


Reinforcement Learning

Ignacio Heredia
Instituto de Física de Cantabria (CSIC-UC)

Master Data Science (UIMP-UC)
March 2025

 iheredia@ifca.unican.es
 [IgnacioHeredia](#)

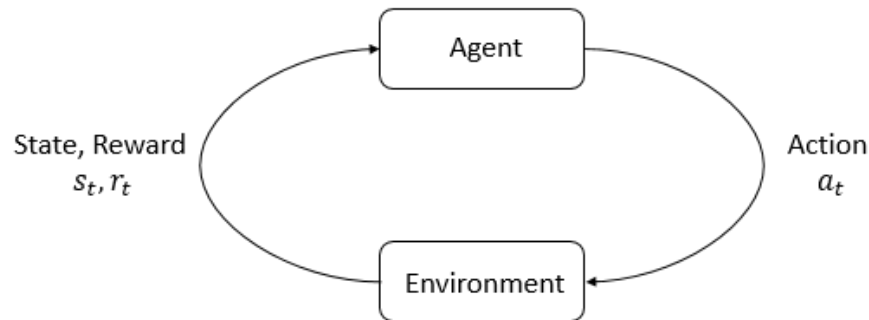
Overview of the course

- Reinforcement Learning (**1h**)
- Transformers (**1h**)
- Large Language Models (**1h**)
- Deploy your LLM (**1h**)

- Introduction
- Concepts
 - States, observations, actions
 - Policy functions
 - Trajectories
 - Rewards, returns
 - Value functions
- Algorithms
 - Taxonomy
 - Model-free vs Model-based
- Challenges
- Notable papers
 - Atari
 - AlphaGo

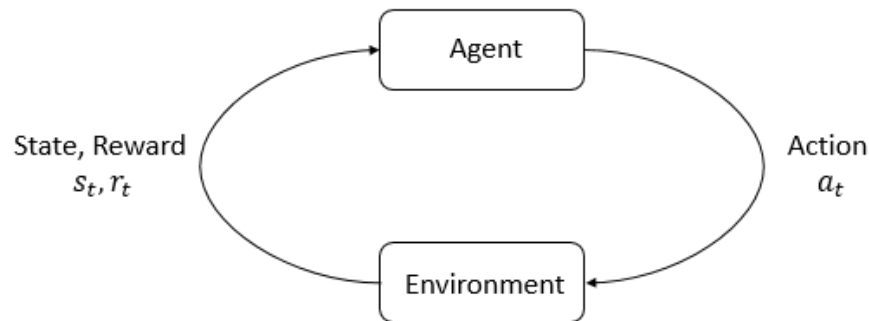
Reinforcement Learning (RL) is a *sequential decision*-making framework in which **agents** learn to perform **actions** in an **environment** with the goal of maximizing cumulative received **rewards** (ie. **return**).

At every step of interaction, the agent sees a (possibly partial) **observation** of the state of the world, and then decides on an action to take. The environment changes when the agent acts on it, but may also change on its own.

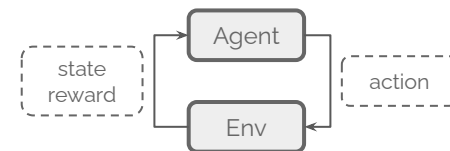


Examples:

- *video games*: control the moves (actions) of a character (the agent) in a video game (the environment), aiming to maximize the score (the reward),
- *robotics*: control the movements (actions) of a robot (the agent) in the world (the environment) to perform a task (earning a reward),
- *finance*: control a virtual trader (the agent) who buys or sells assets (the actions) on a financial exchange (the environment) to maximize profit (the reward)



- A **state** s is a *complete* description of the state of the world. There is no information about the world which is hidden from the state.
- An **observation** o is a *partial* description of a state, which may omit information.
- We represent states and observations by a real-valued tensor.
 - For instance, a visual observation could be represented by the RGB matrix of its pixel values;
 - the state of a robot might be represented by its joint angles and velocities.
- The **action space** is the set of all valid actions. The action space can be *discrete* (Go, chess) or *continuous* (controlling a robot). Not all RL algorithms work for both kind of action spaces.

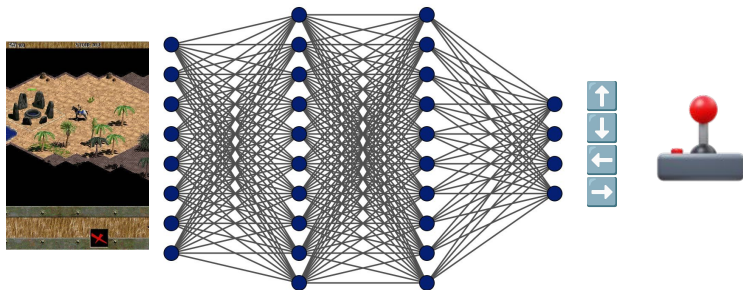


A **policy** is a rule used by an agent to decide what actions to take, in order to *maximize reward*:

- it can be **deterministic**, denoted by μ
- it may be **stochastic**, denoted by π

In **deep RL**, we deal with **parameterized policies**.
For example, a neural network function with weights θ :

$$a_t = \mu_{\theta}(s_t)$$



There are two types of stochastic policies:

- **Categorical policies**: work in *discrete spaces*. It can be a neural network that outputs a softmax over all possible actions. From these probabilities we can sample an action.
- **Diagonal Gaussian policies**: work in *continuous spaces*. It can be a neural network that predict the parameters of a Gaussian distribution. Then sample an action:

$$a_t = \mu_{\theta}(s_t) + \sigma_{\theta}(s) \odot z$$

where $z \sim N(0, I)$

A trajectory is a *sequence of states and actions* in the world, $\tau = (s_0, a_0, s_1, a_1, \dots)$

- The first state is sampled from the start-state distribution.
- State transitions are $s_t \rightarrow s_{t+1}$ are governed by the laws of the environment and by the *last action* taken (**Markov process**).

The transitions can be:

- **deterministic:** $s_{t+1} = f(s_t, a_t)$
- **stochastic:** $s_{t+1} \sim P(\cdot | s_t, a_t)$

They are also called *episodes* or *rollouts*.

The reward function \mathbf{R} depends on the current state of the world, the action just taken, and the next state of the world: $r_t = R(s_t, a_t, s_{t+1})$

The goal of the agent is to maximize some notion of cumulative reward over a trajectory. We can cumulate rewards in two ways:

- *finite-horizon undiscounted* return: $R(\tau) = \sum_{t=0}^T r_t$
- *infinite-horizon discounted* return: $R(\tau) = \sum_{t=0}^{\infty} \gamma^t r_t$

In infinite-horizon discounted return we multiply future rewards by a **discount factor** $\gamma \in (0,1)$

- to balance short-term vs long-term rewards (future rewards are more uncertain)
- because infinite sums might not converge,
- act as a reward regularization by softening noisy rewards,

The goal in RL is to *select a policy which maximizes expected return* when the agent acts according to it. This is called the optimal policy π^* .

The value of a state (or state-action pair) is the *expected return if you start in that state* or state-action pair, and then act according to a particular policy forever after.

There are different value functions:

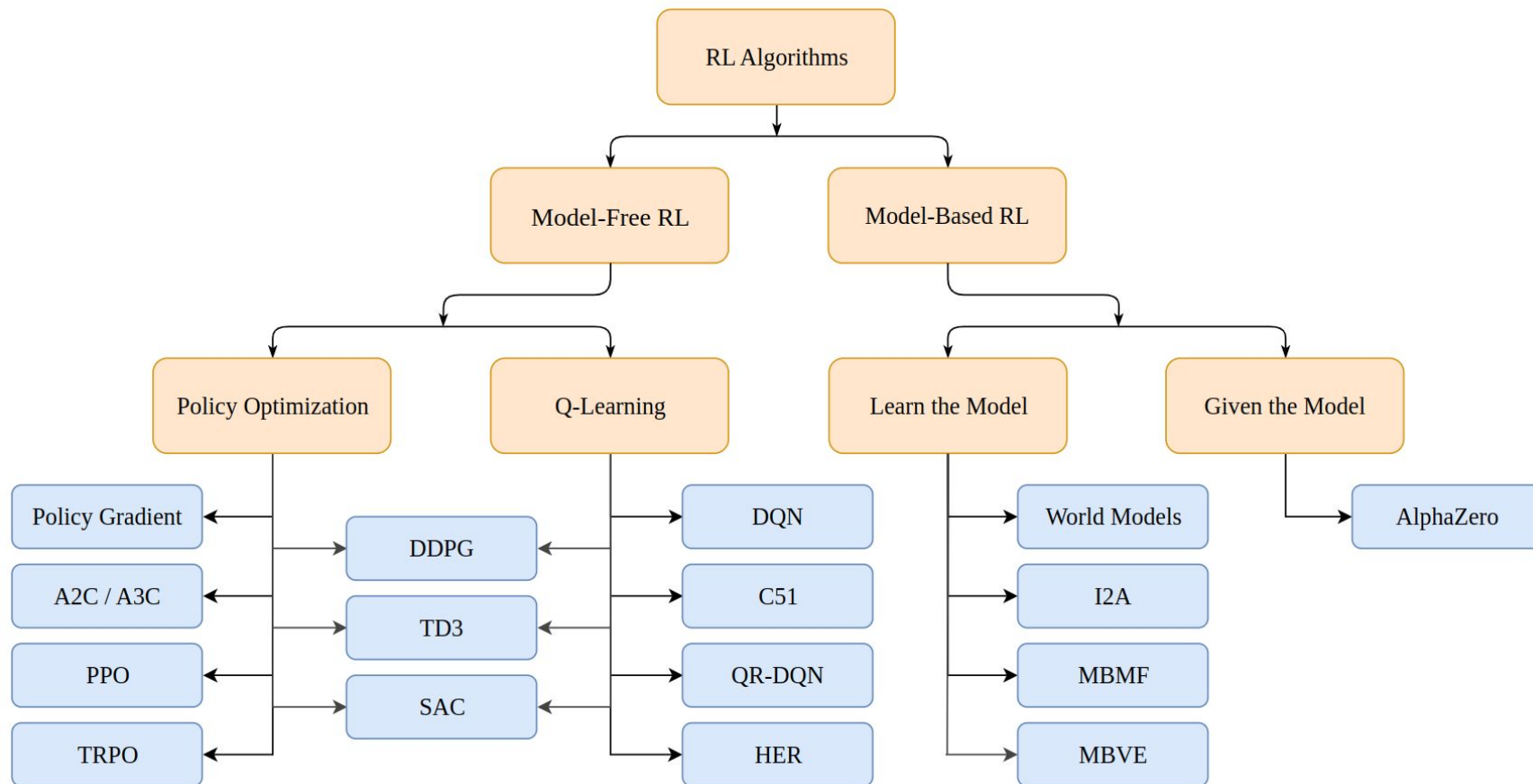
- **On-Policy Value** Function, $V^\pi(s)$, which gives the expected return if you start in state s and always act according to policy π .
- **On-Policy Action-Value** Function, $Q^\pi(s, a)$ which gives the expected return if you start in state s , take an arbitrary action a (which may not have come from the policy), and then forever after act according to policy π .

$$V^\pi(s) = E_{a \sim \pi}[Q^\pi(s, a)]$$

The value function obeys the **Bellman equations** which state that the value of your starting point is the reward you expect to get from being there, plus the value of wherever you land next.

$$V^\pi(s) = E_{a \sim \pi} E_{s' \sim P}[r(s, a) + \gamma V^\pi(s')]$$

- **State** [s] produced by an environment.
- **Observation** [o] is the partial state the agent has access to.
- **Action** [a] an agent can take
- **Policy** [π] function an agent follows to select an action
- **Reward** [r] an agent receives after performing an action
- **Trajectory** [τ] is the concatenations of state actions
- **Return** [R] is the cumulated reward over trajectories, usually using a **discount-rate** [γ].
- **Value** [Q] function, that estimates the return one will receive in a given state is one follows the policy function



Whether the agent has *access to (or learns) a model of the environment*. By a model of the environment, we mean a function which predicts state transitions and rewards.

- **Model-based methods:** Having a model is that it allows the agent to plan by *thinking ahead*, seeing what would happen for a range of possible choices, and explicitly deciding between its options. Agents can then distill the results from planning ahead into a learned policy. This is very *sample efficient*!
- **Model-free methods:** A ground-truth model of the environment is usually not available to the agent. If an agent wants to use a model in this case, it has to *learn the model purely from experience*, which creates several challenges:
 - *bias in the model* can be exploited by the agent, resulting in an agent which performs well with respect to the learned model, but behaves poorly in the real environment.
 - *very compute intensive* and time consuming, and many times it won't even work,

→ but model-free methods they tend to be *easier to implement* and tune!

- **Sparse rewards:** Consider learning to play chess. Here, there is a reward of +1, -1, or 0 at the end of the game if the agent wins, loses, or draws and 0 at every other time step. So you *only receive rewards once*, at the end of the game.
- **Delayed rewards:** You often receive the rewards much later after performing an action. The first moves in a chess game can be crucial, but you will only receive a reward for it *much later on*, when the game has finished. Therefore is hard to *assign credit* to the best moves (did I win because of the first move? the tenth move? the Nth move?)
- **Stochastic environment:** your chess opponent never reacts the same to a given move. Was the move really good or did your opponent replied badly?
- **Exploration vs Exploitation:** when you are learning to play, should you play the moves you are good or should you try new moves? If you try new moves, you can potentially discover better moves, but trying to much will mean that you make a lot of bad moves also. Balancing those two is what is called the *exploration vs exploitation trade-off*. We have to add to this the fact that the action space is huge.

Let's illustrate the concepts by the first big feat in modern RL:

[Human-level control through deep reinforcement learning](#),

Deepmind, *Nature* (2015)

This is the first papers to demonstrate at scale that neural networks can be used as agents to select the policy.

It trained a neural network to play the classic Atari 2600 games, surpassing humans professionals in 49 games.

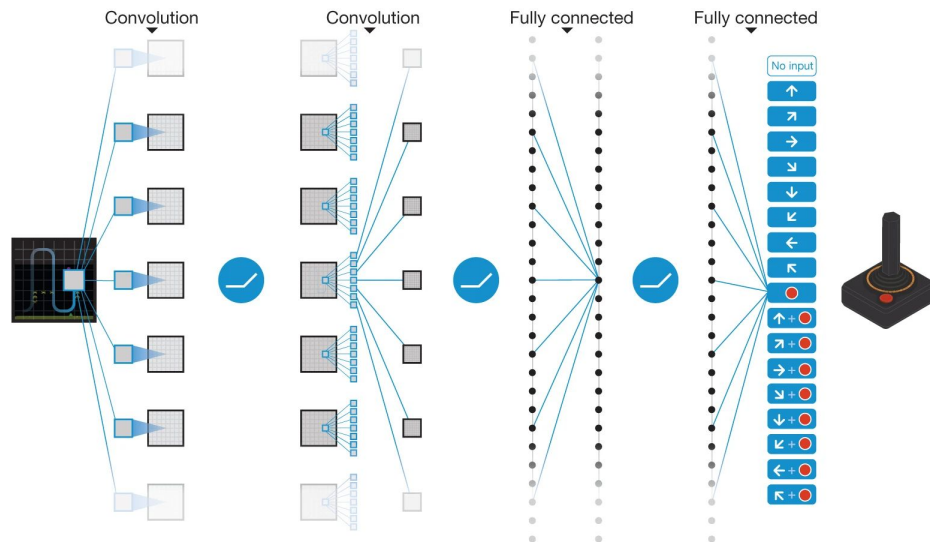


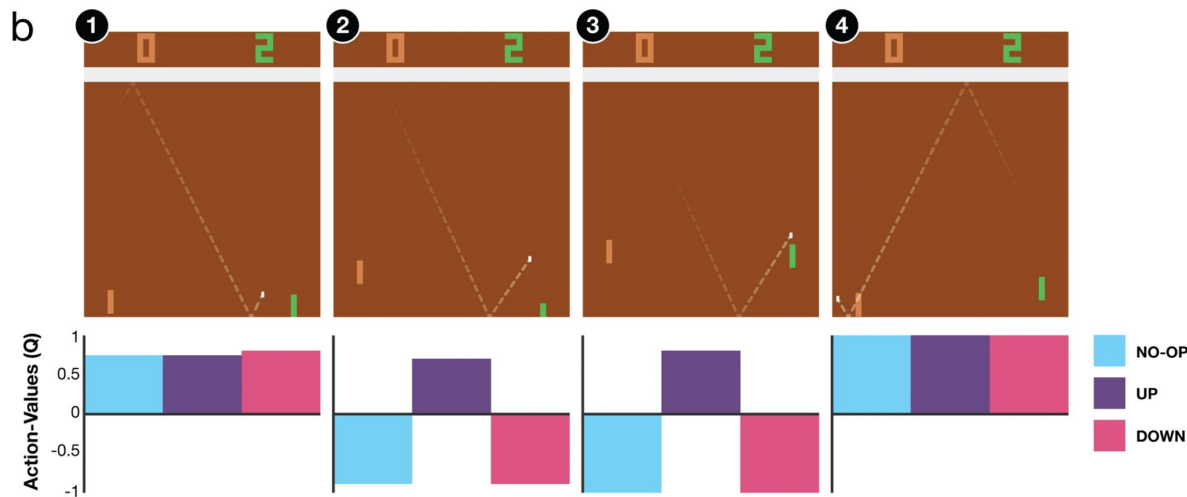
The policy π is a neural network that given a state s outputs an action a .

- **Inputs:**
 - Last 4 frames → we need more than one frame to see dynamics (direction, velocity, acceleration)
 - Game score
- **Network:** Conv, Relu, FC
- **Output:** action to perform

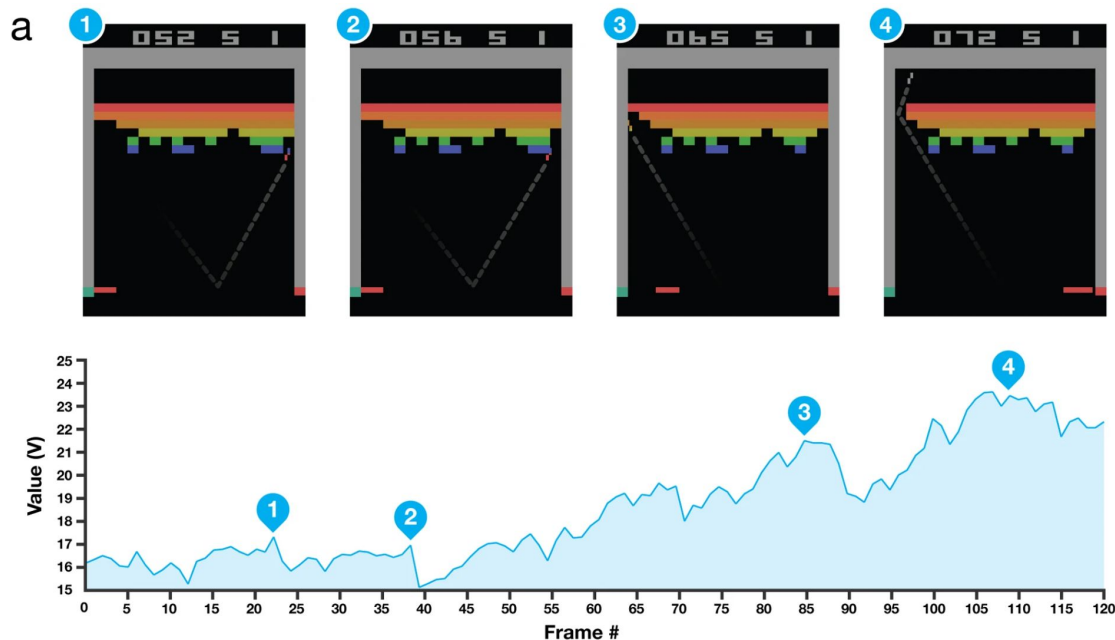
Same hyperparameters across all games, only n° of output actions changed.

No prior knowledge is used!





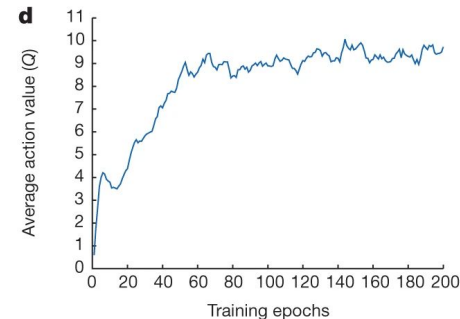
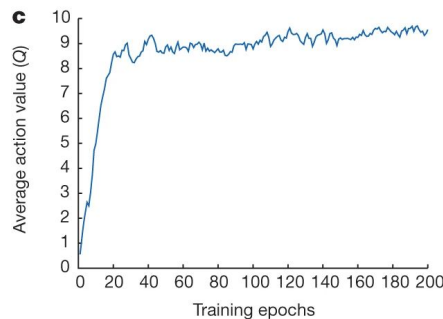
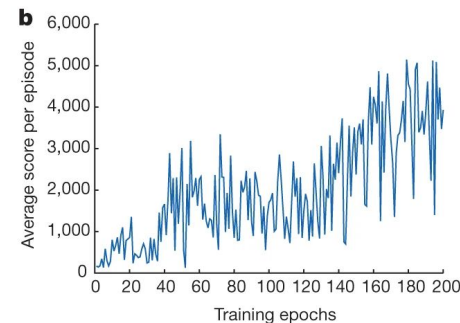
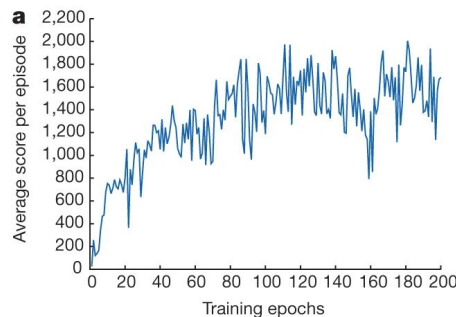
b, A visualization of the learned action-value function on the game Pong. At time point 1, the ball is moving towards the paddle controlled by the agent on the right side of the screen and the values of all actions are around 0.7, reflecting the expected value of this state based on previous experience. At time point 2, the agent starts moving the paddle towards the ball and the value of the 'up' action stays high while the value of the 'down' action falls to -0.9 . This reflects the fact that pressing 'down' would lead to the agent losing the ball and incurring a reward of -1 . At time point 3, the agent hits the ball by pressing 'up' and the expected reward keeps increasing until time point 4, when the ball reaches the left edge of the screen and the value of all actions reflects that the agent is about to receive a reward of 1. Note, the dashed line shows the past trajectory of the ball purely for illustrative purposes (that is, not shown during the game). With permission from Atari Interactive, Inc.



a, A visualization of the learned value function on the game Breakout. At time points 1 and 2, the state value is predicted to be ~ 17 and the agent is clearing the bricks at the lowest level. Each of the peaks in the value function curve corresponds to a reward obtained by clearing a brick. At time point 3, the agent is about to break through to the top level of bricks and the value increases to ~ 21 in anticipation of breaking out and clearing a large set of bricks. At point 4, the value is above 23 and the agent has broken through. After this point, the ball will bounce at the upper part of the bricks clearing many of them by itself.

Experience replay: Sampling from the policy network is expensive. So to make it more *efficient* they save experiences (s, a, r) and do several passes over them.

This also *stabilizes training* by removing correlations in sequences and smoothing data distribution changes,



Atari - Playing Breakout



(*) Games which require *extended planning strategies* (eg. Montezuma's revenge) still constitute a major challenge.

- Go is a game of perfect information,
- Transitions are deterministic,
- You have b^d possible sequences:
 - b : game breadth, legal move
 - d : game depth, game length

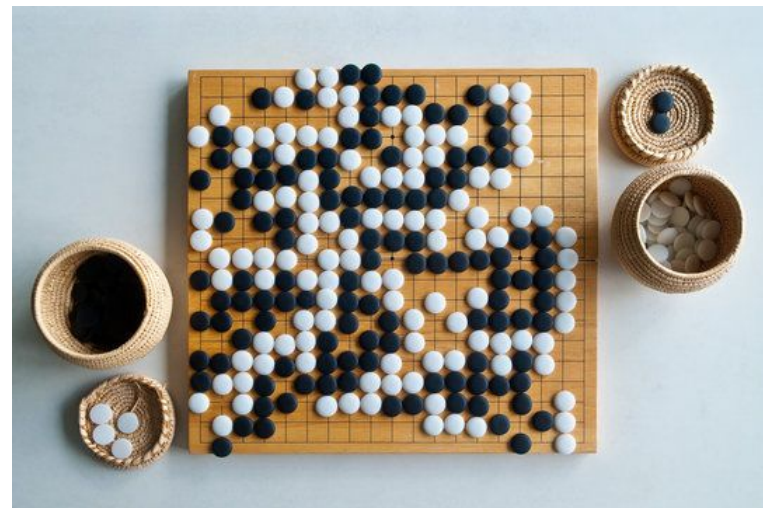
→ **Chess**: 35^{80} , **Go**: 250^{150}

- It's infeasible to play by searching the tree exhaustively. Expert players sometimes talk about intuitions, the “beauty” of some table positions, etc.

→ You have to estimate the positions!

[Mastering the game of Go with deep neural networks and tree search,](#)

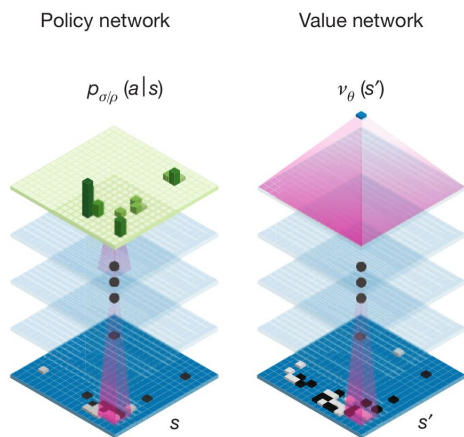
Deepmind, *Nature* (2016)



You have two networks:

- **value networks:** to evaluate board positions
- **policy networks:** to select moves.

Networks receive as input the board position as *48 images* of size *19 x 19* and process it using convolutional layers.



Extended Data Table 2 | Input features for neural networks

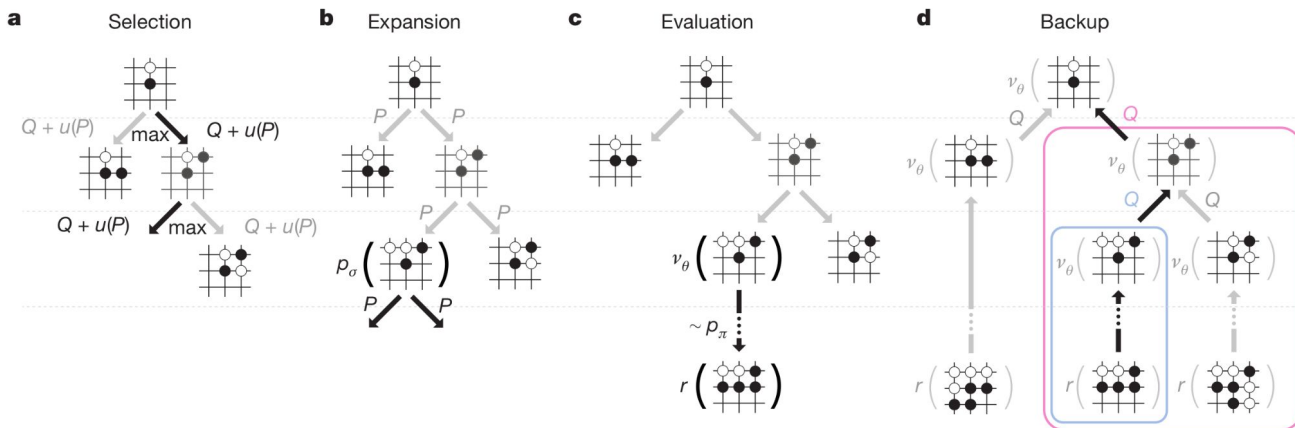
| Feature | # of planes | Description |
|----------------------|-------------|---|
| Stone colour | 3 | Player stone / opponent stone / empty |
| Ones | 1 | A constant plane filled with 1 |
| Turns since | 8 | How many turns since a move was played |
| Liberties | 8 | Number of liberties (empty adjacent points) |
| Capture size | 8 | How many opponent stones would be captured |
| Self-atari size | 8 | How many of own stones would be captured |
| Liberties after move | 8 | Number of liberties after this move is played |
| Ladder capture | 1 | Whether a move at this point is a successful ladder capture |
| Ladder escape | 1 | Whether a move at this point is a successful ladder escape |
| Sensibleness | 1 | Whether a move is legal and does not fill its own eyes |
| Zeros | 1 | A constant plane filled with 0 |
| Player color | 1 | Whether current player is black |

Feature planes used by the policy network (all but last feature) and value network (all features).

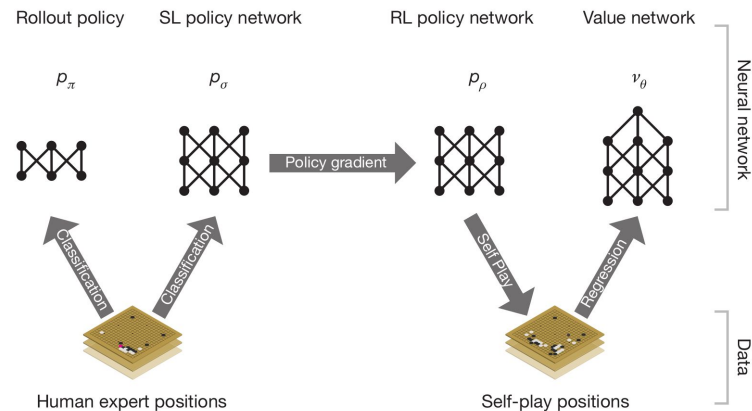
Before selecting your final move you:

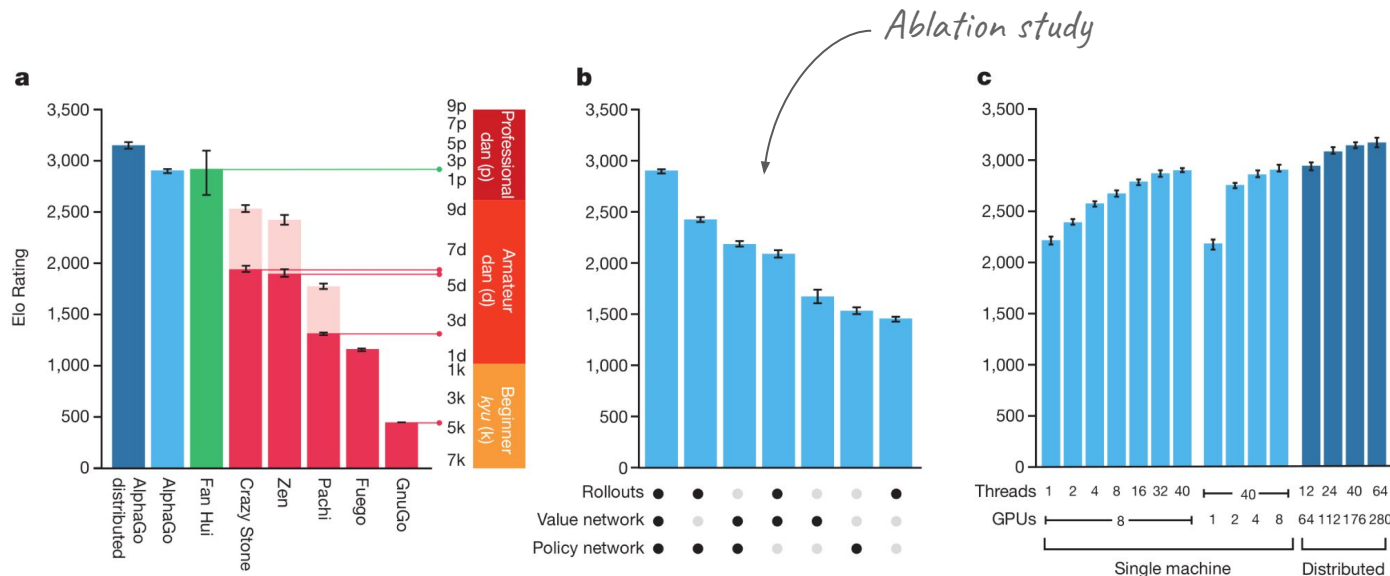
- select promising moves (using the fast rollout policy)
- iteratively expand them
- when you are doing exploring you update the value of the initial set of moves with the mean value of the child nodes.

These are called **Montecarlo rollouts**.



- training a **supervised learning (SL) policy network** p_σ directly from expert human moves. This provides fast, *efficient learning* updates with immediate feedback and high-quality gradients.
- train a **fast policy** p_π that can rapidly *sample actions* during rollouts,
- train a **reinforcement learning (RL) policy network** p_ρ that improves the SL policy network by optimizing the final outcome of games of self-play. This adjusts the policy towards the correct goal of *winning games*, rather than maximizing predictive accuracy
- train a **value network** v_θ that predicts the winner of games played by the RL policy network against itself.





- A more advanced version of AlphaGo not only defeated Fan-Hui (European champion) but also Lee Sedol (World champion). ([AlphaGo documentary](#))
- A subsequent version ([AlphaZero](#)) was able to be trained without the supervised learning from human moves.

- RL is *hard*:
 - things don't always work out,
 - models are hard to debug,
 - algorithms are not always simple,
- But RL is a key piece to achieve **AGI** (Artificial General Intelligence), the point when machines surpass human beings: you need an *autonomous agent* that can make its own decisions to be able to surpass human strategies (typically used in supervised learning).

"It doesn't play like a human, and it doesn't play like a program. It plays in a third, almost alien, way."


-Demis Hassabis on AlphaZero-

- RL works better in combination with other types of learning, as we saw with AlphaGo or we will see in the Large Language Models (LLMs) class.

Y. LeCun

How Much Information is the Machine Given during Learning?

- ▶ **“Pure” Reinforcement Learning (cherry)**
 - ▶ The machine predicts a scalar reward given once in a while.
 - ▶ **A few bits for some samples**
- ▶ **Supervised Learning (icing)**
 - ▶ The machine predicts a category or a few numbers for each input
 - ▶ Predicting human-supplied data
 - ▶ **10→10,000 bits per sample**
- ▶ **Self-Supervised Learning (cake génoise)**
 - ▶ The machine predicts any part of its input for any observed part.
 - ▶ Predicts future frames in videos
 - ▶ **Millions of bits per sample**




© 2019 IEEE International Solid-State Circuits Conference 1.1: Deep Learning Hardware: Past, Present, & Future 59

Yann LeCun's cake (2016)



Study materials:

- [Reinforcement Learning: An introduction](#), Sutton & Barto ←  [Turing Award 2024](#)
- [UCL Course on RL](#), David Silver
- [Spinning Up in Deep RL](#), OpenAI
- [Understanding Deep Learning](#), Simon Prince
- [CleanRL](#) - single file algorithm implementations, useful to learn

Software | Algorithms:

- [Tianshou](#) - RL in Pytorch
- [Stable Baselines3](#) - RL in Pytorch
- [RLlib](#) - RL with Ray
- [Torch RL](#) - official RL library from Pytorch

Software | Environments:

- [Gymnasium](#) (formerly OpenAI Gym)
- [PettingZoo](#) - multi agent version of Gymnasium

Questions

Ignacio Heredia
Instituto de Física de Cantabria (CSIC-UC)



iheredia@ifca.unican.es



[IgnacioHeredia](https://github.com/ignacioheredia)



Image sources