UNIVERSITÀ DEGLI STUDI DI SALERNO
DIPARTIMENTO DI INFORMATICA

# Intelligenza Artificiale

## Model-based Reinforcement Learning

a.a. 2022/2023

# Outline

- Introduzione
- Model-based reinforcement learning
- Architetture integrate
  - Learning e Planning
  - Esperienza reale e simulata
- Monte-Carlo Tree Search
- Use case - Go

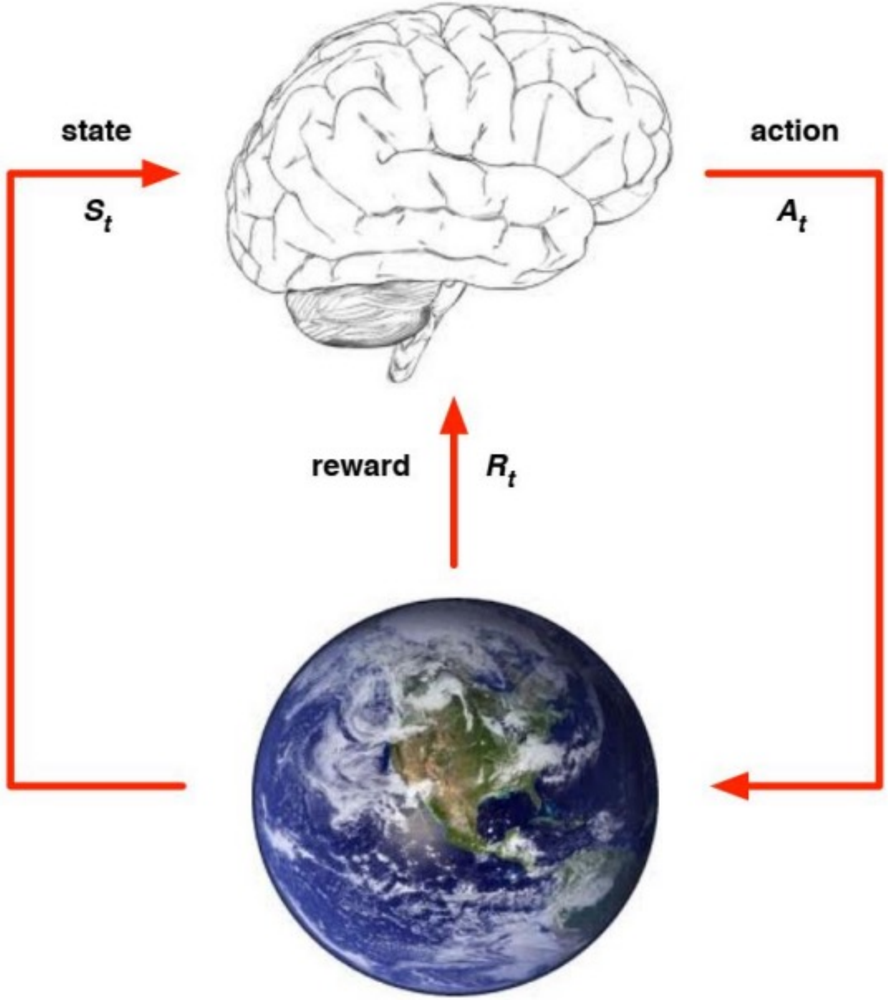
# Model-based Reinforcement Learning

- Lezioni precedenti
  - Apprendimento di una value function dall'esperienza
  - Apprendimento di una policy dall'esperienza
  - Model-free RL
- Oggi
  - Apprendimento di un modello dall'esperienza
  - Planning per costruire una value function o una policy
  - Integrare learning e planning in un'unica architettura

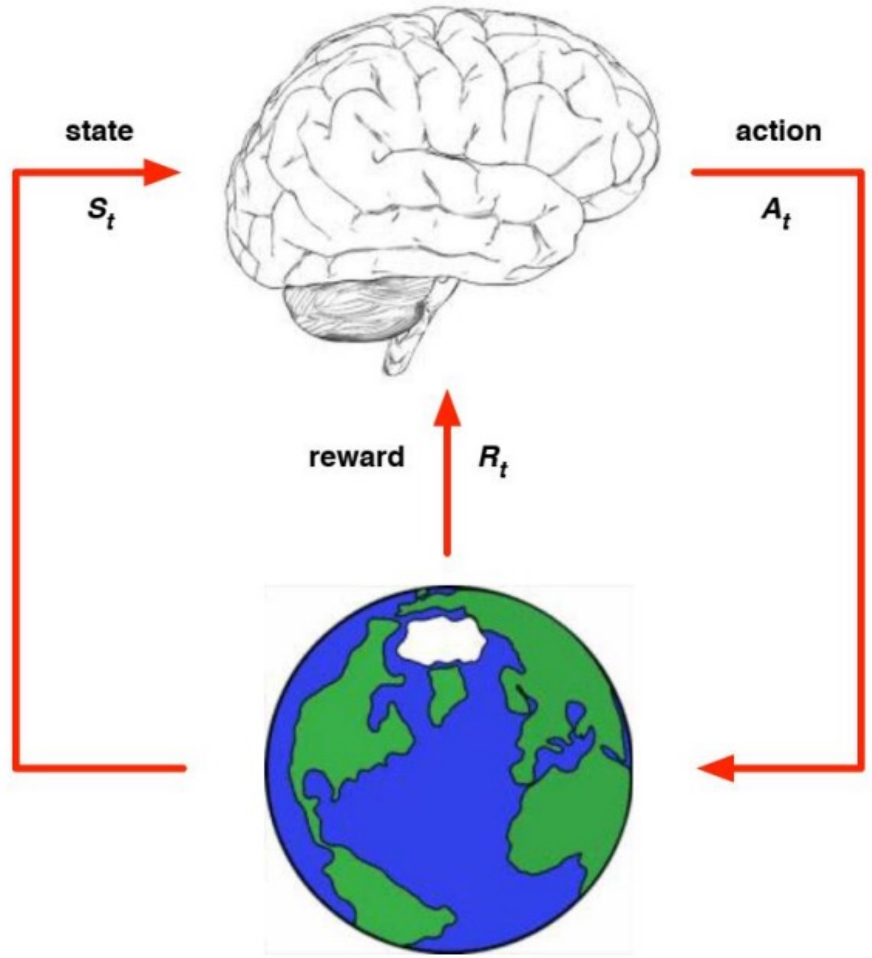
# Model-Based & Model-Free RL

- Model-Free RL
  - Nessun modello
  - Apprendimento della value function (e/o della policy) dall'esperienza
- Model-Based RL
  - Apprendimento di un modello dall'esperienza
  - Pianificare la value function (e/o la policy) dal modello

# Model-Free RL

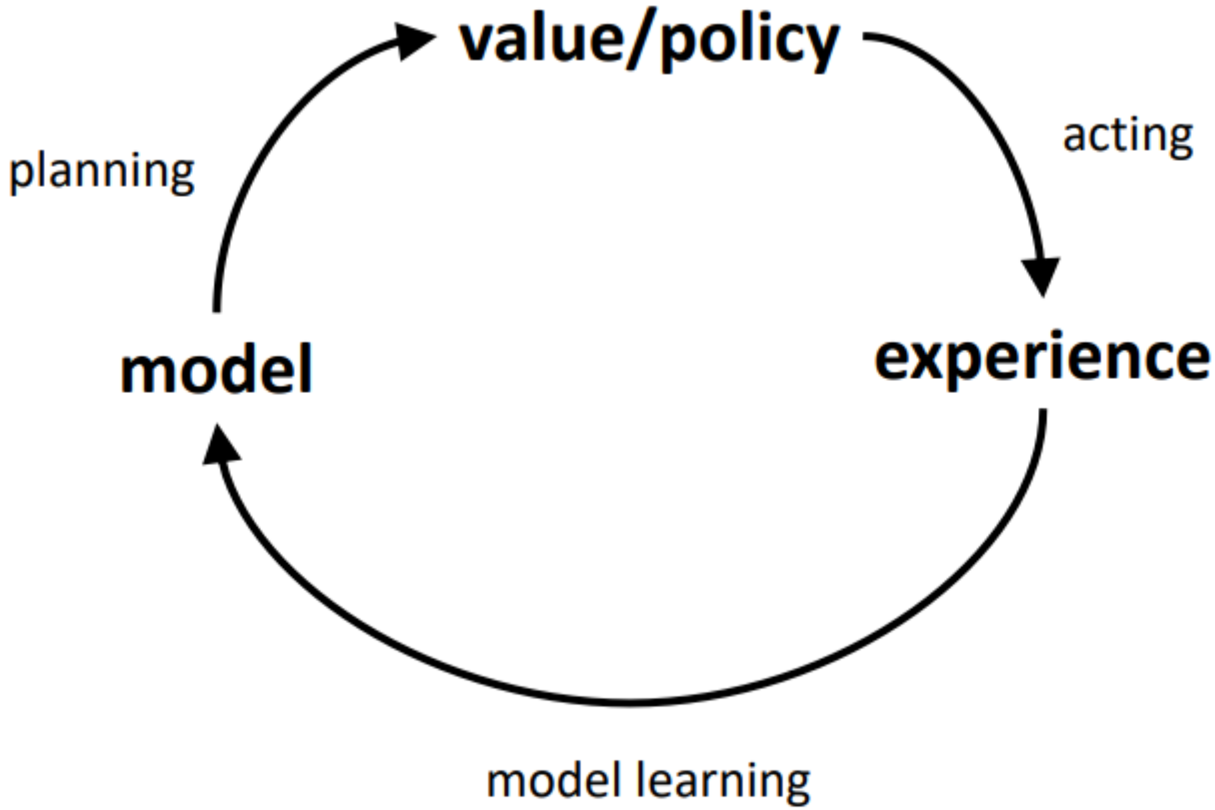


# Model-Based RL



# Model-based RL

# Model-Based RL – Ciclo di vita



# Model-Based RL – Pro e contro

- Vantaggi
  - Può apprendere in modo efficiente il modello con metodi di apprendimento supervisionato
  - Può ragionare sull'incertezza del modello
- Svantaggi
  - Prima si apprende un modello, poi si costruisce una value function

    => Ovvero due fonti di errore di approssimazione

# Cosa è un Modello?

- Un modello $M_\eta$ è una rappresentazione di un MDP <*S, A,* ***P****, R*> parametrizzato tramite $\eta$
- Si supponga che lo spazio degli stati *S* e lo spazio delle azioni *A* siano noti
- Quindi un modello $M_\eta$ = <$P_\eta$, $R_\eta$> rappresenta le transizioni di stato $P_\eta$ ≈ ***P*** e le ricompense $R_\eta$ ≈ *R*

$$S_{t+1} = P_\eta(S_{t+1}|S_t, A_t)$$
$$R_{t+1} = \mathcal{R}_\eta(R_{t+1}|S_t, A_t)$$

- In genere si assume l'indipendenza condizionale tra le transizioni di stato e le ricompense

$$P(S_{t+1}, R_{t+1}|S_t, A_t) = P(S_{t+1}|S_t, A_t)P(R_{t+1}|S_t, A_t)$$

# Model Learning

- Obiettivo: stimare il modello $M_\eta$ dall'esperienza $\{S_1, A_1, R_2, \ldots, S_T\}$
  Questo è un problema di apprendimento supervisionato

$$S_1, A_1 \rightarrow R_2, S_2$$
$$S_2, A_2 \rightarrow R_3, S_3$$
$$\ldots$$
$$S_{T-1}, A_{T-1} \rightarrow R_T, S_T$$

- Apprendere $s,a \rightarrow r$ è un problema di regressione
- L'apprendimento di $s,a \rightarrow s'$ è un problema di stima della densità
- Scegli la loss function, ad es. errore quadratico medio, …
- Trova i parametri η che minimizzano la perdita empirica

# Alcuni Learning Model

- Table Lookup Model
- Linear Expectation Model
- Linear Gaussian Model
- Gaussian Process Model
- Deep Belief Network Model
- …

# Model Learning con Table Lookup

- Il modello è un MDP esplicito <$\hat{P}$, $\hat{R}$>
- Conta le visite N(s,a) per ogni coppia stato-azione

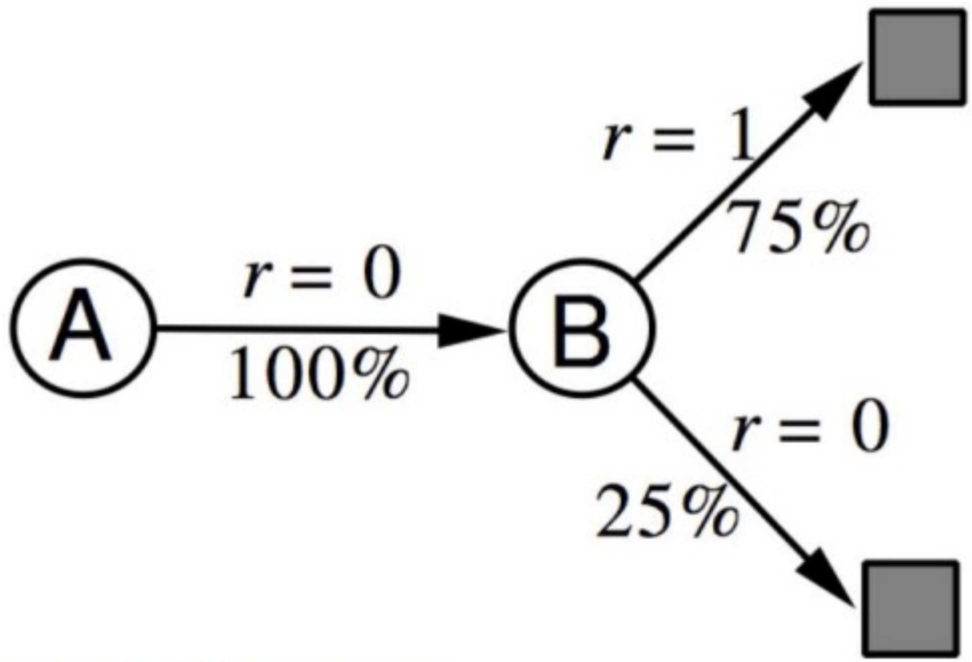
$$\hat{P}^{a}_{s,s'} = \frac{1}{N(s,a)} \sum_{t=1}^{T} \mathbf{1}(S_t, A_t, S_{t+1}; s, a, s')$$

$$\hat{R}^{a}_{s} = \frac{1}{N(s,a)} \sum_{t=1}^{T} \mathbf{1}(S_t, A_t; s, a) R_t$$

- Alternativamente
  - Ad ogni time-step *t* si memorizza la tupla esperienza <$S_t$, $A_t$, $R_{t+1}$, $S_{t+1}$>
  - Per campionare il modello si scelgono casualmente le tuple corrispondenti <s, a,·,·>

# Esempio AB

- Due stati A; B; nessuna scontistica; 8 episodi di esperienza
  - A, 0, B, 0
  - B, 1
  - B, 1
  - B, 1
  - B, 1
  - B, 1
  - B, 1
  - B, 0




- Abbiamo costruito un modello table lookup dall'esperienza

# Planning con un Modello

- Dato un modello $M_\eta = < \hat{P}, \hat{R} >$
- Risolvere il MDP $<S, A, P_\eta, R_\eta>$
- Utilizzando un algoritmo di planning
  - Value iteration
  - Policy iteration
  - Tree search
  - …

# Sample-Based Planning

- Un approccio semplice ma efficace per pianificare
- Utilizzare il modello solo per generare campioni
- Campione di esperienza dal modello

$$S_{t+1} \sim P_\eta(S_{t+1} | S_t, A_t)$$
$$R_{t+1} = \mathcal{R}_\eta(R_{t+1} | S_t, A_t)$$

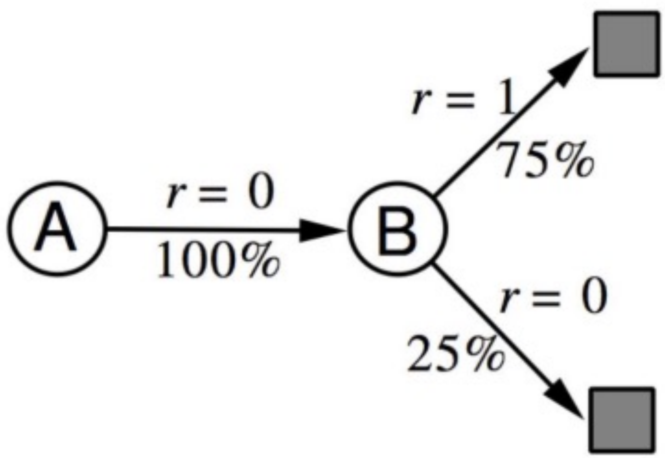
- Applicare il model-free RL ai campioni, ad esempio:
  - Monte-Carlo control
  - Sarsa
  - Q-learning
- I metodi sample-based planning sono spesso più efficienti

# Esempio AB

- Costruire un table-lookup model dall'esperienza reale
- Applicare il model-free RL all'esperienza campionata

Real experience

1. A, 0, B, 0
2. B, 1
3. B, 1
4. B, 1
5. B, 1
6. B, 1
7. B, 1
8. B, 0




e.g. Monte-Carlo learning: V (A) = 1; V (B) = 0:75

Sampled experience

1. B, 1
2. B, 0
3. B, 1
4. A, 0, B, 1
5. B, 1
6. A, 0, B, 1
7. B, 1
8. B, 0

# Planning con un Modello Poco Accurato

- Dato un modello imperfetto $<P_\eta, R_\eta> \neq <P, R>$
  - Le performance del model-based RL sono limitate a policy ottimali per MDP approssimati $<S, A, P_\eta, R_\eta>$
  - Ovvero, il model-based RL è efficace quanto il modello stimato
- Quando il modello è poco accurato, il processo di planning calcolerà una policy sub-ottimale
  - Soluzione 1 – Quando il modello è sbagliato, utilizzare il model-free RL
  - Soluzione 2 – Ragionare esplicitamente sull'incertezza del modello

# Integrated Architectures

# Esperienza reale e simulata

- Consideriamo due fonti di esperienza
- Esperienza reale – Campionata dall'ambiente (true MDP)

$$S' \sim \mathcal{P}^{a}_{s,s'}$$
$$R = \mathcal{R}^{a}_{s}$$

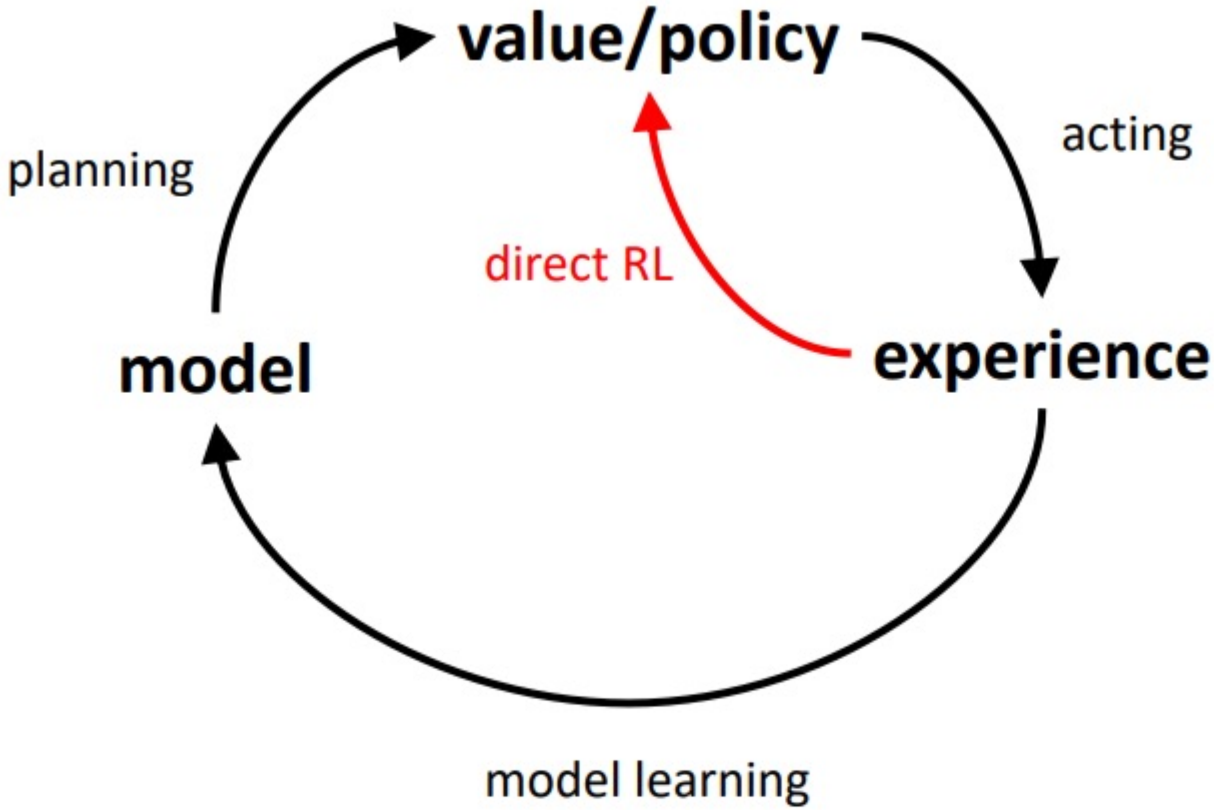
- Esperienza simulata – Campionata dal modello (MDP approssimato)

$$S' \sim P\eta(S'|S, A)$$
$$R = \mathcal{R}\eta(R|S, A)$$

# Integrare Learning e Planning

- Model-Free RL
  - Nessun modello
  - Apprendere la value function (e/o la policy) da un'esperienza reale
- Model-based RL (usando Sample-Based Planning)
  - Apprendere un modello dall'esperienza reale
  - Pianificare la value function (e/o la policy) dall'esperienza simulata
- Dyna
  - Apprendere un modello dall'esperienza reale
  - Apprendere e pianificare la value function (e/o la policy) a partire da esperienze reali e simulate

# Architettura Dyna



# Algoritmo Dyna-Q

Initialize $Q(s,a)$ and $Model(s,a)$ for all $s \in \mathcal{S}$ and $a \in \mathcal{A}(s)$

Do forever:

(a) $S \leftarrow$ current (nonterminal) state

(b) $A \leftarrow \varepsilon\text{-greedy}(S,Q)$

(c) Execute action $A$; observe resultant reward, $R$, and state, $S'$

(d) $Q(S,A) \leftarrow Q(S,A) + \alpha\big[R + \gamma \max_a Q(S',a) - Q(S,A)\big]$

(e) $Model(S,A) \leftarrow R, S'$ (assuming deterministic environment)

(f) Repeat $n$ times:

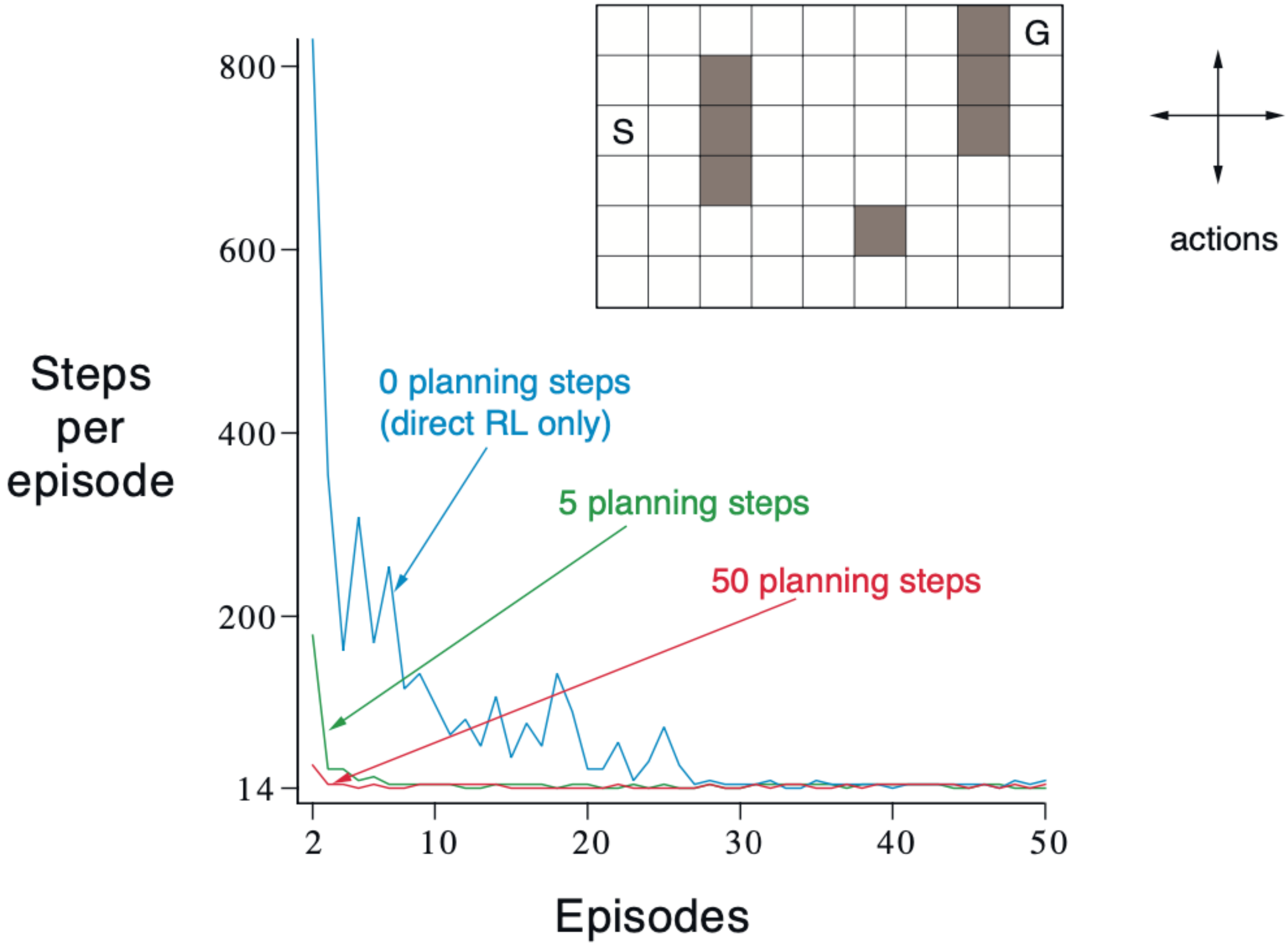
$S \leftarrow$ random previously observed state

$A \leftarrow$ random action previously taken in $S$

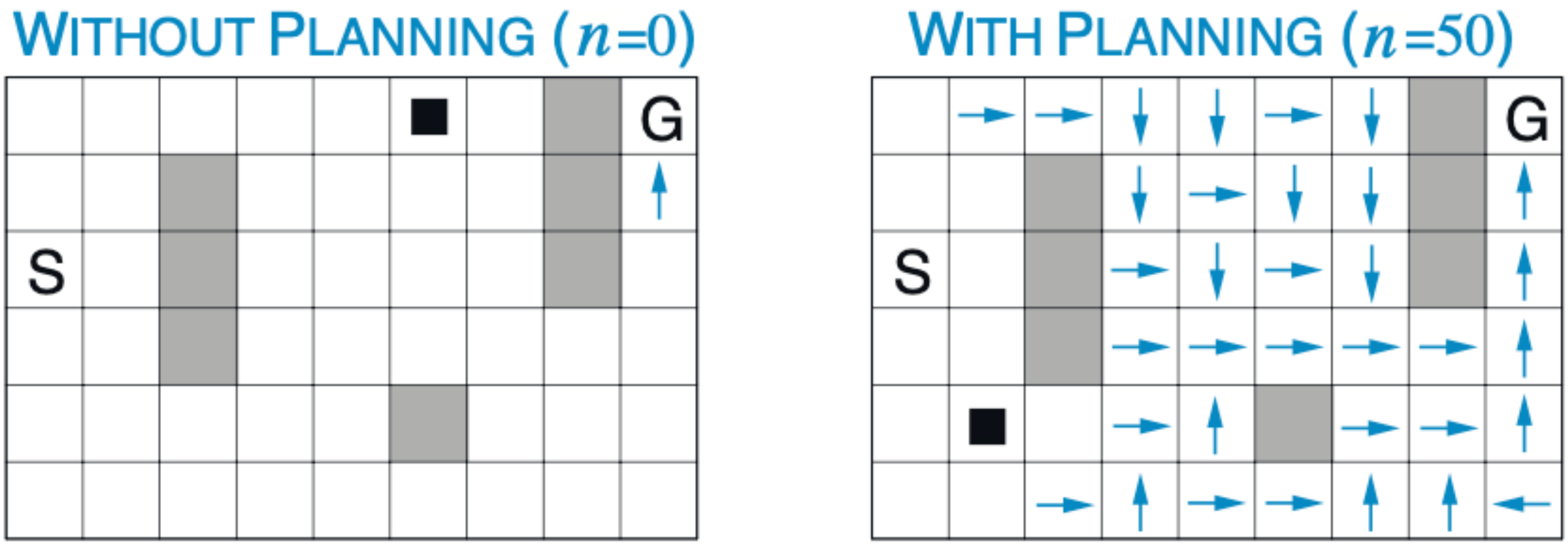
$R, S' \leftarrow Model(S,A)$

$Q(S,A) \leftarrow Q(S,A) + \alpha\big[R + \gamma \max_a Q(S',a) - Q(S,A)\big]$

# Esempio Maze - Learning curve

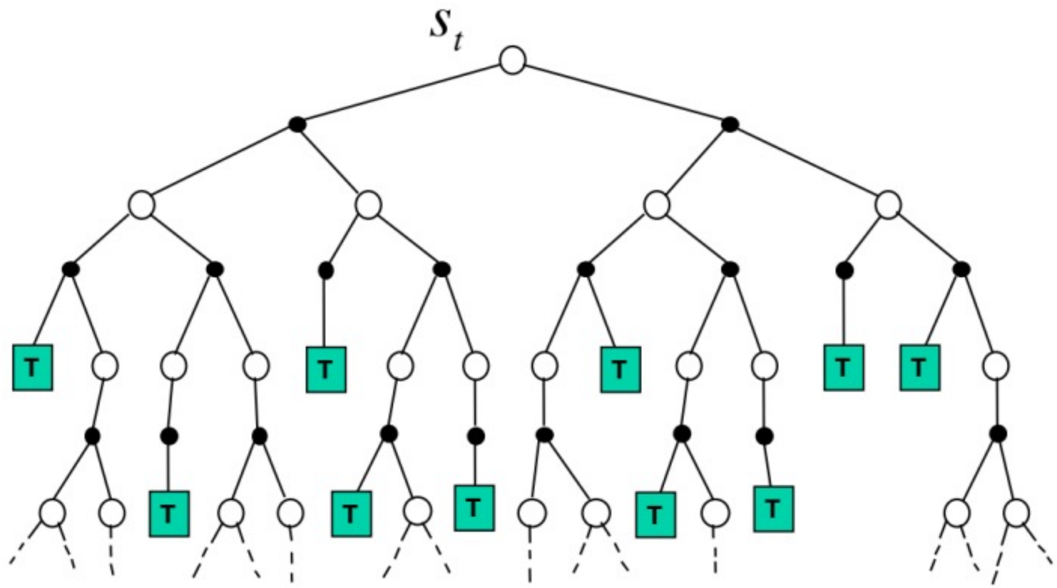


# Esempio Maze – Policy at 2^ episode



# Learning with Simulation

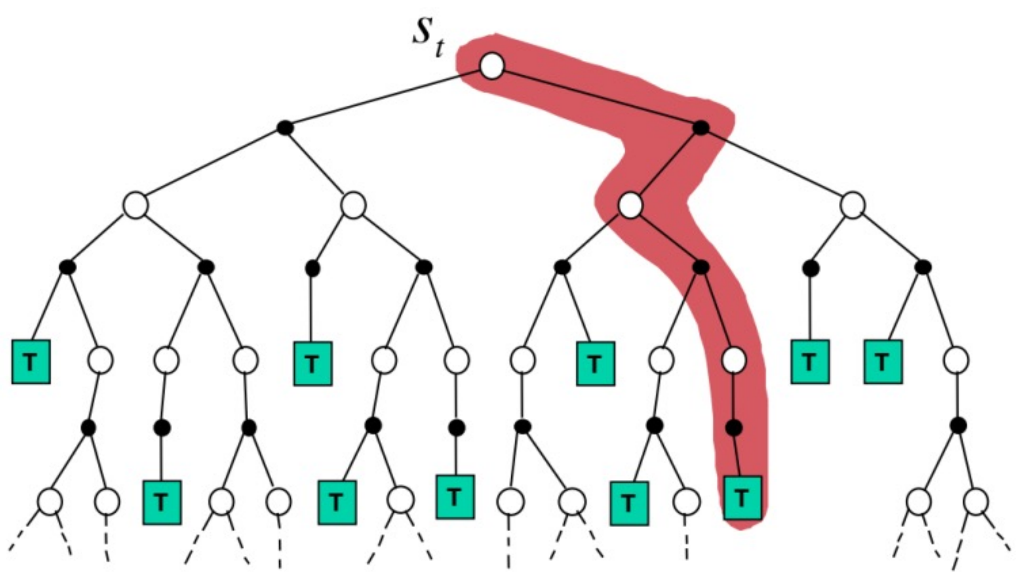
# Forward Search




- Gli algoritmi forward search selezionano l’azione migliore tramite il lookahead
- Costruiscono un albero di ricerca con lo stato corrente $s_t$ alla radice
- Usare un modello del MDP per ‘guardare in avanti’
- Non è necessario risolvere l’intero MDP, ma solo i sub-MDP da ora

# Simulation-Based Search

- Si basano sul paradigma forward search ed usano il sample-based planning
- Simulare episodi di esperienza da ora con il modello
- Applicare il model-free RL a episodi simulati



# Simulation-Based Search (2)

- Simulare episodi di esperienza con il modello

$$\{s_t^k, A_t^k, R_{t+1}^k; \ldots, S_T^k\} \sim \mathcal{M}_v$$

- Applicare il model-free RL a episodi simulati
  - Monte-Carlo control $\Longrightarrow$ Monte-Carlo search
  - SARSA $\Longrightarrow$ TD search

# Simple Monte-Carlo Search

- Dato un modello $M_v$ e una simulation policy $\pi$
- Per ogni azione $a \in A$
  - Simulare *K* episodi a partire dallo stato attuale (reale) $s_t$

$$\{st, a, R_{t+1}^k; S_{t+1}^k, A_{t+1}^k \dots, S_T^k\} \sim \mathcal{M}v, \pi$$

  - Valutare le azioni in base al guadagno medio (Monte-Carlo evaluation)

$$Q(s_t, a) = \frac{1}{K} \sum_{k=1}^{K} G_t \to^P q_\pi(s_t, a)$$

- Selezionare l'azione corrente (reale) con il valore massimo

$$a_t = \arg\max_{a \in \mathcal{A}} Q(s_t, a)$$

- Questo corrisponde ad 1 passo di miglioramento di policy

# Monte-Carlo Tree Search (MCTS) - Evaluate

- Dato un modello $M_v$
- Simulare *K* episodi a partire dallo stato attuale $s_t$ usando la simulation policy $\pi$

$$\{s_t, A_t^k, R_{t+1}^k; S_{t+1}^k, A_{t+1}^k \dots, S_T^k\} \sim \mathcal{M}_v, \pi$$

- Costruire un albero di ricerca contenente gli stati visitati e le azioni
- Valutare gli stati *Q*(*s*, *a*) attraverso il guadagno medio degli episodi da *s*, *a*

$$Q(s, a) = \frac{1}{\mathrm{N(s, a)}} \sum_{k=1}^{K} \sum_{\mathrm{u=t}}^{\mathrm{T}} \mathbf{1}(S_u, A_u; s, a)\, G_{\mathrm{u}} \to^P q_\pi(s, a)$$

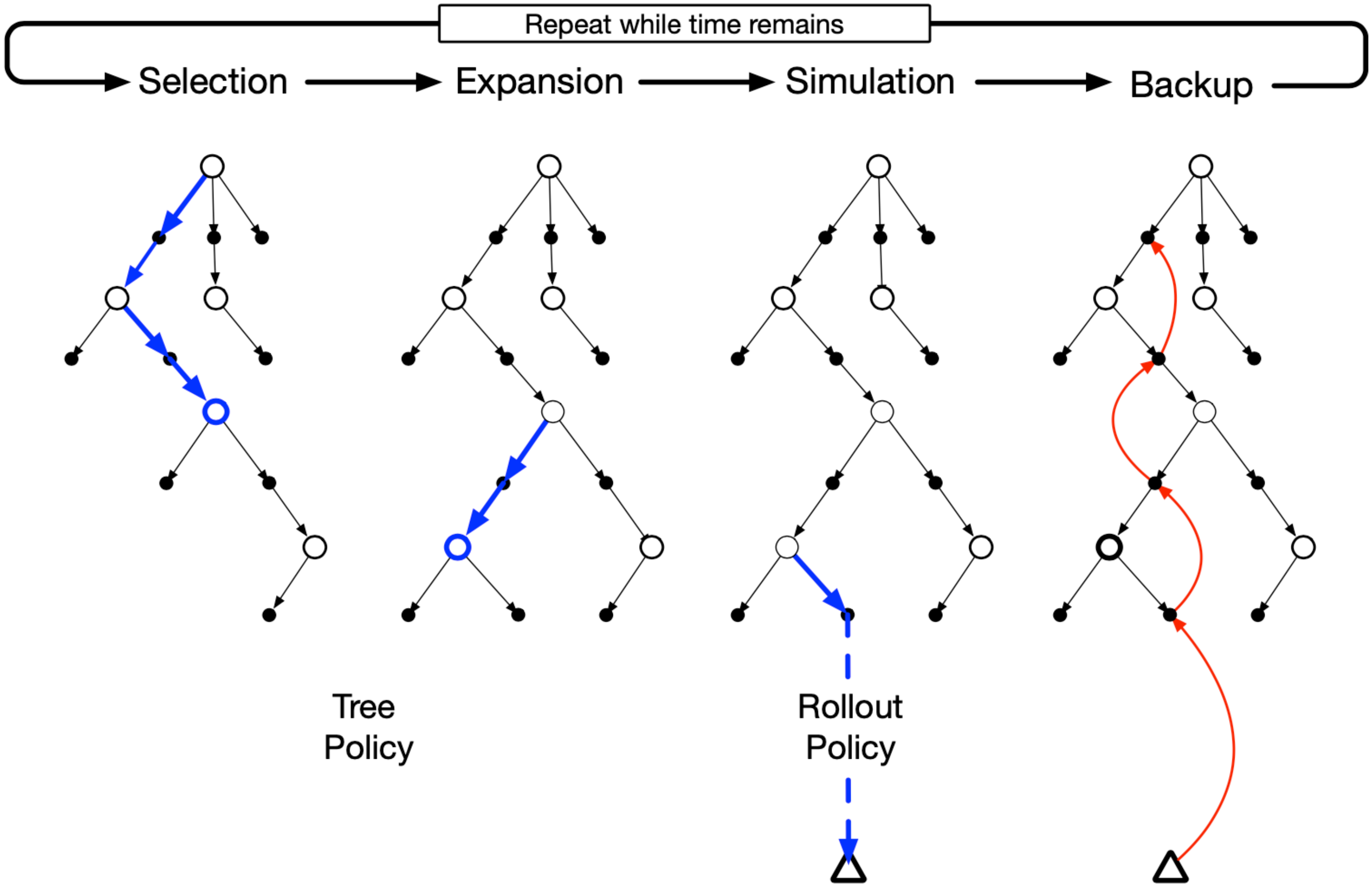
- Al termine della ricerca, selezionare l'azione corrente (reale) con il valore massimo nell'albero di ricerca

$$a_t = \arg\max_{a \in \mathcal{A}} Q(s_t, a)$$

# Monte-Carlo Tree Search (MCTS) - Simulate

- In MCTS la simulation policy $\pi$ migliora
- Ogni simulazione consiste in due fasi (in-tree, out-of-tree)
  - Tree policy (migliora): scegliere le azioni per massimizzare $Q(S, A)$
  - Default policy (fissa): scegliere le azioni in modo casuale
- Ripetere (per ogni simulazione)
  - Valutare gli stati $Q(S, A)$ tramite la Monte-Carlo evaluation
  - Migliorare la tree policy, ad esempio attraverso $\epsilon$-greedy(Q)
- Monte-Carlo control applicato all'esperienza simulata
- Converge all'albero di ricerca ottimale, $Q(S, A) \rightarrow q_*(S, A)$

# Monte-Carlo Tree Search (MCTS)



# Vantaggi di MCTS

- Ricerca best-first altamente selettiva
- Valuta gli stati in modo dinamico (a differenza di DP)
- Utilizza il campionamento per risolvere la curse of dimensionality
- Funziona per modelli black-box (richiede solo i campioni)
- Efficiente dal punto di vista computazionale, in qualsiasi momento, parallelizzabile

# Temporal-Difference Search

- Simulation-based search
- Utilizzo di TD invece di MC (bootstrapping)
- MCTS applica il MC Control al sub-MDP
- La TD search applica SARSA al sub-MDP

# MC vs TD search

- Per il model-free RL, il bootstrapping è utile
  - Il TD learning riduce la varianza ma aumenta il bias
  - Il TD learning è solitamente più efficiente di quello MC
  - TD(λ) può essere molto più efficiente di MC
- Per la simulation-based search, il bootstrapping è utile
  - La TD search riduce la varianza ma aumenta il bias
  - La TD search è solitamente più efficiente della MC search
  - La TD(λ) search può essere molto più efficiente della MC search

# TD search

- Simulare episodi a partire dallo stato attuale (reale) $s_t$
- Stima della action-value function $Q(s, a)$
- Per ogni passo della simulazione, aggiornare gli action-value tramite SARSA

$$\Delta Q(S, A) = \alpha\left(R + \gamma Q(S', A') - Q(S, A)\right)$$

- Selezionare le azioni in base agli action-value $Q(s, a)$ (ad esempio $\epsilon$-greedy)
- Può anche utilizzare una funzione di approssimazione per $Q$

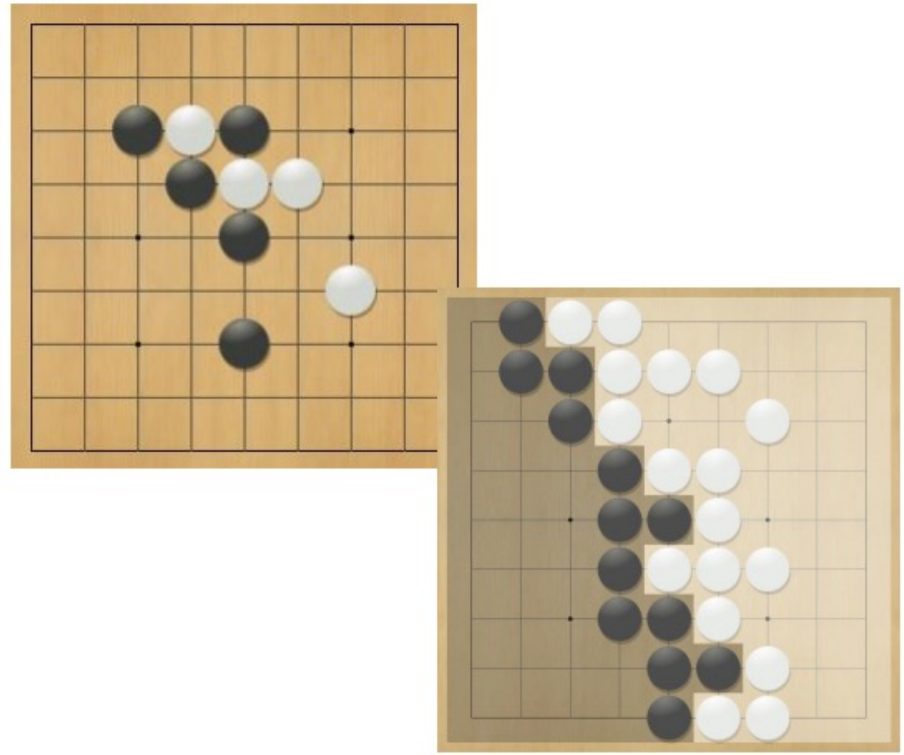
# GO Case Study

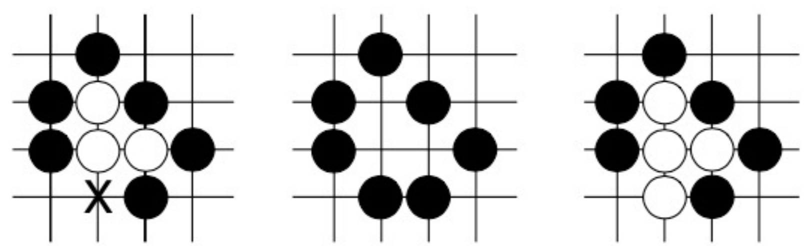
# Go

- L'antico gioco orientale GO risale a 2500 anni fa
- Considerato il più difficile gioco da tavolo classico
- Considerato un compito complesso per l'IA (da McCarthy)
- La ricerca tradizionale basata su game-tree ha da sempre fallito per GO

# Regole

- Di solito si gioca su un tavolo 19x19, ma anche 13x13 o 9x9
- Regole semplici, strategia complessa
- I due giocatori posizionano le pietre alternativamente
- Le pietre circondate vengono catturate e rimosse
- Il giocatore con più territori vince la partita





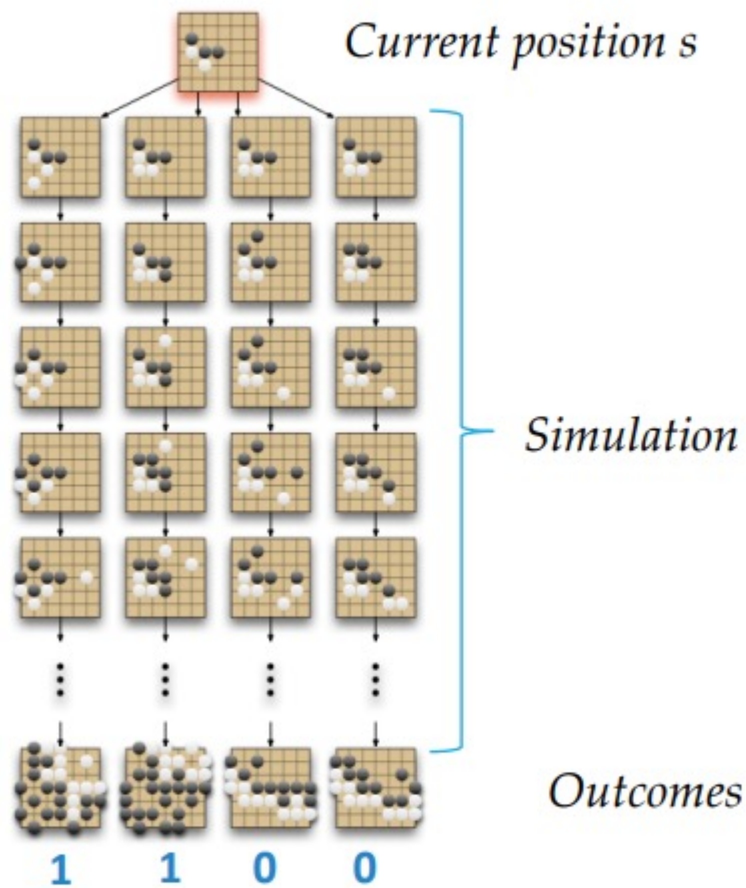
# Valutazione della posizione in GO

- Quanto è valida una posizione $s$?
- Reward function (non scontata):
  - $R_t$ = 0 per tutti gli step non terminali $t < T$
  - $R_T$ = 1 se il giocatore nero vince
  - $R_T$ = 0 se il giocatore bianco vince
- La policy $\pi = \langle \pi_B, \pi_W \rangle$ seleziona le mosse per entrambi i giocatori (B,W)
- Value function (quanto è valida la posizione $s$)

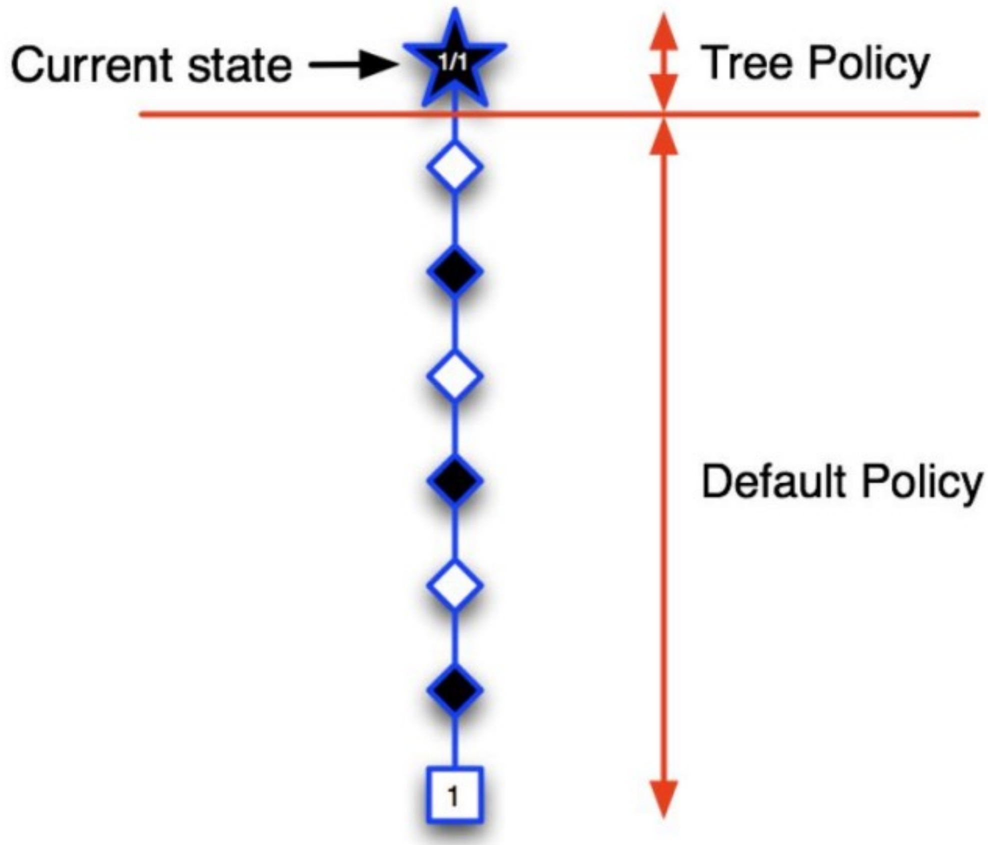
$$v_\pi(s) = \mathbb{E}[R_T | S = s] = P(\text{Black wins} | S = s)$$
$$v_*(s) = \max_{\pi_B} \min_{\pi_W} v_\pi(s)$$

# Go – Monte-Carlo Evaluation

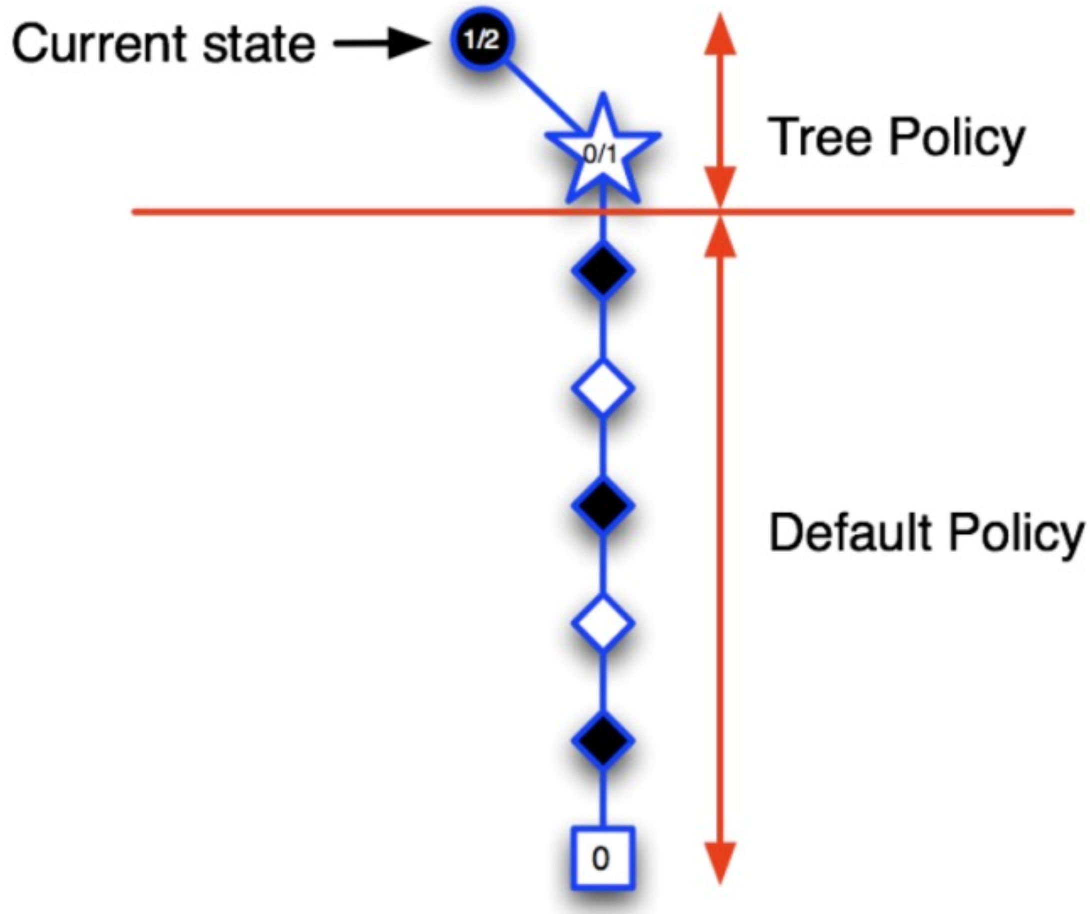
$$V(s) = 2/4 = 0.5$$



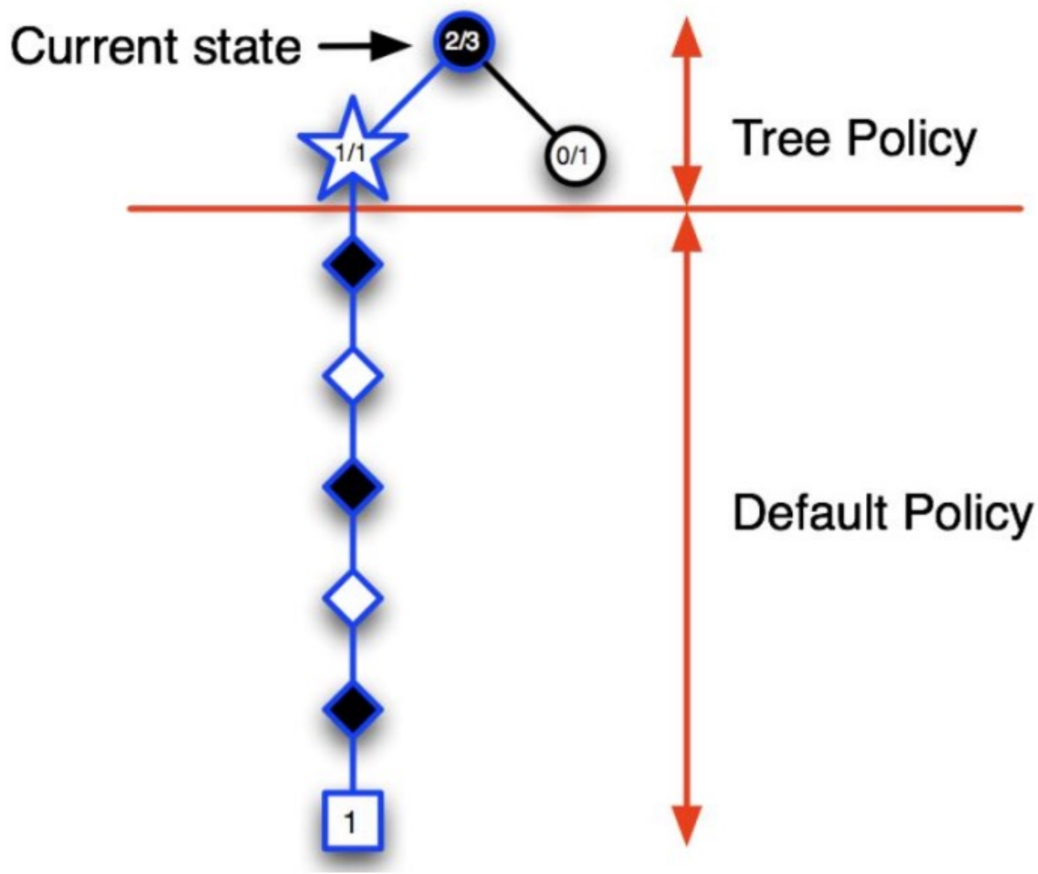
# Applicazione Monte-Carlo Tree Search



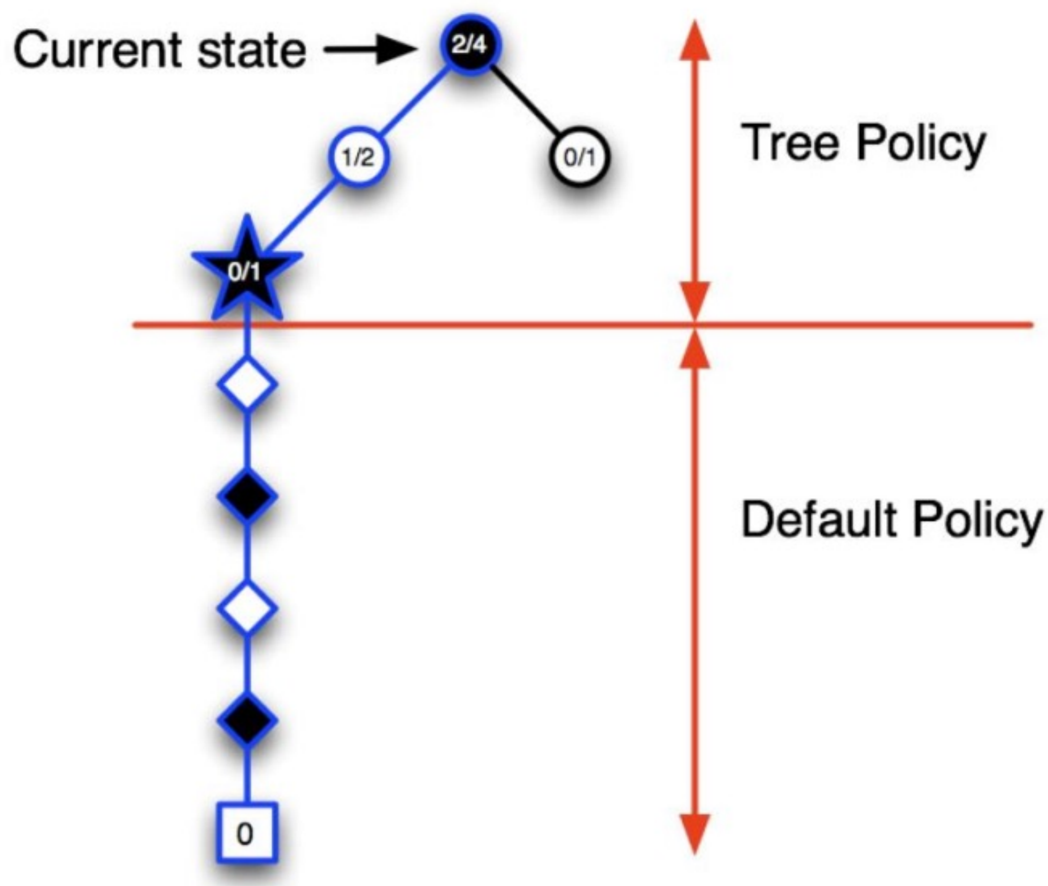
# Applicazione Monte-Carlo Tree Search (2)



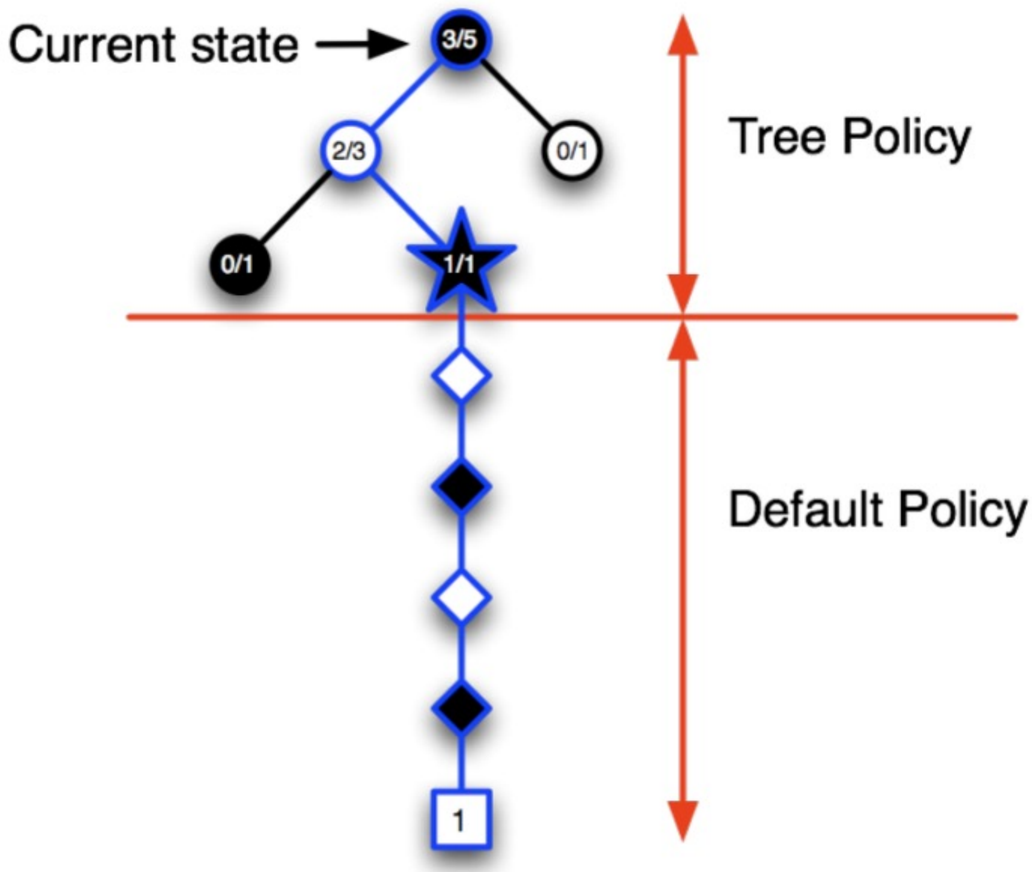
# Applicazione Monte-Carlo Tree Search (3)



# Applicazione Monte-Carlo Tree Search (4)

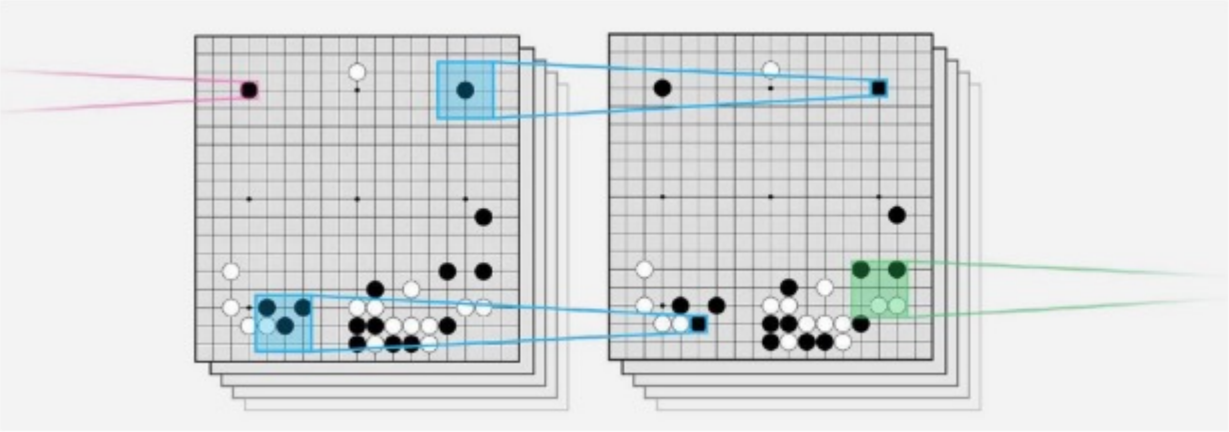


# Applicazione Monte-Carlo Tree Search (5)



# Alpha-Go

Reti neurali convoluzionali per estrarre una rappresentazione significativa dello stato

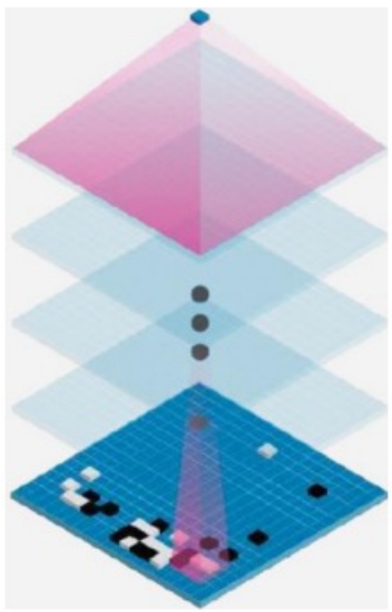


- Combinando quanto visto finora
  - Apprendimento della value function
  - Policy gradient
  - Monte-Carlo Tree Search

# Deep Learning in Alpha-Go

Value network

$V(s)$

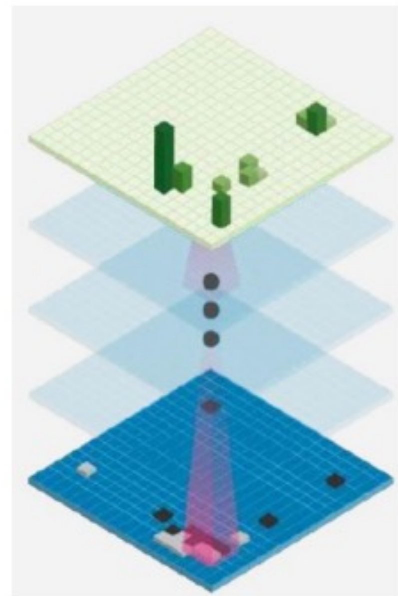


$s$

Il giocatore vincerà con la tavola attuale?

$P(a|s)$

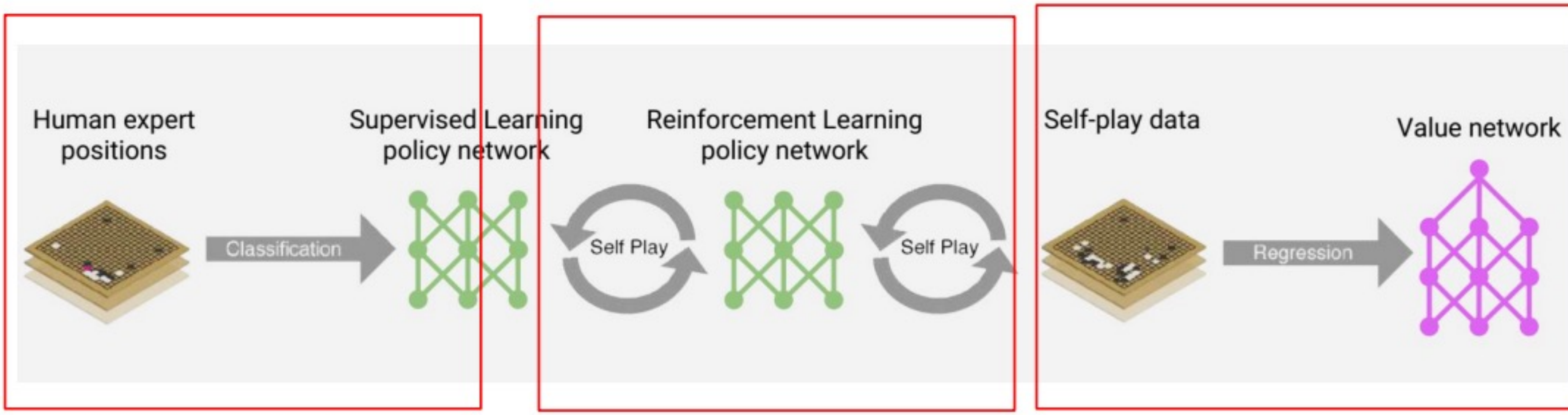
Policy network



$s$

Quanta preferenza per una mossa specifica nella tavola attuale?

# Supervised-Reinforcement Learning Pipeline (Offline phase)

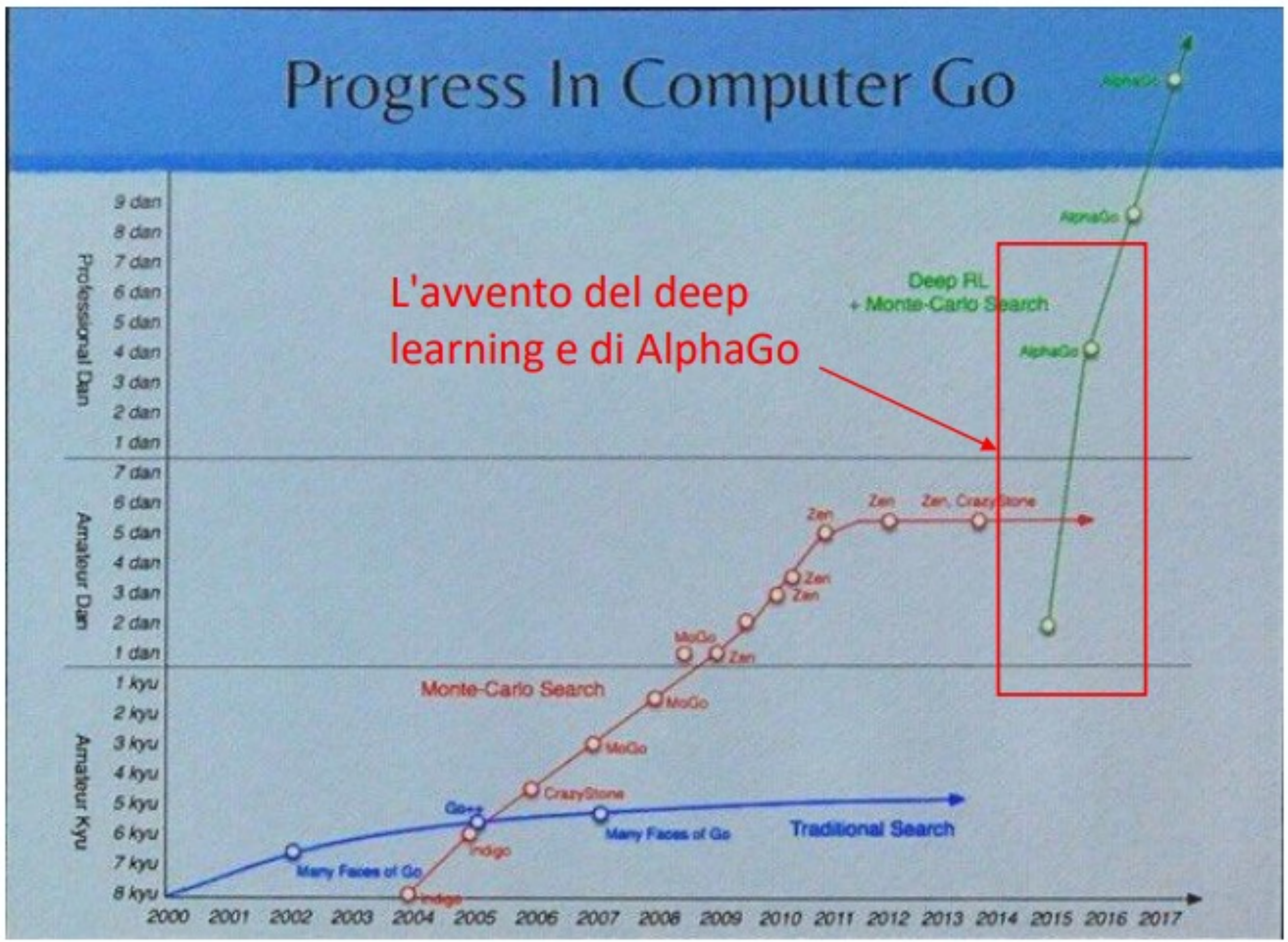



Max likelihood policy on human expert games

Policy gradient on self-play

Value function learning on self-play (by policy)

# Progress in Computer Go



# Take home messages

- Model-based RL è efficace
  - Se si conoscono le regole del mondo (gioco) è possibile utilizzarle per simulare l'esperienza
  - Può utilizzare il modello dell'ambiente per simulare le esperienze
  - Può integrare esperienze simulate con esperienze del mondo reale (Dyna)
- Monte-Carlo Tree Search
  - Valutare il valore di uno stato attuale guardando avanti negli episodi campionati nella simulazione
  - Funziona per modelli black-box ed è altamente efficiente
- TD Search
  - Aggiornamento degli action-state value tramite SARSA su episodi simulati
  - Come al solito, riduce la varianza rispetto a MCTS, ma aumenta il bias
  - Probabilmente più efficiente di MCTS