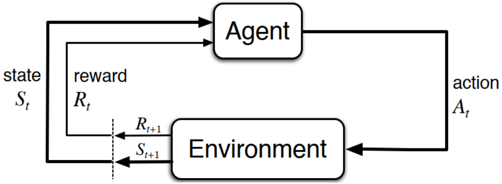
Reinforcement Learning 1

Basic concepts, Bandit algorithm

Pekka Marttinen

Aalto University

Markov decision process (MDP)



- At step t the agent is in state S_t.
  - Executes action A_t
  - Receives reward R_{t+1}
  - Moves to state S_{t+1}

- Multiple steps form a **trajectory**:

$$S_0, A_0, R_1, S_1, A_1, R_2, S_2, \ldots.$$

Dynamics of the MDP

- **Dynamics** (or **Model**) specifies how the reward and next state depend on the action and current state:

  $$p(s', r|s, a) = \Pr\{S_t = s', R_t = r|S_{t-1} = s, A_{t-1} = a\},$$

  for all $s', s \in \mathcal{S}$, $r \in \mathcal{R}$, and $a \in \mathcal{A}(s)$.

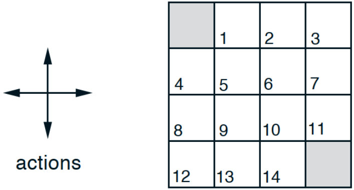
- **State-transition** probabilities:

  $$p(s'|s, a) = \Pr\{S_t = s'|S_{t-1} = s, A_{t-1} = a\} = \sum_{r \in \mathcal{R}} p(s', r|s, a).$$

- **Reward function**:

  $$r(s, a) = E[R_t|S_{t-1} = s, A_{t-1} = a] = \sum_{r \in \mathcal{R}} r \sum_{s' \in \mathcal{S}} p(s', r|s, a)$$

Example, simple gridworld



$R_t = -1$
on all transitions

- Simulate trajectories
  - Start at state 5, denote the terminal state as 15.
  - Select actions uniformly at random.
  - Example state sequences:
    - (5,1,1,5,6,2,2,1,15), Total reward = -8
    - (5,4,5,4,4,8,4,5,9,8,9,10,14,14,13,14,15), Total reward = -16

Episodic and continuing tasks

- In the simple gridworld, the task was completed when the *terminal state* 15 was entered.
- Tasks with a terminal state are called *episodic*
  - Let T denote the step of entering the terminal state.
  - Sequence of steps $1, \ldots, T$ is called an *episode*
  - T often varies between episodes.
  - Examples: winning or losing in chess, exiting a maze.
- In *continuing tasks* there is no clear terminal state ($T = \infty$)
  - Examples:
    - on-going process control
    - a robot with a long life span

Returns, discounting

- **Return** G_t is a function of the rewards after step t, and for episodic tasks it can be define as:

$$G_t = R_{t+1} + R_{t+2} + \ldots + R_T.$$

- For continuing tasks *discounting* is used:

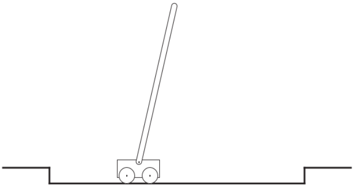
$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1},$$

where γ is the discount rate.

- With discounting and constant reward, the return stays finite. For example if reward is always $+1$, then

$$G_t = \sum_{k=0}^{\infty} \gamma^k = \frac{1}{1-\gamma}.$$

Example: Pole balancing



- Formulate as an episodic task:
  - Reward $+1$ for every time step on which failure did not occur
- Formulate as a continuing task, using discounting.
  - Reward -1 on each failure and zero at all other times.
  - The return at each step would then be related to $-\gamma^{K-1}$, where K is the number of steps before failure.
- Either way, the return is maximized by keeping the pole balanced for as long as possible.

Value function and policy

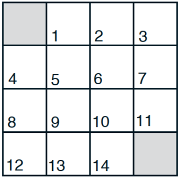
- **State-value function for policy** π, $v_\pi(s)$, specifies how much return the agent can expect when starting in a state s, following a policy π.

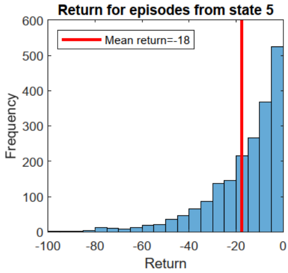
$$v_\pi(s) = E_\pi[G_t | S_t = s] = E_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} | S_t = s\right], \quad \text{for all } s.$$

- Policy $\pi(a|s)$ is a mapping from states s to probabilities of different actions a that the agent uses to select the next action.

Example: simple gridworld

- Multiple simulated episodes from state 5:
  - (5,1,1,5,6,2,2,1,15), Return = -8
  - (5,4,5,4,4,8,4,5,9,8,9,10,14,14,13,14,15), Return = -16
  - $\cdots$
- Average return ≈ -18. Therefore $v_\pi(s_5) = -18$ for a random policy π.





0.0	-14.	-20.	-22.
-14.	-18.	-20.	-20.
-20.	-20.	-18.	-14.
-22.	-20.	-14.	0.0

Value function, random policy.

Bellman Equation

- A recursive relationship between the value of any state s and its possible successor states s':

$$v_\pi(s) = \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)\left[r + \gamma v_\pi(s')\right].$$

$$v_\pi(s_5) = -18 = \frac{1}{4} * (-1 - 14) + \frac{1}{4} * (-1 - 14) + \frac{1}{4} * (-1 - 20) + \frac{1}{4} * (-1 - 20).$$

0.0	-14.	-20.	-22.
-14.	-18.	-20.	-20.
-20.	-20.	-18.	-14.
-22.	-20.	-14.	0.0

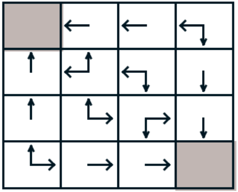
Value function, random policy.

Optimal policy and value function

- A policy π is better than and π' iff $v_\pi(s) \geq v_{\pi'}(s)$ for all $s \in \mathcal{S}$. An optimal policy is better than or equal to all other policies.
- There always exists at least one optimal policy, denoted by π_*.
- Optimal state-value function $v_*(s) \equiv \max_\pi v_\pi(s)$.

0.0	-14.	-20.	-22.
-14.	-18.	-20.	-20.
-20.	-20.	-18.	-14.
-22.	-20.	-14.	0.0

Value function, random policy.



Optimal policy

0	-1	-2	-3
-1	-2	-3	-2
-2	-3	-2	-1
-3	-2	-1	0

Optimal value function

Bellman optimality equation

- For optimal policy π_*

$$v_*(s) = \max_a \sum_{s',r} p(s', r | s, a) \left[r + \gamma v_\pi(s') \right] .$$

- Maximization instead of expectation in Bellman equation.
- $v_*(s_5) = \max\{-1 + v_*(s_1), -1 + v_*(s_4), -1 + v_*(s_6), -1 + v_*(s_9)\} = -2.$

	1	2	3
4	5	6	7
8	9	10	11
12	13	14	

0	-1	-2	-3
-1	-2	-3	-2
-2	-3	-2	-1
-3	-2	-1	0

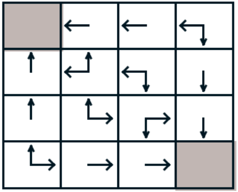
Optimal value function

From optimal value function to optimal action

- Once the optimal value function v_* is estimated or approximated somehow, selecting the optimal action is easy:
  - Choose action that leads to the maximum in the Bellman optimality equation, e.g., UP or LEFT for state s_5.
- The expected return (long-term consequence) is maximized by deciding the short-term action using v_*.
  - From s_5 all 4 actions give reward -1 (short-term consequence)
  - Only actions UP or LEFT give return -2 (optimal long-term consequence)

0	-1	-2	-3
-1	-2	-3	-2
-2	-3	-2	-1
-3	-2	-1	0

Optimal value function



Optimal policy

Prediction vs. Control

- **Prediction**: evaluate the future, given a policy.
  - What is the value function for a random policy?

0.0	-14.	-20.	-22.
-14.	-18.	-20.	-20.
-20.	-20.	-18.	-14.
-22.	-20.	-14.	0.0

Value function a random policy.

- **Control**: optimise the future.
  - What is the optimal policy and value function?

0	-1	-2	-3
-1	-2	-3	-2
-2	-3	-2	-1
-3	-2	-1	0

Optimal value function

Learning vs. planning

- Reinforcement learning:
  - The environment is initially unknown
  - The agent interacts with the environment
  - The agent improves its policy
- Planning:
  - A model of the environment is known
  - The agent performs computations with its model (without any external interaction)
  - The agent improves its policy
  - a.k.a. introspection, thought, search
  - For example the simple grid world above.

Game of Go

- A strategy board game for two players.
- Goal: surround more territory than the opponent.
- Extremely complex: number of legal board positions approximately 2.1×10^{170}.

Source: Wikipedia.

Evolution of modern of Go programs

- **AlphaGo**, 2016.
  - Deep NNs to evaluate states and policies.
  - Tree search to select a move to play.
  - Supervised learning with human moves and RL with self-play.
- **AlphaGo Zero**, 2017.
  - Training uses no human knowledge, just RL with self-play.
  - Rules of the game are assumed known.
- **AlphaZero**, 2018
  - A general RL algorithm for chess, shogi and Go.
- **MuZero**, 2020
  - Rules not supplied, but the model is learned.

Policy iteration

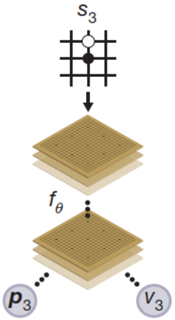
- **Policy iteration**, iterate between:
  - **Policy evaluation**: estimate the value function of the current policy
  - **Policy improvement**: use the current value function to generate a better policy

AlphaGo Zero on high-level (1)

- A combined convolutional neural network f_θ to predict the policy and value:

$$(\mathbf{p}, v) = f_\theta(s).$$

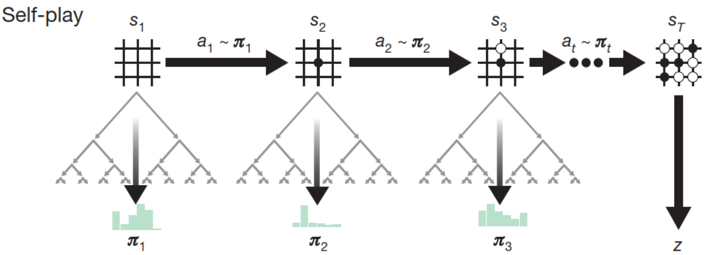
  - s: current state of the game
  - $\mathbf{p}$: a policy, $p_a = \Pr(a|s)$
  - v value, probability of winning from state s.
- The network f_θ is used to search moves during self-play.



Silver et al. (2017). Nature

AlphaGo Zero on high-level (2)

- **Self-play**: create a 'training set'
  - The best current player plays 25,000 games against itself
  - Moves are selected using Monte Carlo Tree Search (MCTS)
  - For each state s, store search probabilities π, and the eventual winner z.



Silver *et al.* (2017). Nature.

AlphaGo Zero on high-level (3)

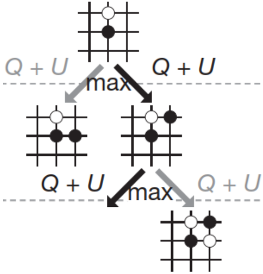
- **Retrain network**:
  - Pick a mini-batch of 2048 states from the last 500,000 games.
  - Retrain the current $(\mathbf{p}, v) = f_\theta(s)$ with these states s.
- **Loss function:**

$$l = (z - v)^2 - \boldsymbol{\pi}^\top \log \mathbf{p} + c \|\theta\|^2$$

  - Compares predictions $(\mathbf{p}, v)$ from the NN with search probabilities $\boldsymbol{\pi}$ and actual winner z.
  - Rationale: search probabilities $\boldsymbol{\pi}$ are decided by looking ahead from the current state; hence that's a better policy than the NN policy p predicted using current state only.
- **Evaluate the new network** before updating:
  - Play 400 games against the current best player.
  - Must win 55% of games to become the new best player.

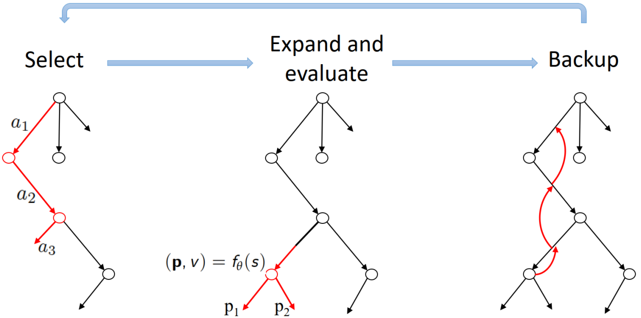
Monte-Carlo tree search

- A search tree for **deciding the next move to play**.
  - Nodes=states
  - Edges=actions.
  - Root node=current state.
- Each edge stores:
  - N number of times action has been selected from state s during search.
  - W total value of the next state
  - Q mean value of the next state.
  - p the prior for selecting action a.



Silver et al. (2017). Nature.

Monte-Carlo tree search



$$a_t = \arg\max_a \{Q(s_t, a) + U(s_t, a)\}$$

$$N(s_t, a_t) \leftarrow N(s_t, a_t) + 1$$
$$W(s_t, a_t) \leftarrow W(s_t, a_t) + v$$
$$Q(s_t, a_t) \leftarrow \frac{W(s_t, a_t)}{N(s_t, a_t)}$$

All of this is *planning*, i.e., happens 'in the head'!

Monte-Carlo tree search - move selection details

Start at the root node

- Choose action that maximizes the UCB $Q + U$
  - Q: mean value of the next state ($Q = 0$, if action has not yet been selected)
  - U : confidence bound that depends on how often the action has been chosen
  - Early in the simulation, U dominates (exploration), later Q dominates (exploitation)
- In detail:

$$U(s,a) = c\underbrace{P(s,a)}_{\text{prior}}\frac{\sqrt{\sum_b N(s,b)}}{1+N(s,a)}.$$

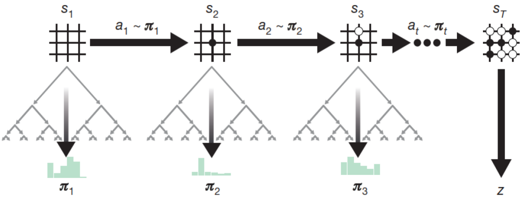
Playing a move

- After MCTS, calculate search probabilities π for root node s_0:

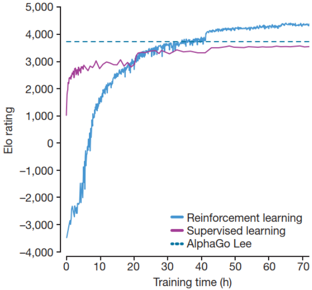
$$\pi(a|s_0) = \frac{N(s_0, a)^{1/\tau}}{\sum_b N(s_0, b)^{1/\tau}}.$$

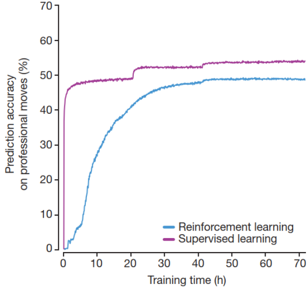
  - τ : temperature to control exploration

- Use π to **select** the next move to play.



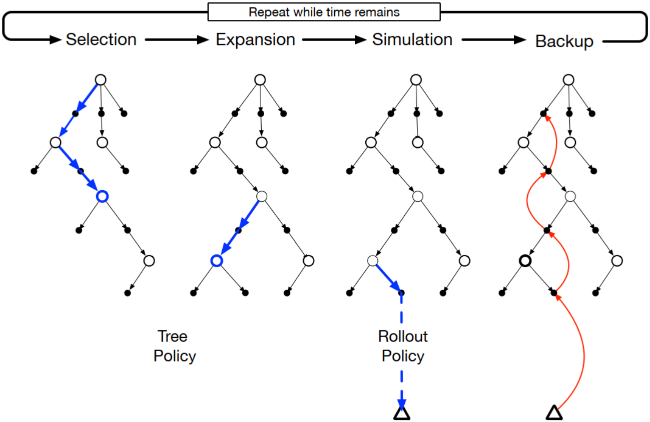
Results





- AlphaGo Lee defeated World Champ Lee Sedol in 2016.
- (left) AlphaGo Zero won AlphaGo Lee just after 30h of self-play RL
- (right) ...using moves different from human players.

Monte Carlo tree search for planning



Sutter, Fig. 8.10

- Leaf evaluation can be done using **rollouts**.
  - Rapid play until the end using a simple policy.

MCTS Applications

- Games: chess, poker, real-time video games, ...
- Robotics: path planning, multi-robot planning, task allocation,...
- Chemical synthesis
- Scheduling
- Vehicle routing
- Etc.[1]

[1] For references, see *Monte Carlo Tree Search: A Review of Recent Modifications and Applications*, https://arxiv.org/abs/2103.04931