Università degli
Studi di Pavia

Deep Learning

13 - AlphaZero

Marco Piastra & Andrea Pedrini(*)

(*) Dipartimento di Matematica F. Casorati

This presentation can be downloaded at:
http://vision.unipv.it/DL

Playing Games with Trees

Tree representation

- **Game Tree:**

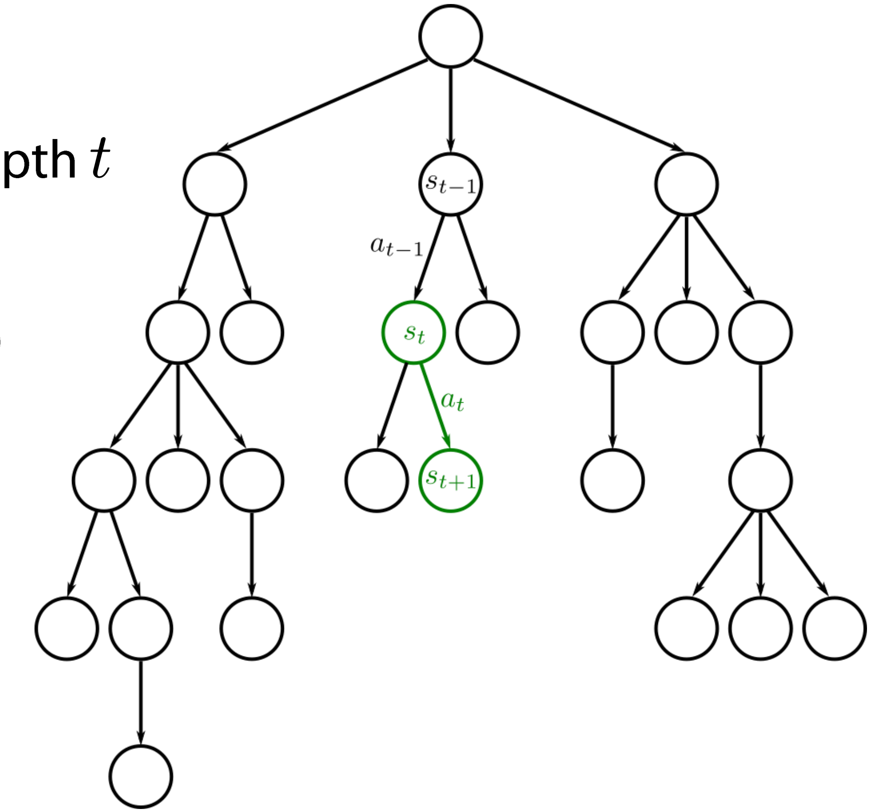
  The *current state* s_t at time t is a ***node*** with depth t

  Any admissible *action* a_t is an ***edge*** of the tree

  (branching factor = number of admissible actions in a state)

  State s_{t+1} obtained from s_t after executing a_t is determined by a *transition function*

$$\tau : \ (s_t, a_t) \mapsto s_{t+1}$$



Tree representation

- **Game Tree:**

The *current state* s_t at time t is a ***node*** with depth t

Any admissible *action* a_t is an ***edge*** of the tree

(branching factor = number of admissible actions in a state)

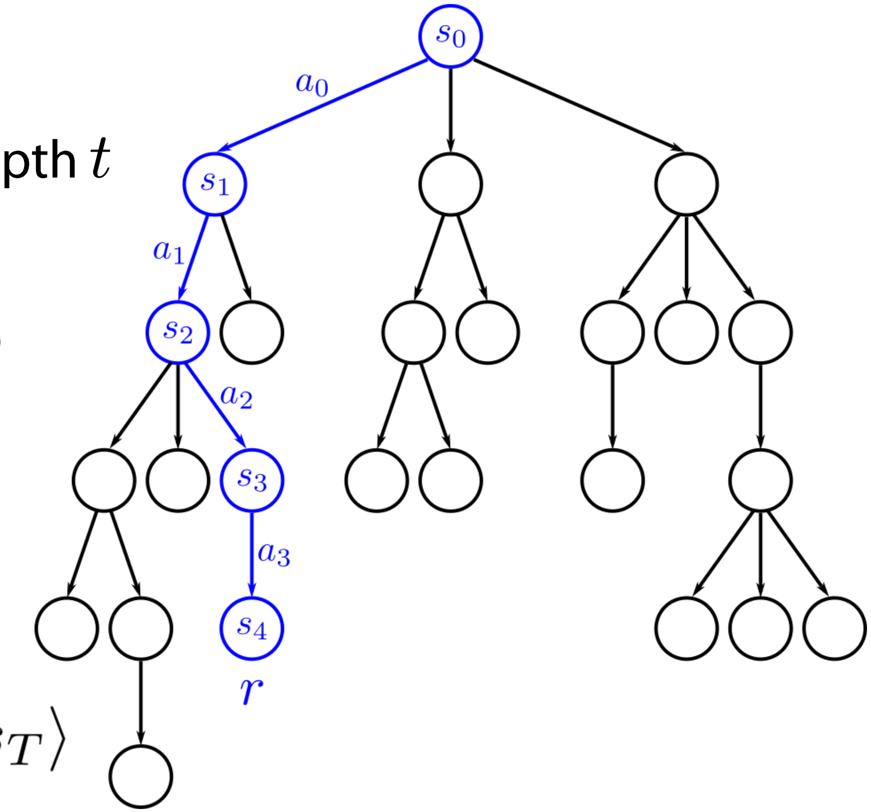
State s_{t+1} obtained from s_t after executing a_t is determined by a *transition function*

$$\tau : \ (s_t, a_t) \mapsto s_{t+1}$$

A *playout* is a ***path*** $\langle s_0, a_0, s_1, \ldots, a_{T-1}, s_T \rangle$ from the *initial state* s_0 to a *terminal state* s_T

A *reward* r is the outcome of a playout

A *policy* is a map $\pi : \ s \mapsto a$ which selects action a to be executed in state s



Policy optimization

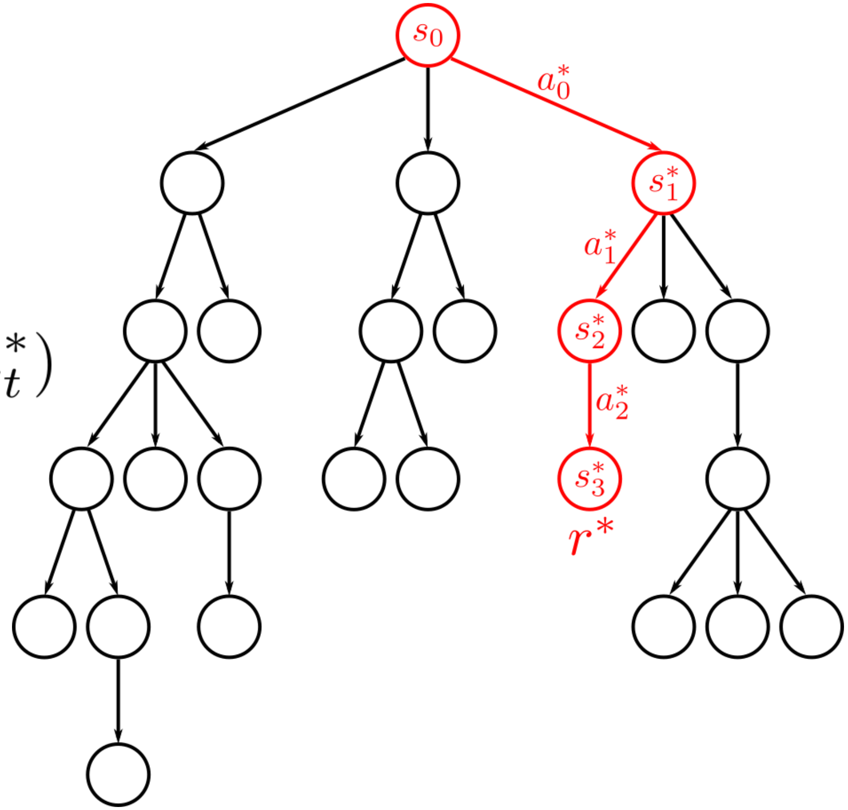
- Goal: finding the *best policy* π^*

  such that the reward r^* of playout

  $$\langle s_0, a_0^*, s_1^*, \ldots, a_{T-1}^*, s_T^* \rangle$$

  with $a_{t+1}^* := \pi^*(s_t^*)$ and $s_{t+1}^* := \tau(s_t^*, a_t^*)$

  is ***maximal***



"Brute Force": a simple (bad) policy optimization

- Goal: finding the <u>*best policy*</u> π^*

- ***"Brute Force"***:
  1. explore the entire tree by following **all** possible paths
  2. select the path(s) with the best outcome (and randomly choose one of them)
  3. play by following the policy associated with that path

- Problems:

  Huge game tree with infeasible full exploration

  (branching factor in Go is around 200)

  Infinitely many admissible actions

  Intrinsic ***stochasticity*** and ***uncertainty*** after playing an action

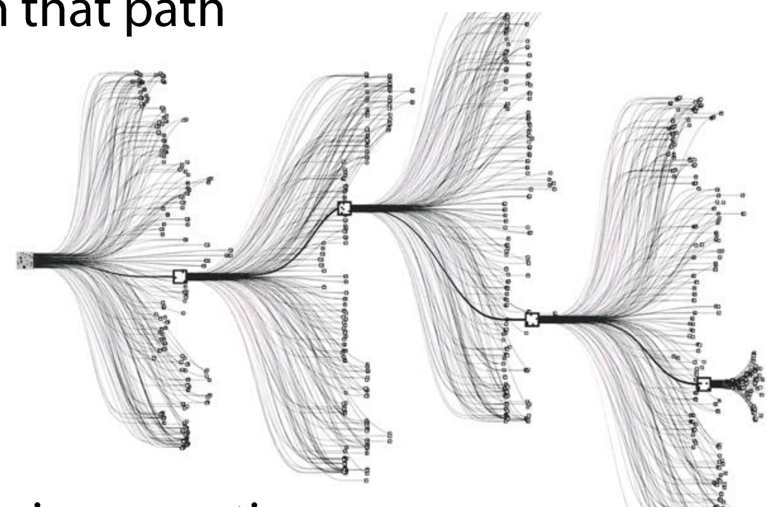


Image from https://thenewstack.io/google-ai-beats-human-champion-complex-game-ever-invented/

Stochasticity and Uncertainty: examples

- ***Multi-armed bandits***

  i.e. which arm to play

  The reward after each action is *stochastic*

  random variable

  probability of reward r for action a

$$Q(s,a) := \mathbb{E}[R \mid s,a] = \sum_r r\, P(r \mid s,a)$$

Q-value (expected reward of action a performed in state s)

- ***Games with two players (White and Black):***

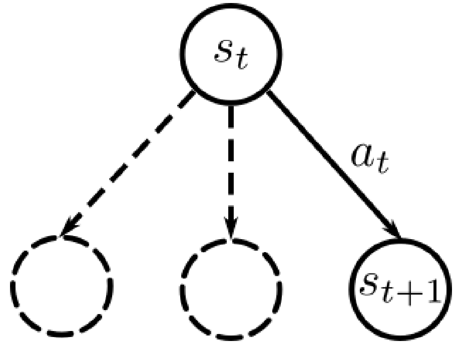
  White plays action a_t in state s_t
  but her next state s_{t+1} depends on Black's next action

  Uncertainty of execution:
  nondeterministic $\tau : (s_t, a_t) \mapsto s_{t+1}$ with $P(s_{t+1} \mid s_t, a_t)$

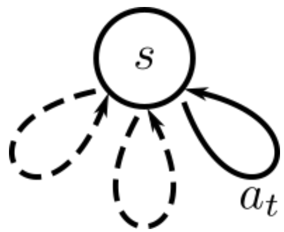
  transition function *probability transition distribution*

Stochasticity and Uncertainty: tree representation

- ***Simplest case scenario***
  - deterministic transition
  - deterministic reward

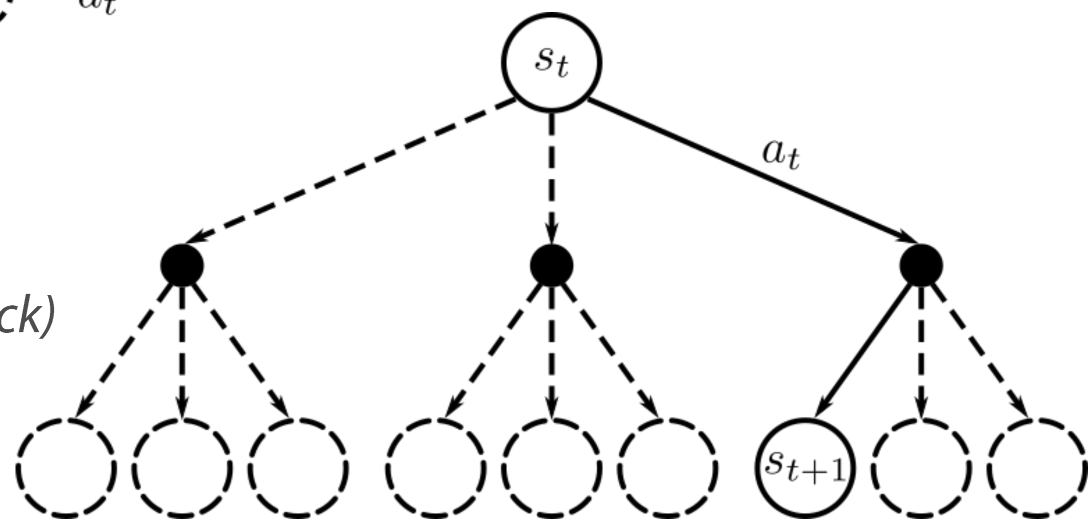


- ***Multi-armed bandits***
  - deterministic transition
  - stochastic reward



Actually, this is not a tree! (but it can be expanded and became one)

- ***Uncertainty of execution:***
  - stochastic transition
  - either deterministic *(White vs Black)* or stochastic reward



Monte Carlo method: step by step simulations

Monte Carlo (MC) step

- Goal: finding the *best policy* π^* *(avoiding brute-force approach)*

  It can be done iteratively, by focusing on the single best action $a^* =: \pi^*(s)$ in the current state s

- ***Monte Carlo (MC)*** *step:* [Abramson 1990]

repeat n times {
1) play a *pseudo-random playout* from current state s
2) compute and save the reward r obtained at the end of the playout
}

3) for each admissible action a in state s compute the mean of the rewards

$$\hat{Q}(s,a) := \frac{1}{N(s,a)} \sum_{i=1}^{N(s,a)} r_{a,i}$$

estimates $Q(s,a)$

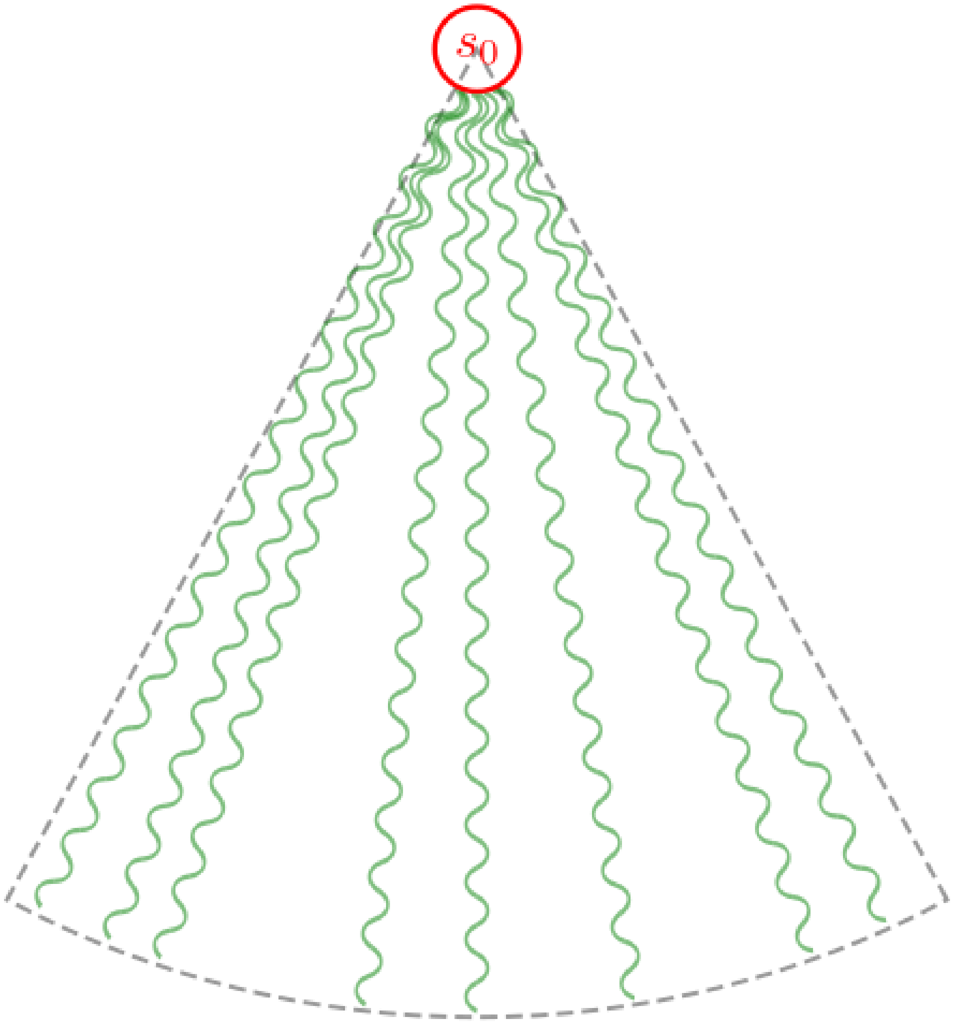
number of playouts with first action a

reward of i^{th} playout with first action a

4) $a^* := \text{argmax}_a \hat{Q}(s,a)$ is the action with the highest mean

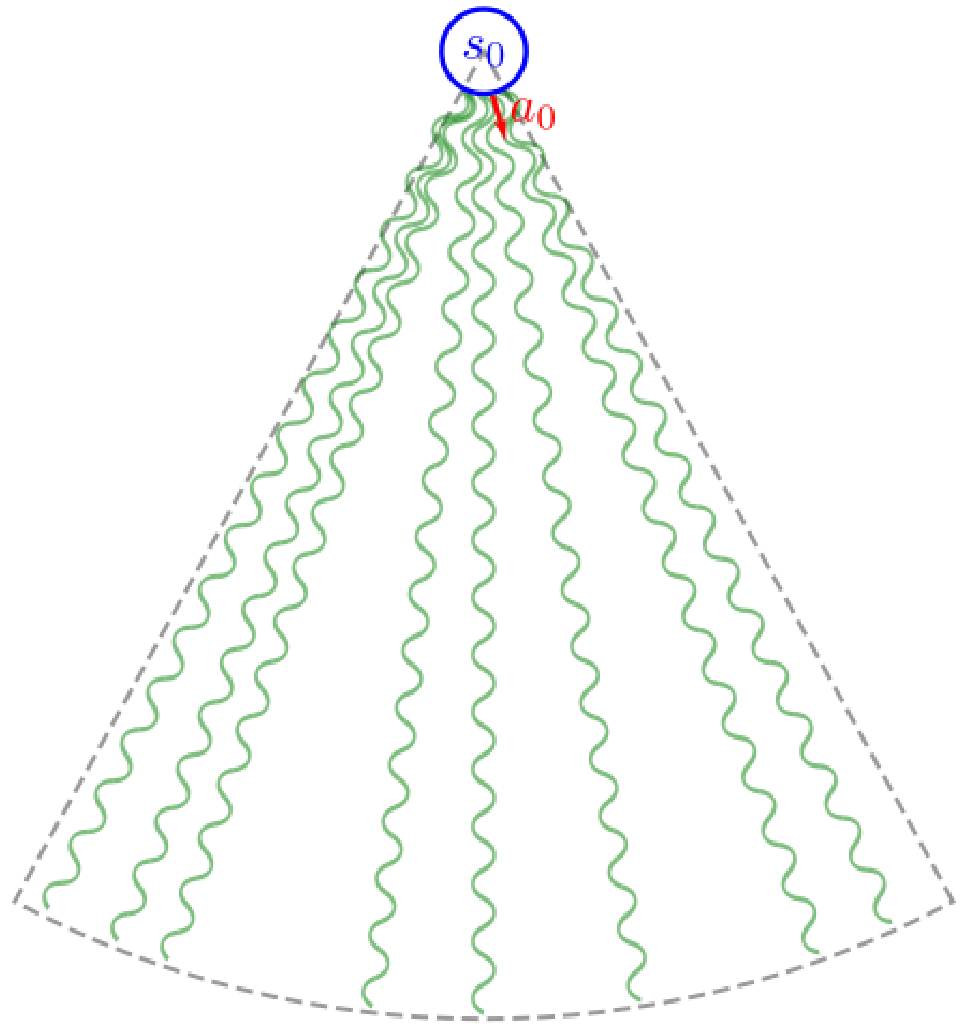
Monte Carlo episode

- ***Monte Carlo*** *episode:*
  1) set t:=0
  2) current state s:=s_t
  3) use *MC step* to decide a_t
  4) compute $s_{t+1} := \tau(s_t, a_t)$
  5) set t:=t+1
  6) repeat 2) to 5) until end game



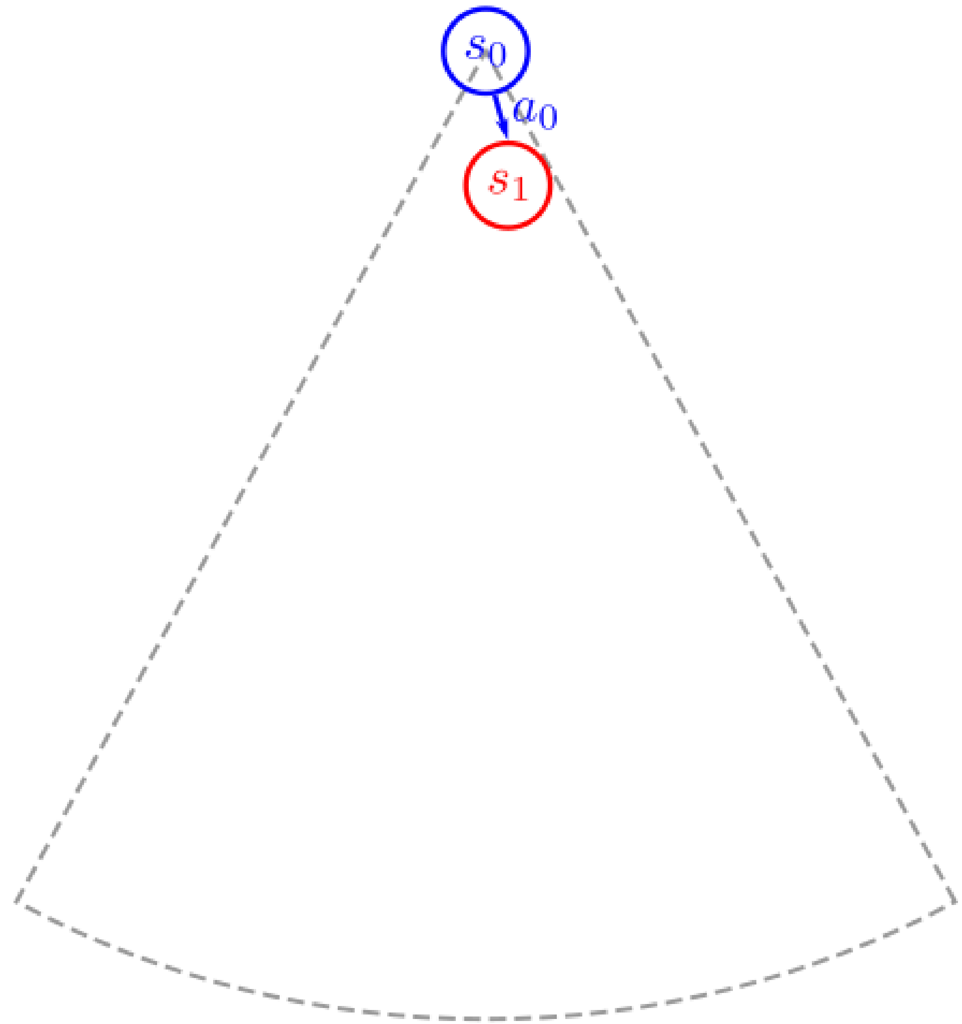
Monte Carlo episode

- ***Monte Carlo** episode:*
  1) set $t{:=}0$
  2) current state $s{:=}s_t$
  3) use *MC step* to decide a_t
  4) compute $s_{t+1} := \tau(s_t, a_t)$
  5) set $t{:=}t{+}1$
  6) repeat 2) to 5) until end game



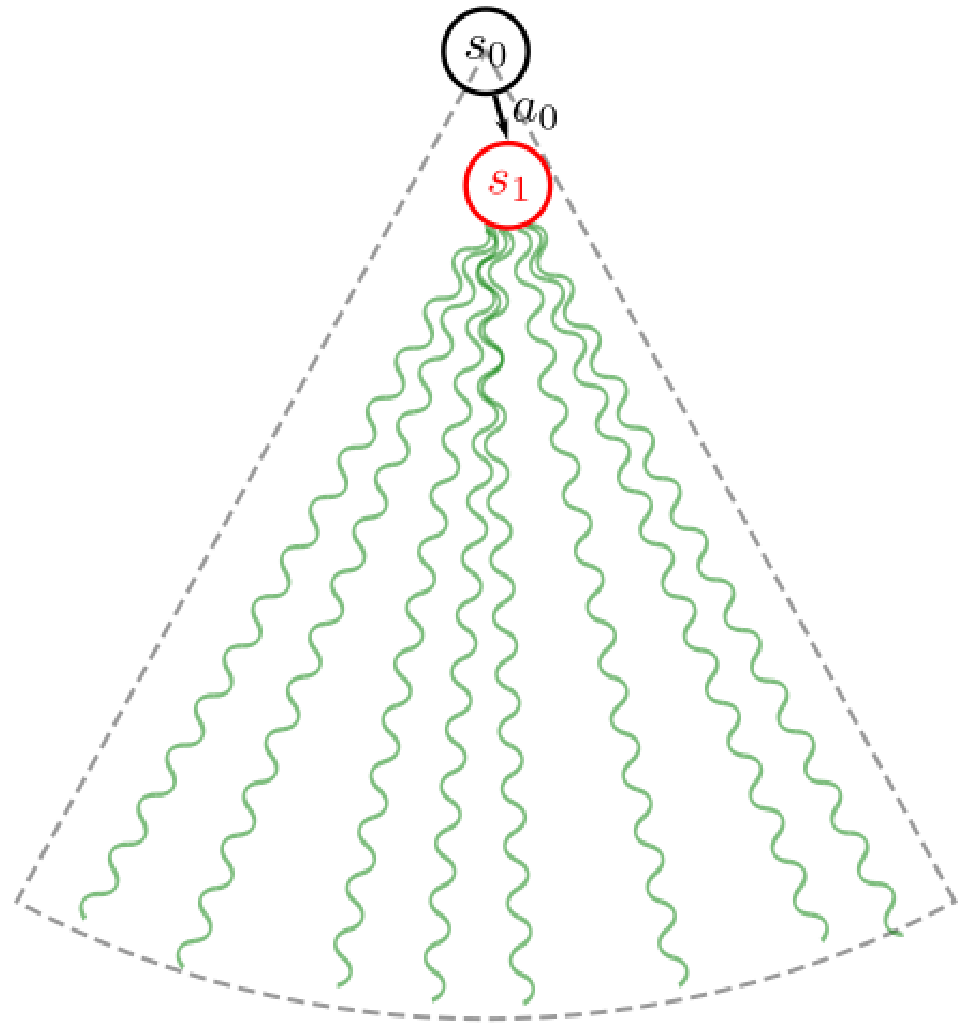
Monte Carlo episode

- ***Monte Carlo*** *episode:*
  1) set $t{:=}0$
  2) current state $s{:=}s_t$
  3) use *MC step* to decide a_t
  4) compute $s_{t+1} := \tau(s_t, a_t)$
  5) set $t{:=}t{+}1$
  6) repeat 2) to 5) until end game



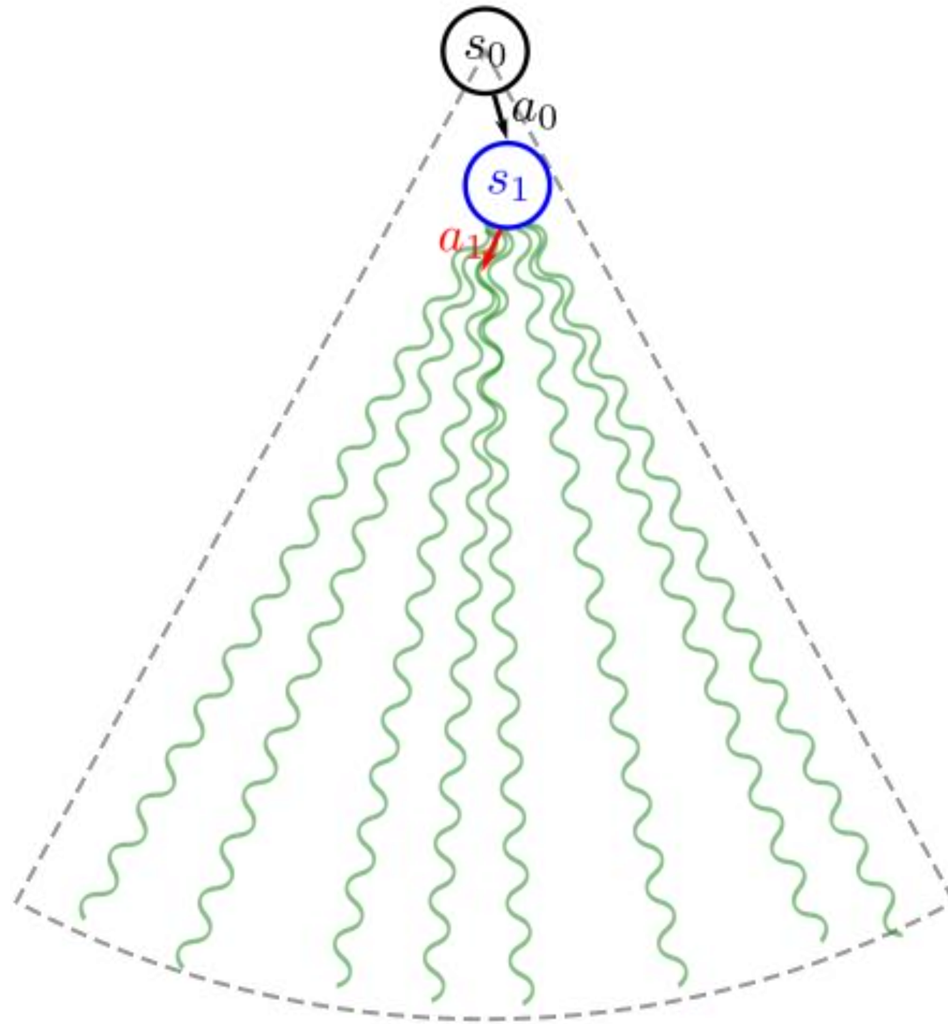
Monte Carlo episode

- ***Monte Carlo** episode:*
  1) set $t := 0$
  2) current state $s := s_t$
  3) use *MC step* to decide a_t
  4) compute $s_{t+1} := \tau(s_t, a_t)$
  5) set $t := t+1$
  6) repeat 2) to 5) until end game



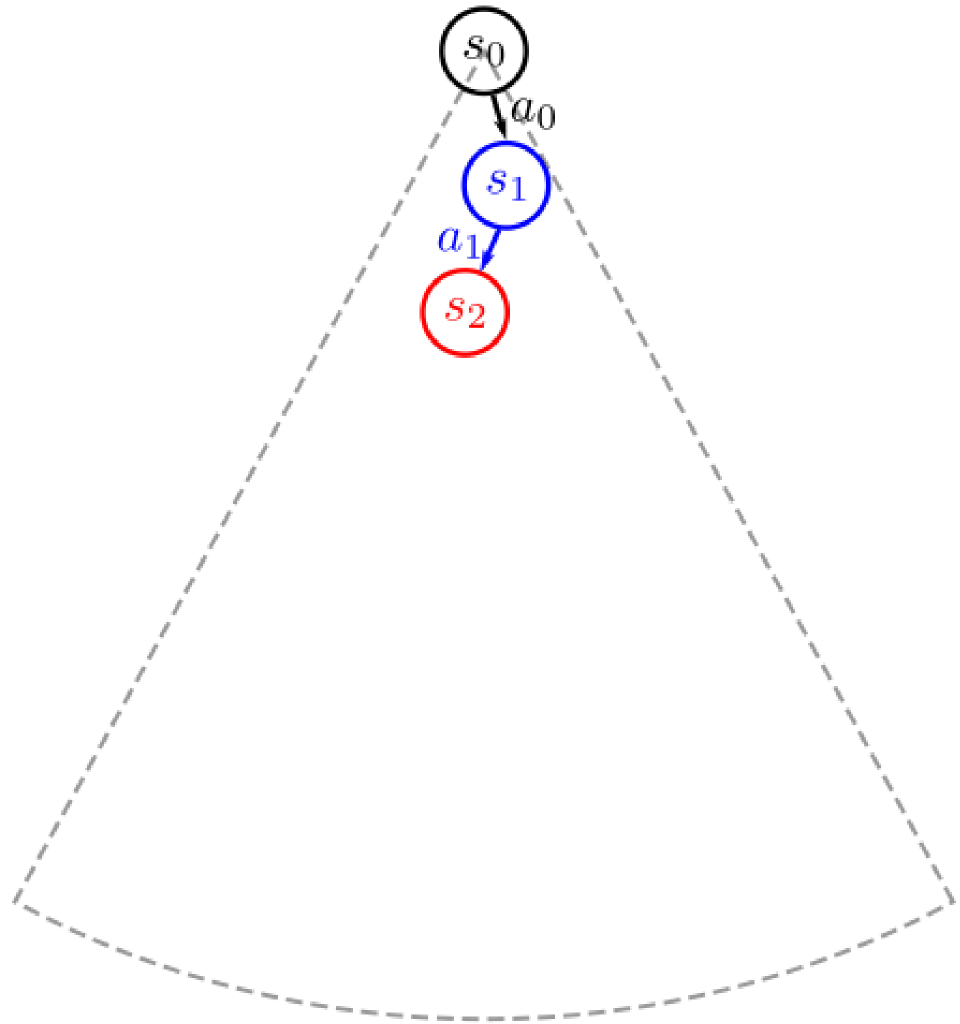
Monte Carlo episode

- ***Monte Carlo** episode:*
  1) set $t := 0$
  2) current state $s := s_t$
  3) use *MC step* to decide a_t
  4) compute $s_{t+1} := \tau(s_t, a_t)$
  5) set $t := t+1$
  6) repeat 2) to 5) until end game



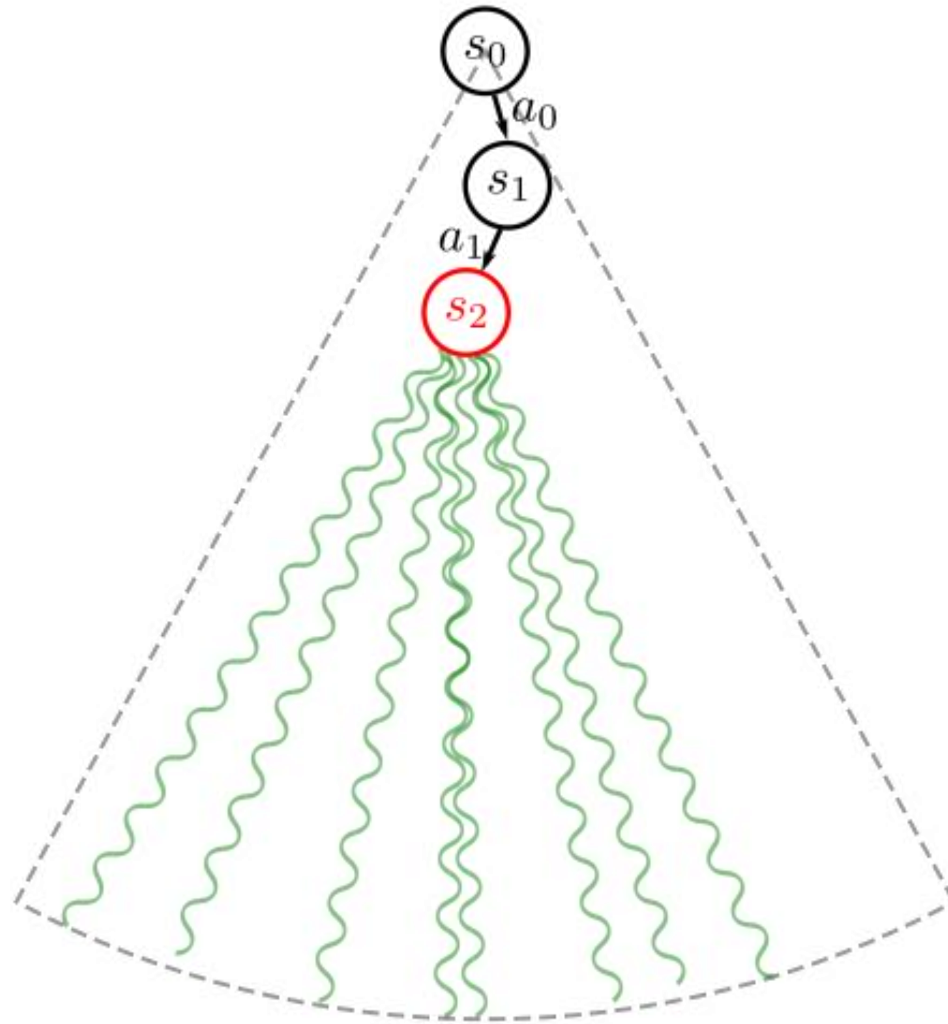
Monte Carlo episode

- ***Monte Carlo** episode:*
  1) set t:=0
  2) current state s:=s_t
  3) use *MC step* to decide a_t
  4) compute $s_{t+1} := \tau(s_t, a_t)$
  5) set t:=t+1
  6) repeat 2) to 5) until end game



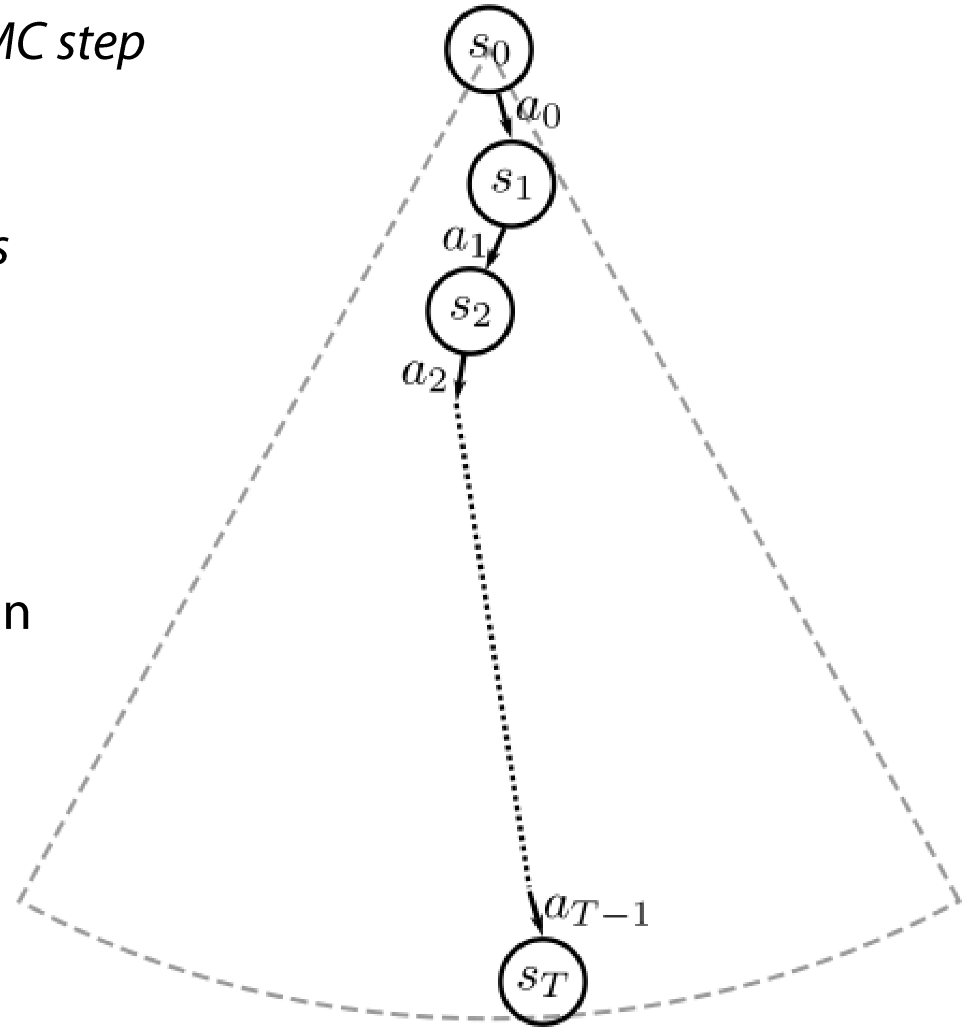
Monte Carlo episode

- ***Monte Carlo*** *episode:*
  1) set $t{:=}0$
  2) current state $s{:=}s_t$
  3) use *MC step* to decide a_t
  4) compute $s_{t+1} := \tau(s_t, a_t)$
  5) set $t{:=}t+1$
  6) repeat 2) to 5) until end game



Monte Carlo method

- ***Monte Carlo*** *method:*
  - *<u>no memory</u>* of past playouts in a single *MC step*
    (only the reward is saved)
  - <u>no *transfer knowledge*</u> between *MC steps*
  - *<u>no construction</u>* of game subtree
  - optimal policy only *<u>partially</u>* defined
    (on actually computed states)
  - *<u>intrinsically stochastic</u>* policy optimization
    (the same initial state
    can give rise to different optimizations)
  - *<u>no knowledge transfer</u>*
    between *MC episodes*

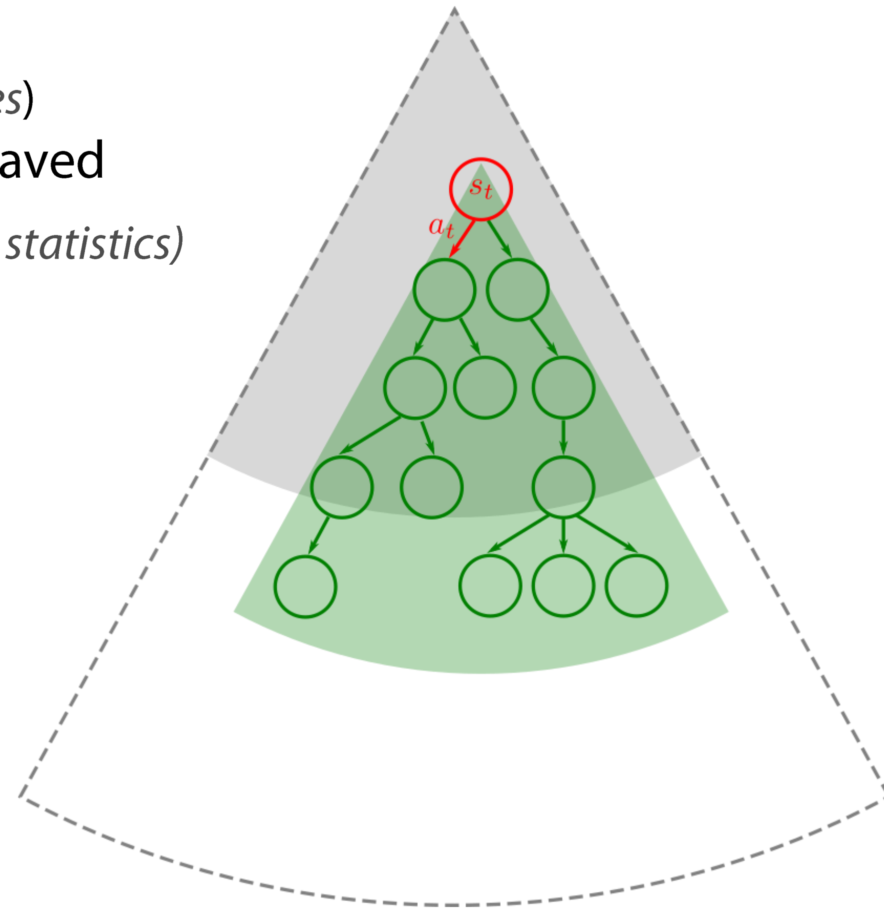


Monte Carlo Tree Search (MCTS):

simulation + partial expansion

MCTS episode: basic idea

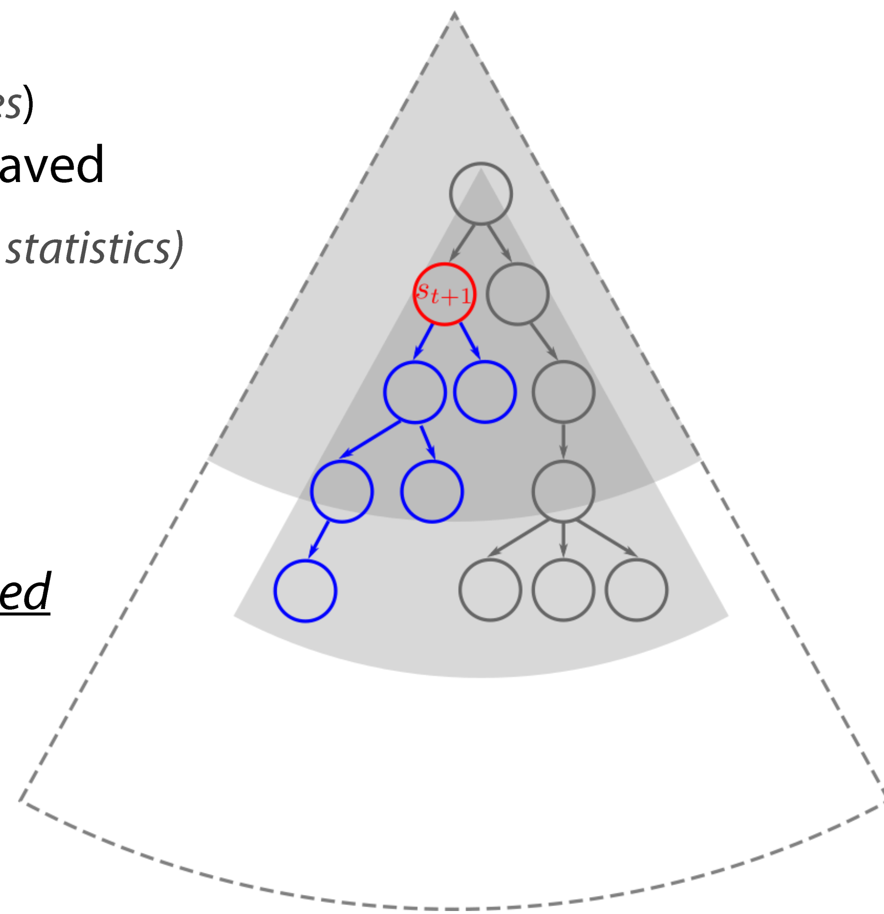
- *At each step (with current state s_t):*
  - a <u>*subgraph*</u> G_t with root s_t is created
  - <u>*statistics*</u> (*number of visits* and *estimate outcomes*) for states and actions in the subgraph are saved
  - best action a_t is decided *(accordingly to those statistics)*
  - next state $s_{t+1} := \tau(s_t, a_t)$ is computed



MCTS episode: basic idea

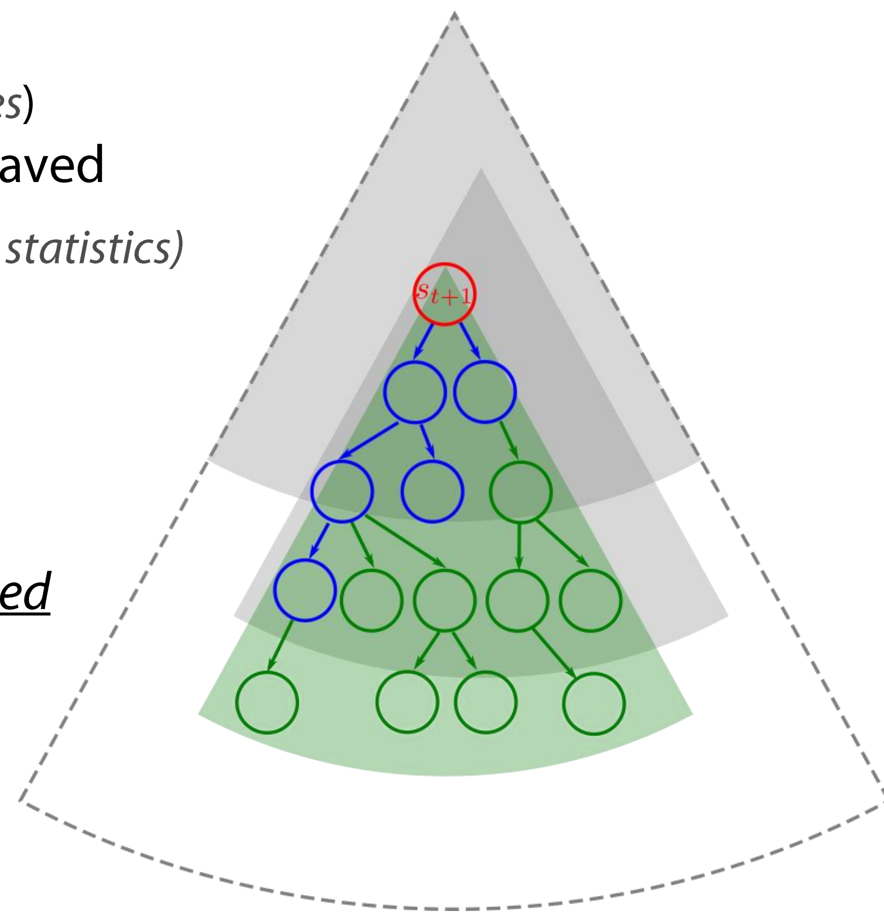
- *At each step (with current state s_t):*
  - a ***subgraph*** G_t with root s_t is created
  - ***statistics*** (*number of visits* and *estimate outcomes*) for states and actions in the subgraph are saved
  - best action a_t is decided *(accordingly to those statistics)*
  - next state $s_{t+1} := \tau(s_t, a_t)$ is computed

- *In the next step (with current state s_{t+1}):*
  - the subgraph of G_t with root s_{t+1} is ***expanded*** to create G_{t+1}
  - the statistics are ***updated*** and saved
  - best action a_{t+1} is decided
  - next state $s_{t+1} := \tau(s_t, a_t)$ is computed



MCTS episode: basic idea

- *At each step (with current state s_t):*
  - a ***subgraph*** G_t with root s_t is created
  - ***statistics*** (*number of visits* and *estimate outcomes*) for states and actions in the subgraph are saved
  - best action a_t is decided *(accordingly to those statistics)*
  - next state $s_{t+1} := \tau(s_t, a_t)$ is computed

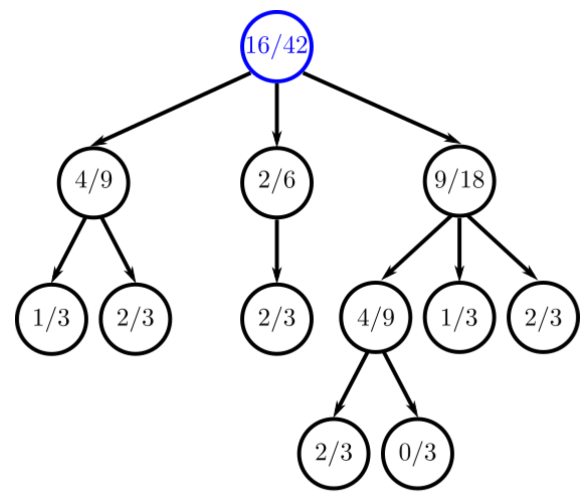


- *In the next step (with current state s_{t+1}):*
  - the subgraph of G_t with root s_{t+1} is ***expanded*** to create G_{t+1}
  - the statistics are ***updated*** and saved
  - best action a_{t+1} is decided
  - next state $s_{t+1} := \tau(s_t, a_t)$ is computed

Monte Carlo Tree Search (MCTS) step

- ***Monte Carlo Tree Search (MCTS)*** *step:* [Coulom 2006]

    1) start from current state s *(and the –possibly empty– stored tree with root s)*



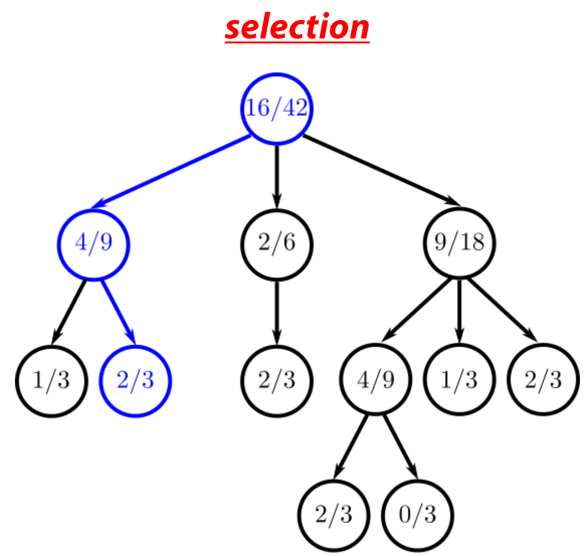
Monte Carlo Tree Search (MCTS) step

- ***Monte Carlo Tree Search (MCTS)*** *step:* [Coulom 2006]

1) start from current state s *(and the –possibly empty– stored tree with root* s*)*
2) traverse the tree by following the <u>*selection policy*</u>

$$\pi^{\mathrm{sel}} : \; s_t \mapsto a_t$$

until encountering a *leaf node* s_L (i.e. a state not stored in the tree)

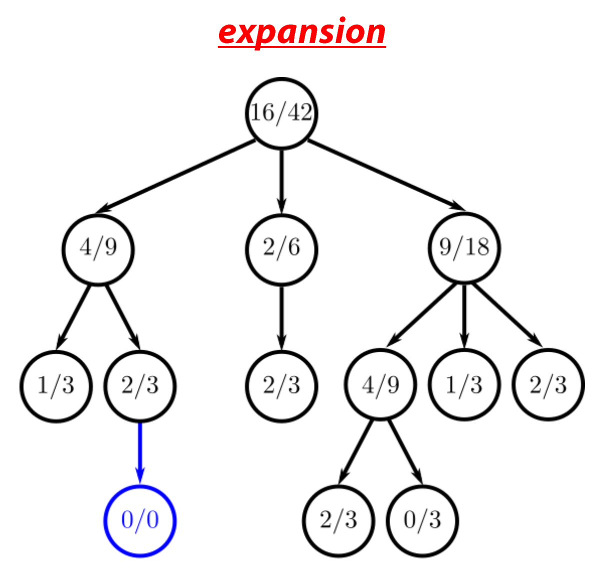


- ***Monte Carlo Tree Search (MCTS)*** *step:* [Coulom 2006]

  1) start from current state s *(and the –possibly empty– stored tree with root s)*
  2) traverse the tree by following the *selection policy*

  $$\pi^{\text{sel}} : \; s_t \mapsto a_t$$

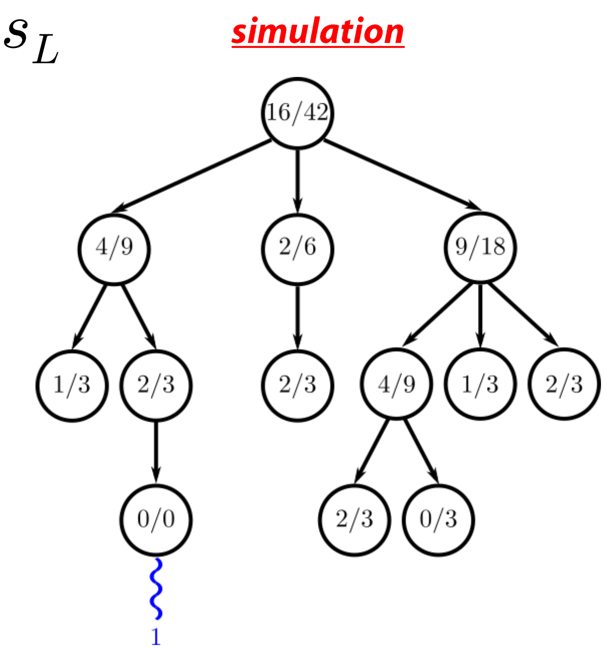
     until encountering a *leaf node* s_L (i.e. a state not stored in the tree)
  3) *expand* the tree by adding s_L



Monte Carlo Tree Search (MCTS) step

- ***Monte Carlo Tree Search (MCTS)*** *step:* [Coulom 2006]

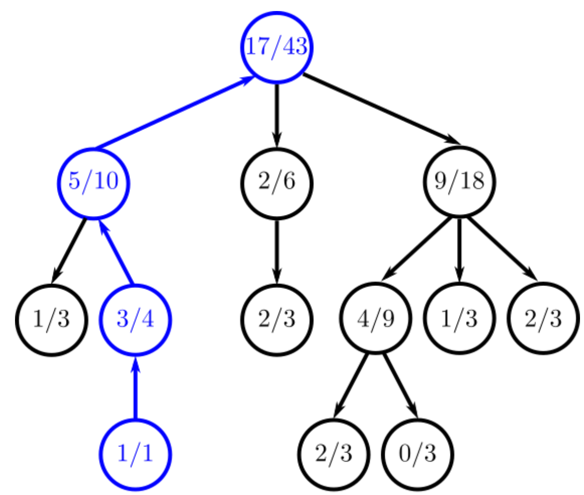
1) start from current state s *(and the –possibly empty– stored tree with root s)*
2) traverse the tree by following the *selection policy*
$$\pi^{\mathrm{sel}} : \; s_t \mapsto a_t$$
until encountering a *leaf node* s_L (i.e. a state not stored in the tree)
3) *expand* the tree by adding s_L
4) play one pseudo-random playout from state s_L by following the *simulation policy*
$$\pi^{\mathrm{sym}} : \; s_t \mapsto a_t$$
and obtain the reward r



- ***Monte Carlo Tree Search (MCTS)*** *step:* [Coulom 2006]

  1) start from current state s *(and the –possibly empty– stored tree with root s)*
  2) traverse the tree by following the *selection policy*
  $$\pi^{\text{sel}} : \; s_t \mapsto a_t$$
  until encountering a *leaf node* s_L (i.e. a state not stored in the tree)
  3) *expand* the tree by adding s_L
  4) play one pseudo-random playout from state s_L by following the *simulation policy*
  $$\pi^{\text{sym}} : \; s_t \mapsto a_t$$
  and obtain the reward r
  5) *backpropagate* r (and update the statistics of each encountered state and action)

backpropagation



Monte Carlo Tree Search (MCTS) step

- ***Monte Carlo Tree Search (MCTS)*** *step:* [Coulom 2006]

repeat m times:

1) start from current state s *(and the –possibly empty– stored tree with root s)*
2) traverse the tree by following the ***selection policy***

$$\pi^{\mathrm{sel}} : \ s_t \mapsto a_t$$

until encountering a *leaf node* s_L (i.e. a state not stored in the tree)

3) ***expand*** the tree by adding s_L

repeat n times:

4) play one pseudo-random playout from state s_L by following the ***simulation policy***

$$\pi^{\mathrm{sym}} : \ s_t \mapsto a_t$$

and obtain the reward r

5) ***backpropagate*** r (and update the statistics of each encountered state and action)

Monte Carlo Tree Search (MCTS) step

- ***Monte Carlo Tree Search (MCTS)*** *step:* [Coulom 2006]

repeat m times:

1) start from current state s *(and the –possibly empty– stored tree with root s)*
2) traverse the tree by following the *selection policy*

$$\pi^{\text{sel}} : \; s_t \mapsto a_t$$

until encountering a *leaf node* s_L (i.e. a state not stored in the tree)

3) *expand* the tree by adding s_L

repeat n times:

4) play one pseudo-random playout from state s_L by following the *simulation policy*

$$\pi^{\text{sym}} : \; s_t \mapsto a_t$$

and obtain the reward r

5) *backpropagate* r (and update the statistics of each encountered state and action)

6) decide the *best* action to be performed in s with the *greedy policy*

$$\pi^{\text{gre}} : \; s \mapsto a$$

MCTS statistics: expansion and backpropagation

- ***MCTS** statistics* for state s and action a:

  $N(s)$ = total number of times state s has been visited

  $N(s, a)$ = number of times action a has been selected in state s

  $\hat{Q}(s, a)$ = estimated outcome of action a when selected in state s

- *<u>Expansion</u> initialization:* $N(s) := 0, \quad N(s,a) := 0, \quad \hat{Q}(s,a) := 0$

- *<u>Backpropagation</u> update* after a single playout with reward r:

$$N(s) := N(s) + 1$$
$$N(s,a) := N(s,a) + 1$$
$$\hat{Q}(s,a) := \hat{Q}(s,a) + \frac{r - \hat{Q}(s,a)}{N(s,a)}$$

MCTS: greedy, selection and simulation policies

- Greedy policy:

$$\pi^{\text{gre}}(s) := \underset{N(s,a)>0}{\text{argmax}}\ \hat{Q}(s,a)$$

- Selection policy: ***Upper Confidence Bound applied to Trees (UCT)***

$$\pi^{\text{sel}}(s) := \pi^{\text{UCT}}(s) := \underset{N(s,a)>0}{\text{argmax}} \left\{ \hat{Q}(s,a) + c\sqrt{\frac{2\log N(s)}{N(s,a)}} \right\}$$

parameter (default=1)

exploitation of actions that look currently the best

exploration of currently suboptimal-looking actions (no good alternatives are missed because of early estimation errors)

Convergence [Kocsis 2006]: for the first state s of a single *MCTS* episode

$$\pi^{\text{UCT}}(s) \to a^* := \pi^*(s) \quad \text{for } n \to +\infty$$

MCTS: greedy, selection and simulation policies

- *Greedy* policy:

$$\pi^{\text{gre}}(s) := \underset{N(s,a)>0}{\text{argmax}}\, \hat{Q}(s,a)$$

- *Selection* policy: ***Upper Confidence Bound applied to Trees (UCT)***

$$\pi^{\text{sel}}(s) := \pi^{\text{UCT}}(s) := \underset{N(s,a)>0}{\text{argmax}} \left\{ \hat{Q}(s,a) + c\sqrt{\frac{2\log N(s)}{N(s,a)}} \right\}$$

- *Simulation* policy: ***Random Uniform Policy***

$$\pi^{\text{sym}}(s) := a \qquad \text{with } P(s,a) = \frac{1}{|\mathcal{A}(s)|}$$

set of admissible actions in state s

Monte Carlo Tree Search (MCTS) step

Algorithm 2 UCT

procedure UCTSEARCH(s_0)
  while time remaining **do**
    $\{s_0, ..., s_T\}, R = \text{SIMULATE}(s_0)$
    $\text{BACKUP}(\{s_0, ..., s_T\}, R)$
  end while
  return $\underset{a \in \mathcal{A}}{\text{argmax}}\ Q(s_0, a)$
end procedure

procedure SIMULATE(s_0)
  $t = 0$
  $R = 0$
  repeat
    if $s_t \in \mathcal{T}$ **then**
      $a = \text{UCB1}(s_t)$
    else
      $\text{NEWNODE}(s_t)$
      $a_t = \text{DEFAULTPOLICY}(s_t)$
    end if
    $s_{t+1} = \text{SAMPLETRANSITION}(s_t, a_t)$
    $r_{t+1} = \text{SAMPLEREWARD}(s_t, a_t, s_{t+1})$
    $R = R + r_{t+1}$
    $t \mathrel{+}= 1$
  until $Terminal(s_t)$
  return $\{s_0, ..., s_t\}, R$
end procedure

procedure UCB1(s)
  $a^* = \underset{a}{\text{argmax}}\ Q(s,a) + c\sqrt{\frac{2\log N(s)}{N(s,a)}}$
  return a^*
end procedure

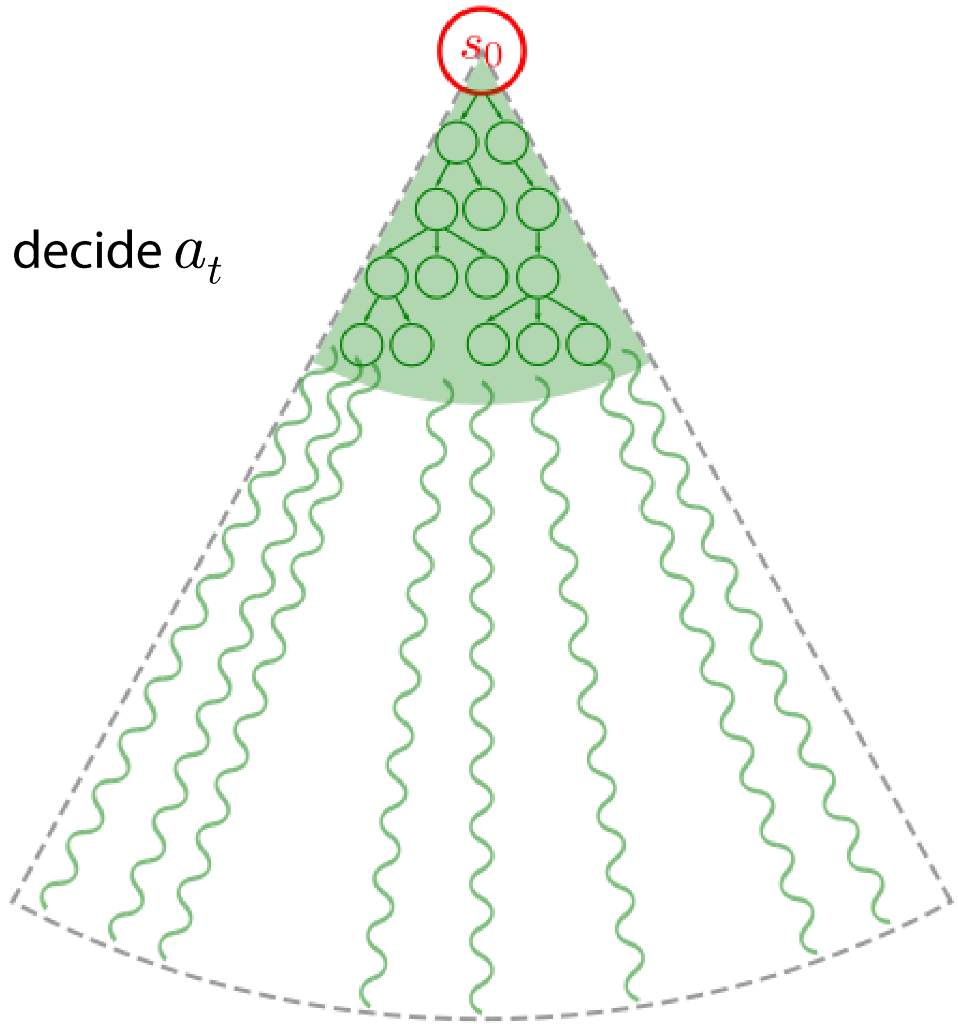
procedure BACKUP($\{s_0, ..., s_T\}, R$)
  for $t = 0$ **to** $T - 1$ **do**
    $N(s_t) \mathrel{+}= 1$
    $N(s_t, a_t) \mathrel{+}= 1$
    $Q(s_t, a_t) \mathrel{+}= \frac{R - Q(s_t, a_t)}{N(s_t, a_t)}$
  end for
end procedure

procedure NEWNODE(s)
  $N(s) = 0$
  for all $a \in \mathcal{A}$ **do**
    $N(s,a) = 0$
    $Q(s,a) = \infty$
  end for
  $\mathcal{T}.Insert(s)$
end procedure

From: D. Silver, Reinforcement Learning and Simulation-Based Search in Computer Go, PhD Thesis, 2009

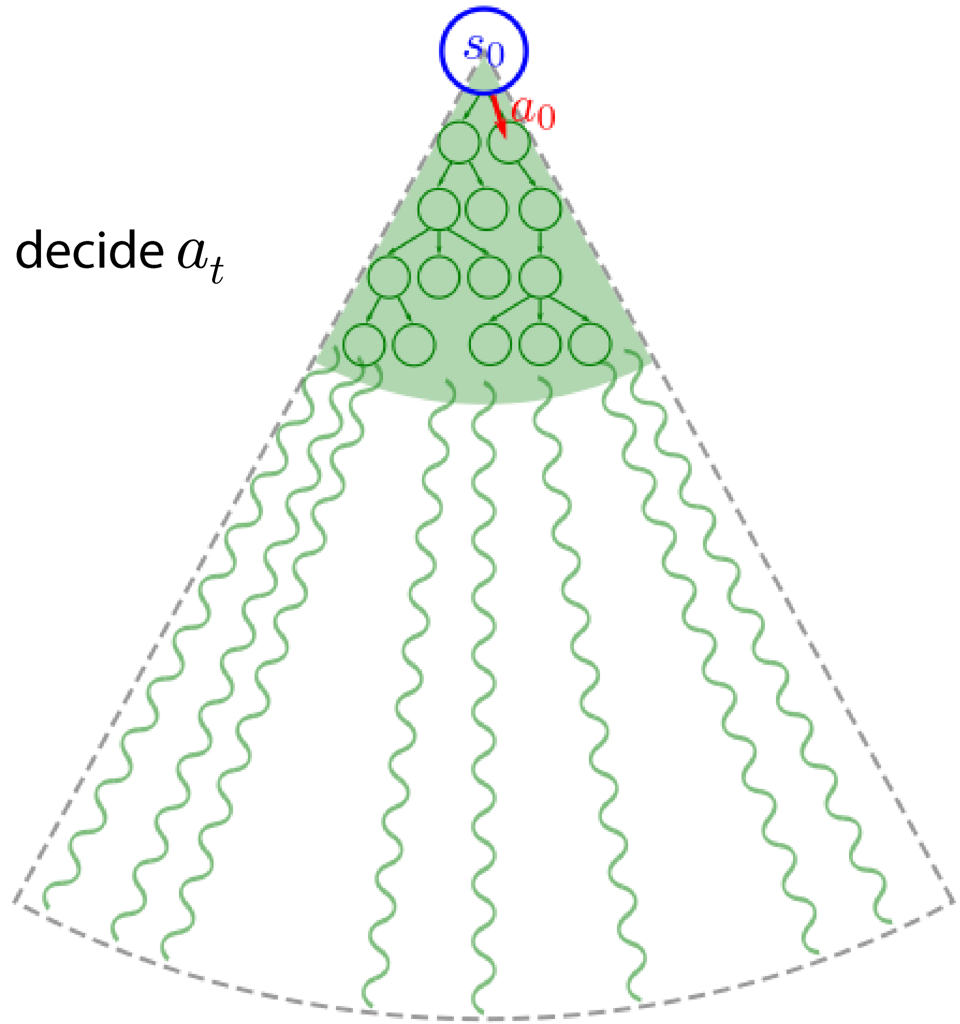
MCTS episode

- ***Monte Carlo Tree Search*** *episode:*
  1) set t:=0
  2) current state s:=s_t
  3) use *MCTS step* to expand the tree and decide a_t
  4) compute $s_{t+1} := \tau(s_t, a_t)$
  5) set t:=t+1
  6) repeat 2-5 until end game



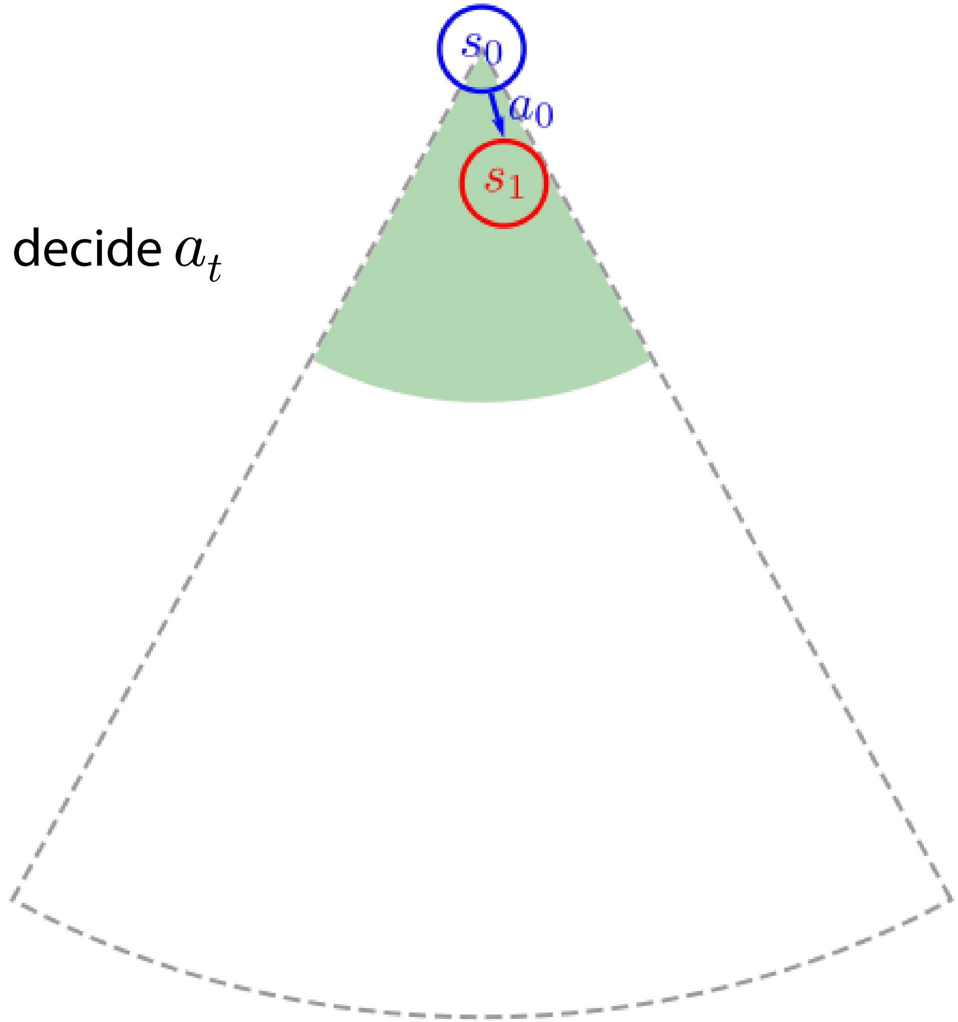
MCTS episode

- ***Monte Carlo Tree Search*** *episode:*
  1) set $t{:=}0$
  2) current state $s{:=}s_t$
  3) use *MCTS step* to expand the tree and decide a_t
  4) compute $s_{t+1} := \tau(s_t, a_t)$
  5) set $t{:=}t{+}1$
  6) repeat 2-5 until end game



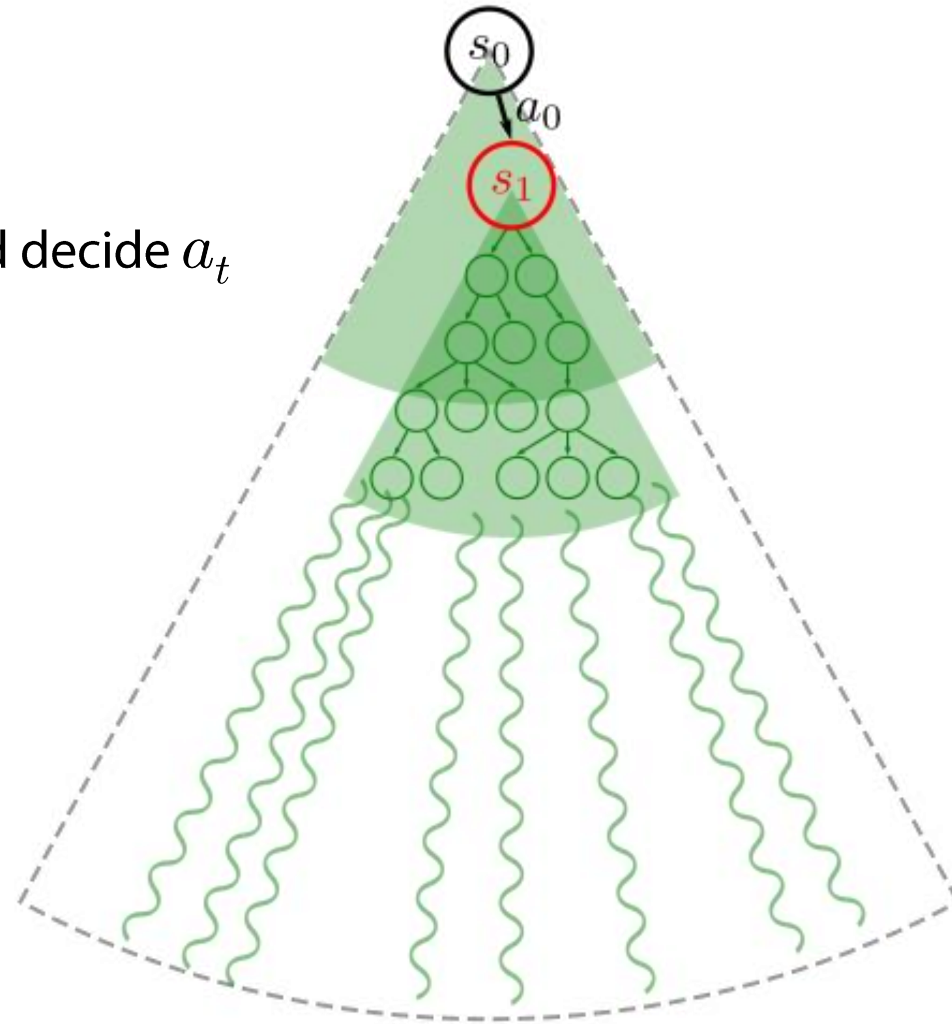
MCTS episode

- ***Monte Carlo Tree Search*** *episode:*
  1) set t:=0
  2) current state s:=s_t
  3) use *MCTS step* to expand the tree and decide a_t
  4) compute $s_{t+1} := \tau(s_t, a_t)$
  5) set t:=$t+1$
  6) repeat 2-5 until end game



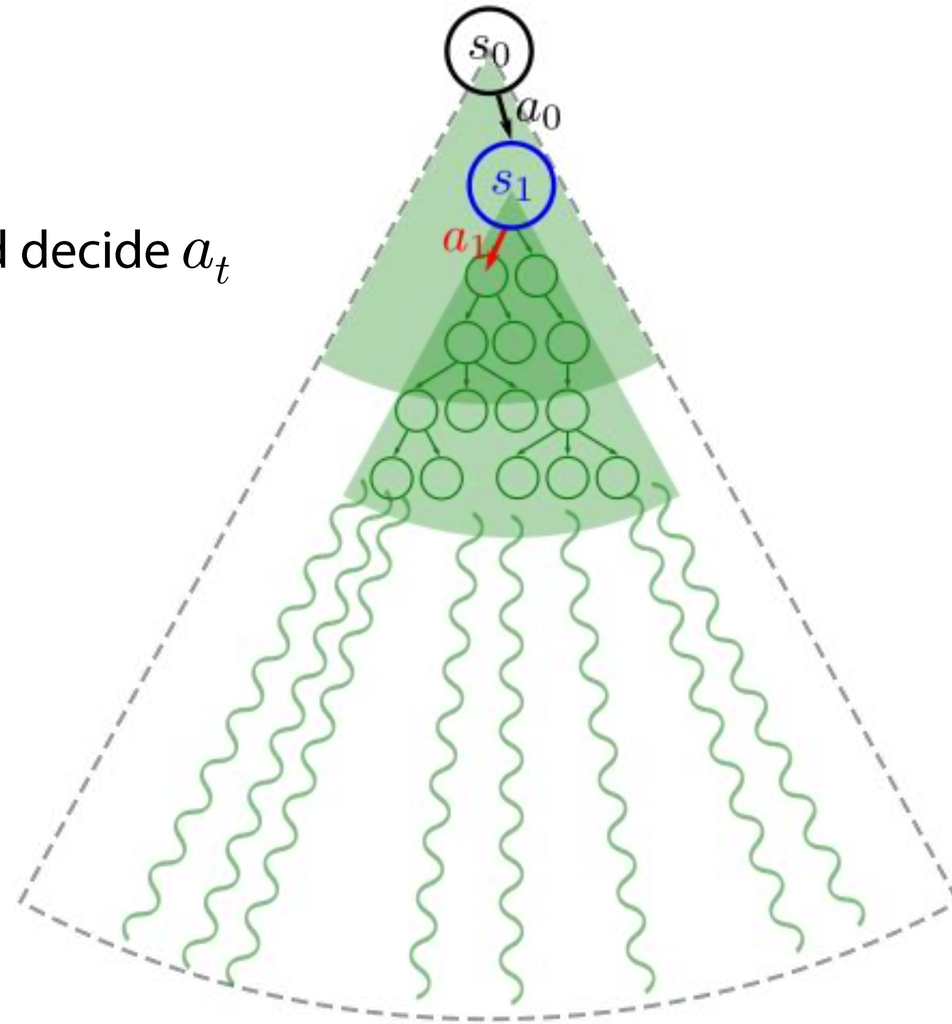
MCTS episode

- ***Monte Carlo Tree Search*** *episode:*
  1) set $t := 0$
  2) current state $s := s_t$
  3) use *MCTS step* to expand the tree and decide a_t
  4) compute $s_{t+1} := \tau(s_t, a_t)$
  5) set $t := t+1$
  6) repeat 2-5 until end game



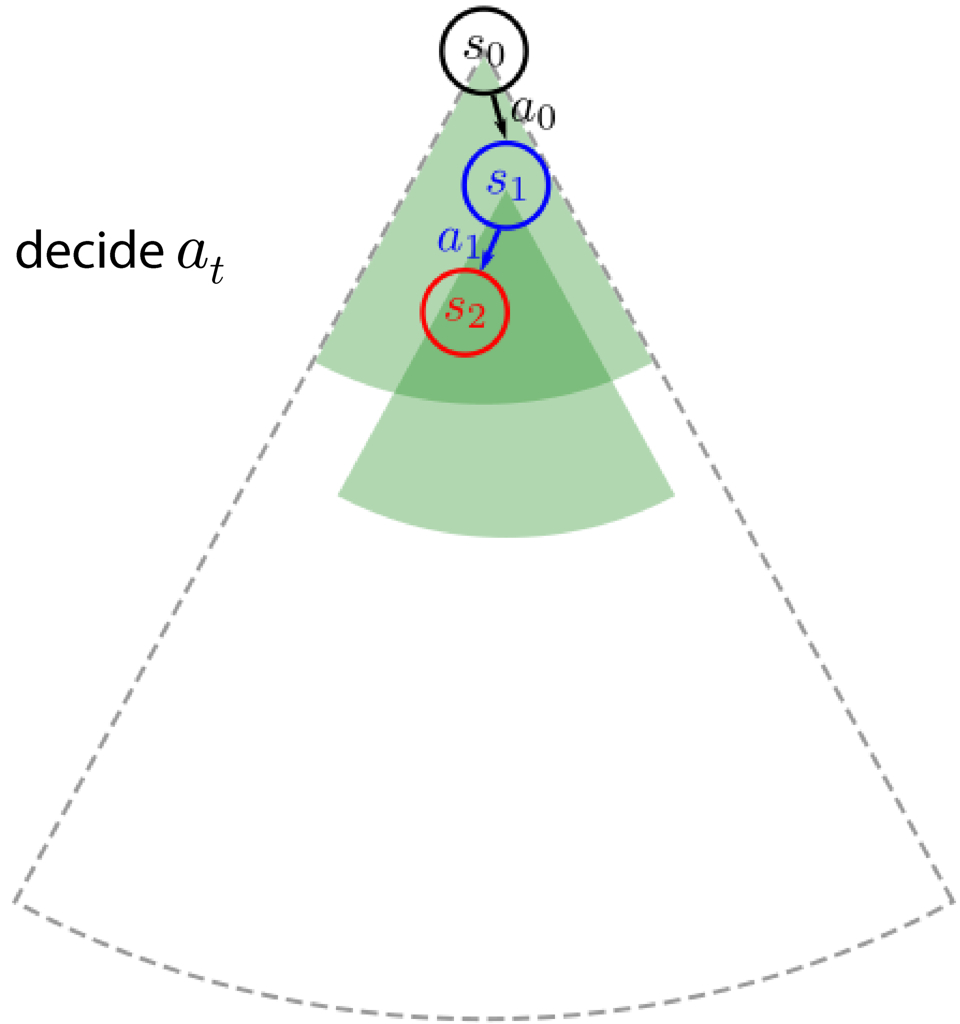
MCTS episode

- ***Monte Carlo Tree Search*** *episode:*
  1) set $t := 0$
  2) current state $s := s_t$
  3) use *MCTS step* to expand the tree and decide a_t
  4) compute $s_{t+1} := \tau(s_t, a_t)$
  5) set $t := t+1$
  6) repeat 2-5 until end game



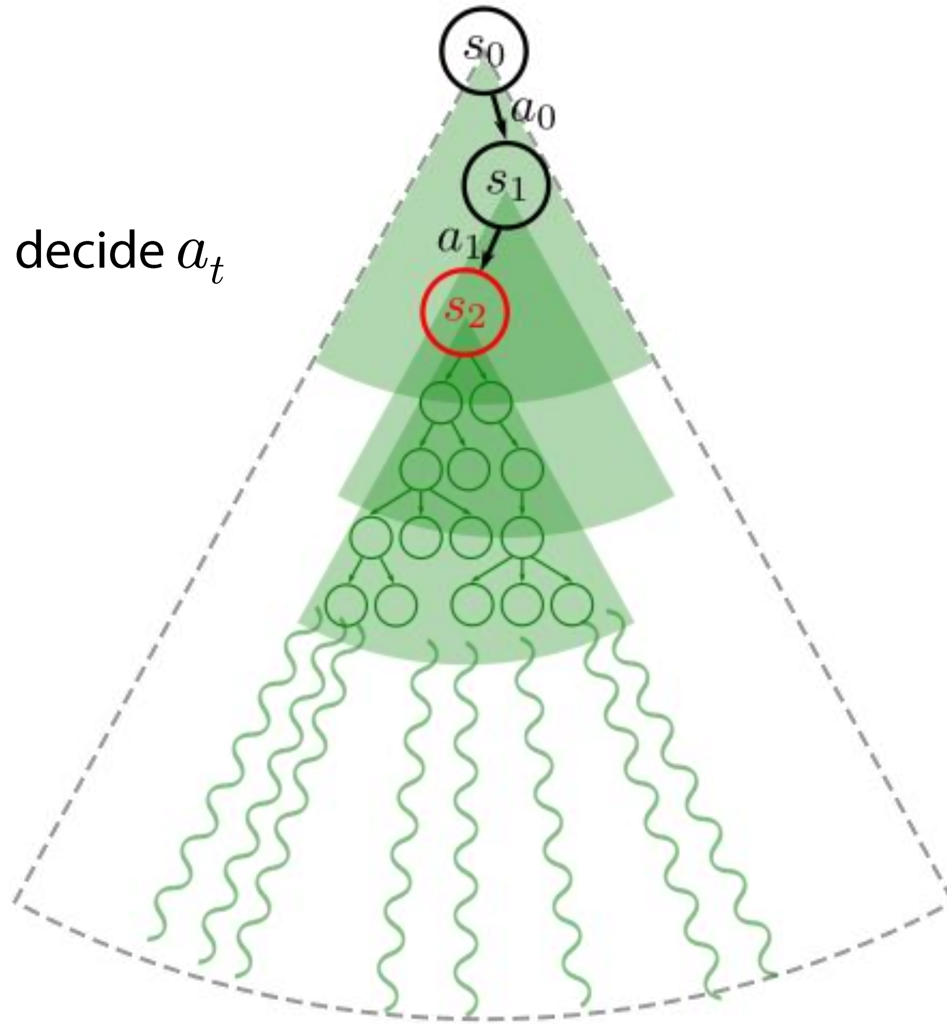
MCTS episode

- ***Monte Carlo Tree Search*** *episode:*
  1) set $t{:=}0$
  2) current state $s{:=}s_t$
  3) use *MCTS step* to expand the tree and decide a_t
  4) compute $s_{t+1} := \tau(s_t, a_t)$
  5) set $t{:=}t{+}1$
  6) repeat 2-5 until end game

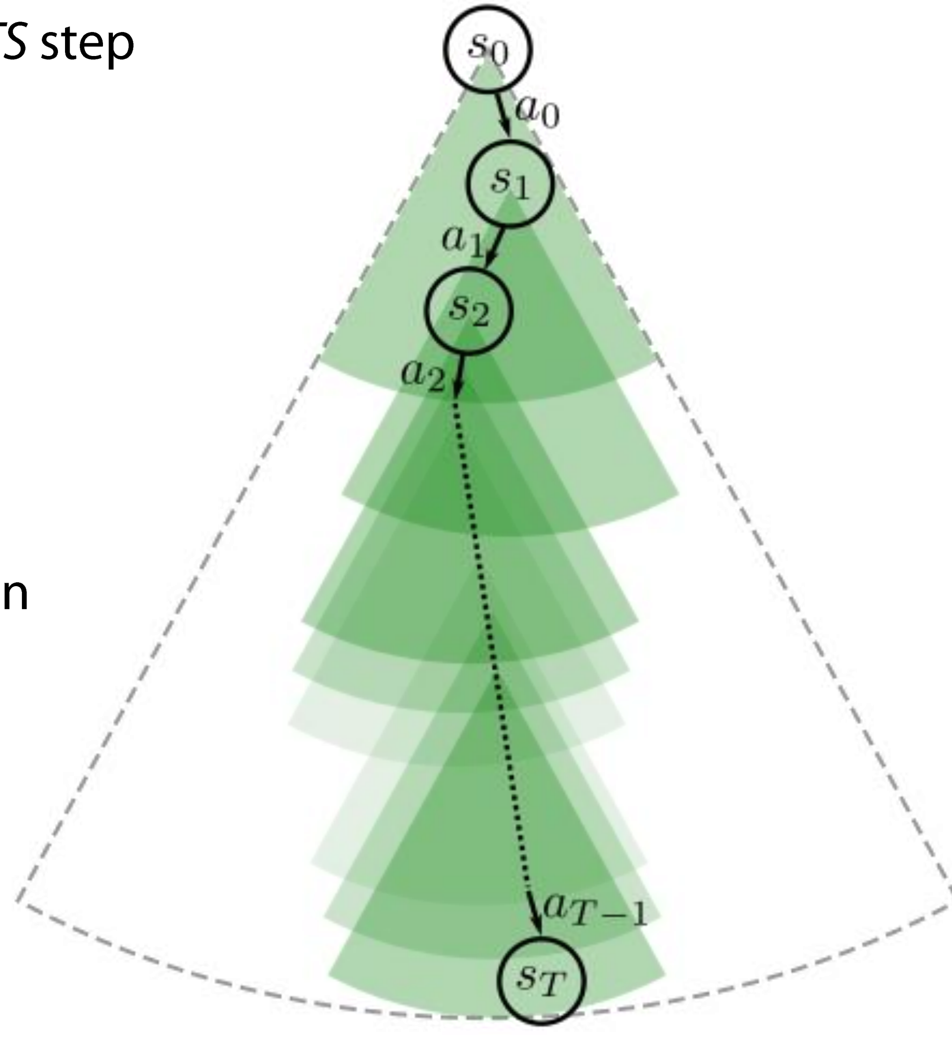


MCTS episode

- ***Monte Carlo Tree Search*** *episode:*
  1) set $t := 0$
  2) current state $s := s_t$
  3) use *MCTS step* to expand the tree and decide a_t
  4) compute $s_{t+1} := \tau(s_t, a_t)$
  5) set $t := t+1$
  6) repeat 2-5 until end game



- ***Monte Carlo Tree Search*** *method:*
  - *memory* of past playouts in a single *MCTS* step
    (collected in the tree statistics)
  - *knowledge transfer* between *MCTS* steps
    (by reusing subtrees already explored)
  - optimal policy only *partially* defined
    (on actually computed states)
  - *intrinsically stochastic* policy optimization
    (the same initial state
    can give rise to different optimizations)
  - What about *knowledge transfer*
    between *MCTS episodes*?
    transferring the entire MCTS tree
    would rapidly cause its explosive growth…



Dealing with Stochasticity and Uncertainty

Stochasticity and Uncertainty: general setting

- Stochastic reward:
  - *immediate reward* $r(s_t, a_t)$ is obtained when performing action a_t in state s_t
  - *delayed reward* is obtained only at the end of the game

$$r(s_t) := \begin{cases} 0 & \text{if } s_t \text{ is not a terminal state} \\ r & \text{otherwise} \end{cases}$$

possibly with $P(r \mid s_t, a_t)$ or $P(r \mid s_t)$ respectively

- Stochastic policy:

policy $\pi(s, a) := P(a \mid s)$ is a *probability distribution*

- Uncertainty of execution:

stochastic transition function $\tau : (s_t, a_t) \mapsto s_{t+1}$ with $P(s_{t+1} \mid s_t, a_t)$

Reinforcement Learning (RL)

- *Value function:*

$$V^\pi(s) := \mathbb{E}_\pi[R \mid s_0 = s]$$

mean over the trajectories following policy π

Optimal value: $V^*(s) := \max_\pi V^\pi(s) \ \forall s$

- *Action-value function:*

$$Q^\pi(s_t, a) := \mathbb{E}_\pi[R \mid s_0 = s, a_0 = a]$$

Optimal action-value: $Q^*(s,a) := \max_\pi Q^\pi(s,a) \ \forall s, a$

<u>Optimal policy:</u> $a^*(s) = \underset{a}{\operatorname{argmax}}[Q^\pi(s,a)]$

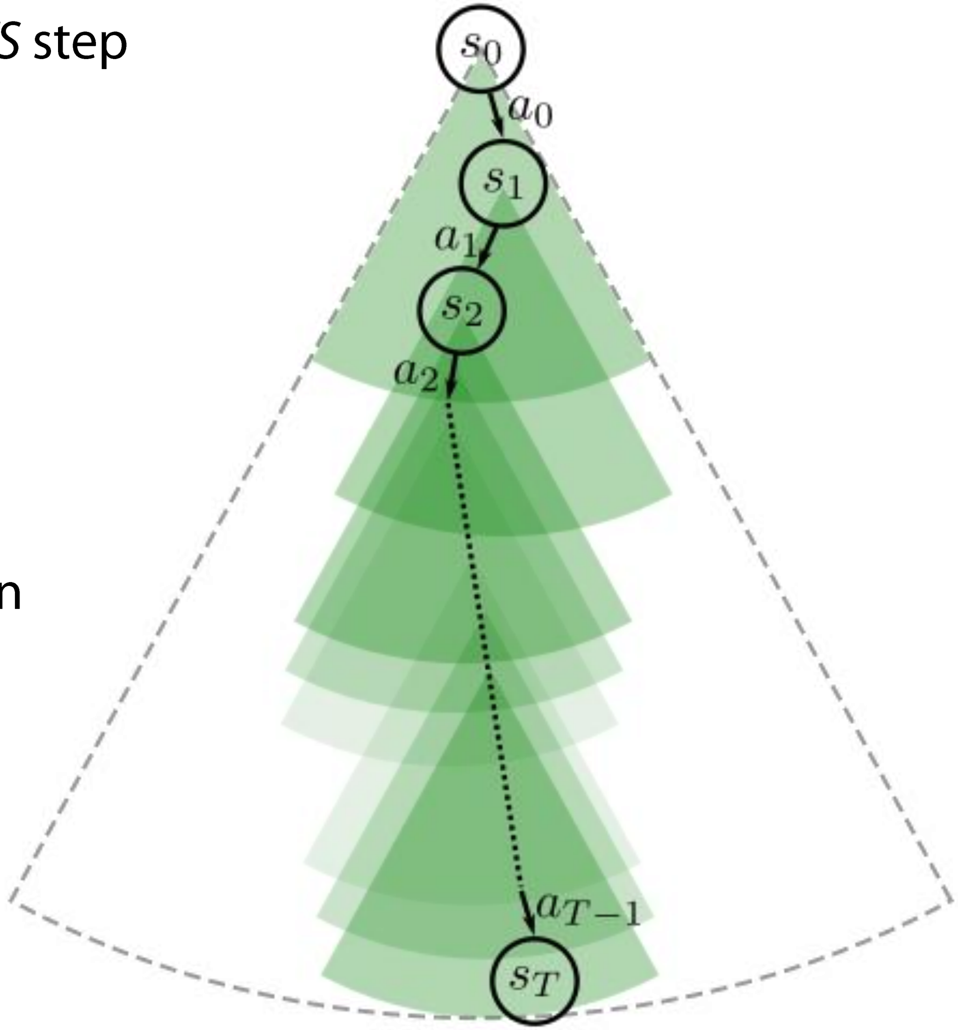
Connection: $V^\pi(s) = \mathbb{E}_\pi[Q^\pi(s,a)]$ and $V^*(s) = \max_a [Q^*(s,a)]$

AlphaZero:

MCTS + DNN

Monte Carlo Tree Search (MCTS) method

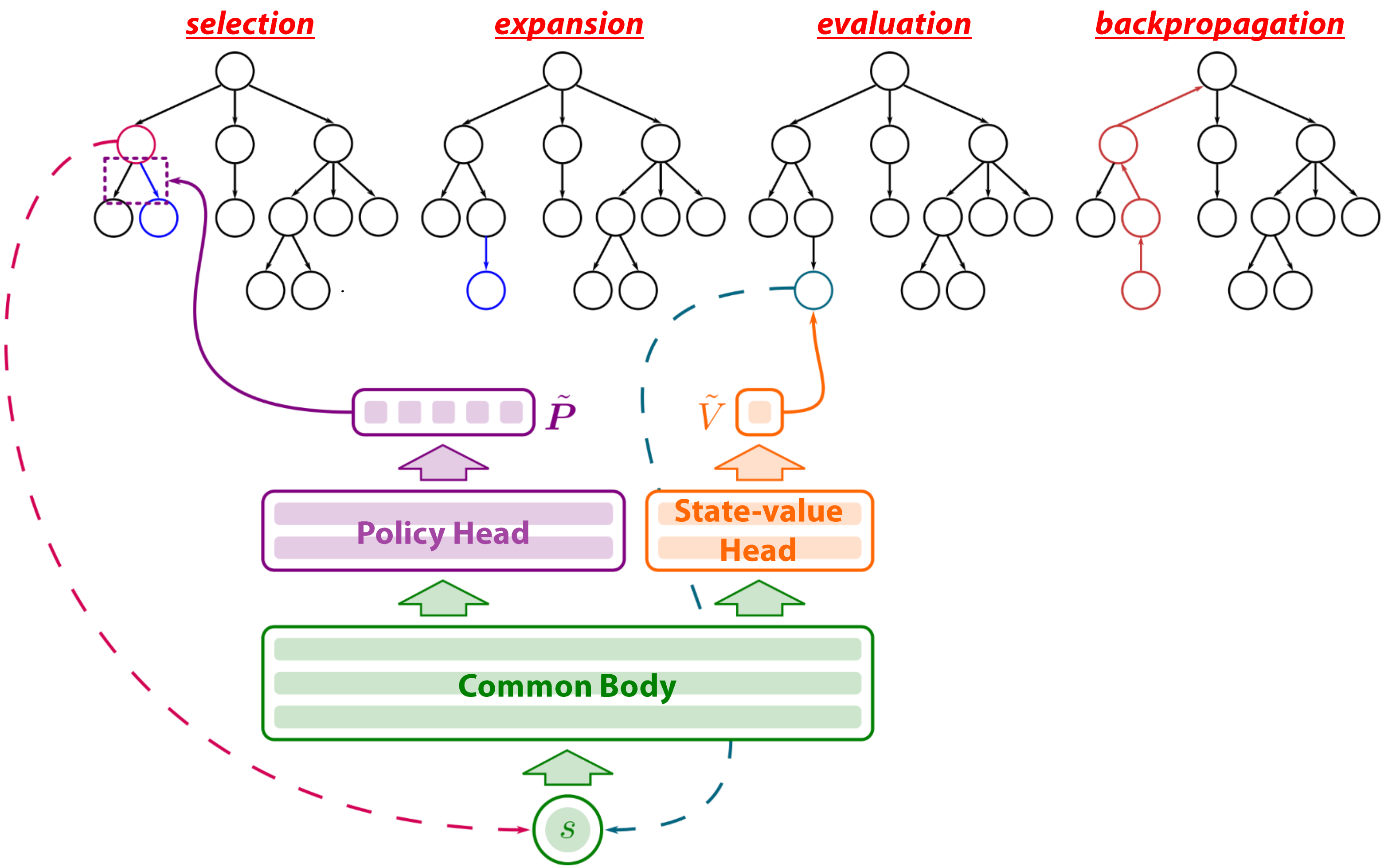
- ***MCTS** method:*
  - *memory* of past playouts in a single *MCTS* step
    (collected in the tree statistics)
  - *knowledge transfer* between *MCTS* steps
    (by reusing subtrees already explored)
  - optimal policy only *partially* defined
    (on actually computed states)
  - *intrinsically stochastic* policy optimization
    (the same initial state
    can give rise to different optimizations)
  - What about *knowledge transfer* between *MCTS episodes?*
    transferring the entire MCTS tree
    would rapidly cause its explosive growth…



Knowledge transfer between MCTS episodes

- ***AlphaZero*** [Silver et al. 2017]
  - *Monte Carlo Tree Search (MCTS):*
    improves the policy by focusing on the most promising actions
  - *Deep Neural Network (DNN):*
    learns the improved policy and transfers it between MCTS episodes

- **AlphaZero = MCTS + DNN**

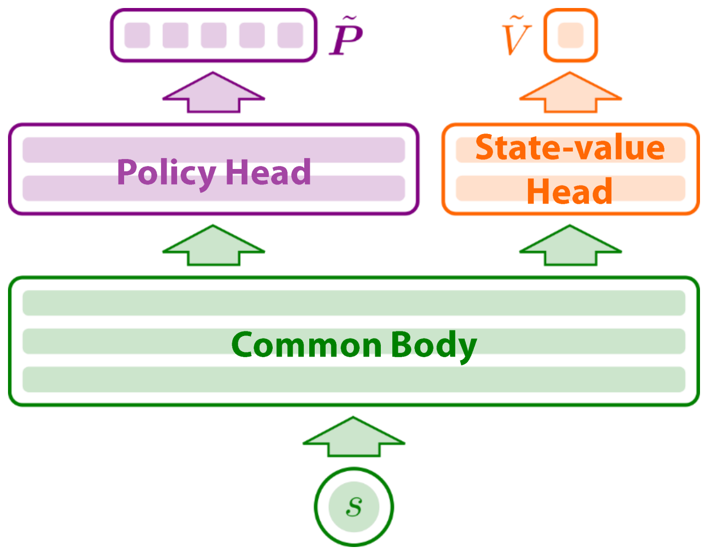


DNN in AlphaZero

- **DNN in AlphaZero**
  - *input:* a *state* s
  - *output:* a *probability distribution* $\tilde{\boldsymbol{P}}(s) := [\tilde{P}(a \mid s)]_{a \in \mathcal{A}(S)}$ — stochastic policy (a vector of probabilities)

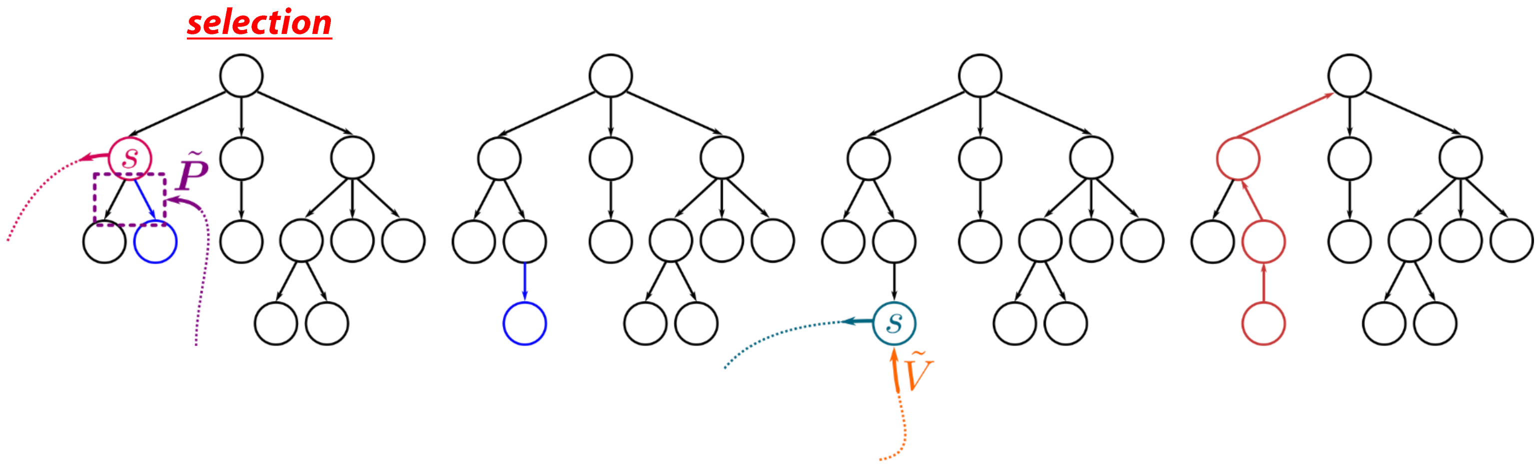
    and a *state-value* $\tilde{V}(s)$
    - predicts the expected reward for state s
    - acts as an ***actor-critic*** in the *training* of ***parameters*** ϑ of the net

$\tilde{V}$ is compared with the *actual* reward r, which also impacts on training $\tilde{\boldsymbol{P}}$ by *backpropagating* through the *Common Body*



"Y" shape

- ***MCTS step*** **in** ***AlphaZero***



- <u>*selection:*</u> *UCT* policy is replaced with ***PUCT*** *("Predictor" + UCT)*

MCTS estimation of $Q(s,a)$ for *DNN* policy

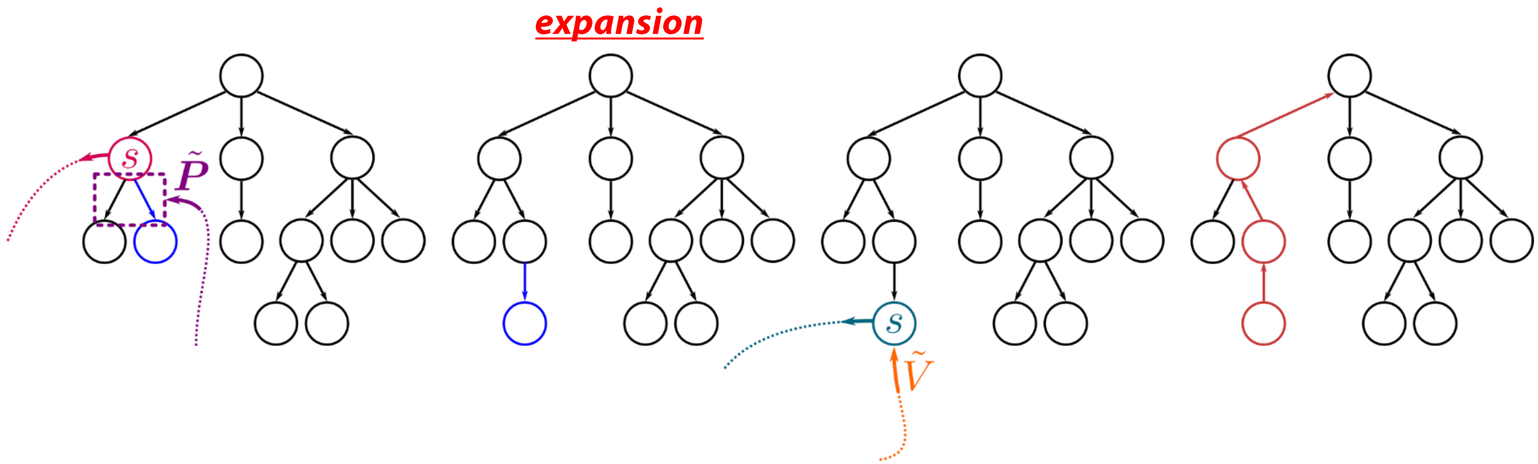
DNN policy

$$\pi^{\mathrm{PUCT}}(s) := \underset{a}{\operatorname{argmax}} \left\{ \hat{Q}(s,a) + c(s)\tilde{P}(a \mid s)\frac{\sqrt{N(s)}}{N(s,a)+1} \right\}$$

avoids division by 0

exploration rate $c(s) := \log \frac{1 + N(s) + c_{\mathrm{base}}}{c_{\mathrm{base}}} + c_{\mathrm{init}}$
(slowly grows with search time)

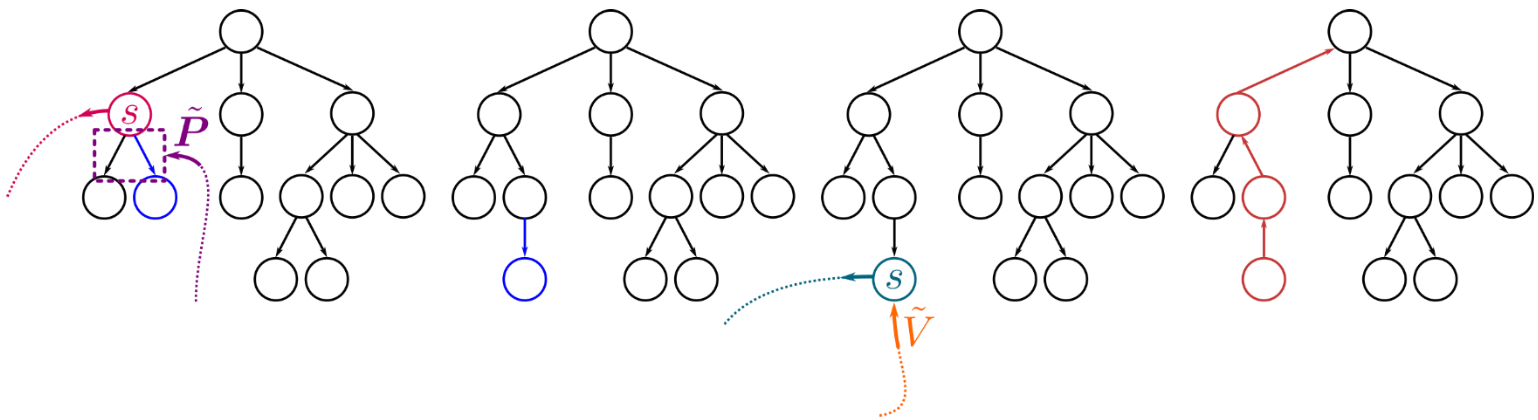
- ***MCTS step* in *AlphaZero***



- *expansion:* initialization of the leaf new node s_L:

$$N(s_L) := 0 \quad \text{and} \quad \forall\, a \in \mathcal{A}(s_L) \qquad N(s_L, a_L) := 0, \quad \hat{Q}(s_L, a_L) := +\infty$$

- ***MCTS step*** **in *AlphaZero***

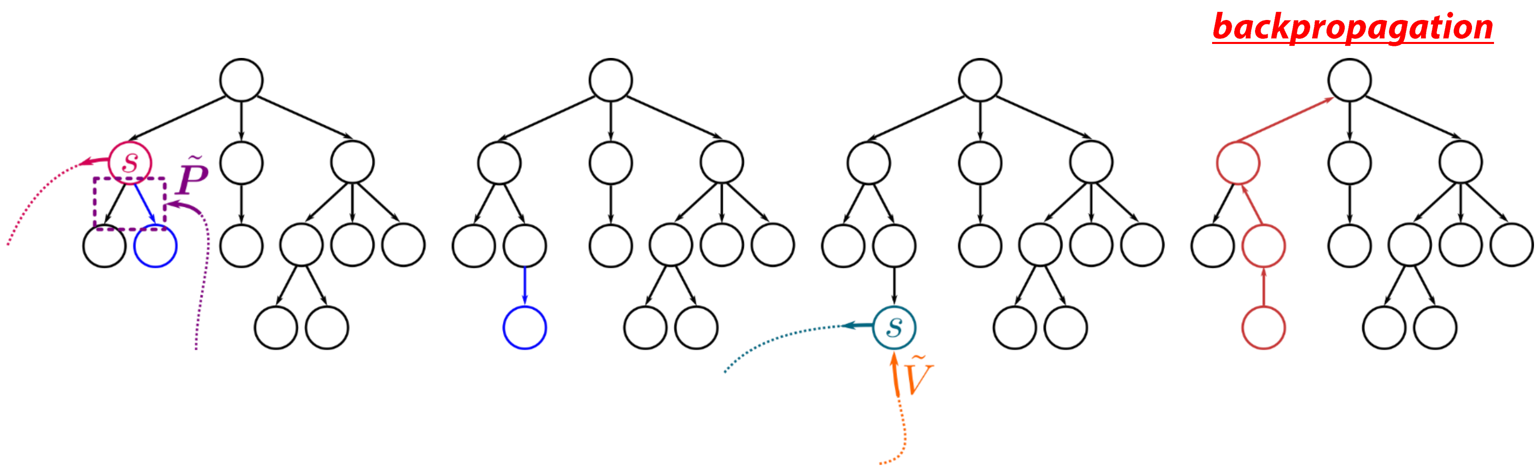


- *expansion:* initialization of the leaf new node s_L:

$$N(s_L) := 0 \quad \text{and} \quad \forall\, a \in \mathcal{A}(s_L) \qquad N(s_L, a_L) := 0, \quad \hat{Q}(s_L, a_L) := +\infty$$

- *evaluation* (in place of *simulation*): expected reward is $\tilde{V}(s_L)$

- **MCTS step in AlphaZero**



- *expansion:* initialization of the leaf new node s_L:

$$N(s_L) := 0 \quad \text{and} \quad \forall a \in \mathcal{A}(s_L) \qquad N(s_L, a_L) := 0, \quad \hat{Q}(s_L, a_L) := +\infty$$

- *evaluation* (in place of *simulation*): expected reward is $\tilde{V}(s_L)$
- *backpropagation:* for each state s and action a visited in selection/expansion:

$$\begin{aligned} N(s) &:= N(s) + 1, \\ N(s,a) &:= N(s,a) + 1 \end{aligned} \quad \text{and} \quad \hat{Q}(s,a) := \hat{Q}(s,a) + \frac{\boxed{\tilde{V}(s_L)} - \hat{Q}(s,a)}{N(s,a)}$$

MCTS step in AlphaZero: policies

- Selection policy: **PUCT**

$$\pi^{\text{sel}}(s) := \pi^{\text{PUCT}}(s) := \operatorname*{argmax}_{a} \left\{ \hat{Q}(s,a) + c(s)\tilde{P}(a \mid s) \frac{\sqrt{N(s)}}{N(s,a)+1} \right\}$$

- Output policy:

$$\pi^{\text{out}}(s) \sim \left[\hat{P}(a \mid s) := \frac{N(s,a)}{N(s)} \right]_{a \in \mathcal{A}(s)}$$

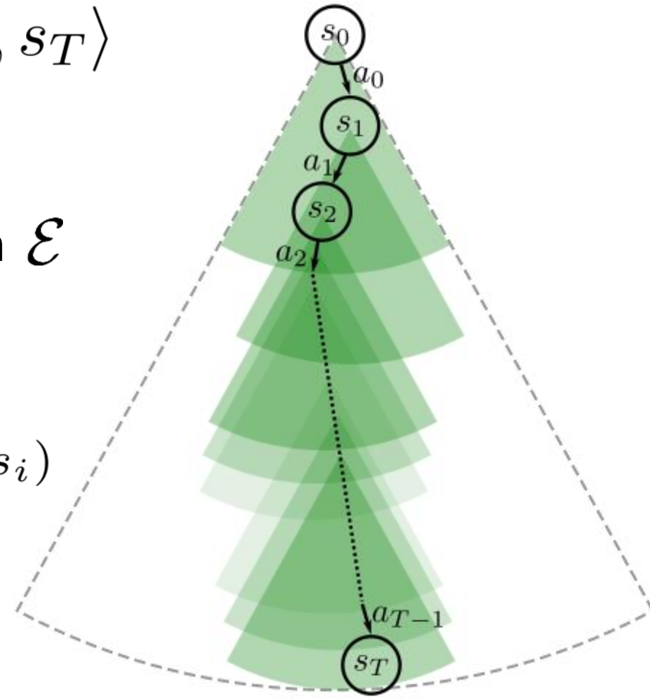
taking frequencies as probabilities
(in place of their argmax as output action)
ensures ***exploration***

(the simulation policy does not exist anymore)

DNN training in AlphaZero

- ***Data items*** from a <u>single</u> *MCTS episode:*

  After an *MCTS episode* $\mathcal{E} := \langle s_0, a_0, s_1, \ldots, a_{T-1}, s_T \rangle$
  with actual reward $\hat{V}^{\mathcal{E}} := r(s_T)$:



  - for each *<u>non-terminal</u>* state $s_i \ (i = 0 \ldots T-1)$ in $\mathcal{E}$

$$\hat{\boldsymbol{P}}(s_i) := \left[\hat{P}(a \mid s_i) := \frac{N(s_i, a)}{N(s_i)} \right]_{a \in \mathcal{A}(s_i)}$$

<u>vector</u> of frequencies

  - the ***output*** of $\mathcal{E}$ is

$$D^{\mathcal{E}} := \{ \langle s_i, \hat{\boldsymbol{P}}(s_i), \hat{V}^{\mathcal{E}} \rangle \}_{i=0 \ldots T-1}$$

data item

DNN training in AlphaZero

- ***Iteration:***

  K times:
  1) play one *MCTS episode* $\mathcal{E}_j$
  2) collect data items $D^{\mathcal{E}_j}$

  3) train the parameters of the *DNN* by using as ***dataset***

$$D := \bigcup_{j=1}^{K} D^{\mathcal{E}_j}$$

- After <u>enough</u> iterations:

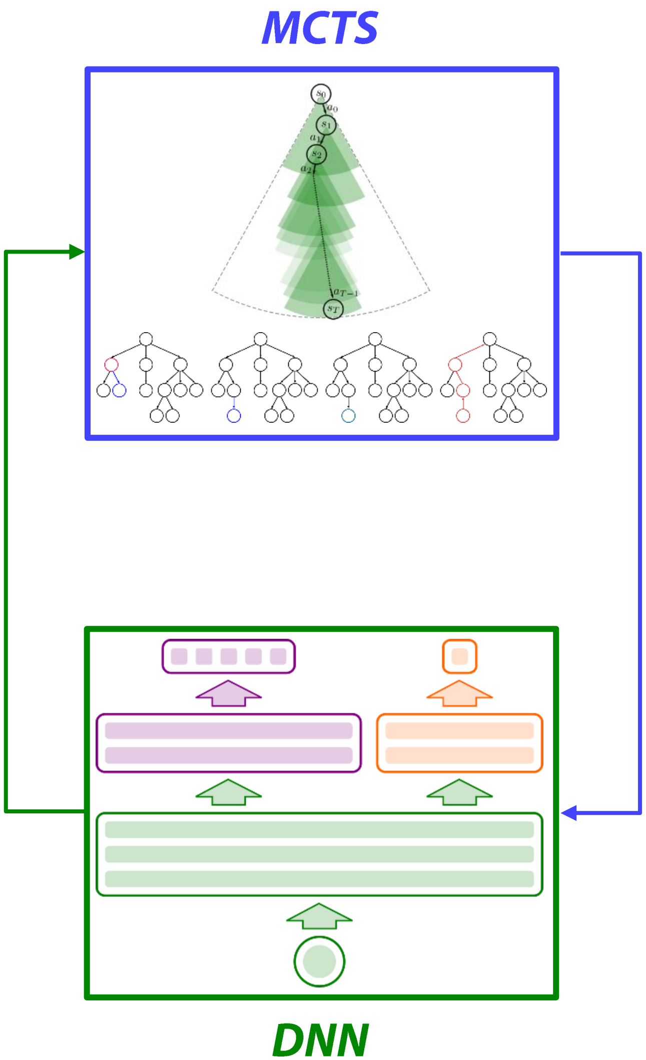
$$\pi^{\mathrm{DNN}}(s) := \operatorname*{argmax}_{a \in \mathcal{A}(s)} \tilde{P}(a \mid s) \to \pi^*(s) \quad \forall\, s$$

AlphaZero

- ***AlphaZero:***
  - *memory* of past playouts in a single *MCTS* step
    (collected in the tree statistics)
  - *knowledge transfer* between *MCTS* steps
    (by reusing subtrees already explored)
  - *knowledge transfer* between *MCTS* episodes
    (provided by DNN)

  - *deterministic* policy optimization
    with policy defined for all states s :

$$\pi^{\mathrm{DNN}}(s) := \operatorname*{argmax}_{a \in \mathcal{A}(s)} \tilde{P}(a \mid s)$$



AlphaZero

in Continuous Spaces

Continuous Action Spaces

- *What happens when the space $\mathcal{A}(s)$ of admissible actions is continuous?*
  - How to compute the deterministic *policy optimization* in practice?

$$\pi^{\text{DNN}}(s) = \underset{a \in \mathcal{A}(s)}{\text{argmax}}\, \tilde{P}(a \mid s)$$

it could be a high-dimensional space

continuous and analytic, but in general with a lot of (local) maxima

  - How to initialize (and deal with) a *new node* s in the MCTS *expansion* phase?

    Standard initialization requires:

$$\forall\, a \in \mathcal{A}(s) \qquad N(s,a) := 0, \quad \hat{Q}(s,a) := +\infty$$

each admissible action is initialized

each admissible action will be evaluated at least once

Cross-Entropy Maximization (CEM)

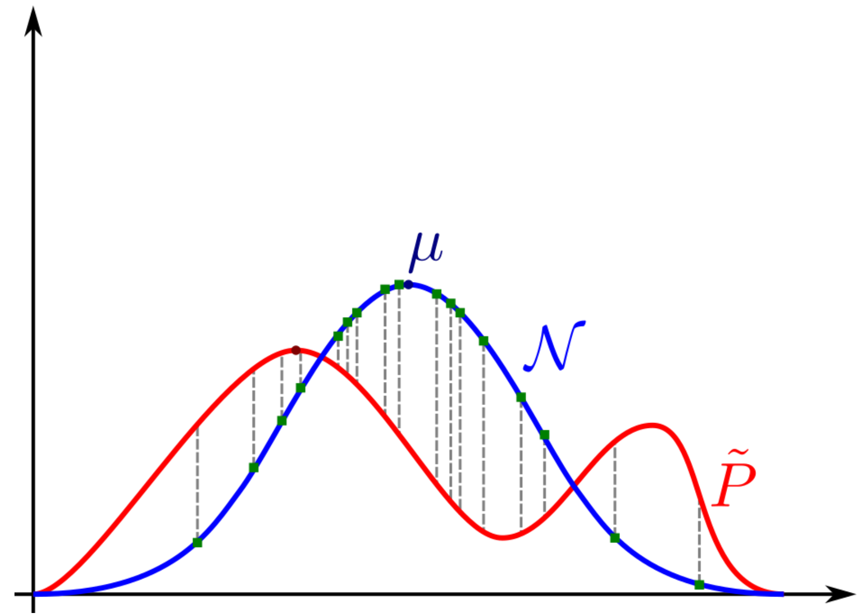
- ***CEM Method:***

1) choose *<u>at random</u>* initial values $\mu, \sigma \in \mathbb{R}^d$

2) <u>sample</u> m actions from

normal probability distribution $\mathcal{N}(\mu, \mathrm{diag}(\sigma))$ — *mean* (μ), *variances (diagonal matrix)* ($\mathrm{diag}(\sigma)$)

3) evaluate $\left\{ \tilde{P}(a_i \mid s) \right\}_{i=1}^{m}$



Cross-Entropy Maximization (CEM)

- ***CEM Method:***

1) choose *at random* initial values $\mu, \sigma \in \mathbb{R}^d$

2) sample m actions from

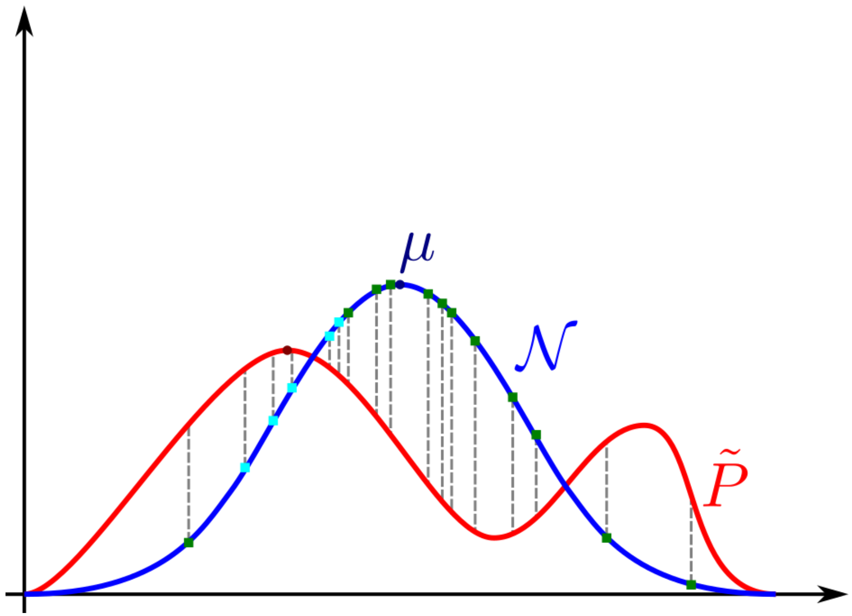
mean

variances (diagonal matrix)

normal probability distribution $\mathcal{N}(\mu, \mathrm{diag}(\sigma))$

3) evaluate $\left\{\tilde{P}(a_i \mid s)\right\}_{i=1}^{m}$

4) select $k < m$ actions with *highest probability*



- ***CEM Method:***

1) choose _at random_ initial values $\mu, \sigma \in \mathbb{R}^d$

2) _*sample*_ m actions from

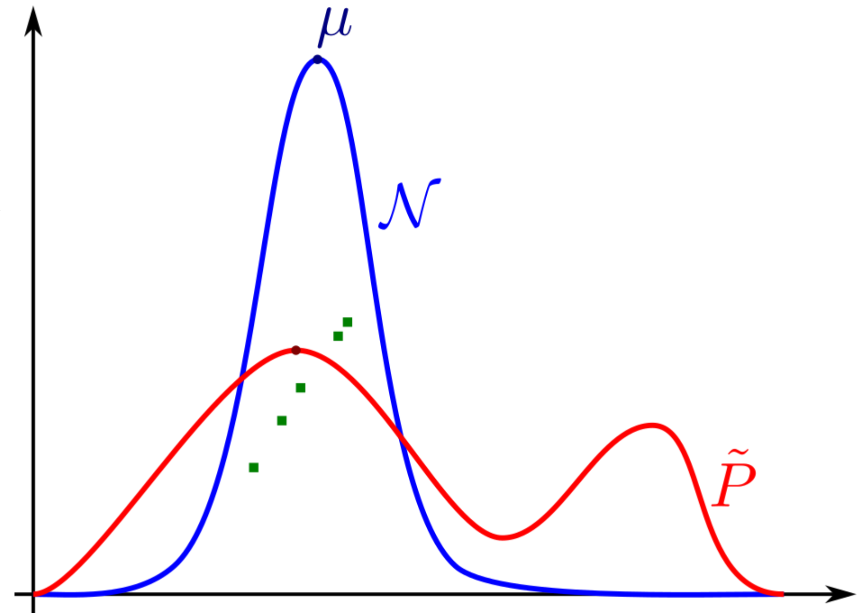
normal *probability distribution* $\mathcal{N}(\mu, \mathrm{diag}(\sigma))$ (*mean*: μ; *variances (diagonal matrix)*: $\mathrm{diag}(\sigma)$)

3) evaluate $\left\{\tilde{P}(a_i \mid s)\right\}_{i=1}^{m}$

4) select $k < m$ actions with *highest probability*

5) fit new μ, σ

6) if terminated, return μ otherwise go to 2)



Progressive Widening (PW)

- ***Progressive Widening (PW)*** *of action space* $\mathcal{A}(s)$ [Chaslot et al., 2007]:
  - For any *new node* s created in the MCTS *expansion* phase
    1. initialize $\mathcal{A}(s) := \{a_1, \dots, a_k\}$ with k admissible actions by ***sampling*** the ***probability*** $\tilde{P}(a \mid s)$ (given by the DNN)
    2. initialize the statistics for each action $a \in \mathcal{A}(s)$ as usual:

$$N(s, a) := 0, \quad \hat{Q}(s, a) := +\infty$$

Progressive Widening (PW)

- ***Progressive Widening (PW)*** *of action space* $\mathcal{A}(s)$ [Chaslot et al., 2007]:
  - For any *new node* s created in the MCTS *expansion* phase
    1. initialize $\mathcal{A}(s) := \{a_1, \ldots, a_k\}$ with k admissible actions by ***sampling*** the ***probability*** $\tilde{P}(a \mid s)$ (given by the DNN)
    2. initialize the statistics for each action $a \in \mathcal{A}(s)$ as usual:

$$N(s,a) := 0, \quad \hat{Q}(s,a) := +\infty$$

  - Before any *selection* phase in state s, compare number of actions $|\mathcal{A}(s)|$ and number of visits $N(s)$:
    1. if $|\mathcal{A}(s)|^2 \leq N(s)$ add a *new action* a' by sampling the probability $\tilde{P}(a \mid s)$

not enough actions, a lot of visits

a' will be the next selected action

$$\mathcal{A}(s) := \mathcal{A}(s) \cup \{a'\} \quad \text{with} \quad N(s,a') := 0, \quad \hat{Q}(s,a') := +\infty$$

    2. proceed with the usual selection phase

Sampling DNN probability

- ***How to sample the DNN probability*** $\tilde{P}(a \mid s)$ ***?***
  - Probability $\tilde{P}(a \mid s)$ could be the *normalization* of a function such as

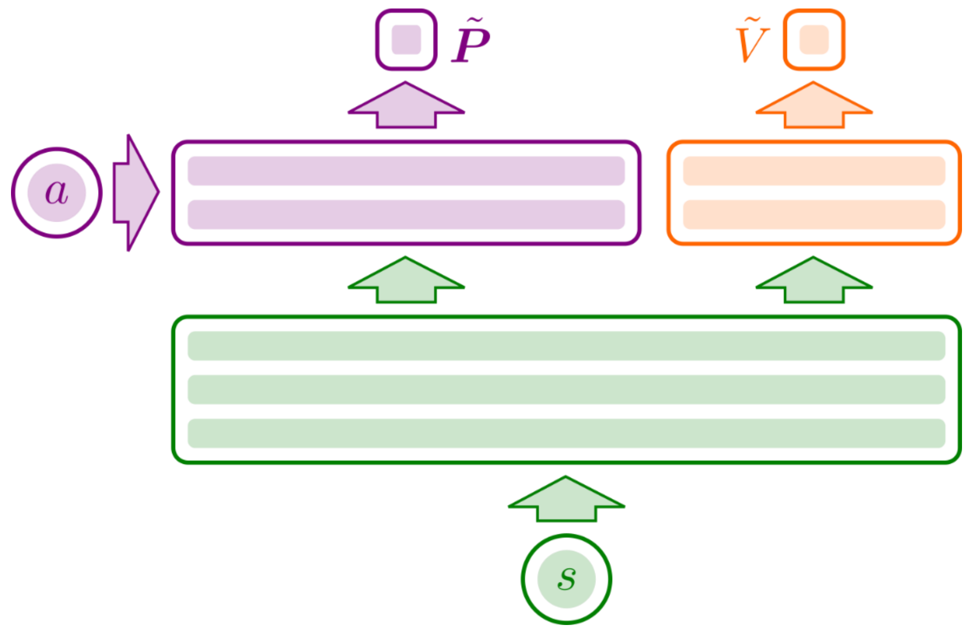
vector representing action a

$$p(a\,;s) = \boldsymbol{w} \cdot g(\boldsymbol{W}^{[\ell]} g(\cdots g(\boldsymbol{W}_s^{[1]} \boldsymbol{a} + \boldsymbol{b}_s^{[1]}) + \cdots) + \boldsymbol{b}^{[\ell]}) + b$$

non-linear continuous function

depending on state s

  - Probability $\tilde{P}(a \mid s)$ is *computable* <u>*given*</u> the state s and the action a



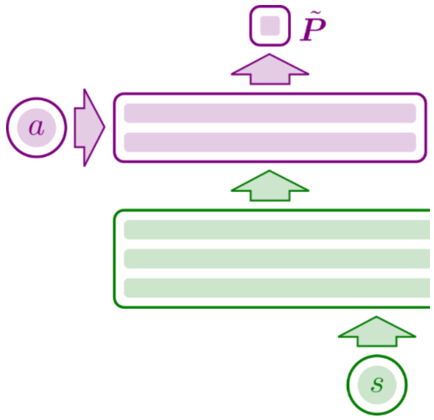
  - *What about sampling* $\tilde{P}(a \mid s)$ *?*

Advanced methods: Neural Importance Sampling

Neural Importance Sampling

- ***How to sample the DNN probability*** $\tilde{P}(a \mid s)$ ***?***

  we can use the Importance Sampling!



- ***Neural Importance Sampling***
  1) choose a suitable ***bijector*** $\boldsymbol{\mathcal{T}}$
  2) sample $\boldsymbol{y} \in [0,1]^d$ with uniform probability distribution u
  3) apply $\boldsymbol{\mathcal{T}}$ and compute the (vector representing the) action

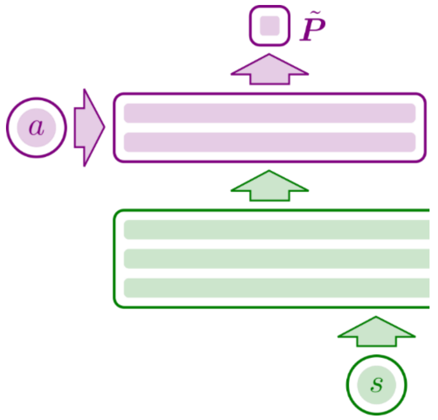
$$\boldsymbol{a} := \boldsymbol{\mathcal{T}}(\boldsymbol{y} \mid s)$$

  Then

$$\tilde{P}(a \mid s) = \left|\det\left(\frac{\partial \boldsymbol{\mathcal{T}}(\boldsymbol{y})}{\partial \boldsymbol{y}}\bigg|_{\boldsymbol{y}=\boldsymbol{\mathcal{T}}^{-1}(\boldsymbol{a}|s)}\right)\right|^{-1} u(\boldsymbol{\mathcal{T}}^{-1}(\boldsymbol{a} \mid s))$$

Neural Importance Sampling

- *Training:*
  - minimize a suitable *loss*:



$$L_{\mathrm{KL}}(\hat{P}||\tilde{P}) := \mathbb{E}_{\hat{P}}[\log(\hat{P}(a \mid s)) - \log(\tilde{P}(a \mid s))]$$

$$= \int \hat{P}(a \mid s) \log\left(\frac{\hat{P}(a \mid s)}{\tilde{P}(a \mid s)}\right) \mathrm{d}a$$

e.g. ***Kullback-Leibler (KL) divergence***

it can be approximated by a *discrete sum*

  - over the *dataset*

$$D^f := \left\{\langle a_j, s_i, \hat{P}(a_j \mid s_i)\rangle\right\}$$