TÉCNICO LISBOA

Planning, Learning and Intelligent Decision Making

Lecture 12
(at last!!)

PADInt 2024

Announcements

- Don't forget to register for the exam!
  - Registration is **mandatory!**
- Registrations are already open
  - **Registrations close on April 7th, at 17h00**

Announcements

- **Make sure to register in the right campus!**

TagusPark

Alameda

TagusPark

Alameda

Evaluations

Projects	Beginning	End
Project: Lab 1 (including homework 1)	20/02/2024 21:00	05/03/2024 21:00
Project: Lab 2 (including homework 2)	27/02/2024 21:00	12/03/2024 21:00
Project: Lab 3 (including homework 3)	05/03/2024 21:00	19/03/2024 21:00
Project: Lab 4 (including homework 4)	12/03/2024 21:00	26/03/2024 21:00

Tests/Exams	Day	Beginning	End	Enrolment period	Rooms	Courses
Exam: 1ª Época	11/04/2024	13:00	15:00	25/03/2024 08:00 - 07/04/2024 17:00	A1	MECD, Min-EG, Min-RSI, MEIC-A, MEIC-T
Exam: 1ª Época	11/04/2024	13:00	15:00	25/03/2024 08:00 - 07/04/2024 17:00	FA2 F4 F3 F2	MECD, Min-EG, Min-RSI, MEIC-A, MEIC-T
Exam: 2ª Época	01/07/2024	10:30	12:30	01/06/2024 09:00 - 25/06/2024 17:00	A5	MECD, Min-EG, Min-RSI, MEIC-A, MEIC-T
Exam: 2ª Época	01/07/2024	10:30	12:30	01/06/2024 09:00 - 25/06/2024 17:00		MECD, Min-EG, Min-RSI, MEIC-A, MEIC-T
Exam: Época Especial	22/07/2024	10:30	12:30		0 - 25	MECD, Min-EG, Min-RSI, MEIC-A, MEIC-T
Exam: Época Especial	22/07/2024	10:30	12:30		F2	MECD, Min-EG, Min-RSI, MEIC-A, MEIC-T

Rules/tips for the exam

- The exam is open book/open notes
  - You can take **any written material** you want
  - You can take a **calculator**
  - You **cannot take communication devices** (laptop, notepad, cell phone, etc.)

Rules/tips (cont.)

- You can answer in Portuguese or in English
- Questions have space for answering:

(a) **(1.0 pts.)** Suppose that you want to learn the XOR function using a linear classifier such as logistic regression or an SVM. Is the data linearly separable? Explain.

Rules/tips (cont.)

- You can answer in Portuguese or in English
- Questions have space for answering
  - If you make a mess, **indicate clearly** where the answer is

Rules/tips (cont.)

- Make sure that you identify **all the pages**
  - Tests will be scanned for grading, so every page must have an id

Rules/tips (cont.)

- Make sure that you identify **all the pages**
  - Tests will be scanned for grading, so every page must have an id
- At the end, please make sure to provide feedback on how the test went:

The grade that you give yourself + justification

Rules/tips (cont.)

- **Start with the easy/quick questions**
- **If you get stuck in a question, move on**
  - You can revisit it later if you have the time
- **Have your notes organized and indexed**
- **Make sure you can use your calculator efficiently**

Exploration vs Exploitation

Exploration vs exploitation

- You have 3 machines in the casino

Machine 1

Machine 2

Machine 3

Exploration vs exploitation

- You play each machine once
  - Machine 1: You gain 10$
  - Machine 2: You gain 2$
  - Machine 3: You lose 1$
- Which machine should you play next?

Exploration vs exploitation

- You play each machine **twice**
  - Machine 1: You gain 10$, 2$
  - Machine 2: You gain 2$, 4$
  - Machine 3: You lose 1$, win 15$
- Which machine should you play next?

Exploration vs exploitation

- Pure exploration vs exploitation problem
  - When to explore (try new machines)?
  - When to stop exploring and start exploiting (play the apparently best machine)?

Multi-armed bandit

Multi-armed bandit

Multi-armed bandit

Multi-armed bandit

- Sequential decision problem
- Game between agent and "nature"
- At each time step t:
  - Agent selects an action
  - Nature selects cost function
  - Agent gets the cost for its action

Multi-armed bandit

- How can we play this game?

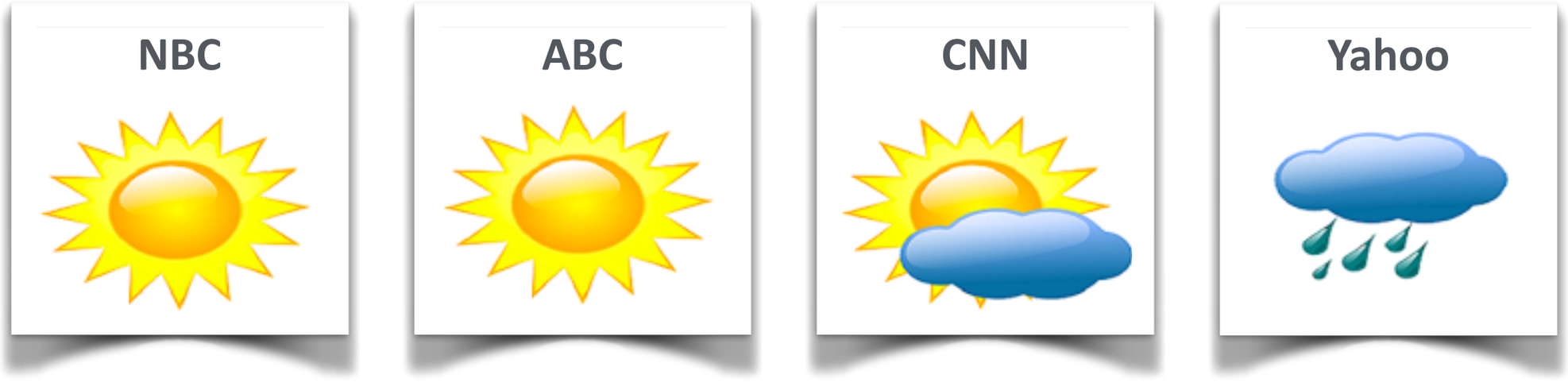
... hard!

Sequential prediction

- Let's play a simpler game

Example

- You are a weather forecaster
- You have access to forecasts from different sources



Which source should you follow?

Example

- Suppose that you known that one source is always right
- How would you do this?

Follow the majority vote!

Example

- Possibility 1:
  - You get the prediction right
  - The cost is 0!


HELL YEAH!

Example

- Possibility 2:
  - You get the prediction wrong
  - You can eliminate half of your sources

How many mistakes?

- Number of sources: N
- Maximum number of (valid) sources after M mistakes:

$$\frac{N}{2^M}$$

- There is always at least one valid source (always right)

$$\frac{N}{2^M} \geq 1$$

- Maximum number of mistakes:

$$M \leq \log_2(N)$$

Nice!

Example

- Even if you have exponentially many sources, you can still manage

What if no source is always right?

Example

- Define a "confidence-level" for your sources
- Still follow the majority vote

Confidence: 1

Confidence: 1

Confidence: 1

Confidence: 1

Example

- Possibility 1:
  - You get the prediction right
  - The cost is 0!


HELL YEAH!

Example

- Possibility 2:
  - You get the prediction wrong
  - You can no longer eliminate half of the sources

Example

- ... but you can decrease by η your confidence in those that failed

Confidence: 1

Confidence: 1

Confidence: 1

Confidence: 1

Example

- ... but you can decrease by η your confidence in those that failed

Confidence: 1

Confidence: 1

Confidence: 0.8

Confidence: 0.8

How many mistakes?

- Total confidence before mistake:

$$W_t = \sum_{n=1}^{N} w_t(n)$$

- After a mistake, at least $1/2$ of the sources decrease by a factor of $(1-\eta)$, with $\eta < 1/2$ so

$$W_{t+1} \leq \underbrace{\left(\frac{1}{2} + \frac{1}{2}(1-\eta)\right)}_{<1} W_t$$

How many mistakes?

- Total confidence before mistake:

$$W_t = \sum_{n=1}^{N} w_t(n)$$

- After a mistake, at least $1/2$ of the sources decrease by a factor of $(1-\eta)$, with $\eta < 1/2$ so

$$W_{t+1} \leq \left(\frac{1}{2} + \frac{1}{2}(1-\eta)\right) W_t \leq \left(1 - \frac{\eta}{2}\right) W_t$$

At every mistake, total weight decreases by a factor of $1 - \frac{\eta}{2}$

How many mistakes?

- Number of sources: N
- Total confidence after M mistakes:

$$N\left(1-\frac{\eta}{2}\right)^{M}$$

- If the best source made m mistakes, then

$$N\left(1-\frac{\eta}{2}\right)^{M} \geq (1-\eta)^{m}$$

- Maximum number of mistakes:

$$M \leq 2(1+\eta)m + \frac{2\log N}{\eta}$$

Logarithmic in number of sources

Important aspects

- We measure our performance compared against that of the best "guess"
- Usually, performance of the best guess can only be assessed a posteriori

Summarizing...

Weighted majority algorithm:

- Given a set of N "predictors" and $\eta < 1/2$
- Initialize predictor weights to $w_0(n) = 1$, $n = 1, ..., N$
- Make prediction based on the (weighted) majority vote
- Update weights of all wrong predictors as

$$w_{t+1}(n) = w_t(n)(1 - \eta)$$

Sequential prediction

- Let's consider a slightly more complex game
- At each time step t:
  - Agent selects an action a_t from a finite set $\mathcal{A}$
  - Nature selects cost function
  - Nature discloses cost function
- We make no assumptions on how nature selects cost function

Sequential prediction

- Use a similar principle:
  - Define a "confidence-level" for each action

Action 1	**Action 2**	**Action 3**	**Action 4**
			
Confidence: 1	Confidence: 1	Confidence: 1	Confidence: 1

Sequential prediction

- Select each action "proportionally" to its confidence:

$$p_t(a) = \frac{w_t(a)}{\sum_{a' \in \mathcal{A}} w_t(a')}$$

Sequential prediction

- When cost is revealed, we update each "confidence" according to the corresponding cost:

Cost: 0.1	**Cost: 0.3**	**Cost: 1**	**Cost: 0.7**
			
Confidence: 1	Confidence: 1	Confidence: 1	Confidence: 1

Sequential prediction

- When cost is revealed, we update each "confidence" according to the corresponding cost:

Cost: 0.1	**Cost: 0.3**	**Cost: 1**	**Cost: 0.7**
			
Confidence: 0.9	Confidence: 0.7	Confidence: 0.4	Confidence: 0.5

Sequential prediction

- When cost is revealed, we update each "confidence" according to the corresponding cost:

$$w_{t+1}(a) = w_t(a)e^{-\eta c_t(a)}$$

Cost

Sequential prediction

- When cost is revealed, we update each "confidence" according to the corresponding cost:

$$w_{t+1}(a) = w_t(a) \underbrace{e^{-\eta c_t(a)}}_{<1}$$

- Then, at each step t,

$$w_t(a) = e^{-\eta \sum_{\tau=0}^{t-1} c_\tau(a)}$$

Total cost

Sequential prediction

- To measure the performance, we compare our total (expected) cost with that of the best action (in hindsight):

$$R_T = \mathbb{E}\left[\sum_{t=0}^{T-1} c_t(a_t)\right] - \min_{a \in \mathcal{A}} \sum_{t=0}^{T-1} c_t(a)$$

- The value R_T is called the regret at time T
  - It measures how much the agent regrets, in hindsight, not having followed the minimizing action

How much regret?

- Initial weights: N
- Total confidence after T steps smaller than:

$$N e^{-\eta \mathbb{E}\left[\sum_{t=0}^{T-1} c_t(a_t)\right]} \rho$$

Constant that depends on η and N

How much regret?

- Initial weights: N
- Total confidence after T steps smaller than:

$$Ne^{-\eta\mathbb{E}\left[\sum_{t=0}^{T-1} c_t(a_t)\right]}\rho$$

- Comparing with the best action (in hindsight):

$$Ne^{-\eta\mathbb{E}\left[\sum_{t=0}^{T-1} c_t(a_t)\right]}\rho \geq \min_{a\in\mathcal{A}} e^{-\eta\sum_{t=0}^{T-1} c_t(a)}$$

- Finally,

$$R_t \leq \frac{\log N}{\eta} + \frac{\rho}{\eta}$$

How much regret?

- Selecting η properly, we get the final bound:

$$R_T \leq \sqrt{\frac{T}{2} \log N}$$

Exponentially Weighted Averager (EWA)

- This algorithm for sequential prediction is called **exponentially weighted averager**
  - It makes no assumptions on the process by which costs are selected (can be adversarial)
  - Depends logarithmically on the number of actions (works well even if there is an exponentially large number of actions to try)
  - Its regret is **sublinear in** T

No-regret prediction

- What does it mean that the regret is sublinear in T?
- Recall that, for the EWA,

$$R_T \leq \sqrt{\frac{T}{2} \log N}$$

Grows with $\sqrt{T}$
(slower than T)

No-regret prediction

- What does it mean that the regret is sublinear in T?
- Recall that, for the EWA,

$$R_T \leq \sqrt{\frac{T}{2} \log N}$$

- If we compute the average regret per step:

$$\frac{R_T}{T} \leq \sqrt{\frac{\log N}{2T}} \xrightarrow[T \to \infty]{} 0$$

No regret algorithm

Summarizing

Exponentially weighted averager:

- Given a set of N actions and $\eta > 0$
- Initialize weights to $w_0(a) = 1$, $a \in \mathcal{A}$
- Select an action according to the probabilities

$$p_t(a) = \frac{w_t(a)}{\sum_{a' \in \mathcal{A}} w_t(a')}$$

- Update weights of all actions as

$$w_{t+1}(a) = w_t(a) e^{-\eta c_t(a)}$$

Multi-armed bandit

Multi-armed bandit

Multi-armed bandit

- Sequential decision problem
- Game between agent and "nature"
- At each time step t:
  - Agent selects an action a_t from a finite set $\mathcal{A}$
  - Nature selects cost function
  - Agent gets the cost for its action

Multi-armed bandit

- How can we play this game?

Adversarial bandits

Multi-armed bandit

- Sequential decision problem
- Game between agent and "nature"
- At each time step t:
  - Agent selects an action a_t from a finite set $\mathcal{A}$
  - Nature selects cost function
  - Agent gets the cost for its action
- We make no assumptions on how the cost is selected (can be adversarial)

Setting is quite similar to EWA...

In EWA...

- We define a confidence level for each action
- Select each action "proportionally" to its confidence:

$$p_t(a) = \frac{w_t(a)}{\sum_{a' \in \mathcal{A}} w_t(a')}$$

- When cost is revealed, we update each "confidence" according to the corresponding cost:

$$w_{t+1}(a) = w_t(a) e^{-\eta c_t(a)}$$

Cost

However…

However...

- We do not observe the cost for all actions, so we can only update weight for current action

$$w_{t+1}(a_t) = w_t(a_t)e^{-\eta c_t(a_t)}$$

- Actions experimented more often will have more updates, which will unbalance weights!

We must compensate

EXP3 - Step 1

- Ideally, as in EWA, we would like

$$w_t(a) = e^{-\eta \sum_{t=0}^{T-1} c_t(a)}$$

- However, we have

$$w_t(a) = e^{-\eta \sum_{t=0}^{T-1} c_t(a)\mathbb{I}(a_t=a)}$$

Only updated
if $a_t = a$

EXP3

- By construction,

$$\mathbb{P}\left[a_t = a\right] = p_t(a)$$

- So, in expectation,

$$w_t(a) \approx e^{-\eta \sum_{t=0}^{T-1} c_t(a) p_t(a)}$$

- **Change:**
  - Use the update:

$$w_{t+1}(a_t) = w_t(a_t) e^{-\eta \frac{c_t(a_t)}{p_t(a_t)}}$$

Compensate

Summarizing

EXP3

- Given a set of N actions and $\eta > 0$
- Initialize weights to $w_0(a) = 1$, $a \in \mathcal{A}$
- Select an action according to the probabilities

$$p_t(a) = \frac{w_t(a)}{\sum_{a' \in \mathcal{A}} w_t(a')}$$

- Update weights of all actions as

$$w_{t+1}(a_t) = w_t(a_t) e^{-\eta \frac{c_t(a_t)}{p_t(a_t)}}$$

EXP3

- To measure the performance, we compare the total (expected) cost with that of the best action (in hindsight):

$$R_T = \mathbb{E}\left[\sum_{t=0}^{T-1} c_t(a_t)\right] - \min_{a \in \mathcal{A}} \sum_{t=0}^{T-1} c_t(a)$$

How much regret?

- How much regret?
  - The analysis is similar to the WM and EWA, but involves trickier manipulations
  - The final bound on the regret is

$$R_T \leq \sqrt{2TN \log N}$$

EXP3

- This algorithm for adversarial multi-armed bandits is called **"EXPonentially weighted algorithm for EXPloration and EXPloitation"**, or EXP3

- It makes no assumptions on the process by which costs are selected (can be adversarial)

- Depends sublinearly on the number of actions

- Its regret is **sublinear in** T

No-regret prediction

- We have that

$$R_T \leq \sqrt{2TN \log N}$$

Grows with $\sqrt{T}$
(slower than T)

Stochastic bandits

Stochastic bandits

- We have available a set of actions
- The cost for each action comes from a **distribution** (fixed through time)

Action 1

Action 2

Action 3

Action 4

Stochastic bandits

- What is the goal?
  - Select action with smallest average cost
- How can we compute the cost?
  - Get samples; average

Stochastic bandits

- After 20 steps:

?

Action 1

Avg. cost: 1

Action 2

Avg. cost: 0.3

Action 3

Avg. cost: 0

Action 4

Avg. cost: 0.5

Which action would you chose?

Why?

Situation 1:

- After 20 steps:

Action 1	Action 2	Action 3	Action 4
			
Avg. cost: 1	Avg. cost: 0.3	Avg. cost: 0	Avg. cost: 0.5
↓	↓	↓	↓
Played 9 times	Played 1 time	Played 9 times	Played 1 time

Situation 1:

- After 20 steps:

Action 1	Action 2	Action 3	Action 4
			
Avg. cost: 1	Avg. cost: 0.3	Avg. cost: 0	Avg. cost: 0.5

Situation 2:

- After 20 steps:

Action 1	Action 2	Action 3	Action 4
			
Avg. cost: 1	Avg. cost: 0.3	Avg. cost: 0	Avg. cost: 0.5
Played 1 time	Played 9 times	Played 1 time	Played 9 times

Situation 1:

- After 20 steps:

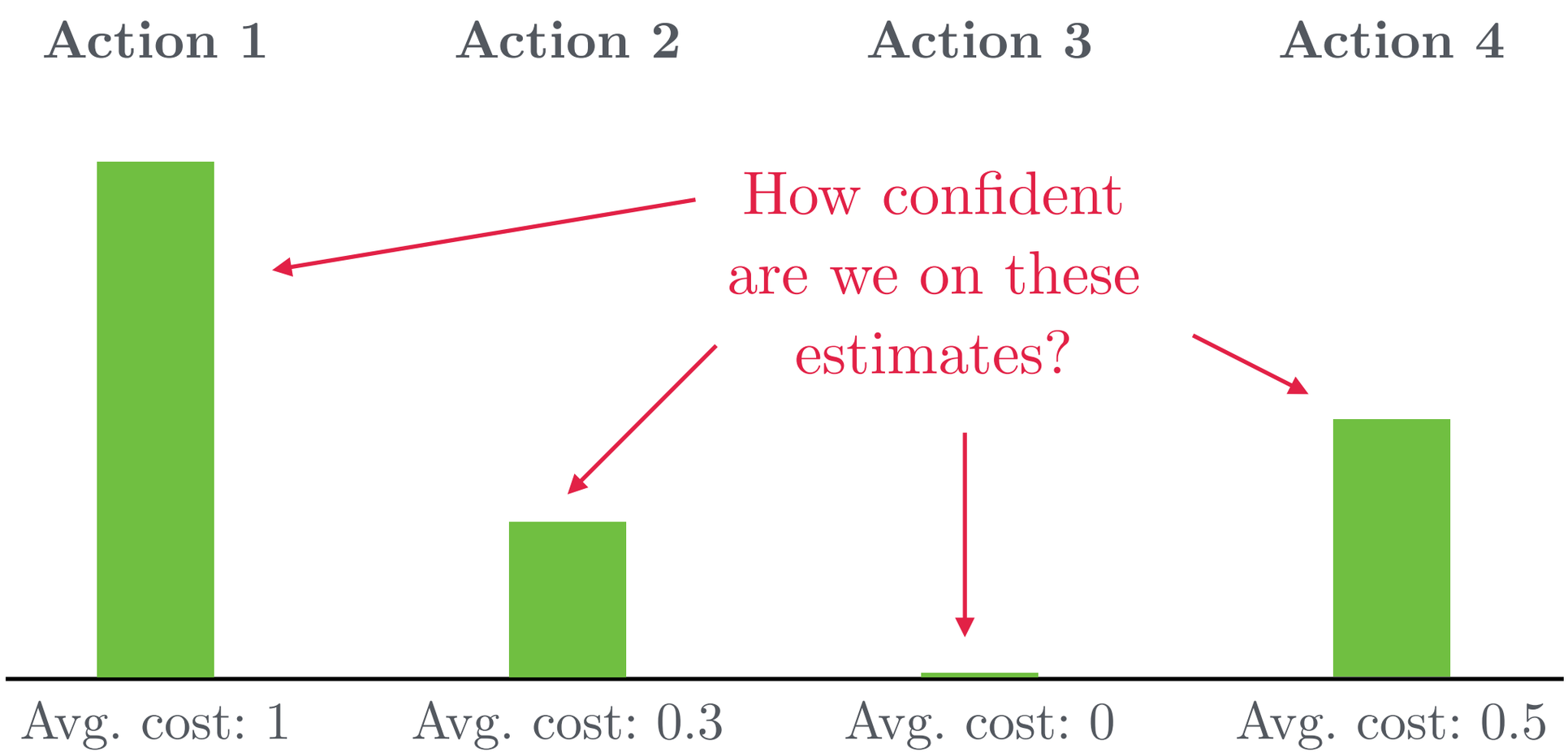
?

Action 1	Action 2	Action 3	Action 4
			
Avg. cost: 1	Avg. cost: 0.3	Avg. cost: 0	Avg. cost: 0.5

Just the mean is not enough!

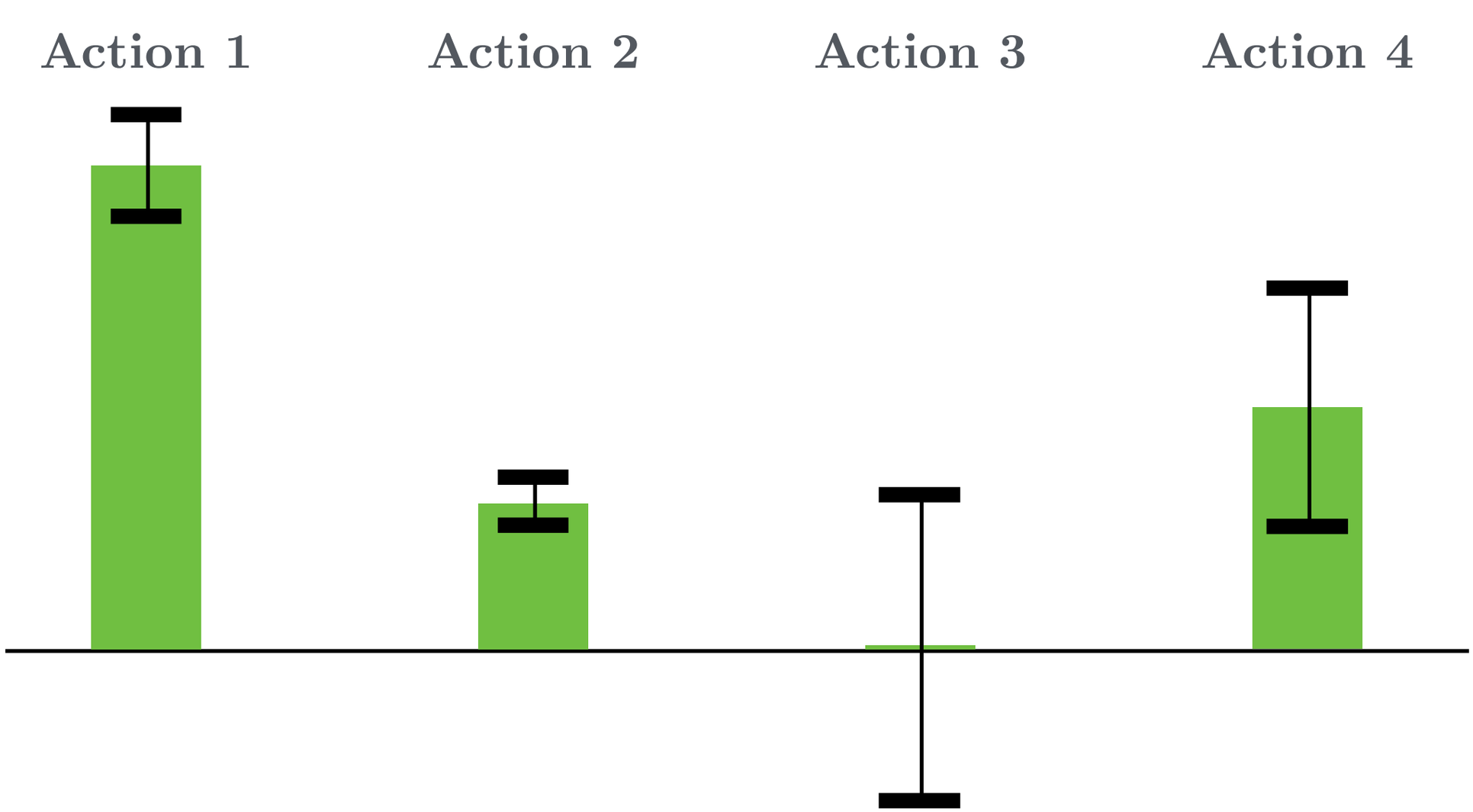
A different perspective

- After 20 steps:



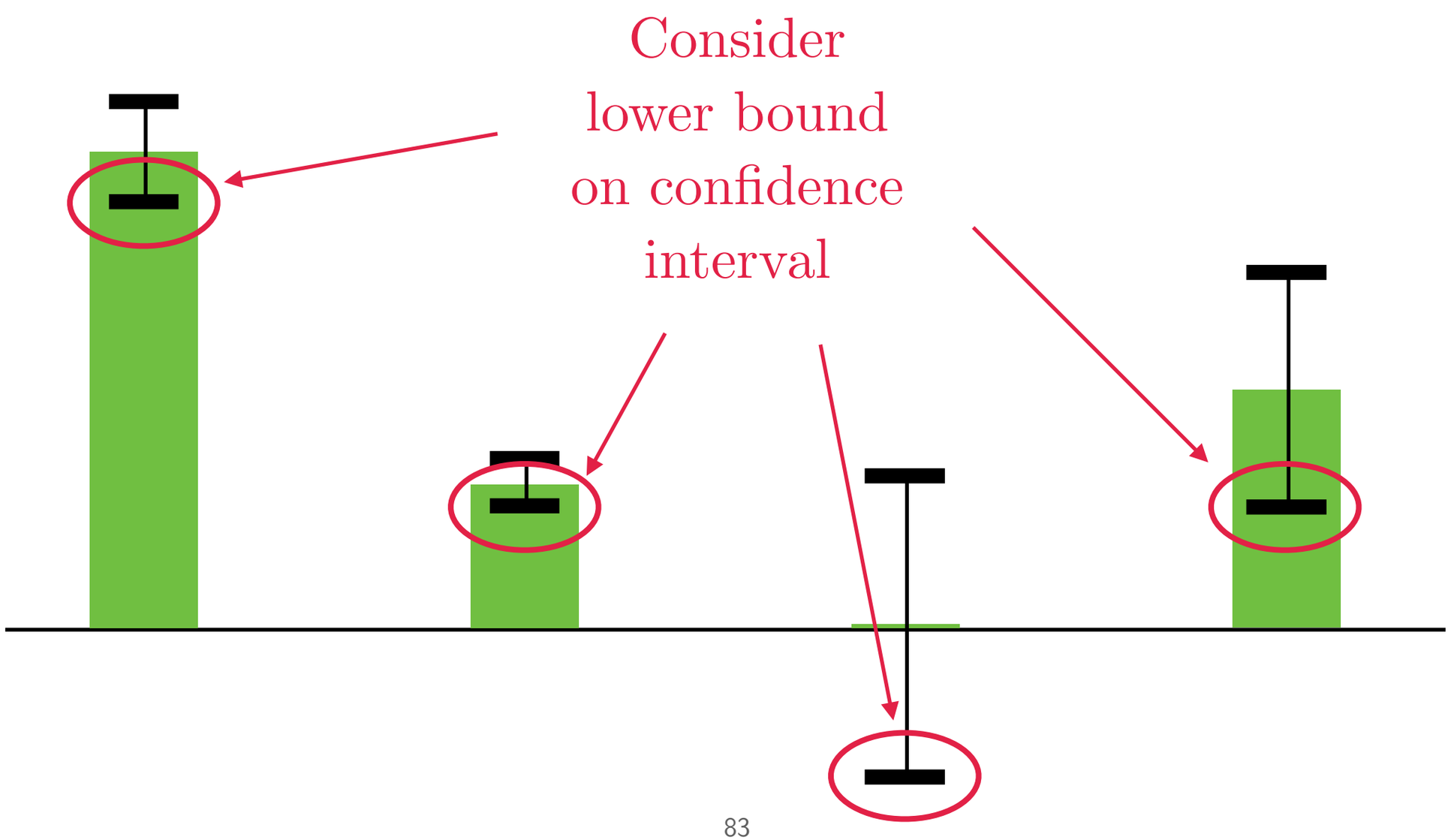
A different perspective

- After 20 steps (with confidence intervals):



A different perspective

- **Principle:** Optimism in the face of uncertainty



UCB algorithm

- Execute each action once
- From then on, at each step t, select action

$$a^* = \operatorname{argmin} \hat{c}(a) - \sqrt{\frac{2 \log t}{N_t(a)}}$$

Average cost

Confidence interval

Important notes

- In the literature, this algorithm was first proposed with rewards instead of costs
- With rewards...
- ... instead of minimizing the lower confidence bound, the algorithm maximizes the **upper confidence bound**
- ... hence the name of the algorithm

Does this work?

- To measure the performance of UCB, we compare its total expected cost with the one optimal:

$$R_T = \mathbb{E}\left[\sum_{t=0}^{T-1} \mathrm{c}(a_t) - \min_{a \in \mathcal{A}} \sum_{t=0}^{T-1} \mathrm{c}(a)\right]$$

Random variables

Does this work?

- To measure the performance of UCB, we compare its total expected cost with the one optimal:

$$R_T = \mathbb{E}\left[\sum_{t=0}^{T-1} \mathrm{c}(a_t) - \min_{a \in \mathcal{A}} \sum_{t=0}^{T-1} \mathrm{c}(a)\right]$$

- Equivalently:

$$R_T = \sum_{a \in \mathcal{A}} \mathbb{E}\left[N_T(a)\right] c(a) - T c(a^*)$$

Expected n. of times that a is selected

Expected cost of optimal action

Does this work?

- To measure the performance of UCB, we compare its total expected cost with the one optimal:

$$R_T = \mathbb{E}\left[\sum_{t=0}^{T-1} \mathrm{c}(a_t) - \min_{a \in \mathcal{A}} \sum_{t=0}^{T-1} \mathrm{c}(a)\right]$$

- Equivalently:

$$R_T = \sum_{a \in \mathcal{A}} \mathbb{E}\left[N_T(a)\right] c(a) - T c(a^*)$$

$$= \sum_{a \in \mathcal{A}} \mathbb{E}\left[N_T(a)\right] \underbrace{(c(a) - c(a^*))}_{\Delta(a)}$$

Does this work?

- Expanding $\mathbb{E}[N_T(a)]$, we get the bound

$$R_T \leq \sum_{a \neq a^*} \frac{8 \log T}{\Delta(a)} + 2 \sum_{a \in \mathcal{A}} \Delta(a)$$

UCB

- Only observes cost for selected actions (bandit problem)
- It assumes that costs follow an unknown (but fixed) distribution
- Its regret is **sublinear in T**

No-regret prediction

- With some tricky manipulation, we can turn the bound into

$$R_T \leq \sqrt{CNT \log T}$$

Constant

Grows with $\sqrt{T \log T}$ (slower than T)

Overview:

- No assumptions on the cost process; observation of complete costs: EWA

$$R_T \leq \sqrt{\frac{T}{2} \log N}$$

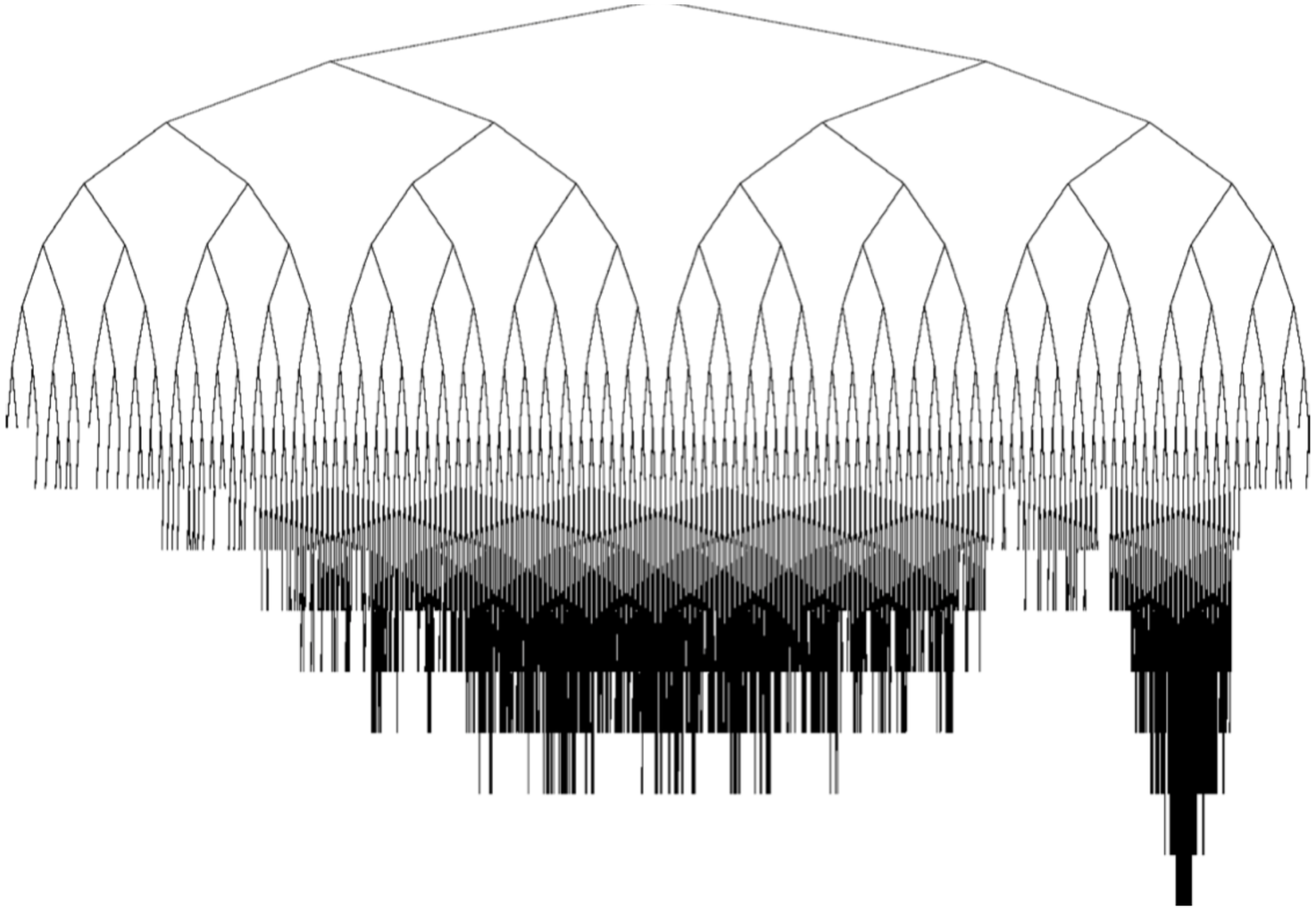
- No assumptions on the cost process; observation of single-action cost (bandit setting): EXP3

$$R_T \leq \sqrt{2TN \log N}$$

- Costs sampled from distribution; observation of single-action cost (bandit setting): UCB

$$R_T \leq \sqrt{CNT \log T}$$

Applications



Monte Carlo Tree Search

Monte Carlo Tree Search

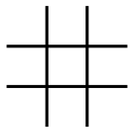
- Application of **UCB**
- Very efficient planning method for large domains
- **Idea:**
  - Build a search tree incrementally
  - Use sampling to compute node values

MCTS (cont.)

- Four stages:
  - Select a node to expand (**SELECTION**)
  - Expand it (**EXPANSION**)
  - Simulate the process starting from that node (**SIMULATION**)
  - Back-propagate the resulting value (**BACK-PROPAGATION**)

Example (Tic-Tac-Toe)

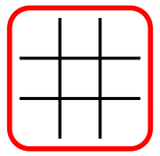
1. Select node to expand



$Q = 0$
$N = 0$

Example (Tic-Tac-Toe)

1. Select node to expand



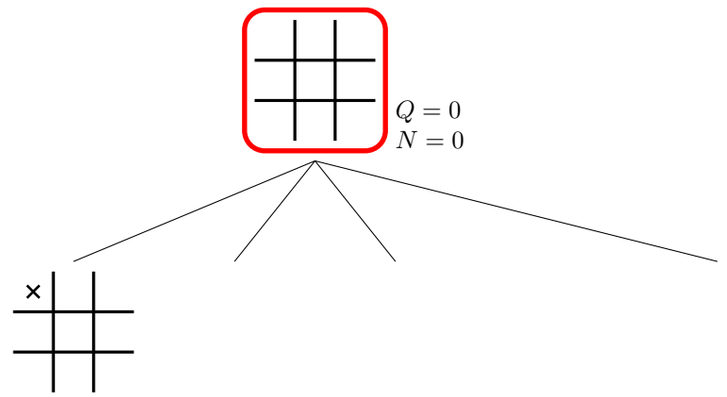
2. Expand it

Example (Tic-Tac-Toe)

1. Select node to expand

2. Expand it

3. Simulate games from the leaves



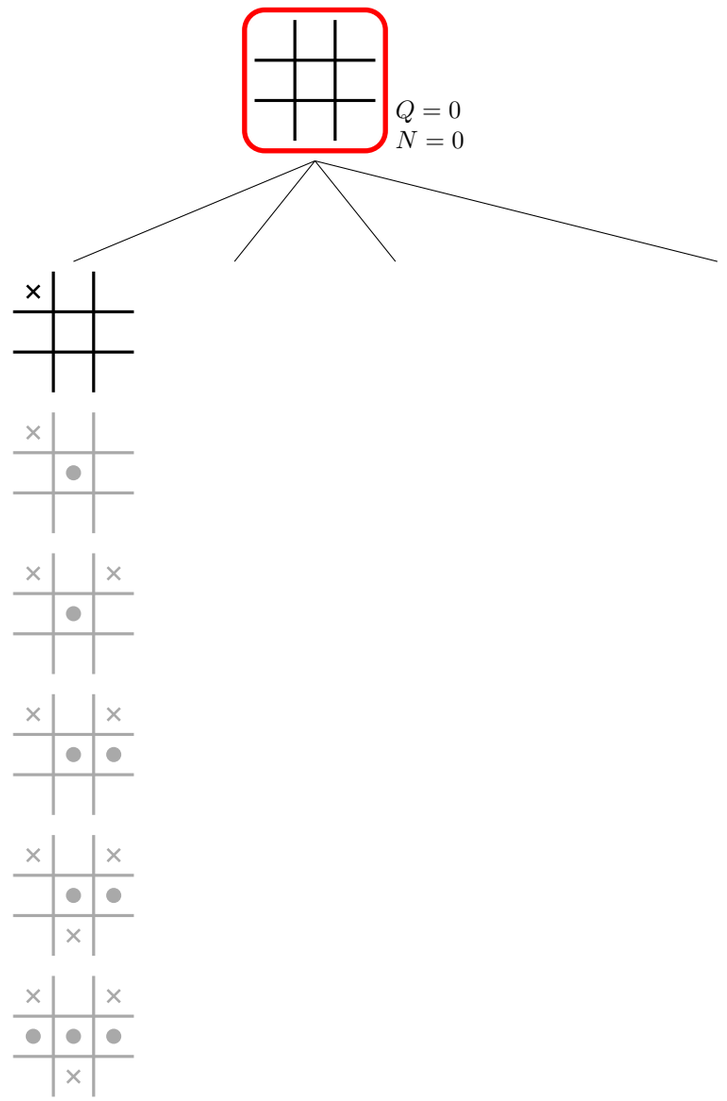
Example (Tic-Tac-Toe)

1. Select node to expand

2. Expand it

3. Simulate games from the leaves

4. Back-propagate values



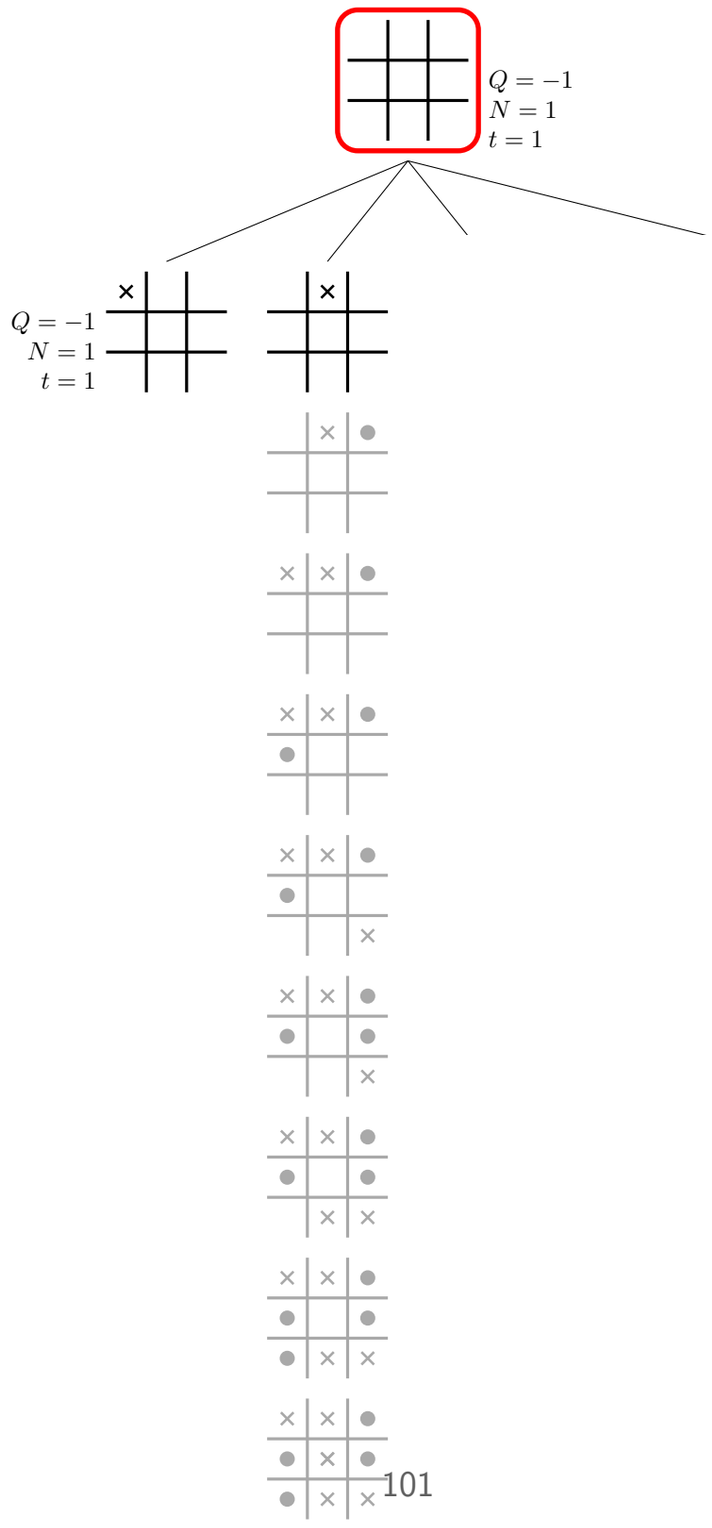
Example (Tic-Tac-Toe)

1. Select node to expand

2. Expand it

3. Simulate games from the leaves

4. Back-propagate values

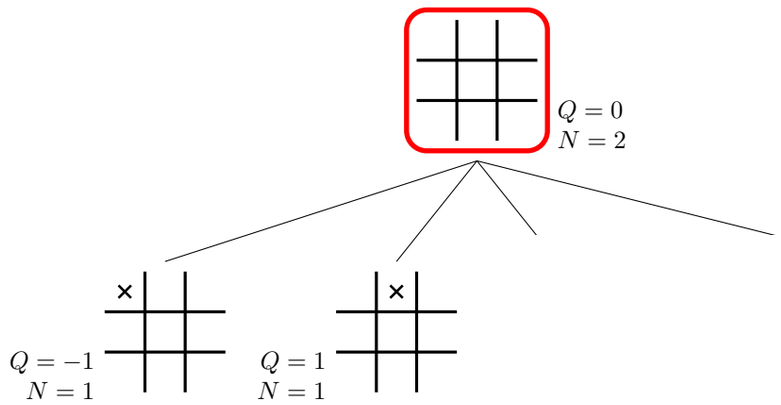


Example (Tic-Tac-Toe)

1. Select node to expand

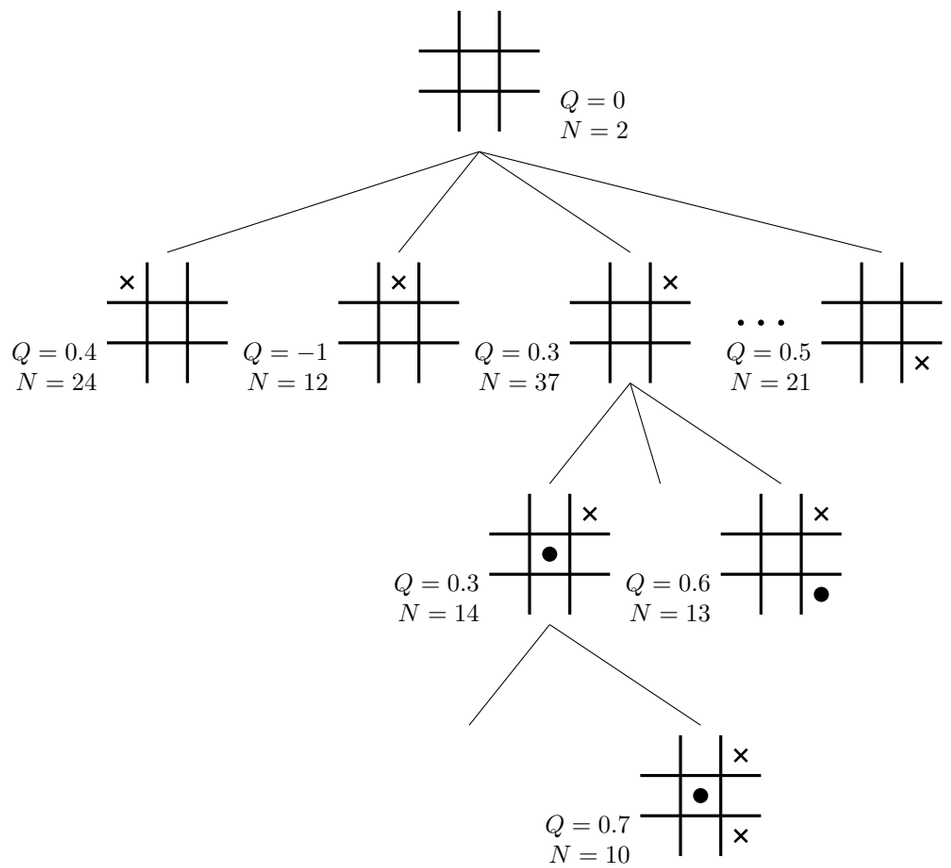
2. Expand it

3. Simulate games from the leaves



4. Back-propagate values

After several iterations



Selection

- How is the node to be expanded selected?
  - Several approaches exist
  - Most standard nowadays is **UCB**
  - The resulting MCTS algorithm is called **UCT** (UCB for Trees)
  - At each node, selects the node i minimizing

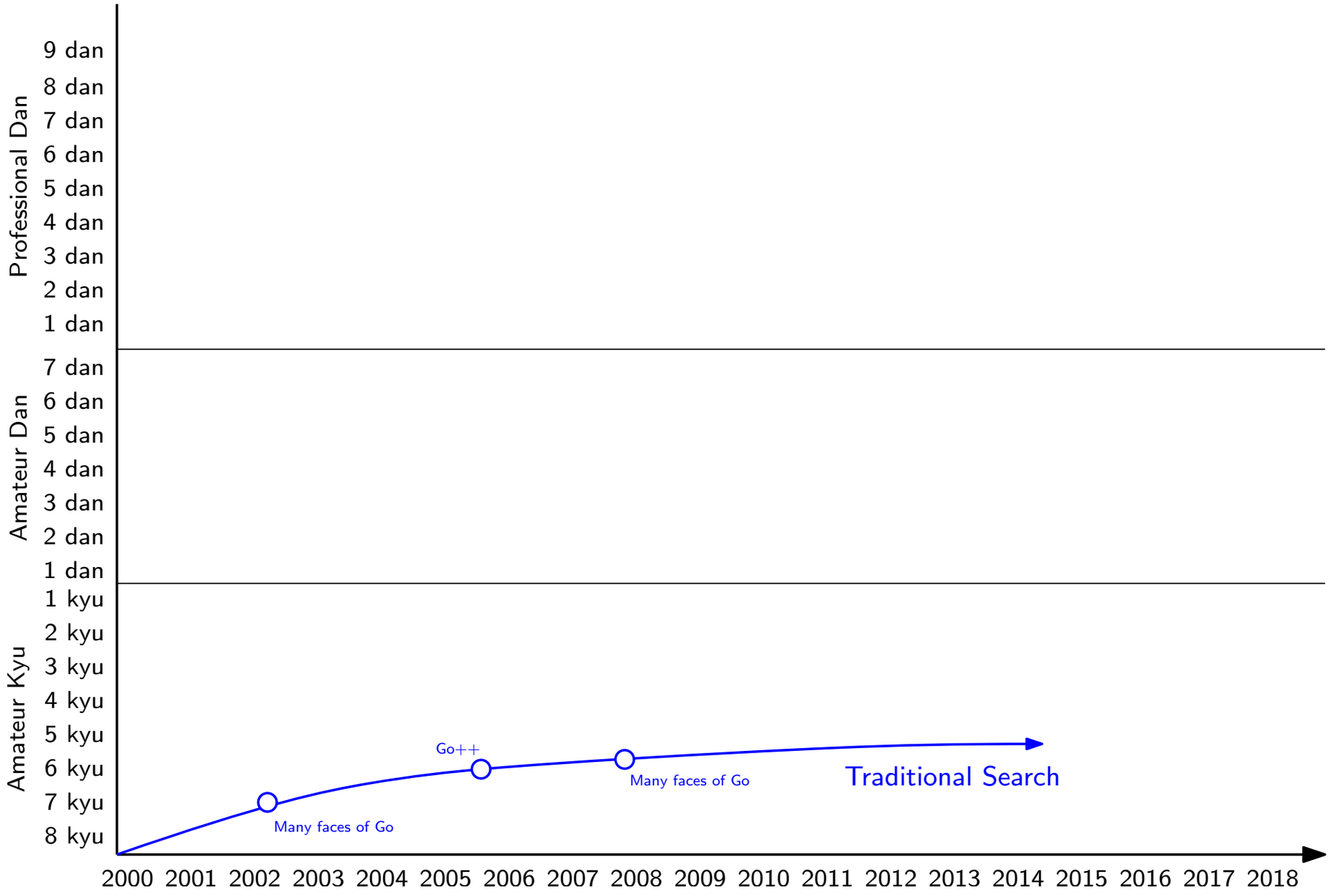
$$Q(i) - c\sqrt{\frac{\log(t)}{N(i)}}$$

UCB expression!

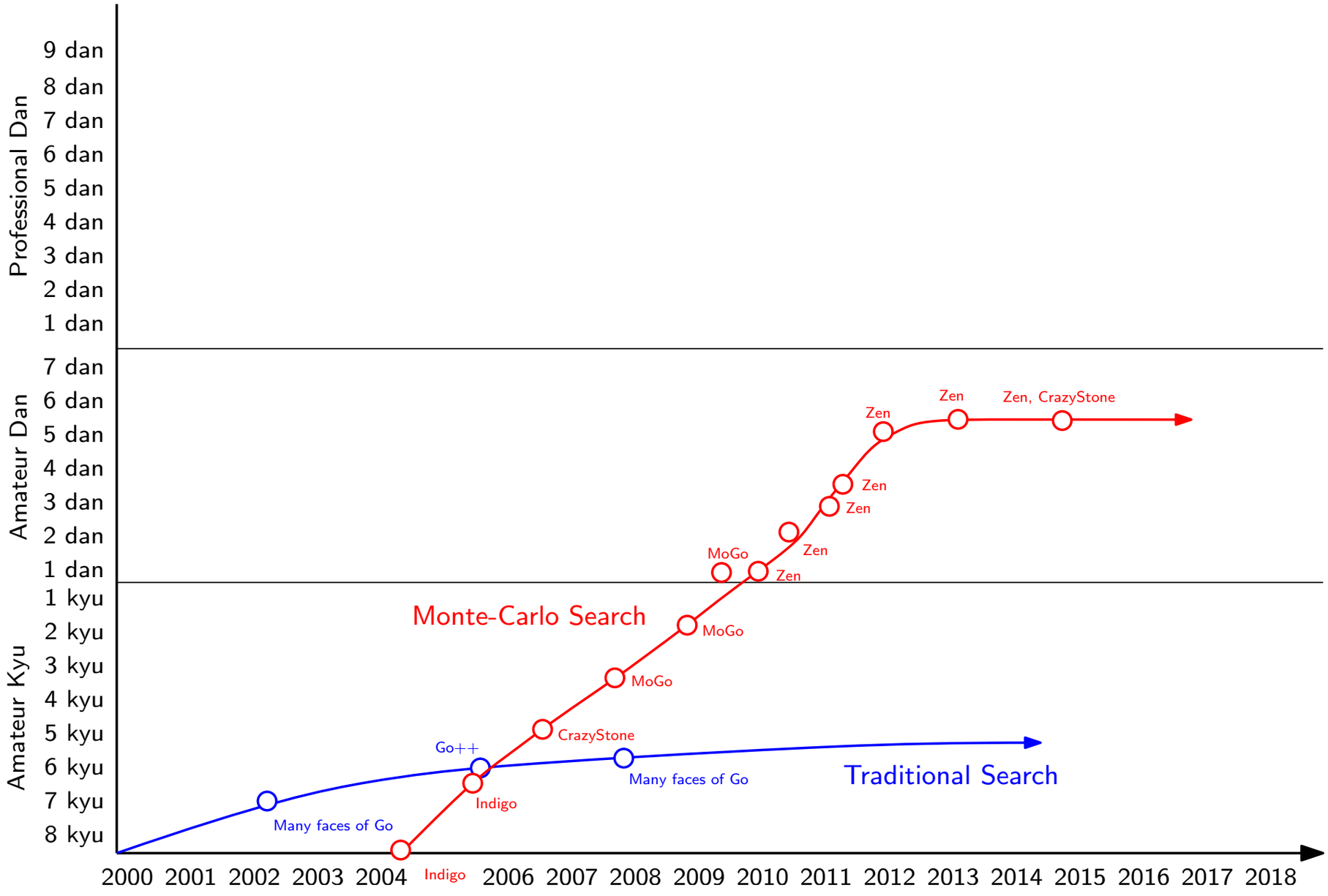
MCTS

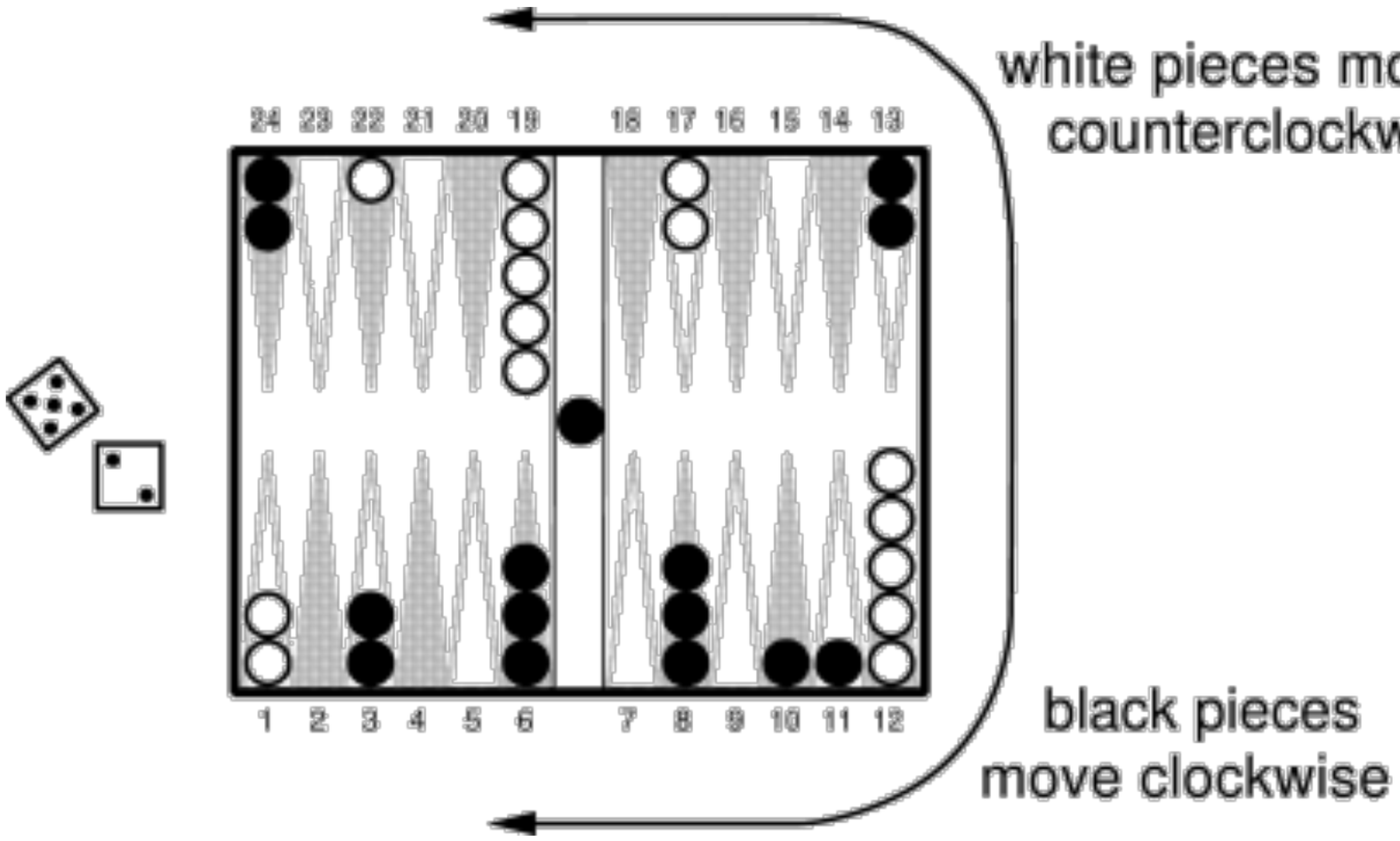
- The use of the "UCB heuristic" allows the tree expansion to follow the most promising route
- It is able to effectively "navigate" the game tree to and focus on areas that are more likely to reach victory

Evolution of Computer Go



Evolution of Computer Go



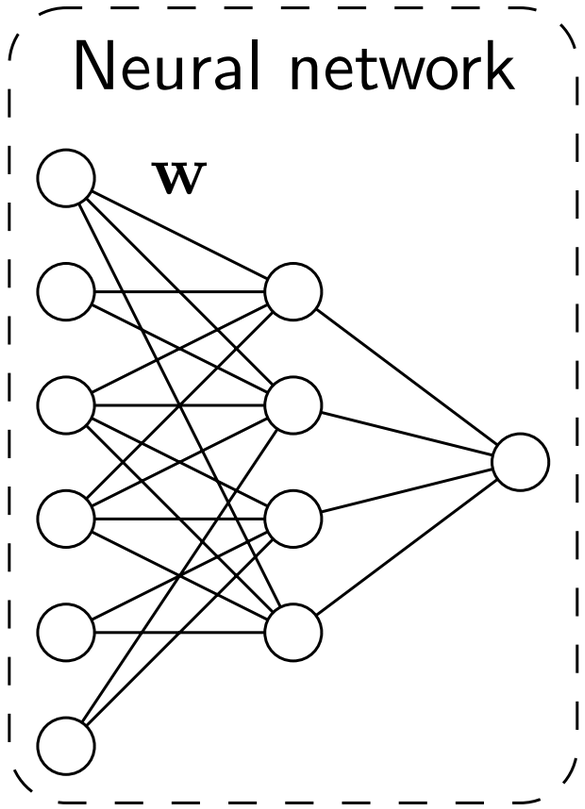

white pieces move
counterclockwise
24 23 22 21 20 19 18 17 16 15 14 13
1 2 3 4 5 6 7 8 9 10 11 12
black pieces
move clockwise

TD Gammon

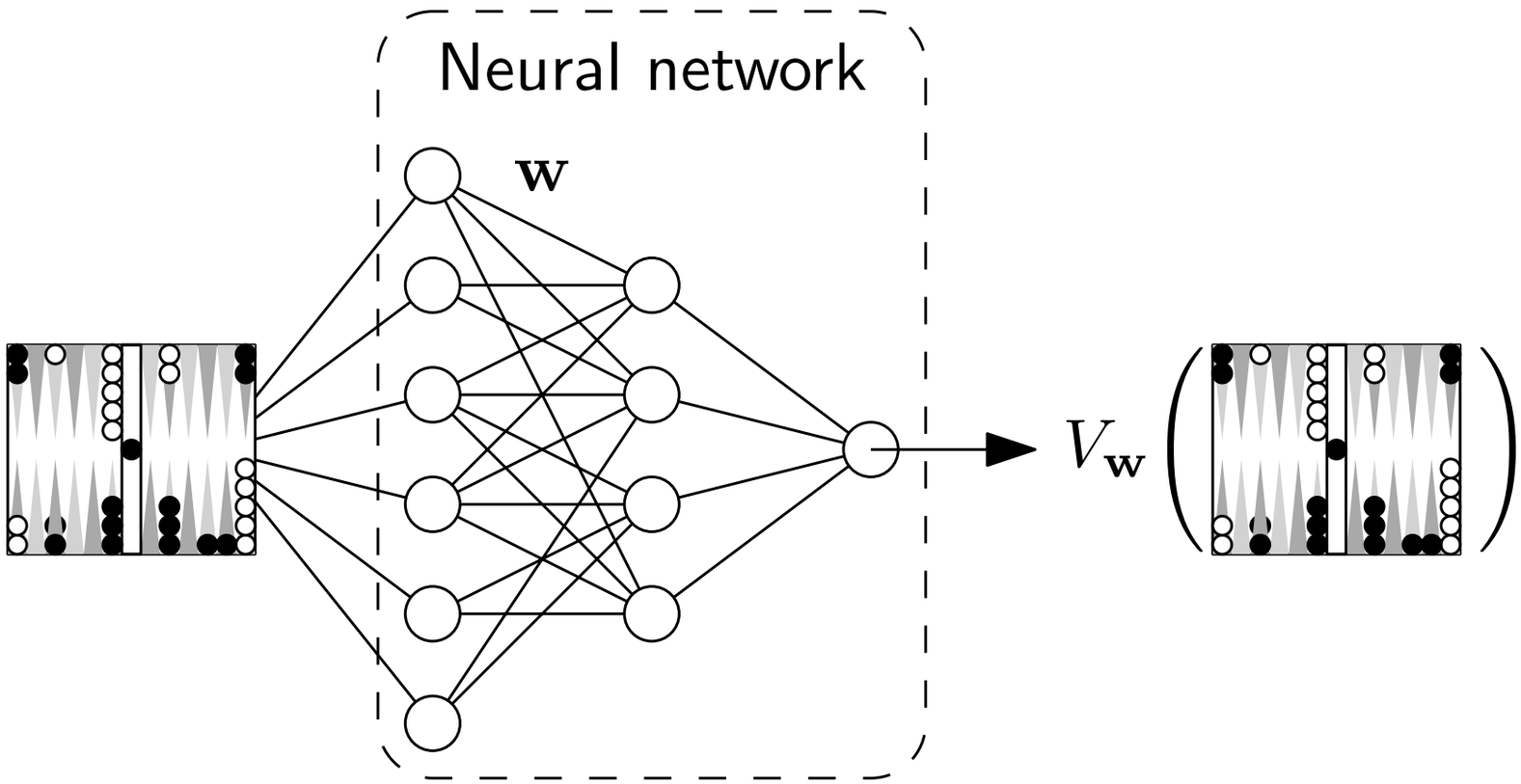
TD-Gammon

- Combines **neural networks** with **TD-learning**
- First computer program to achieve top human-level play in backgammon
- Neural network with a total of 80 hidden units
- Learned by playing games against itself

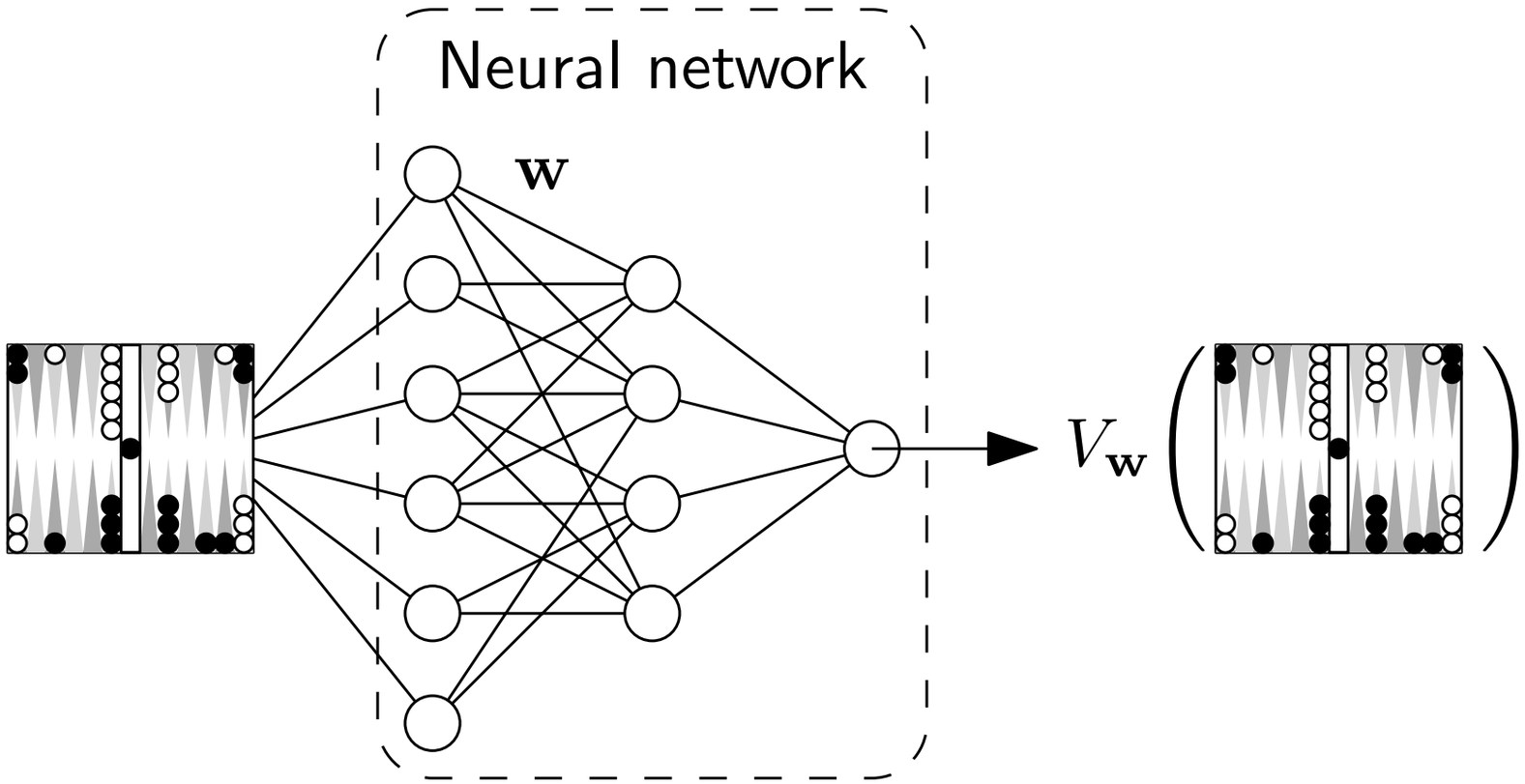
Illustration



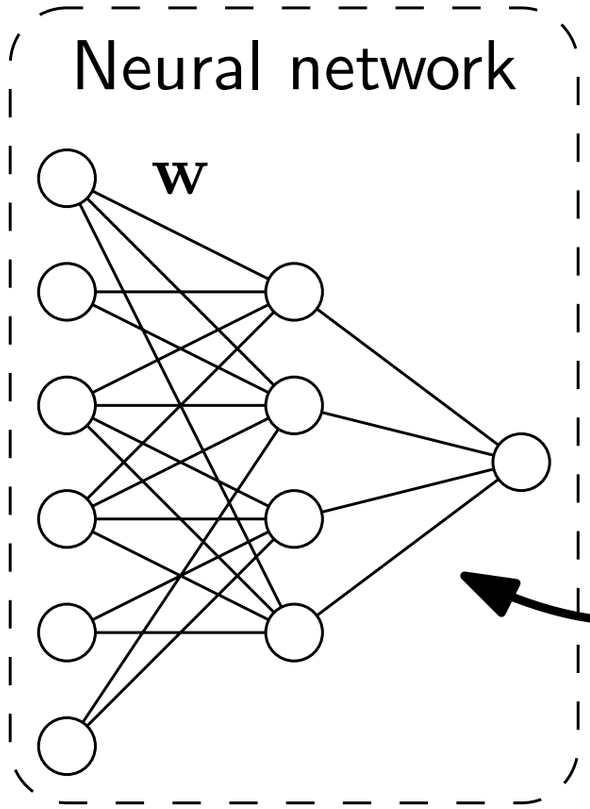
Illustration



Illustration



Illustration



$$\text{Error} = V_{\mathbf{w}}\left(\right) - V_{\mathbf{w}}\left(\right)$$

Training

- Neural network update:

$$\theta_{t+1} = \theta_t + \alpha_t z_{t+1}(x_t)(v_\theta(x_{t+1}) - v_\theta(x_t)) \quad \text{TD update}$$

where z_t is the eligibility trace, given as

$$z_{t+1}(x_t) = \lambda z_t(x_t) + \nabla_\theta v_\theta(x_t)$$

Other facts

- TD-Gammon was developed at IBM by Gerald Tesauro, in 1992
- The program played several world top players, attaining similar performance
- The program introduced several important changes to the way the game is played—for example, TD-Gammon used an opening previously thought to be poor but which was eventually adopted by top players

AlphaGo

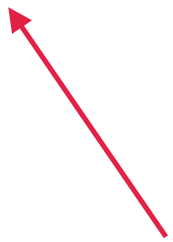
Some facts...

- **AlphaGo** is a Go playing program developed by UK-based company **Deep Mind**
- Deep Mind was also behind the system that learned to play **Atari games** at human level

Google DeepMind

How AlphaGo works...

- Basic idea: **MCTS**
  - Simulate **good opponents** (for simulation)
  - Compute **robust optimal** board values (to perform selection, etc.)



Don't know optimal policy!

How AlphaGo works...

- Basic idea: **MCTS**
  - Simulate good opponents
  - Compute robust optimal board values
  - Compute a great policy
  - Evaluate great policy

How AlphaGo works...

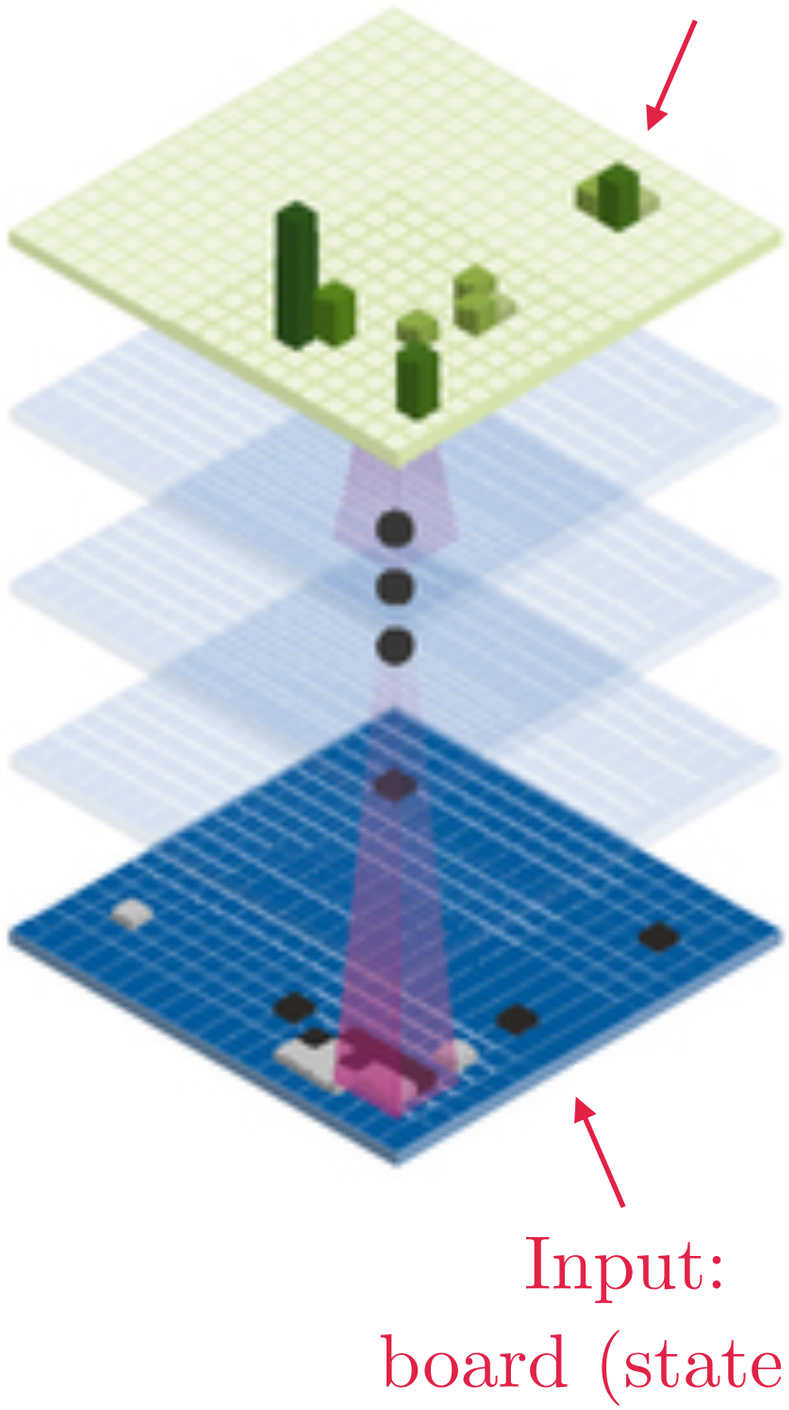
- Basic idea: **MCTS**
  - Simulate good opponents → Policy network
  - Compute robust optimal board values
  - Compute a great policy → Policy network 2
  - Evaluate great policy → Value network

Modules of AlphaGo

- **Policy network** trained to predict human plays
- **Policy network** trained to select "winning" plays
- **Value network** to evaluate board positions

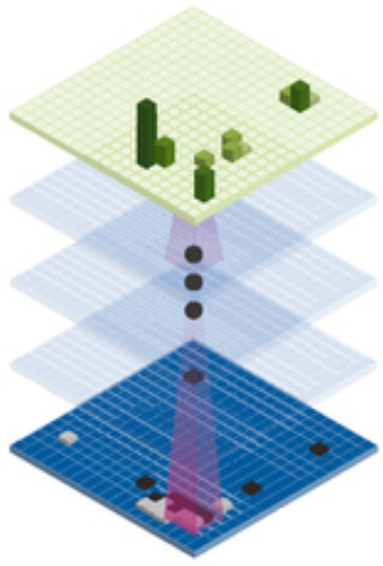
SL policy network

- Optimized to predict human moves
- Trained from 30×10^6 boards from experts
- Attained a prediction accuracy of 57% (almost 15% above existing predictors)



SL policy network

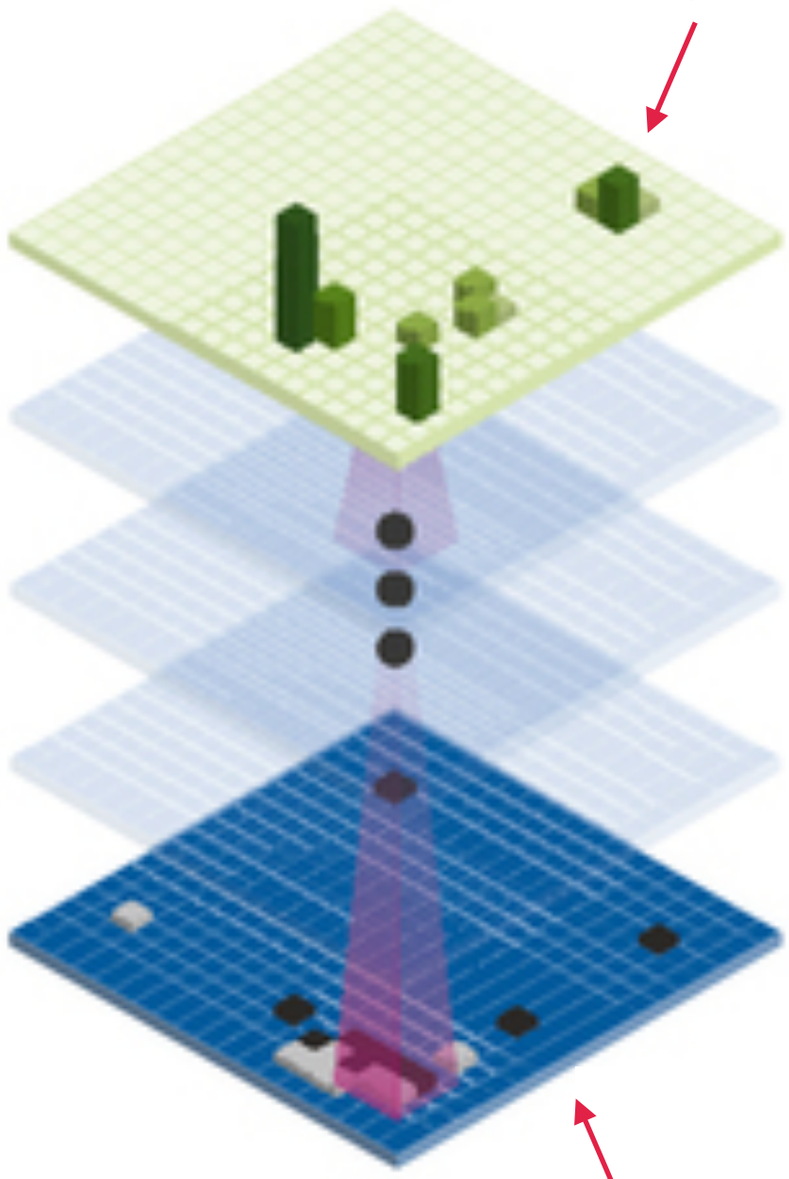
- Optimized to predict human moves
- Trained from 30×10^6 boards from experts
- Attained a prediction accuracy of 57% (almost 15% above existing predictors)



Also trained a second (smaller) network for faster simulation

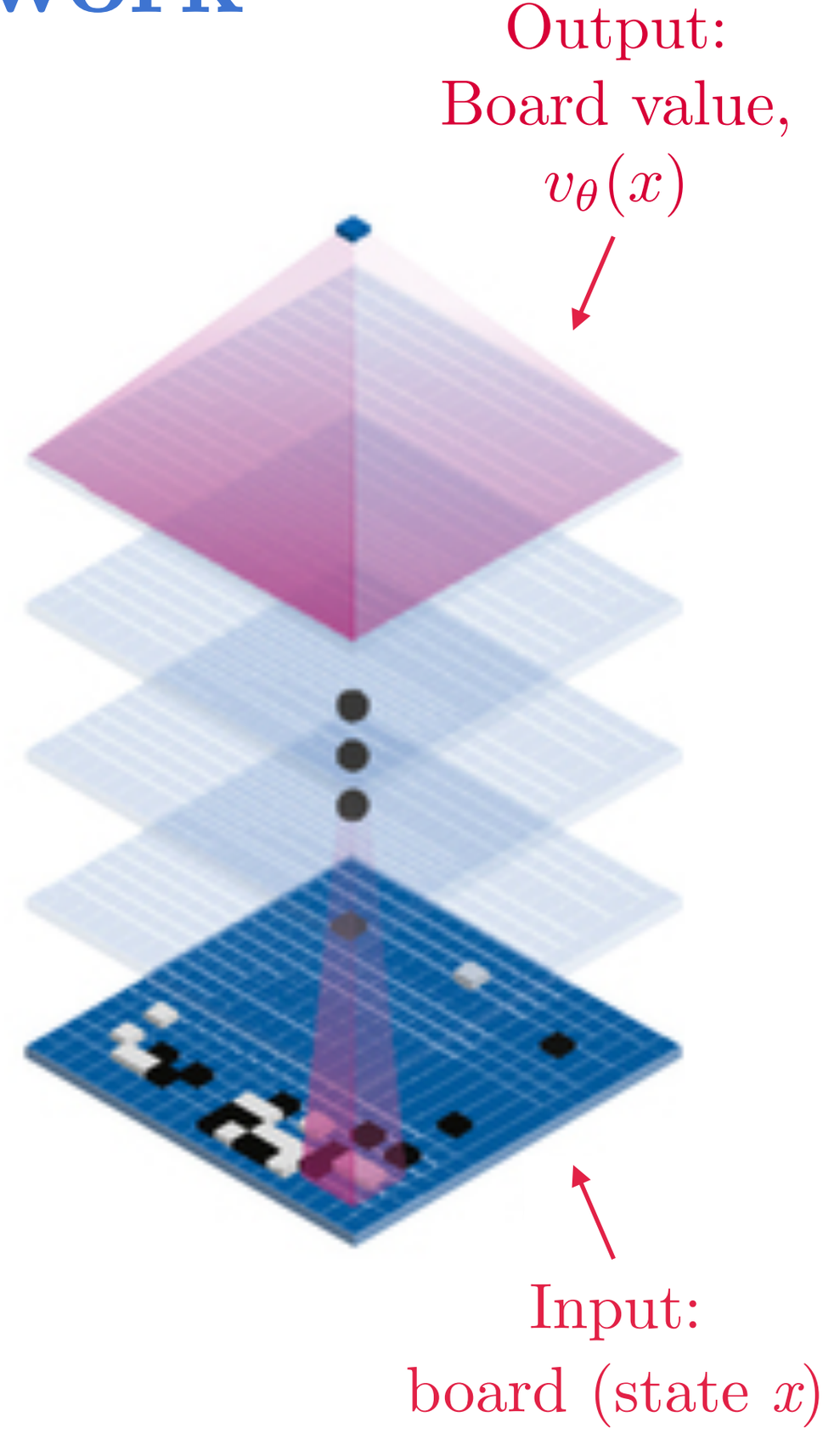
RL policy network

- Optimized to "win"
- Trained in self play (against previous versions of itself)
- Network weights updated using **policy gradient**



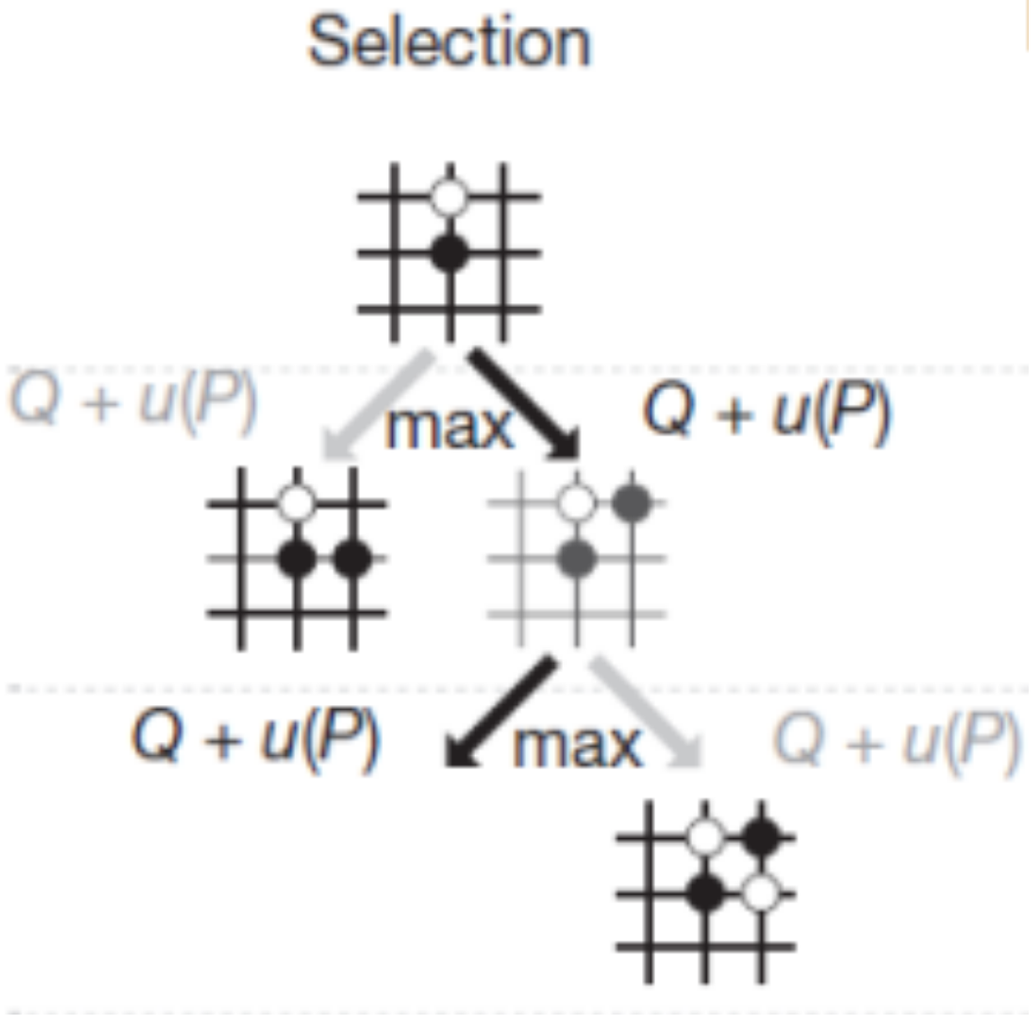
Value network

- Used to evaluate board positions
- Trained from (sampled) board positions obtained during self-play games using the policy obtained from the RL network



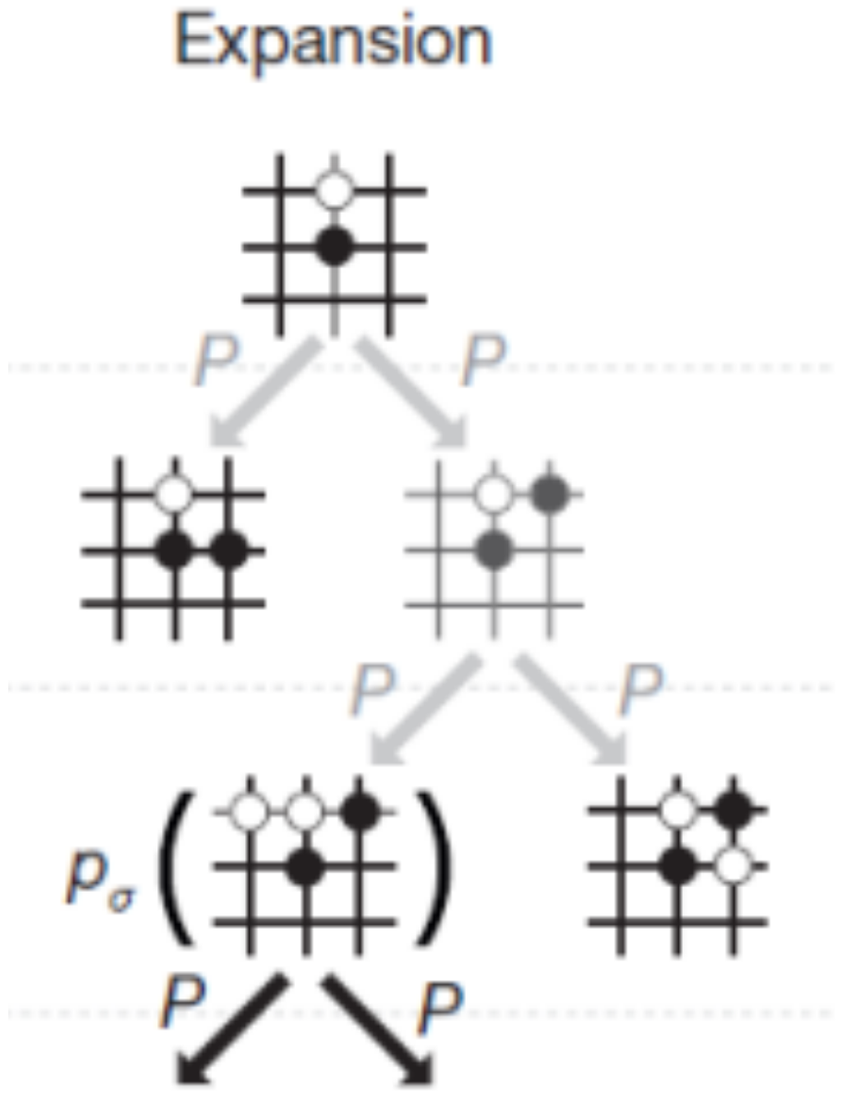
MCTS in AlphaGo

- **Selection:**
  - Tree is traversed from root to leaf by selecting UCB-like heuristic
  - The "bonus" $u(P)$ depends on the prior probability P, provided by the SL policy network



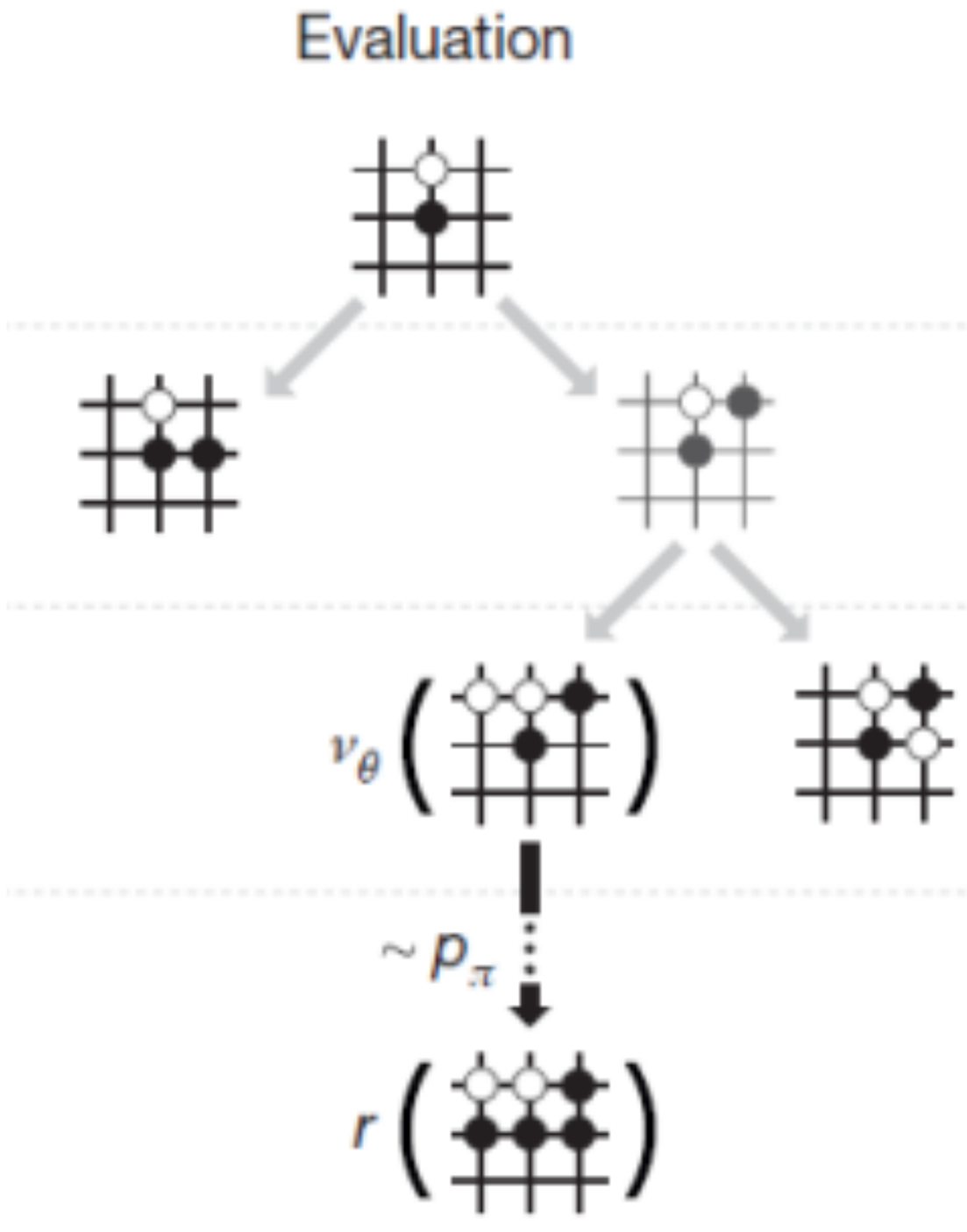
MCTS in AlphaGo

- **Expansion:**
  - New leaf nodes are created and assigned prior probabilities P from SL network



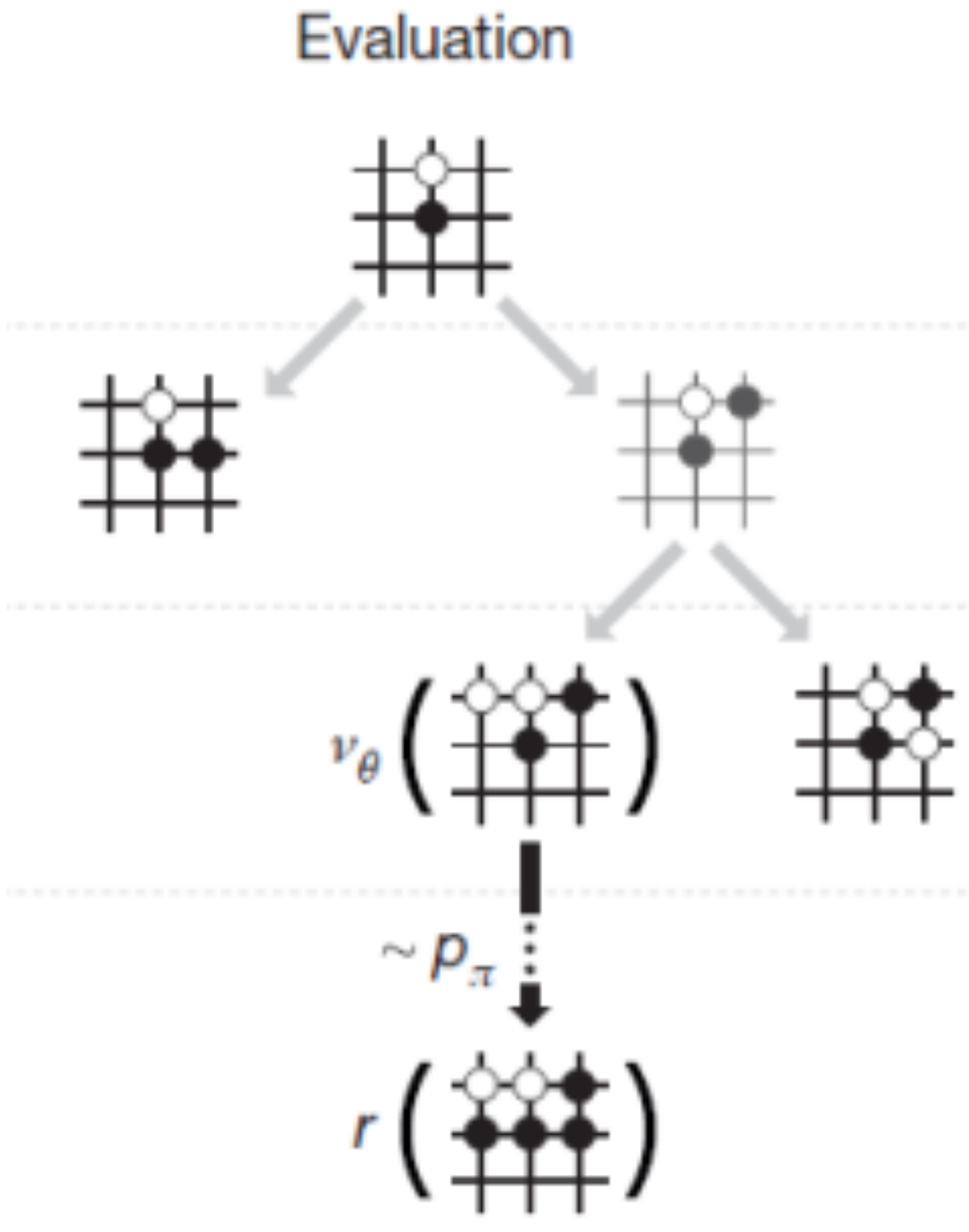
MCTS in AlphaGo

- **Evaluation:**
  - Games are "simulated" until the end using the small network



MCTS in AlphaGo

- **Evaluation:**
  - Value obtained from simulation is combined with value from value network
  - Q values are computed by averaging the resulting evaluations



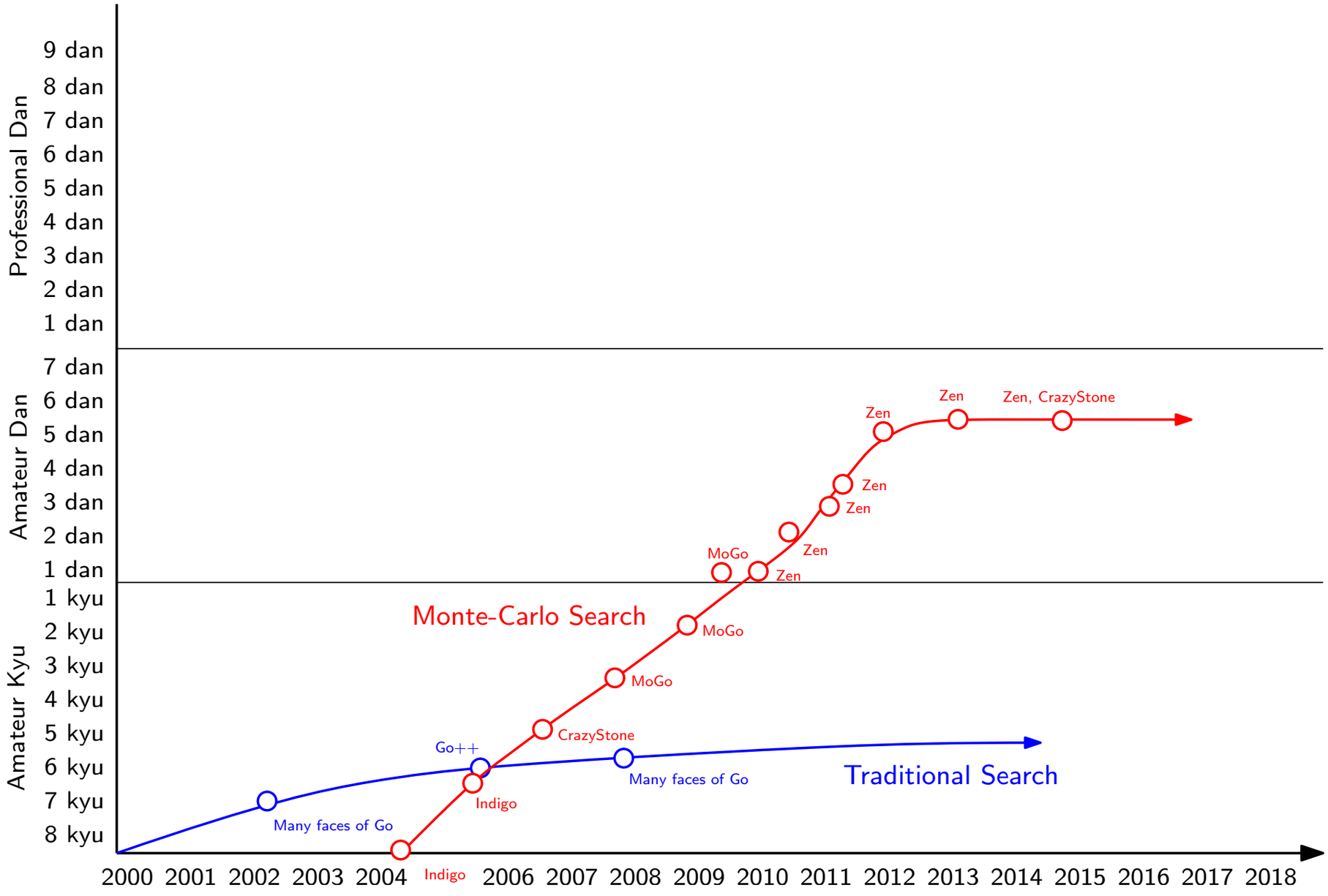
AlphaGo

- Other subsequent architectures extend AlphaGo in different ways
  - AlphaGo Zero learned how to play go from scratch (no human knowledge)
  - AlphaZero learned Go, Shogi, and Chess from scratch
  - MuZero is a model-based extension of AlphaZero that learned how to play Atari, Go, Shogi, and Chess from scratch

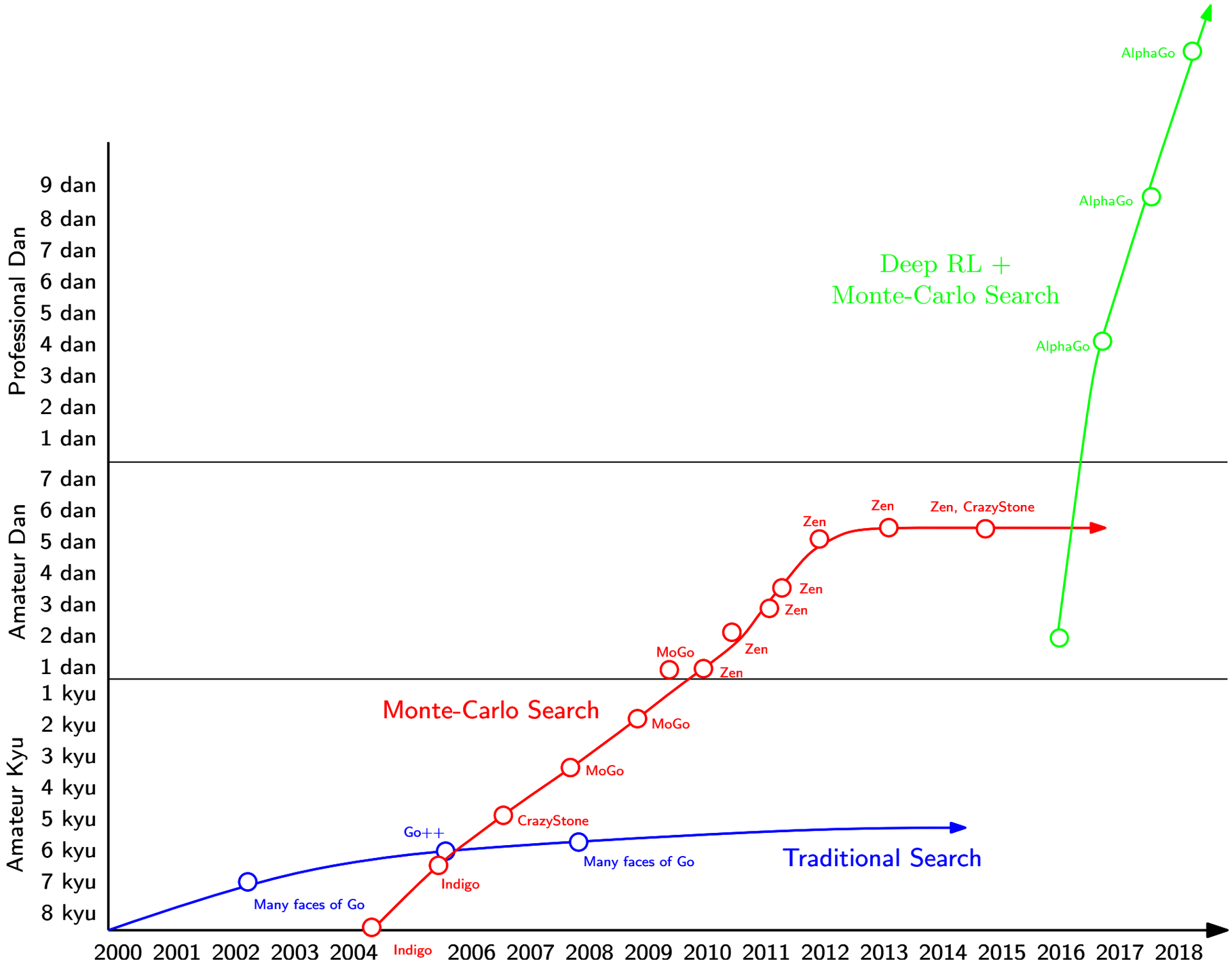
Meet AlphaGo

Distributed AlphaGo: 1202 CPUs and 176 GPUs

Evolution of Computer Go



Evolution of Computer Go

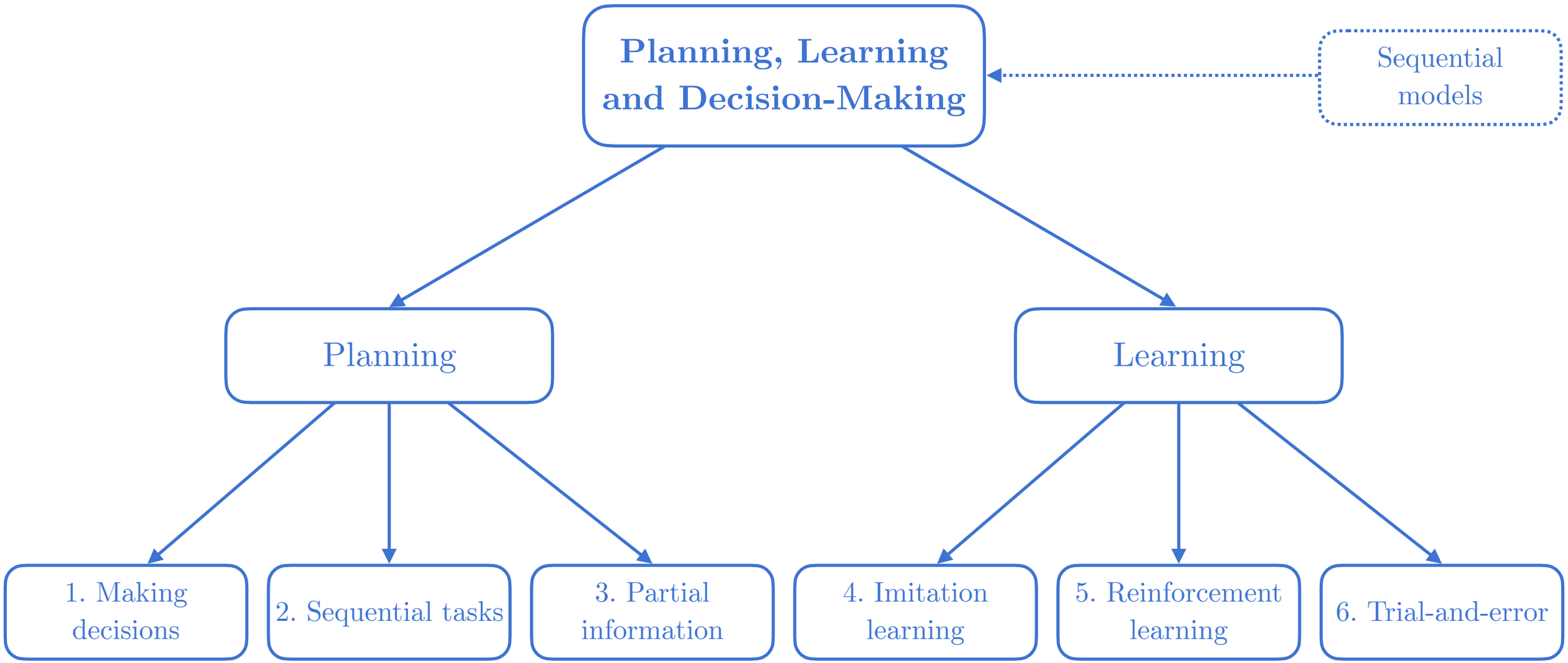


To conclude...

What we set out to do...

- Understand the main issues involved in **decision making**, in face of **uncertainty**
- Familiarize with some of the main tools for **planning** and **learning** in such settings

Learning and Decision Making



“There is no end to education. It is not that you read a book, pass an examination, and finish with education. The whole of life, from the moment you are born to the moment you die, is a process of learning.”

– Jiddu Krishnamurti