AIMLC ZG512/ZG525 -

Deep Reinforcement Learning / Computer Vision

Games

S. P. Vimal, Department of CSIS, WILP Division (vimalsp@wilp.bits-pilani.ac.in)

1

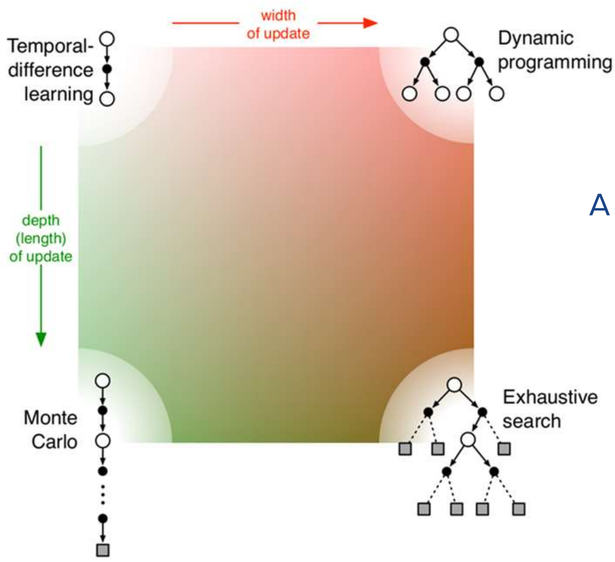
Agenda for the class (1/2)

- ➔ Monte-Carlo Tree Search [MCTS]
- ➔ AlphaGo
- ➔ AlphaGo Zero
- ➔ MuZero

S. P. Vimal, Department of CSIS, WILP Division (vimalsp@wilp.bits-pilani.ac.in)

2

Monte-Carlo Tree Search (MCTS)



A summary of pre-mid sem coverage !!!

S. P. Vimal, Department of CSIS, WILP Division (vimalsp@wilp.bits-pilani.ac.in)

Monte-Carlo Tree Search (MCTS)

Rollout Algorithms:

- Decision-time planning algorithms
- Produce Monte-Carlo estimates of action values only for each current state and for a given policy (Rollout policy)
- Simple, as there is no need to approximate a function over either the
  - entire state space (or)
  - state-action space

- How & Why?
  - Averaging the returns of the simulated trajectories produces estimates of $q\pi(s, a')$ for each action $a' \in A(s)$.
  - The policy selects an action in s that maximizes these estimates & then follows π
- Aim of a rollout algorithm is to improve upon the rollout policy
  - Rollout policy could be completely random !!!

S. P. Vimal, Department of CSIS, WILP Division (vimalsp@wilp.bits-pilani.ac.in)

Monte-Carlo Tree Search (MCTS)

Rollout Algorithms:

- Decision-time planning algorithms
- Produce Monte-Carlo estimates of action values only for each current state and for a given policy (Rollout policy)
- Simple, as there is no need to approximate a function over either the
  - entire state space (or)
  - state-action space

- How & Why?
  - Averaging the returns of the simulated trajectories produces estimates of $q\pi(s, a')$ for each action $a' \in A(s)$.
  - The policy selects an action in s that maximizes these estimates & then follows π
- Aim of a rollout algorithm is to improve upon the rollout policy
  - Rollout policy could be completely random !!!

- MCTS is a recent and strikingly successful example of decision-time planning
- An enhanced rollout algorithm
  - Accumulates value estimates obtained from the simulations to successively direct simulations toward more highly-rewarding trajectories

S. P. Vimal, Department of CSIS, WILP Division (vimalsp@wilp.bits-pilani.ac.in)

Monte-Carlo Tree Search (MCTS)

How MCTS works?

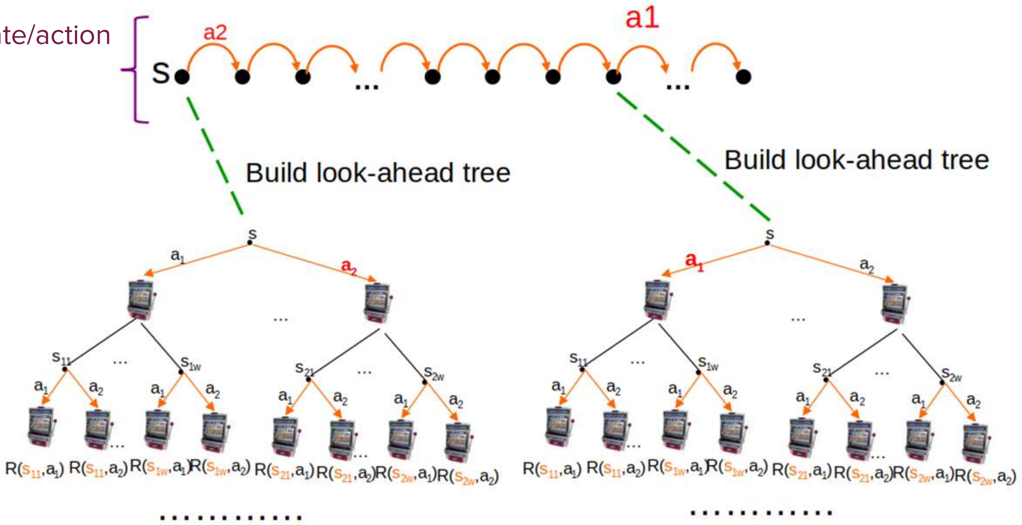
- MCTS is *executed* after encountering each new state (s)
  - [?] to select the agent's action for s
- *Each execution is an iterative process* that simulates many trajectories starting from s and
  - running to a terminal state (or)
  - until discounting makes any further reward negligible to the return
- Focus on multiple simulations starting at s by extending the initial portions of trajectories that have received high evaluations from earlier simulations.

S. P. Vimal, Department of CSIS, WILP Division (vimalsp@wilp.bits-pilani.ac.in)

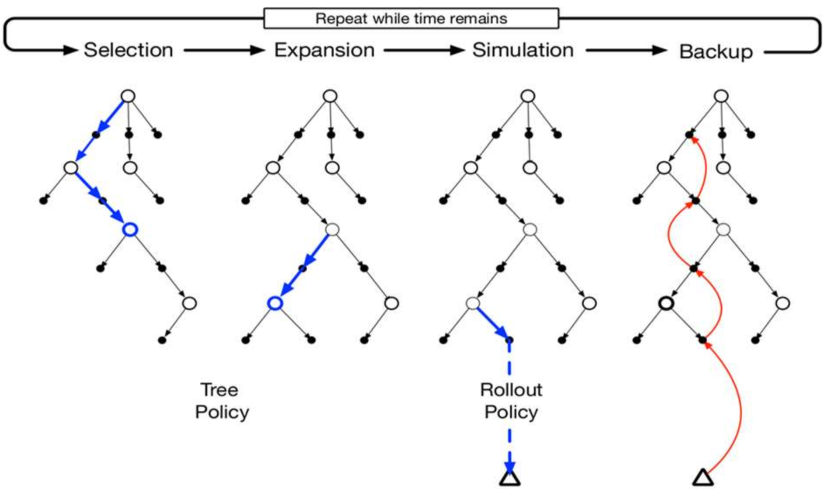
Slide from Prof.Alan Fern, Oregon State University

Monte-Carlo Tree Search (MCTS)

Real world state/action sequence



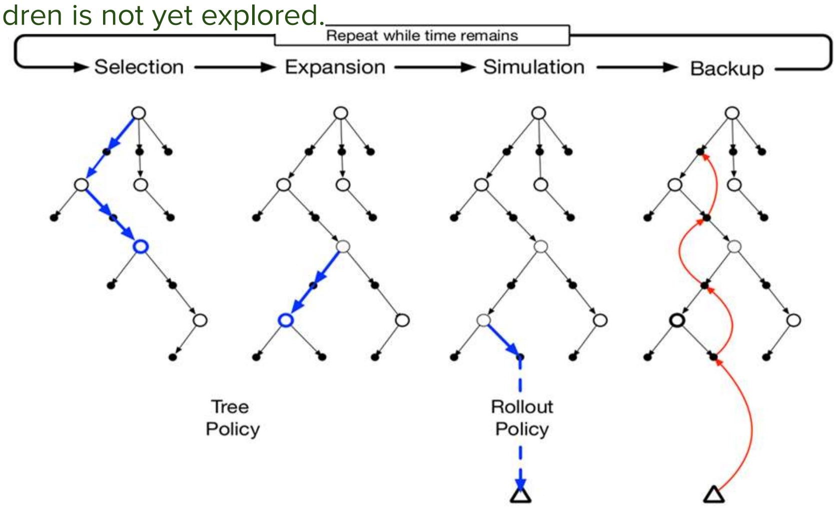
Monte-Carlo Tree Search (MCTS)



$$S_i = x_i + C\sqrt{\frac{\ln(t)}{n_i}}$$

Monte-Carlo Tree Search (MCTS) -- Selection

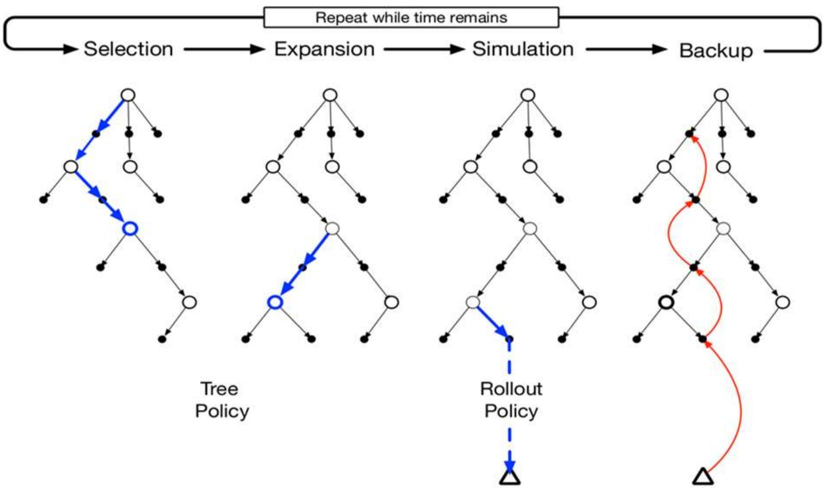
Select: Select a single node in the tree that is *not fully expanded*. By this, we mean at least one of its children is not yet explored.



S. P. Vimal, Department of CSIS, WILP Division (vimalsp@wilp.bits-pilani.ac.in)

Monte-Carlo Tree Search (MCTS) -- Expansion

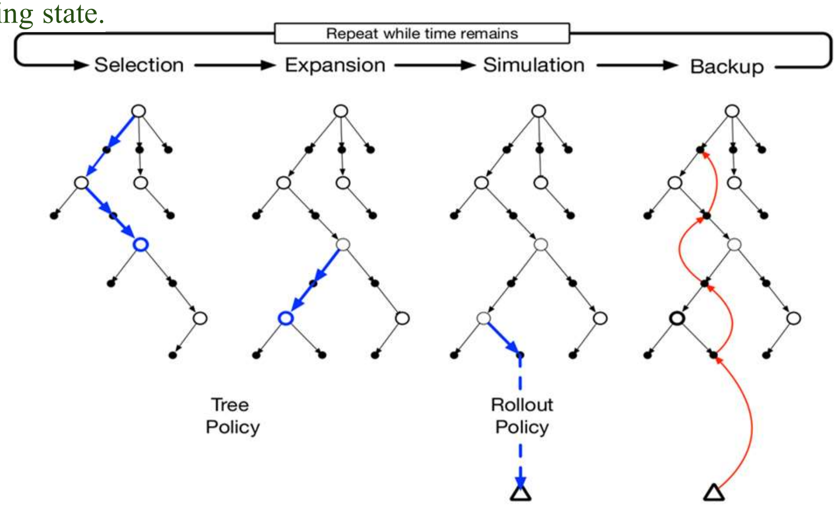
Expand: Expand this node by applying one available action (as defined by the MDP) from the node.



S. P. Vimal, Department of CSIS, WILP Division (vimalsp@wilp.bits-pilani.ac.in)

Monte-Carlo Tree Search (MCTS) -- Simulation

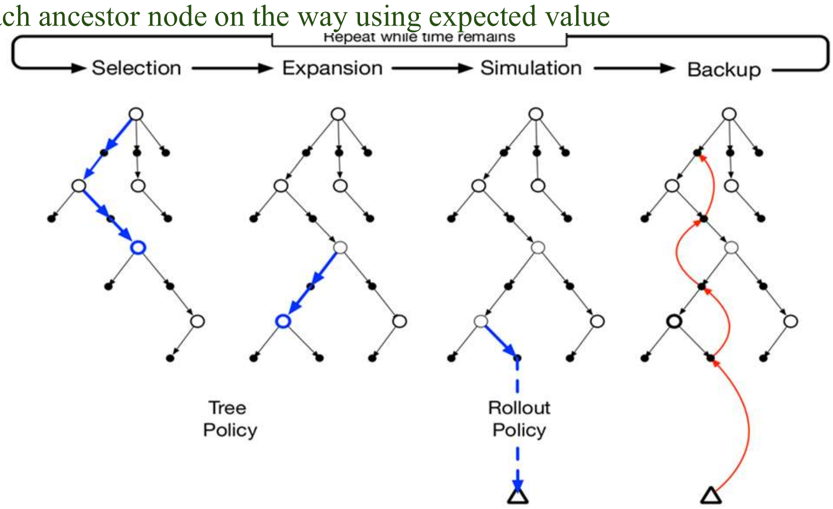
Simulation: From one of the outcomes of the expanded, perform a complete random simulation oto a terminating state.



S. P. Vimal, Department of CSIS, WILP Division (vimalsp@wilp.bits-pilani.ac.in)

Monte-Carlo Tree Search (MCTS) -- Backup

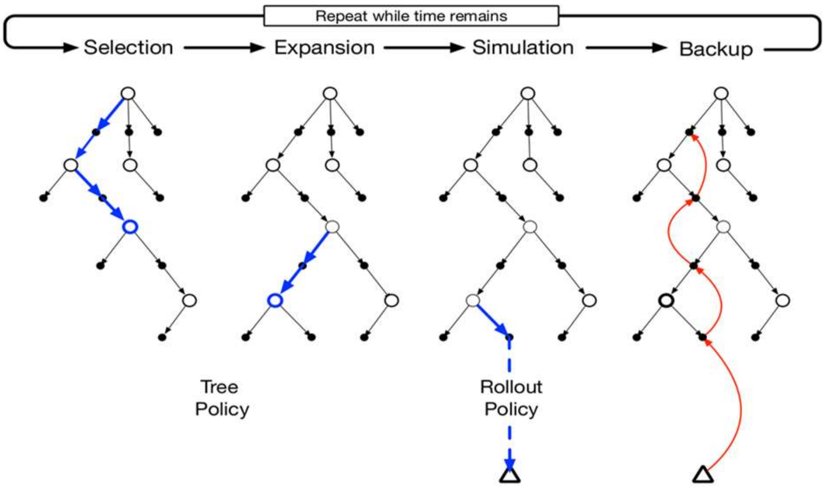
Backup/ Backpropagate: The value of the node is *back propagated* to the root node, updating the value of each ancestor node on the way using expected value



S. P. Vimal, Department of CSIS, WILP Division (vimalsp@wilp.bits-pilani.ac.in)

Monte-Carlo Tree Search (MCTS) -- Summarizing

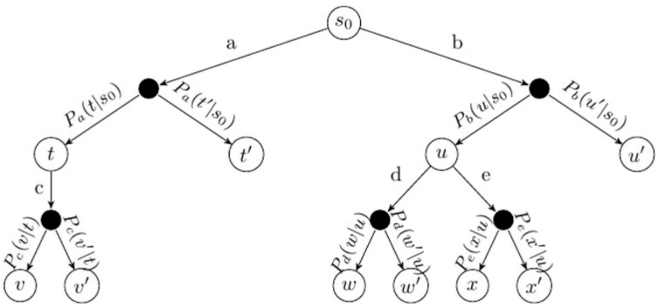
Comments on the overall approach,,,,



S. P. Vimal, Department of CSIS, WILP Division (vimalsp@wilp.bits-pilani.ac.in)

COMP90054: AI Planning for Autonomy ; https://gibberblot.github.io/rl-notes/single-agent/mcts.html

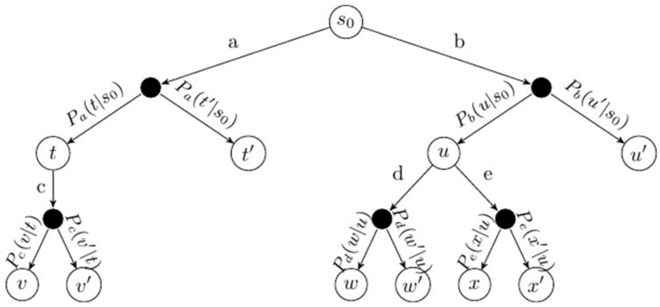
Monte-Carlo Tree Search (MCTS) -- Selection

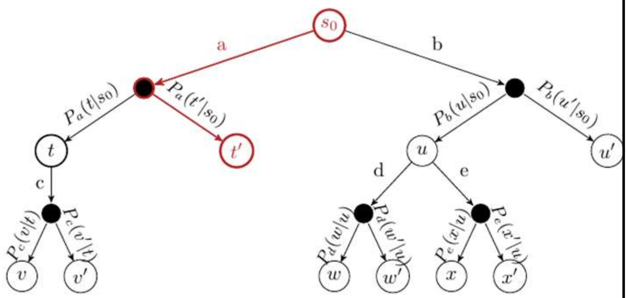


S. P. Vimal, Department of CSIS, WILP Division (vimalsp@wilp.bits-pilani.ac.in)

COMP90054: AI Planning for Autonomy ;
https://gibberblot.github.io/rl-notes/single-agent/mcts.html

Monte-Carlo Tree Search (MCTS) -- Selection

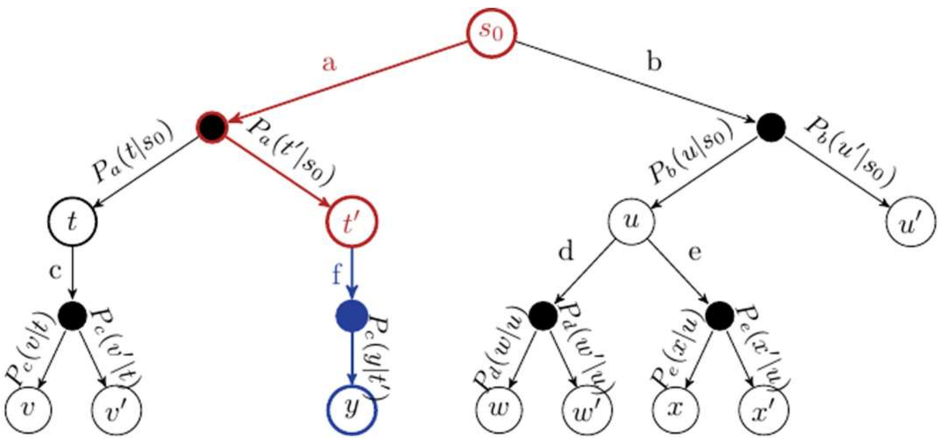




S. P. Vimal, Department of CSIS, WILP Division (vimalsp@wilp.bits-pilani.ac.in)

COMP90054: AI Planning for Autonomy ;
https://gibberblot.github.io/rl-notes/single-agent/mcts.html

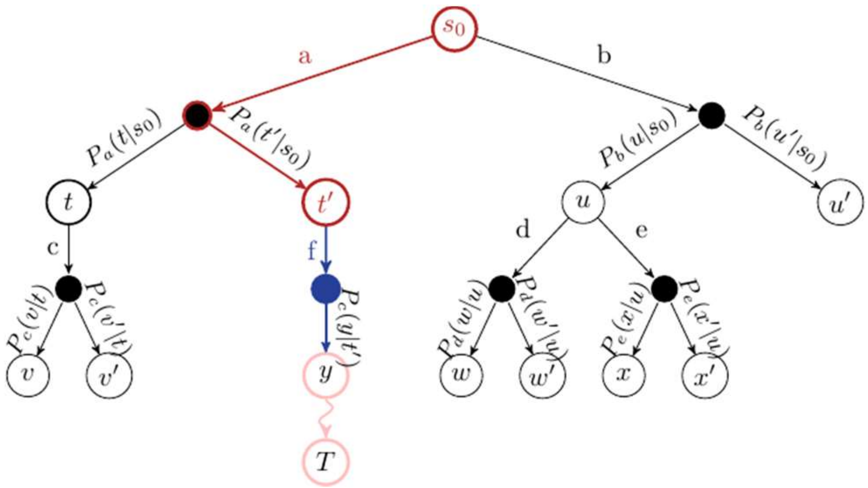
Monte-Carlo Tree Search (MCTS) -- Expansion



S. P. Vimal, Department of CSIS, WILP Division (vimalsp@wilp.bits-pilani.ac.in)

COMP90054: AI Planning for Autonomy ; https://gibberblot.github.io/rl-notes/single-agent/mcts.html

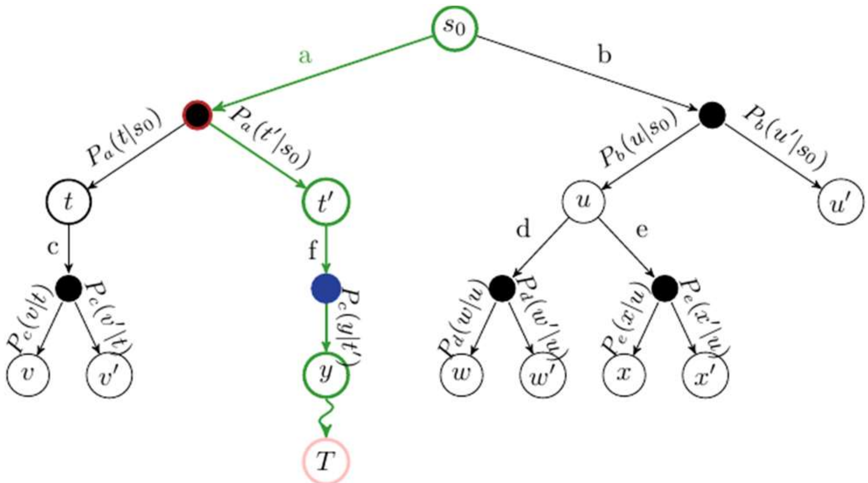
Monte-Carlo Tree Search (MCTS) -- Simulation



S. P. Vimal, Department of CSIS, WILP Division (vimalsp@wilp.bits-pilani.ac.in)

COMP90054: AI Planning for Autonomy ; https://gibberblot.github.io/rl-notes/single-agent/mcts.html

Monte-Carlo Tree Search (MCTS) -- Backup



S. P. Vimal, Department of CSIS, WILP Division (vimalsp@wilp.bits-pilani.ac.in)

COMP90054: AI Planning for Autonomy ;
https://gibberblot.github.io/rl-notes/single-agent/mcts.html

Monte-Carlo Tree Search (MCTS)

Algorithm – Monte-Carlo Tree Search

Input: MDP $M = \langle S, s_0, A, P_a(s' \mid s), r(s,a,s') \rangle$, base value function Q, time limit T.
Output: updated Q-function Q

while $currentTime < T$
    $selected\_node \leftarrow \text{Select}(s_0)$
    $child \leftarrow \text{Expand}(selected\_node)$ – expand and choose a child to simulate
    $G \leftarrow \text{Simulate}(child)$ – simulate from $child$
    $\text{Backpropagate}(selected\_node, child, G)$
return Q

S. P. Vimal, Department of CSIS, WILP Division (vimalsp@wilp.bits-pilani.ac.in)

COMP90054: AI Planning for Autonomy ;
https://gibberblot.github.io/rl-notes/single-agent/mcts.html

Monte-Carlo Tree Search (MCTS)

Function – $\text{Select}(s : S)$

Input: state s
Output: unexpanded state

while s is fully expanded
    Select action a to apply in s using a multi-armed bandit algorithm
    Choose one outcome s' according to $P_a(s' \mid s)$
    $s \leftarrow s'$
return s

S. P. Vimal, Department of CSIS, WILP Division (vimalsp@wilp.bits-pilani.ac.in)

COMP90054: AI Planning for Autonomy ;
https://gibberblot.github.io/rl-notes/single-agent/mcts.html

Monte-Carlo Tree Search (MCTS)

Function – $\text{Expand}(s : S)$

Input: state s
Output: expanded state s'

Select an action a from s to apply
Expand one outcome s' according to the distribution $P_a(s' \mid s)$ and observe reward r
return s'

S. P. Vimal, Department of CSIS, WILP Division (vimalsp@wilp.bits-pilani.ac.in)

COMP90054: AI Planning for Autonomy ;
https://gibberblot.github.io/rl-notes/single-agent/mcts.html

Monte-Carlo Tree Search (MCTS)

Procedure – $\text{Backpropagation}(s : S; a : A)$

Input: state-action pair (s, a)
Output: none

do

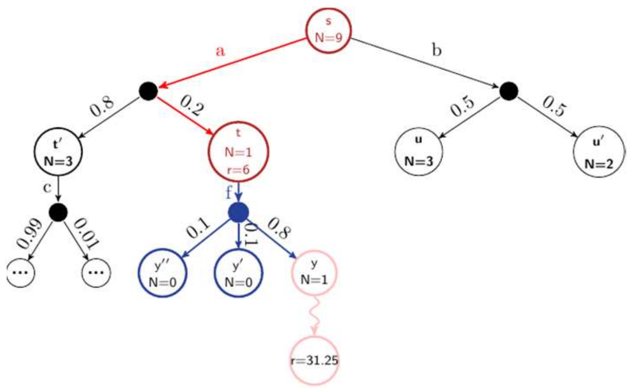
$N(s,a) \leftarrow N(s,a) + 1$
$G \leftarrow r + \gamma G$
$Q(s,a) \leftarrow Q(s,a) + \frac{1}{N(s,a)}[G - Q(s,a)]$
$s \leftarrow$ parent of s
$a \leftarrow$ parent action of s

while $s \neq s_0$

S. P. Vimal, Department of CSIS, WILP Division (vimalsp@wilp.bits-pilani.ac.in)

COMP90054: AI Planning for Autonomy ; https://gibberblot.github.io/rl-notes/single-agent/mcts.html

Monte-Carlo Tree Search (MCTS)



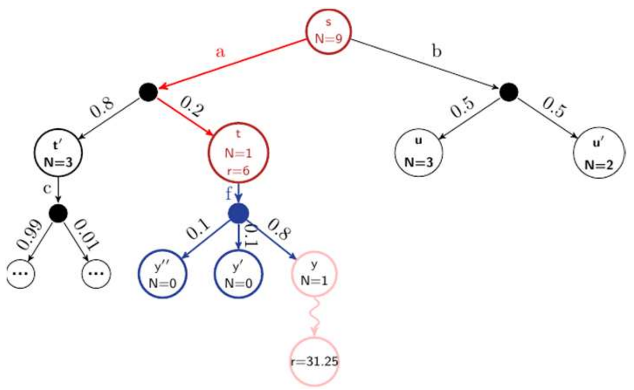
Before backpropagation

$$
\begin{aligned}
Q(s,a) &= 18 \\
Q(t,f) &= 0
\end{aligned}
$$

S. P. Vimal, Department of CSIS, WILP Division (vimalsp@wilp.bits-pilani.ac.in)

COMP90054: AI Planning for Autonomy ; https://gibberblot.github.io/rl-notes/single-agent/mcts.html

Monte-Carlo Tree Search (MCTS)



The backpropagation step is then calculated for the nodes y, t, and s as follows:

$$
\begin{aligned}
Q(y,g) &= \gamma^2 \times 31.25 \;\; \text{(simulation is 3 steps long and receives reward of 31.25)} \\
&= 20 \\
\\
N(t,f) &\leftarrow N(t,f) + 1 = N(y) + N(y') + N(y'') + 1 = 2 \\
Q(t,f) &= Q(t,f) + \frac{1}{N(t,f)}[r + \gamma G - Q(t,f)] \\
&= 0 + \frac{1}{2}[0 + 0.8 \cdot 20 - 0] \\
&= 8 \\
\\
N(s,a) &\leftarrow N(s,a) + 1 = N(t) + N(t') + 1 = 5 \\
Q(s,a) &= Q(s,a) + \frac{1}{N(s,a)}[r + \gamma G - Q(s,a)] \\
&= 18 + \frac{1}{5}[6 + 0.8 \cdot (0.8 \cdot 20) - 18] \\
&= 18 + \frac{1}{5}[6 + 12.8 - 18] \\
&= 18.16
\end{aligned}
$$

Before backpropagation

$$
\begin{aligned}
Q(s,a) &= 18 \\
Q(t,f) &= 0
\end{aligned}
$$

S. P. Vimal, Department of CSIS, WILP Division (vimalsp@wilp.bits-pilani.ac.in)

Upper-Confidence-Bound Action Selection

- ε-greedy action selection forces the non-greedy actions to be tried,
  - Indiscriminately, with no preference for those that are nearly greedy or particularly uncertain
- It would be better to select among the non-greedy actions according to their potential for actually being optimal
  - Take into account both how close their estimates are to being maximal and the uncertainties in those estimates.

$$A_t \doteq \underset{a}{\arg\max}\left[Q_t(a) + c\sqrt{\frac{\ln t}{N_t(a)}}\right]$$

Upper-Confidence-Bound Action Selection

- Each time a is selected the uncertainty is presumably reduced
- Each time an action other than a is selected, t increases but $N_t(a)$ does not; because t appears in the numerator, the uncertainty estimate increases.
- Actions with lower value estimates, or that have already been selected frequently, will be selected with decreasing frequency over time

Action Value at time t for a

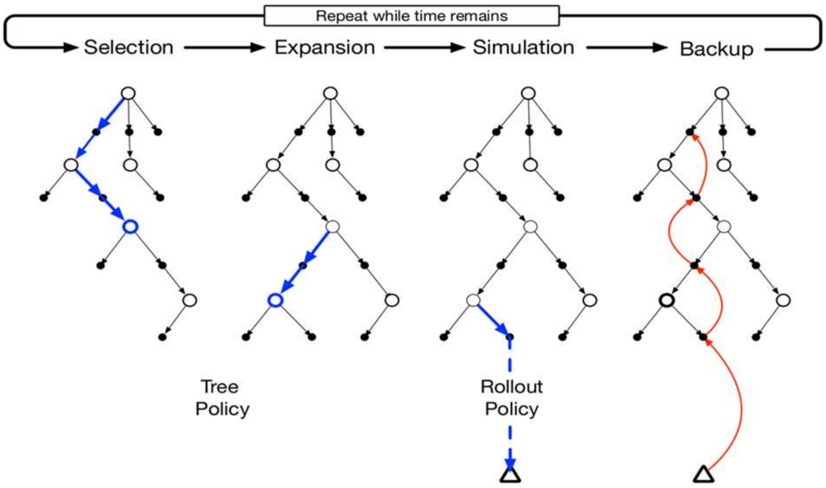
Confidence Level

Measure of Uncertainty

$$A_t \doteq \underset{a}{\arg\max}\left[Q_t(a) + c\boxed{\sqrt{\frac{\ln t}{N_t(a)}}}\right]$$

Monte-Carlo Tree Search (MCTS)

Can the selection of action in Tree policy use UCB?

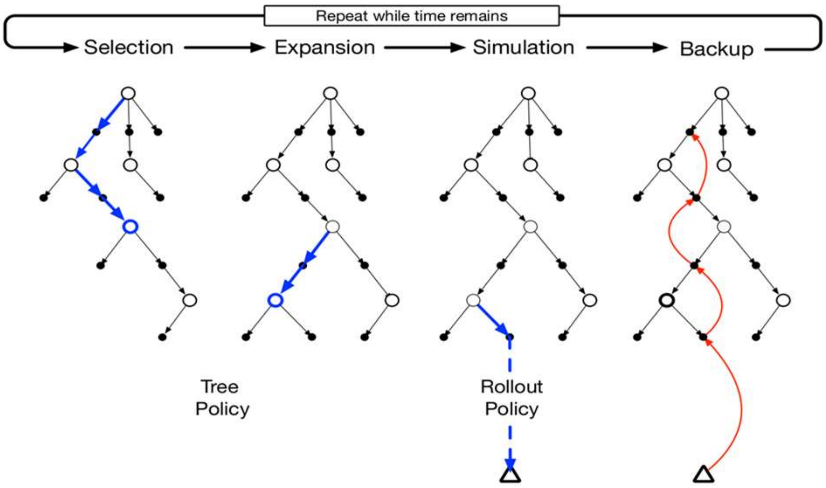


$$S_i = x_i + C\sqrt{\frac{\ln(t)}{n_i}}$$

S. P. Vimal, Department of CSIS, WILP Division (vimalsp@wilp.bits-pilani.ac.in)

Monte-Carlo Tree Search (MCTS)

Can the selection of action in Tree policy use UCB?



Upper Confidence Trees (UCT):
MCTS with UCB for Tree policy

S. P. Vimal, Department of CSIS, WILP Division (vimalsp@wilp.bits-pilani.ac.in)

AlphaGo

To discuss:

- How & Why AlphaGo was significant?
- Working of AlphaGo
- Why is Go a difficult game for AI?
- Role of MCTC in the AlphaGo

ARTICLE

doi:10.1038/nature16961

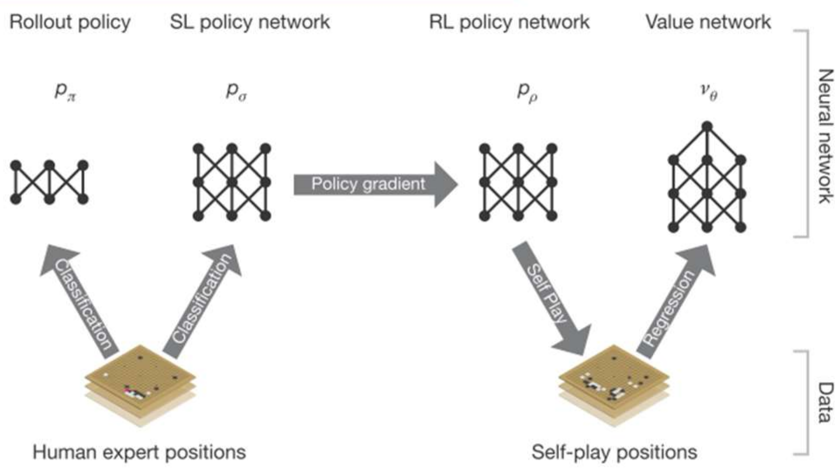
Mastering the game of Go with deep neural networks and tree search

David Silver[1]*, Aja Huang[1]*, Chris J. Maddison[1], Arthur Guez[1], Laurent Sifre[1], George van den Driessche[1], Julian Schrittwieser[1], Ioannis Antonoglou[1], Veda Panneershelvam[1], Marc Lanctot[1], Sander Dieleman[1], Dominik Grewe[1], John Nham[2], Nal Kalchbrenner[1], Ilya Sutskever[2], Timothy Lillicrap[1], Madeleine Leach[1], Koray Kavukcuoglu[1], Thore Graepel[1] & Demis Hassabis[1]

The game of Go has long been viewed as the most challenging of classic games for artificial intelligence owing to its enormous search space and the difficulty of evaluating board positions and moves. Here we introduce a new approach to computer Go that uses 'value networks' to evaluate board positions and 'policy networks' to select moves. These deep neural networks are trained by a novel combination of supervised learning from human expert games, and reinforcement learning from games of self-play. Without any lookahead search, the neural networks play Go at the level of state-of-the-art Monte Carlo tree search programs that simulate thousands of random games of self-play. We also introduce a new search algorithm that combines Monte Carlo simulation with value and policy networks. Using this search algorithm, our program AlphaGo achieved a 99.8% winning rate against other Go programs, and defeated the human European Go champion by 5 games to 0. This is the first time that a computer program has defeated a human professional player in the full-sized game of Go, a feat previously thought to be at least a decade away.

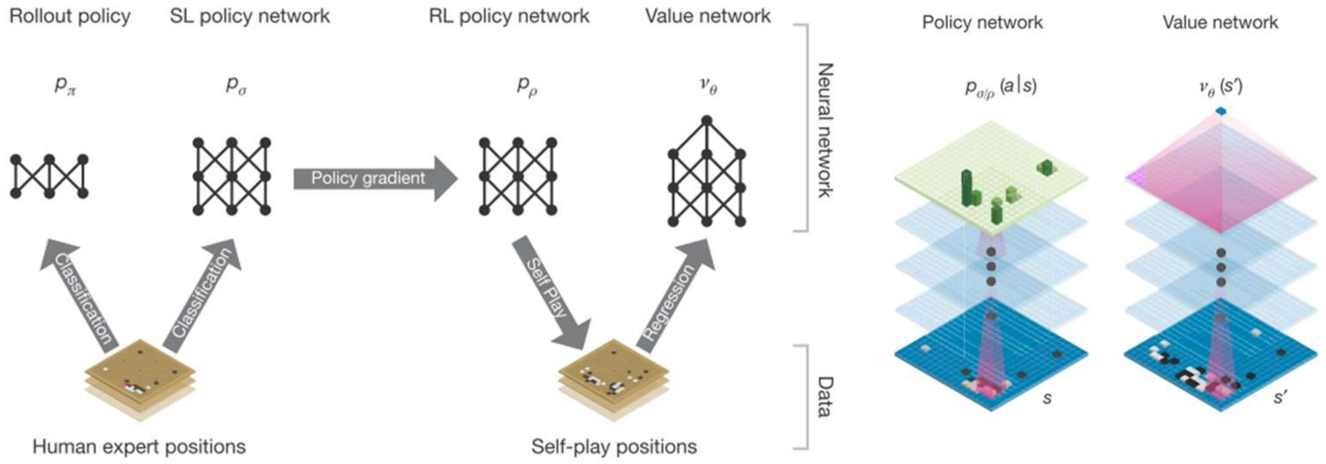
AlphaGo

#1 - Components of the Pipeline



AlphaGo

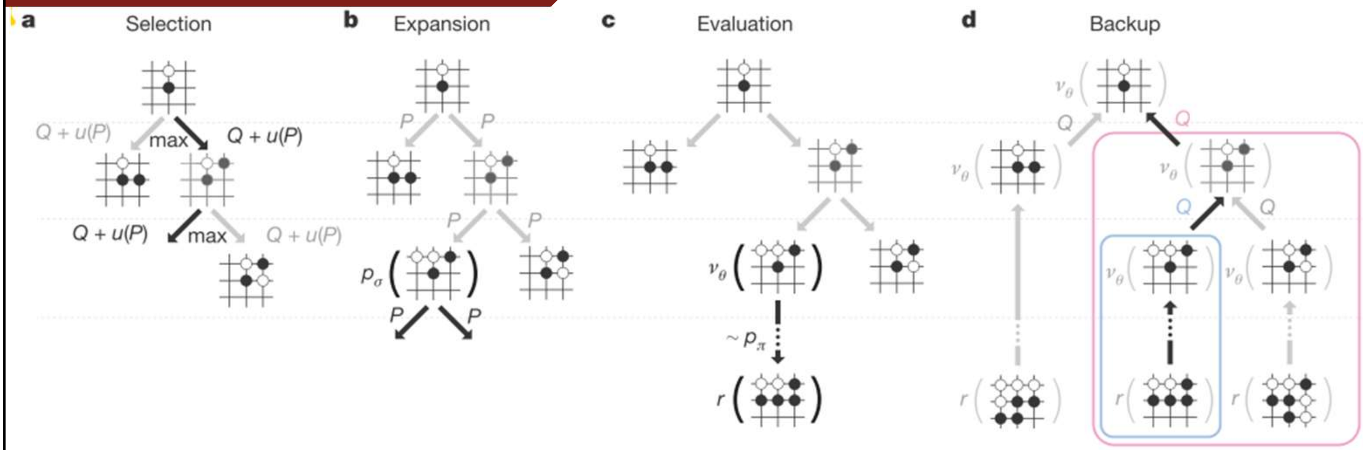
#1 - Components of the Pipeline



S. P. Vimal, Department of CSIS, WILP Division (vimalsp@wilp.bits-pilani.ac.in)

AlphaGo

#2 - MCTS in AlphaGo



S. P. Vimal, Department of CSIS, WILP Division (vimalsp@wilp.bits-pilani.ac.in)

AlphaGo Zero

To discuss:
- How AlphaGo Zero is different from AlphaGo?
- In what way AlphaGo Zero is significant to the field of AI?

ARTICLE

doi:10.1038/nature24270

Mastering the game of Go without human knowledge

David Silver[1*], Julian Schrittwieser[1*], Karen Simonyan[1*], Ioannis Antonoglou[1], Aja Huang[1], Arthur Guez[1], Thomas Hubert[1], Lucas Baker[1], Matthew Lai[1], Adrian Bolton[1], Yutian Chen[1], Timothy Lillicrap[1], Fan Hui[1], Laurent Sifre[1], George van den Driessche[1], Thore Graepel[1] & Demis Hassabis[1]

A long-standing goal of artificial intelligence is an algorithm that learns, *tabula rasa*, superhuman proficiency in challenging domains. Recently, AlphaGo became the first program to defeat a world champion in the game of Go. The tree search in AlphaGo evaluated positions and selected moves using deep neural networks. These neural networks were trained by supervised learning from human expert moves, and by reinforcement learning from self-play. Here we introduce an algorithm based solely on reinforcement learning, without human data, guidance or domain knowledge beyond game rules. AlphaGo becomes its own teacher: a neural network is trained to predict AlphaGo's own move selections and also the winner of AlphaGo's games. This neural network improves the strength of the tree search, resulting in higher quality move selection and stronger self-play in the next iteration. Starting *tabula rasa*, our new program AlphaGo Zero achieved superhuman performance, winning 100–0 against the previously published, champion-defeating AlphaGo.

AlphaGo Zero

#1 - How AlphaGo Zero is different?

1. Trained solely by self-play RL [self-play]
2. Only black & white board position as input
3. Uses only single neural network (policy & value)
4. Simpler tree search without Monte-carlo rollouts

Our program, AlphaGo Zero, differs from AlphaGo Fan and AlphaGo Lee[12] in several important aspects. First and foremost, it is trained solely by self-play reinforcement learning, starting from random play, without any supervision or use of human data. Second, it uses only the black and white stones from the board as input features. Third, it uses a single neural network, rather than separate policy and value networks. Finally, it uses a simpler tree search that relies upon this single neural network to evaluate positions and sample moves, without performing any Monte Carlo rollouts. To achieve these results, we introduce a new reinforcement learning algorithm that incorporates lookahead search inside the training loop, resulting in rapid improvement and precise and stable learning. Further technical differences in the search algorithm, training procedure and network architecture are described in Methods.

AlphaGo Zero

#1 - How AlphaGo Zero is different?

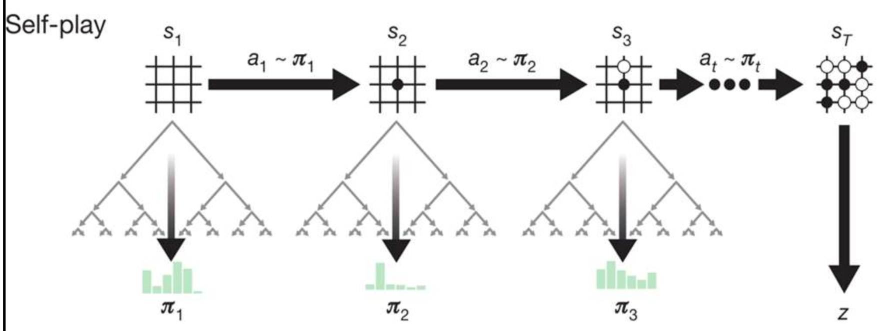
1. Trained solely by self-play RL [self-play]
2. Only black & white board position as input
3. Uses only single neural network (policy & value)
4. Simpler tree search without Monte-carlo rollouts

About the network:

Our new method uses a deep neural network f_θ with parameters θ. This neural network takes as an input the raw board representation s of the position and its history, and outputs both move probabilities and a value, $(\mathbf{p}, v) = f_\theta(s)$. The vector of move probabilities $\mathbf{p}$ represents the probability of selecting each move a (including pass), $p_a = \Pr(a|s)$. The value v is a scalar evaluation, estimating the probability of the current player winning from position s. This neural network combines the roles of both policy network and value network[12] into a single architecture. The neural network consists of many residual blocks[4] of convolutional layers[16,17] with batch normalization[18] and rectifier nonlinearities[19] (see Methods).

AlphaGo Zero

#2 - Self-play Pipeline

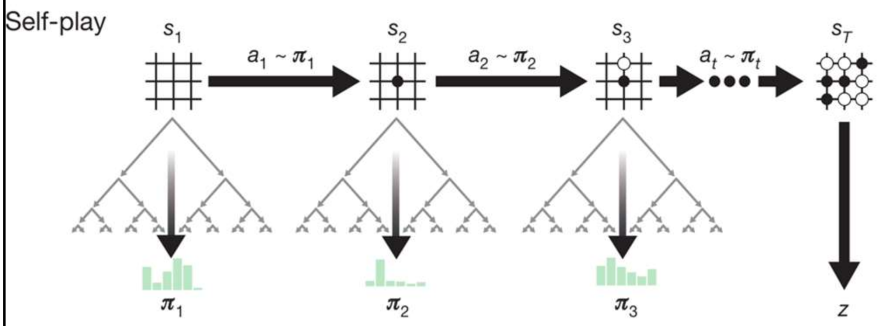


1. In each position st, an MCTS α_θ is executed using the latest neural network f_θ
1. Moves are selected according to the search probabilities computed by the MCTS,$a_t \sim \pi_t$.
2. The terminal position s_T is scored according to the rules of the game to compute the game winner z

AlphaGo Zero

#2 - Self-play Pipeline

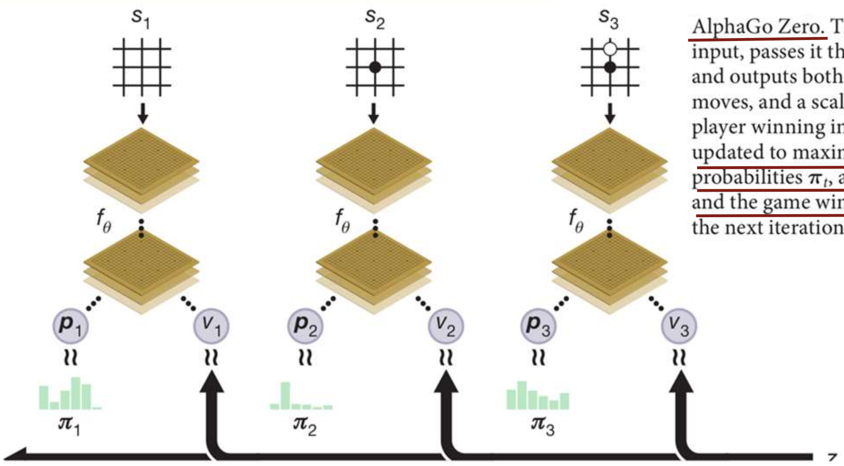
repeatedly in a policy iteration procedure[22,23]: the neural network's parameters are updated to make the move probabilities and value ($\boldsymbol{p}$, v) $= f_\theta(s)$ more closely match the improved search probabilities and self-play winner ($\boldsymbol{\pi}$, z); these new parameters are used in the next iteration of self-play to make the search even stronger. Figure 1 illustrates the self-play training pipeline.



1. In each position st, an MCTS α_θ is executed using the latest neural network f_θ
1. Moves are selected according to the search probabilities computed by the MCTS, $a_t \sim \pi_t$.
2. The terminal position s_T is scored according to the rules of the game to compute the game winner z

AlphaGo Zero

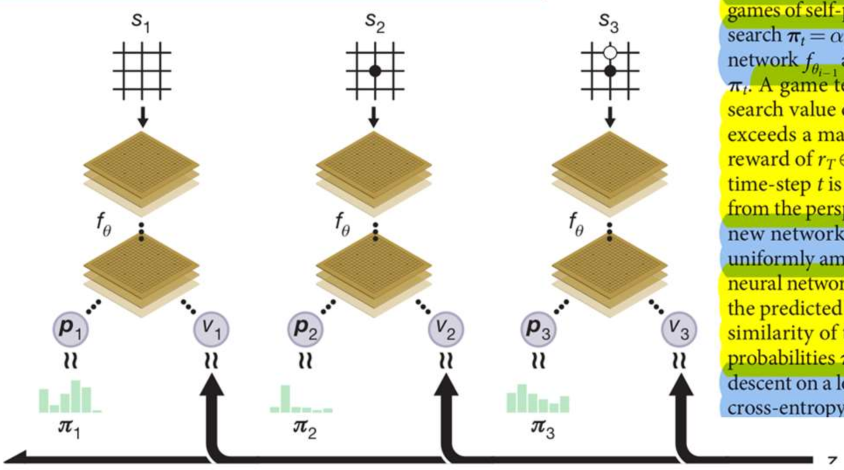
#3 - Neural Network Training



Neural network training in AlphaGo Zero. The neural network takes the raw board position s_t as its input, passes it through many convolutional layers with parameters θ, and outputs both a vector $\boldsymbol{p}_t$, representing a probability distribution over moves, and a scalar value v_t, representing the probability of the current player winning in position s_t. The neural network parameters θ are updated to maximize the similarity of the policy vector $\boldsymbol{p}_t$ to the search probabilities $\boldsymbol{\pi}_t$, and to minimize the error between the predicted winner v_t and the game winner z (see equation (1)). The new parameters are used in the next iteration of self-play as in **a**.

AlphaGo Zero

#3 - Neural Network Training



The neural network is trained by a self-play reinforcement learning algorithm that uses MCTS to play each move. First, the neural network is initialized to random weights θ_0. At each subsequent iteration $i \geq 1$, games of self-play are generated (Fig. 1a). At each time-step t, an MCTS search $\pi_t = \alpha_{\theta_{i-1}}(s_t)$ is executed using the previous iteration of neural network $f_{\theta_{i-1}}$ and a move is played by sampling the search probabilities π_t. A game terminates at step T when both players pass, when the search value drops below a resignation threshold or when the game exceeds a maximum length; the game is then scored to give a final reward of $r_T \in \{-1,+1\}$ (see Methods for details). The data for each time-step t is stored as (s_t, π_t, z_t), where $z_t = \pm r_T$ is the game winner from the perspective of the current player at step t. In parallel (Fig. 1b), new network parameters θ_i are trained from data (s, π, z) sampled uniformly among all time-steps of the last iteration(s) of self-play. The neural network $(\mathbf{p}, v) = f_{\theta_i}(s)$ is adjusted to minimize the error between the predicted value v and the self-play winner z, and to maximize the similarity of the neural network move probabilities $\mathbf{p}$ to the search probabilities π. Specifically, the parameters θ are adjusted by gradient descent on a loss function l that sums over the mean-squared error and cross-entropy losses, respectively:

S. P. Vimal, Department of CSIS, WILP Division (vimalsp@wilp.bits-pilani.ac.in)

AlphaGo Zero

#4 - MCTS in AlphaGo Zero

Fig. 2). Each edge (s, a) in the search tree stores a prior probability $P(s, a)$, a visit count $N(s, a)$, and an action value $Q(s, a)$. Each simulation

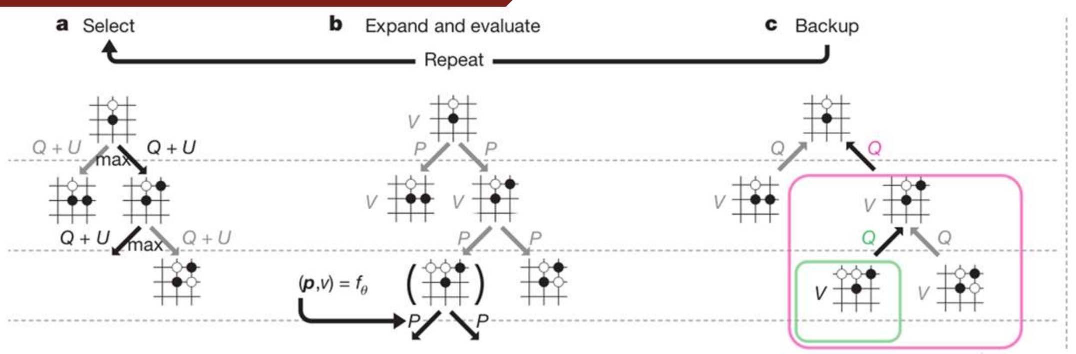


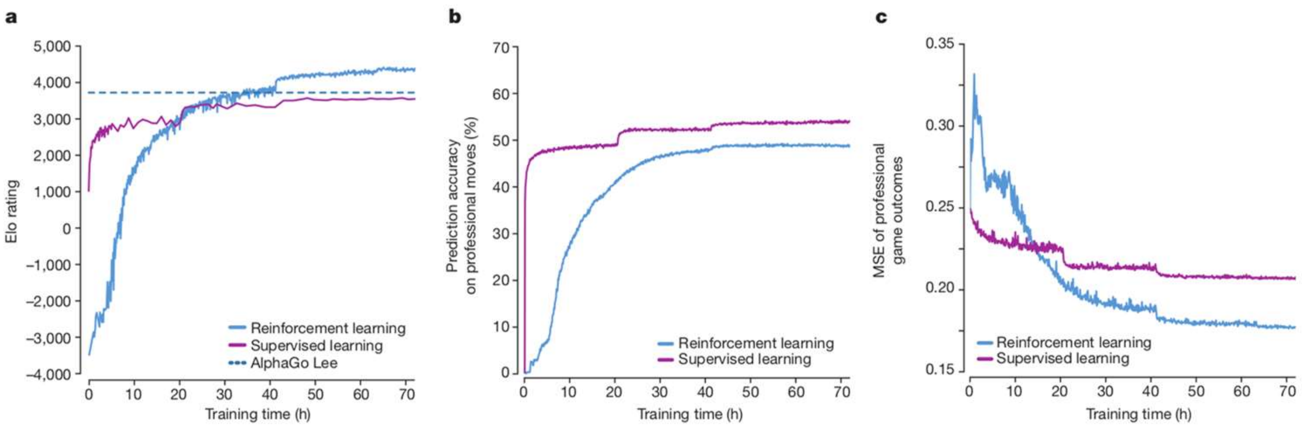
Figure 2 | MCTS in AlphaGo Zero. **a**, Each simulation traverses the tree by selecting the edge with maximum action value Q, plus an upper confidence bound U that depends on a stored prior probability P and visit count N for that edge (which is incremented once traversed). **b**, The leaf node is expanded and the associated position s is evaluated by the neural network $(P(s, \cdot), V(s)) = f_\theta(s)$; the vector of P values are stored in the outgoing edges from s. **c**, Action value Q is updated to track the mean of all evaluations V in the subtree below that action. **d**, Once the search is complete, search probabilities π are returned, proportional to $N^{1/\tau}$, where N is the visit count of each move from the root state and τ is a parameter controlling temperature.

S. P. Vimal, Department of CSIS, WILP Division (vimalsp@wilp.bits-pilani.ac.in)

AlphaGo Zero

#5 - Some Analysis

Over the course of training, 4.9 million games of self-play were generated, using 1,600 simulations for each MCTS, which corresponds to approximately 0.4 s thinking time per move. Parameters were updated from 700,000 mini-batches of 2,048 positions. The neural network contained 20 residual blocks (see Methods for further details).



AlphaGo Zero

#5 - Some Analysis

Conclusion

Our results comprehensively demonstrate that a pure reinforcement learning approach is fully feasible, even in the most challenging of domains: it is possible to train to superhuman level, without human examples or guidance, given no knowledge of the domain beyond basic rules. Furthermore, a pure reinforcement learning approach requires just a few more hours to train, and achieves much better asymptotic performance, compared to training on human expert data. Using this approach, AlphaGo Zero defeated the strongest previous versions of AlphaGo, which were trained from human data using handcrafted features, by a large margin.

MuZero

To discuss:

- How AlphaGo Zero is different from AlphaGo?
- In what way AlphaGo Zero is significant to the field of AI?

Mastering Atari, Go, Chess and Shogi by Planning with a Learned Model

Julian Schrittwieser,[1*] Ioannis Antonoglou,[1,2*] Thomas Hubert,[1*]
Karen Simonyan,[1] Laurent Sifre,[1] Simon Schmitt,[1] Arthur Guez,[1]
Edward Lockhart,[1] Demis Hassabis,[1] Thore Graepel,[1,2] Timothy Lillicrap,[1]
David Silver[1,2*]

[1]DeepMind, 6 Pancras Square, London N1C 4AG.
[2]University College London, Gower Street, London WC1E 6BT.
*These authors contributed equally to this work.

Abstract

Constructing agents with planning capabilities has long been one of the main challenges in the pursuit of artificial intelligence. Tree-based planning methods have enjoyed huge success in challenging domains, such as chess and Go, where a perfect simulator is available. However, in real-world problems the dynamics governing the environment are often complex and unknown. In this work we present the *MuZero* algorithm which, by combining a tree-based search with a learned model, achieves superhuman performance in a range of challenging and visually complex domains, without any knowledge of their underlying dynamics. *MuZero* learns a model that, when applied iteratively, predicts the quantities most directly relevant to planning: the reward, the action-selection policy, and the value function. When evaluated on 57 different Atari games - the canonical video game environment for testing AI techniques, in which model-based planning approaches have historically struggled - our new algorithm achieved a new state of the art. When evaluated on Go, chess and shogi, without any knowledge of the game rules, *MuZero* matched the superhuman performance of the *AlphaZero* algorithm that was supplied with the game rules.

A model that learns the game rules/environment

S. P. Vimal, Department of CSIS, WILP Division (vimalsp@wilp.bits-pilani.ac.in)

Required Readings and references

1. https://rl-lab.com/#play
2. https://www.aionlinecourse.com/tutorial/machine-learning/upper-confidence-bound-%28ucb%29
3. https://towardsdatascience.com/monte-carlo-tree-search-in-reinforcement-learning-b97d3e743d0f
4. https://gibberblot.github.io/rl-notes/single-agent/mcts.html
5. https://towardsdatascience.com/alphazero-chess-how-it-works-what-sets-it-apart-and-what-it-can-tell-us-4ab3d2d08867
6. https://medium.com/geekculture/muzero-explained-a04cb1bad4d4
7. https://towardsdatascience.com/everything-you-need-to-know-about-googles-new-planet-reinforcement-learning-network-144c2ca3f284
8. https://blog.research.google/2019/02/introducing-planet-deep-planning.html?m=1

BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Thank you