \hat{V}(\tilde{s}'')\}$$

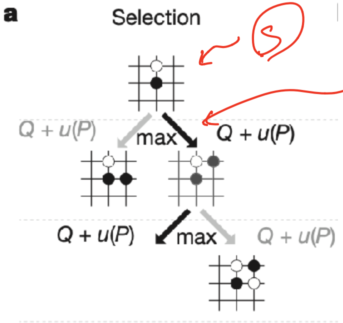


$V^* \approx \hat{V}(s') \qquad \hat{V}(s'')$

$\hat{V}(s')$: win rate of red player starting at s'

AlphaGo uses Monte-Carlo Tree Search algorithm:

AlphaGo uses Monte-Carlo Tree Search algorithm:

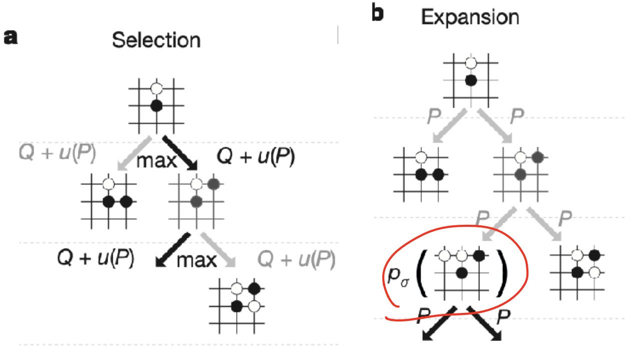


s

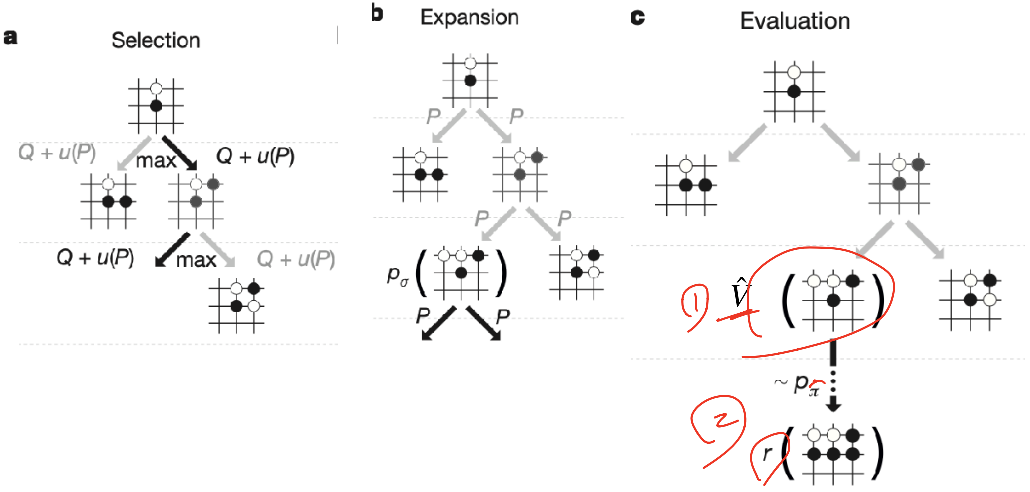
$$Q(s,a) + \sqrt{\frac{1}{N(s,a)}}$$

← # of times we tried (s,a)

AlphaGo uses Monte-Carlo Tree Search algorithm:

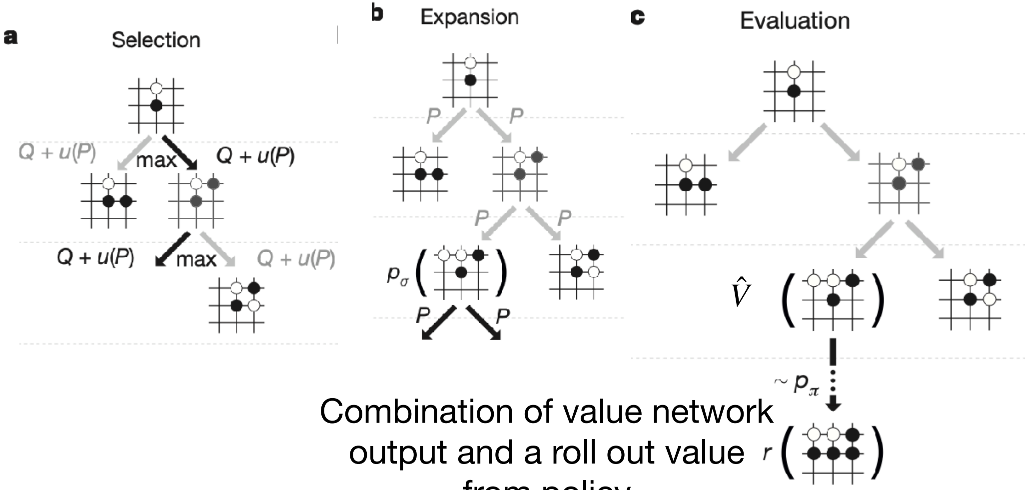


AlphaGo uses Monte-Carlo Tree Search algorithm:



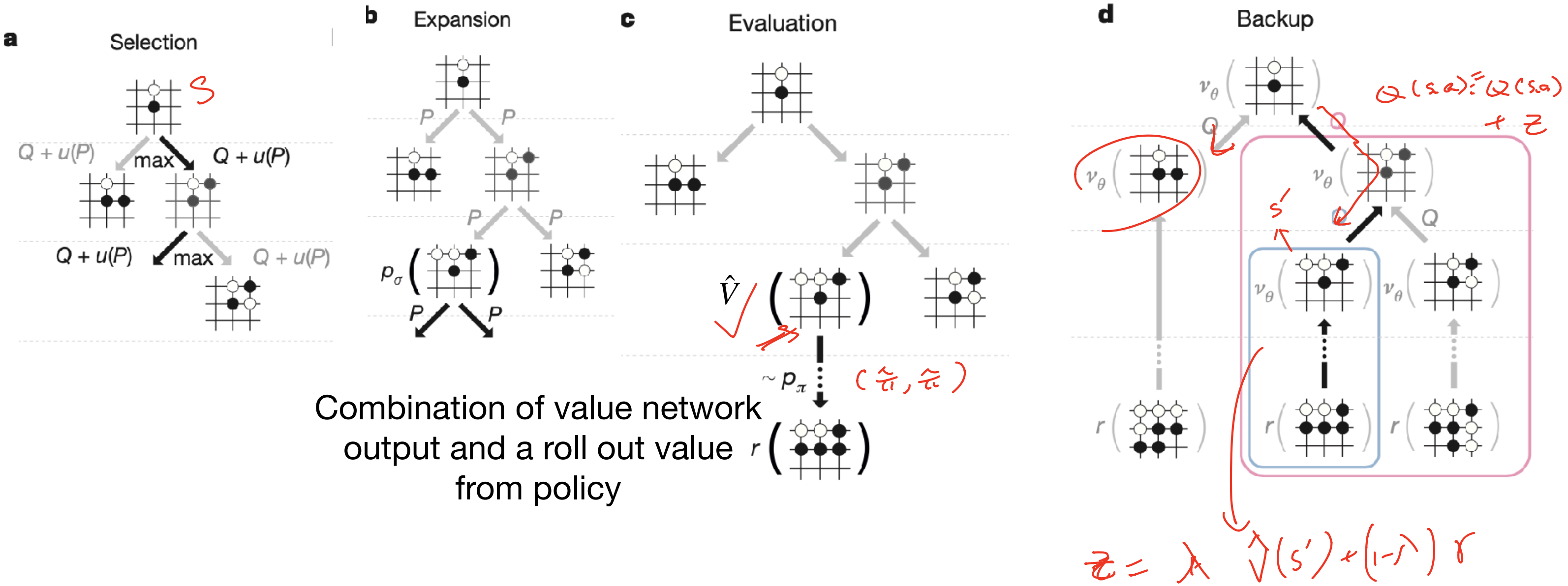
$$\lambda \hat{V}(s') + (1-\lambda) r$$

AlphaGo uses Monte-Carlo Tree Search algorithm:



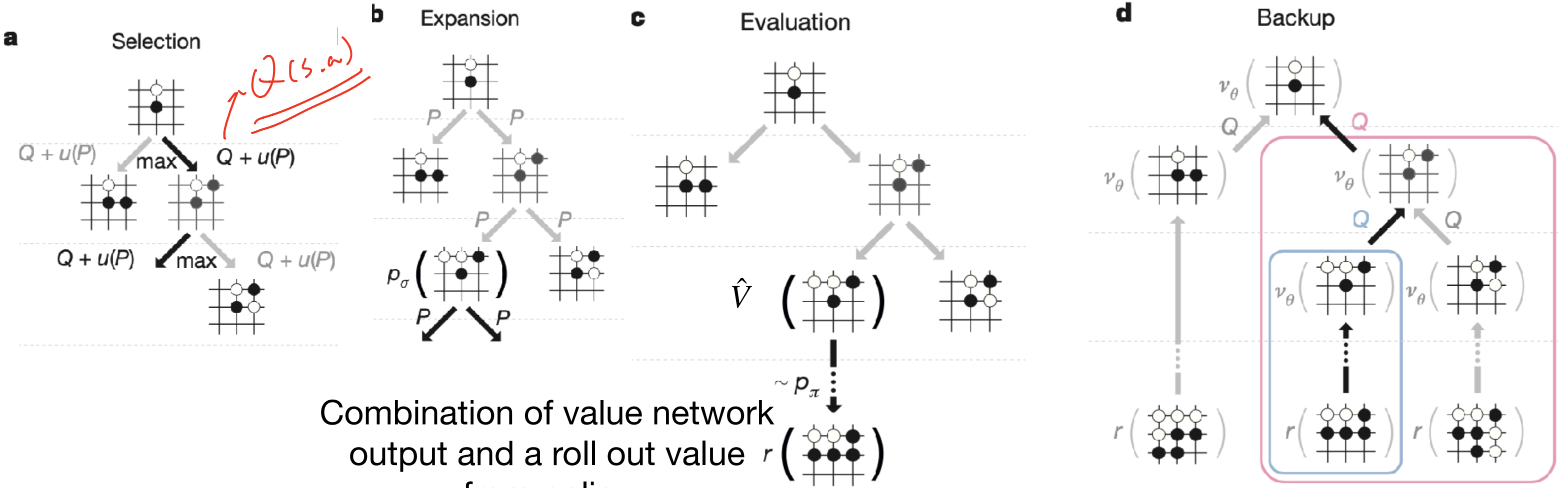
Combination of value network output and a roll out value from policy

AlphaGo uses Monte-Carlo Tree Search algorithm:



Combination of value network output and a roll out value from policy

AlphaGo uses Monte-Carlo Tree Search algorithm:



Combination of value network output and a roll out value from policy

i.e., we enumerate and plan for several steps into the future, and bottom up by a predicted outcome

Summary of the AlphaGo Program

1. Behavior cloning on 30m expert data samples

2. Classic Policy gradient on self-play games

3. Train a value network $\hat{V}$ to predict PG policy's outcome (on 30m self-played games)

4. Build search tree and use $\hat{V}$ to significantly reduce the search tree depth