Deep Reinforcement Learning

TIN0145 - TAA - Aprendizagem Profunda

2023.1 - Prof. Pedro Moura

Deep Reinforcement Learning

- **Deep reinforcement learning** has produced some of the most surprising artificial-neural-network advances.
  - Including the lion's share of the headline-grabbing "artificial intelligence" breakthroughs of recent years.

- We will now introduce what **reinforcement learning** is and how its fusion with deep learning has enabled machines to meet or surpass human-level performance on a **diverse range of complex challenges**:
  - Atari videogames
  - The board game Go; and
  - Subtle physical-manipulation tasks.

Video Games

- When learning a new vídeo game as a child, you quickly became aware that missing the ball in Pong or Breakout was an unproductive move.
- You **processed the visual information on the screen** and, yearning for a **score in excess** of your friends', devised strategies to manipulate the controller effectively and achieve this aim.
- Recently, researchers at DeepMind have been producing software that likewise learns how to play classic Atari games.

Video Games

- DeepMind was a British technology startup founded by Demis Hassabis, Shane Legg, and Mustafa Suleyman in London in 2010.
- Their stated mission was to “solve intelligence”: they were interested in extending the field of AI by developing increasingly general-purpose learning algorithms.
- One of their early contributions was the introduction of **Deep Q-Learning Networks (DQNs)**.
- DQN: a single model architecture was able to learn to play multiple Atari 2600 games well.
  - From scratch, through trial and error.

Video Games

Demis Hassabis cofounded DeepMind in 2010 after completing his PhD in cognitive neuroscience at University College London.

Video Games

- In 2013, Volodymyr Mnih and his DeepMind colleagues published an article on their DQN agent.
- Their agent received raw pixel values from its **environment**, a video game emulator, as its **state** information.
  - Akin to the way human players of Atari games view a TV screen
- Process visual information → Mnih et al.'s DQN included a **Convolutional Neural Network (CNN)**.
  - A common tactic for any deep reinforcement learning model that is fed visual data.

Video Games

- The **handling of the flood of visual input** from Atari games (~ 2M pixels per second) underscores how well suited deep learning in general is to filtering out pertinent features from noise.

- Playing Atari games within an emulator is a problem that is **well suited to deep reinforcement learning** in particular:
  - While they provide a **rich set of possible actions** that are engineered to be challenging to master…
  - There is thankfully **no finite limit** on the amount of training data available because the agent can engage in **endless rounds of play**.

Video Games

- During training, the DeepMind DQN was **not provided** any hints or strategies.

- It was provided only with:
  - **State** (screen pixels);
  - **Reward** (its point score, which it is programmed to maximize); and
  - Range of possible **actions** (game-controller buttons) available in a given Atari game.

Video Games

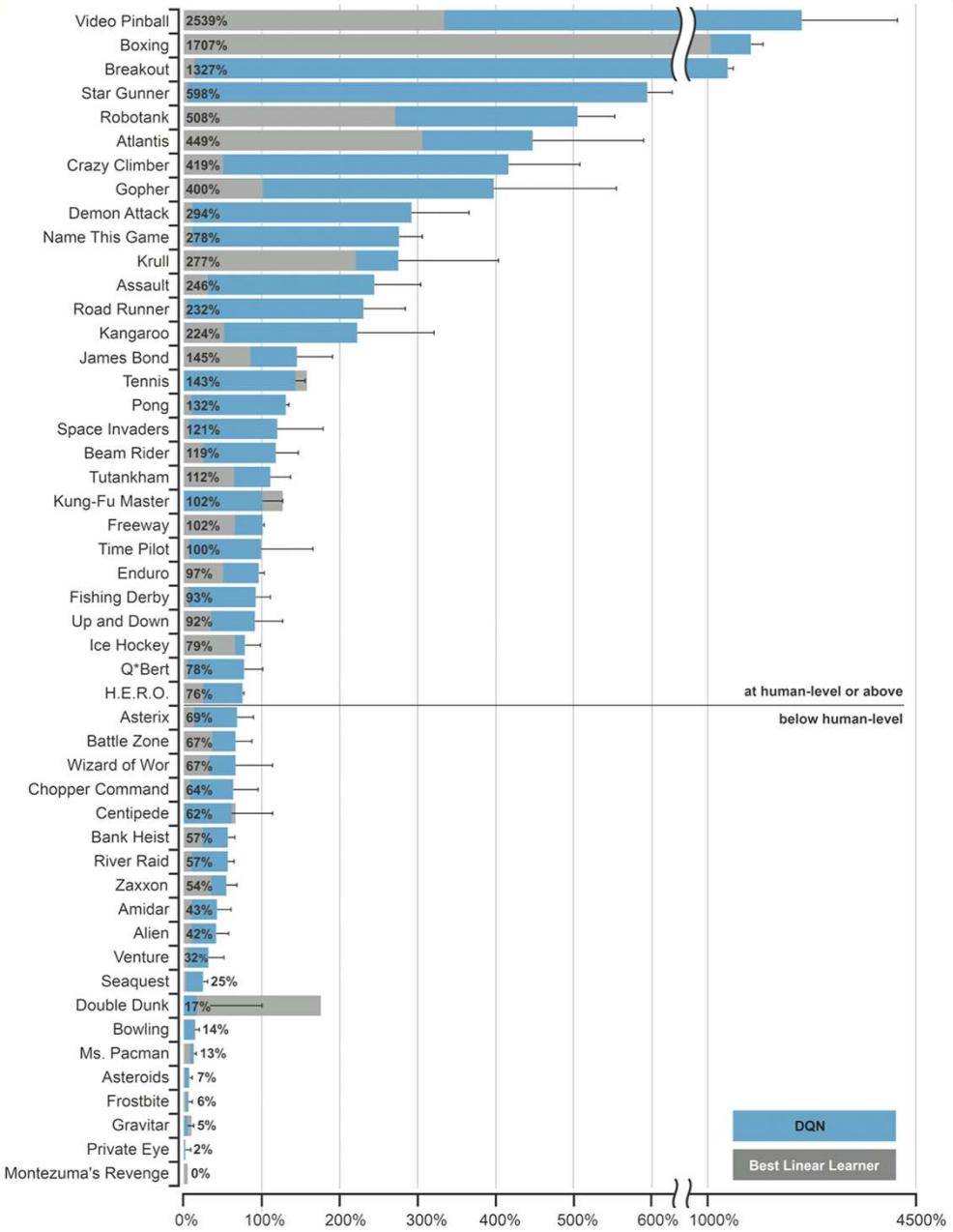
- **The model was not altered for specific games.**
- It was able to **outperform** existing machine learning approaches in **six of the seven games** Mnih and his coworkers tested it on, even **surpassing the performance of expert human players on three**.
- Perhaps influenced by this conspicuous progress, Google acquired DeepMind in 2014 for the equivalent of half a billion U.S. dollars.

Video Games

- Follow-up paper → published in the journal *Nature*, Mnih and his teammates at now-Google DeepMind assessed their DQN algorithm **across 49 Atari games**.

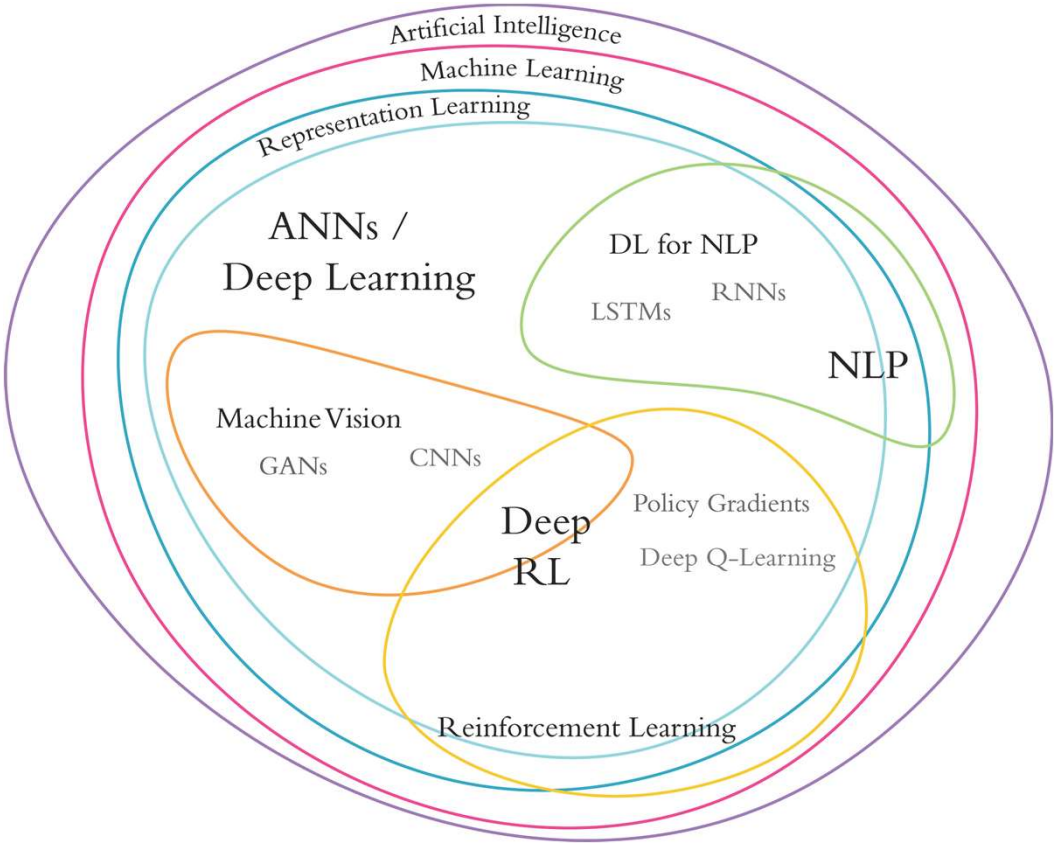
- It **outperformed** other machine learning approaches **on all but three of the games** (94 percent of them).

- Astonishingly, it scored **above human level on the majority of them** (59 percent).



The normalized performance scores of Mnih and colleagues' (2015) DQN relative to a professional game tester: 0% is random play, and 100% is pro's best performance. The horizontal line: the 75th percentile of professionals' scores.

Reinforcement Learning



Venn diagram showing the relative positioning of the major concepts covered over the course of this book.

Essential Theory of Reinforcement Learning

- **Reinforcement Learning** is a machine learnig paradigm involving:
- An **agent** taking an **action** within an **environment** (in a timestep ***t***).
- Environment returns two types of information to the agent:
- **Reward**: a scalar value that provides **quantitative feedback** on the action that the agent took at timestep **t**.
  - In the video-game Pac-Man, it could be 100 points as a reward for acquiring cherries.
  - In this video, the reward is tailored according to the objective of teaching the agent how to properly walk (one should read the first comment).

Essential Theory of Reinforcement Learning

- **The agent's objective is to maximize the rewards it accumulates.**
- Rewards are what reinforce productive behaviors that the agent discovers under particular environmental conditions.

- **State**: the environment changes in response to an agent's action.
  - In the next timestep (**t+1**), these will be the conditions for the agent to choose an action in.

Essential Theory of Reinforcement Learning

- Repeating the above two steps in a loop **until reaching some terminal state**.

- This terminal state could be reached by, for example:
  - Attaining the maximum possible reward;
  - Attaining some specific desired outcome (such as a self-driving car reaching its programmed destination);
  - Running out of allotted time;
  - Using up the maximum number of permitted moves in a game; or
  - The agent dying in a game.

Essential Theory of Reinforcement Learning

- Reinforcement learning problems are sequential decision-making problems.

- Some applications:
  - Atari video games, such as Pac-Man, Pong, and Breakout
  - Autonomous vehicles, such as self-driving cars and aerial drones.
  - Board games, such as Go, chess, and shogi.
  - Robot-arm manipulation tasks, such as removing a nail with a hammer.

The Cart-Pole Game

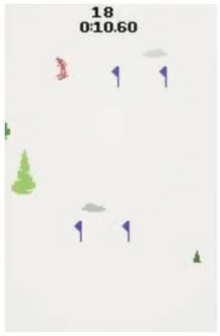
- We will now use **OpenAI Gym -** a popular library of reinforcement learning environments - to train an agent to play Cart-Pole.

- Cart-Pole is a classic problem among academics working in the field of control theory.

The Cart-Po

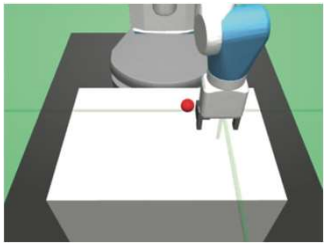
(a)

(b)

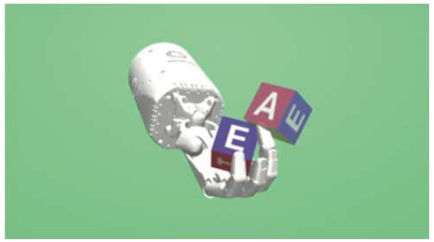


(c)

(d)



(e)



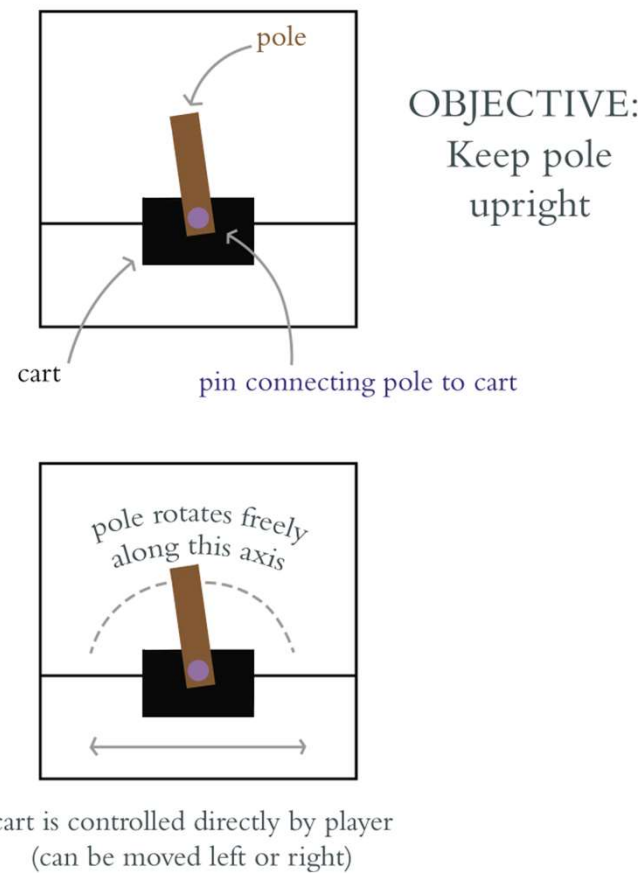
(f)

A sampling of OpenAI Gym environments: (a) CartPole; (b) LunarLander; (c) Skiing, an Atari 2600 game; (d) Humanoid; (e) FetchPickAndPlace, one of several available simulations of real-world robot arms; and (f) HandManipulateBlock, another practical simulation of a robotic arm, the Shadow Dexterous Hand.

The Cart-Pole Game

- The objective is to balance a pole on top of a cart.
  - Pole is connected to the cart at a purple dot, which functions as a pin that permits the pole to rotate along the horizontal axis.
  - The cart itself can only move horizontally (to the left or right). At any given timestep, the cart must be moved. **It can't remain stationary.**
- Each **episode** of the game begins with the cart positioned at a random point near the center of the screen and with the pole at a random angle near vertical.

The Cart-Pole Game

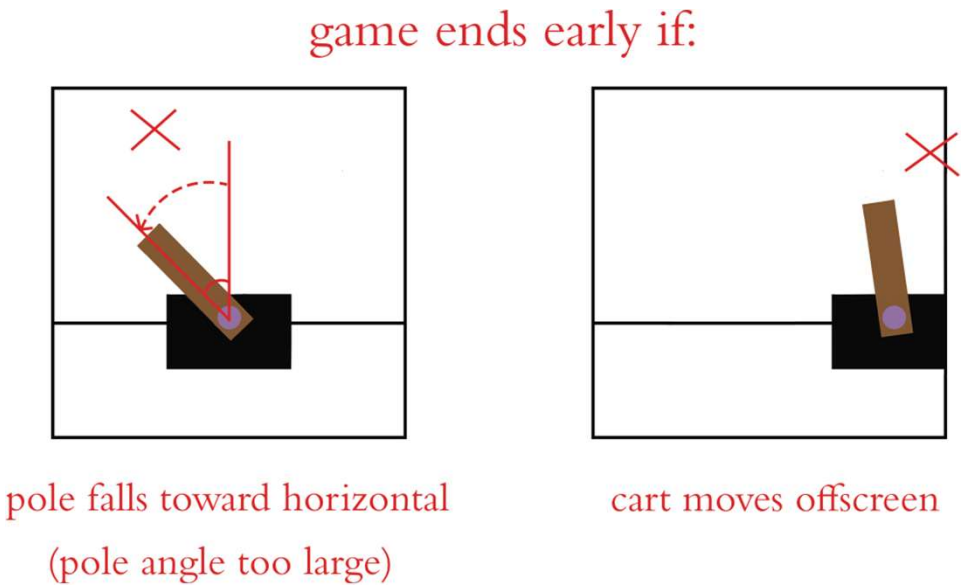


The objective of the Cart-Pole game is to keep the brown pole balanced upright on top of the black cart for as long as possible.

The Cart-Pole Game

- An episode **ends** when either:
  - The pole is no longer balanced on the cart -> when the angle of the pole moves too far away from vertical toward horizontal.
  - The cart touches the boundaries - the far right or far left of the screen.
- In this version of the game, the **maximum number of timesteps** in an episode is 200.
  - If the episode does not end early, then the game will end after 200 timesteps.
- One point of **reward** is provided for every timestep that the episode lasts, so the maximum possible reward is 200 points.

The Cart-Pole Game



The Cart-Pole game ends early if the pole falls toward horizontal or the cart is navigated off-screen.

The Cart-Pole Game

- The Cart-Pole game is a popular introductory reinforcement learning problem because it's so simple.

- With a self-driving car, there are effectively an infinite number of possible environmental **states**:
  - It can have a huge number of sensors: cameras, radar, lidar, accelerometers, microphones, and so on.
  - On the order of a gigabyte of data per second.

- A number of fairly nuanced **actions** are possible with a self-driving car: accelerating, braking, and steering right or left.

The Cart-Pole Game

- In contrast, the Cart-Pole game has four pieces of state information:
  - 1. The position of the cart along the one-dimensional horizontal axis;
  - 2. The cart's velocity;
  - 3. The angle of the pole; and
  - 4. The pole's angular velocity.

- At any given timestep *t*, exactly one **action** can be taken from only two possible actions: **move left or move right**.

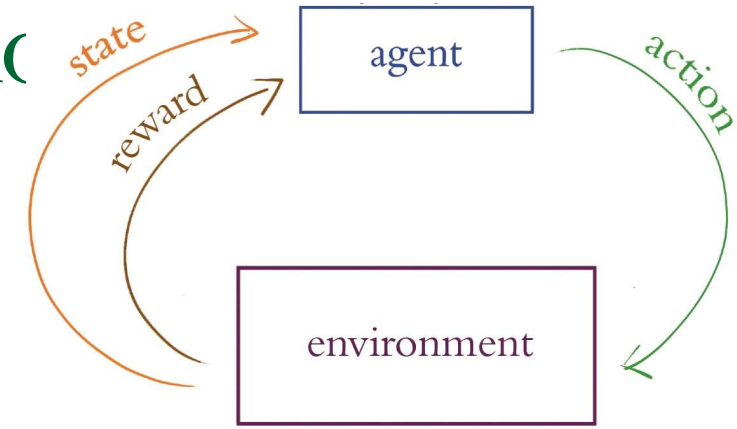
Markov Decision Processes

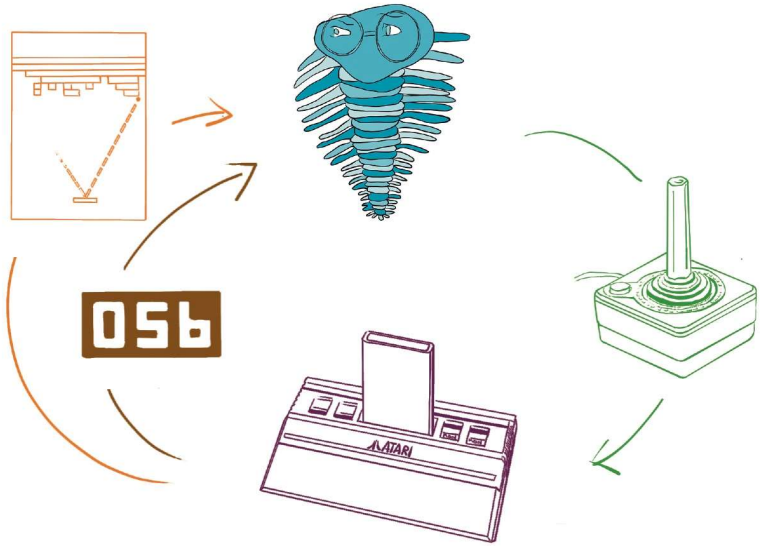
- Reinforcement learning problems can be defined mathematically as a **Markov decision process**.

- MDPs feature the so-called **Markov property**:
  - an assumption that the current timestep contains all of the pertinent information about the state of the environment from previous timesteps.

  - Our agent would elect to move right or left at a given timestep *t* by considering only the attributes of the cart (e.g., its location) and the pole (e.g., its angle) at that particular timestep *t*.

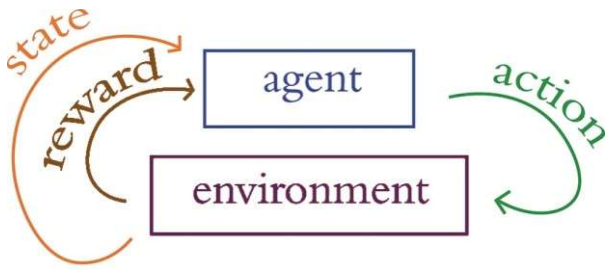
Markc cesses





The reinforcement learning loop.

Markov Decision Processes



"Markov Decision Process"

S: all possible states
A: all possible actions
R: reward distribution given (s,a)
$\mathbb{P}$: transition probability to s_{t+1} given (s,a)
γ: discount factor

The reinforcement learning loop can be considered a Markov decision process, which is defined by the five components **S**, **A**, **R**, **P**, and **γ** (bottom).

Markov Decision Processes

- **S** is the set of all possible states.
  - Each individual possible state is represented by the lowercase **s**.
  - Number of possible recombinations of its four state dimensions is enormous: the cart could be moving slowly near the far-right of the screen with the pole balanced vertically, or the cart could be moving rapidly toward the left edge of the screen with the pole at a wide angle turning clockwise with pace.
- **A** is the set of all possible actions.
  - Cart-Pole game: this set contains only two elements (left and right).
  - Other environments have many more.
  - Each individual possible action is denoted as **a**.

Markov Decision Processes

- **R** is the distribution of reward given a **state-action pair** denoted as (**s**; **a**).
  - It's a probability distribution: the exact same state-action pair (**s**; **a**) might randomly result in different amounts of reward **r** on different occasions.
  - The details of the reward distribution **R** are hidden from the agent but **can be glimpsed by taking actions within the environment**.
    - Exploration of the environment is a way of achieving this objective.

Markov Decision Processes

- In the last figure, the cart is centered within the screen and the pole is angled slightly to the left.
  - Pairing the action of **moving left** with this state **s** on average, correspond to a higher expected reward **r** relative to pairing the action of moving right with this state.
  - Moving left in this state **s** should cause the pole to stand more upright

- Pairing the action of **moving left** with this state **s** would, on average, correspond to a higher expected reward **r than moving right**.
  - Left → causes the pole to stand more upright and increases the number of timesteps that the pole is kept balanced for → higher reward **r**.
  - Right → increases the probability that the pole would fall toward horizontal → early end to the game and a smaller reward **r**.

Markov Decision Processes

- **P** is also a probability distribution. It represents the probability of the next state (s_{t+1}) given a particular state-action pair (**s**, **a**) in the current timestep *t*.
- The **P** distribution is also hidden from the agent, but again aspects of it can be inferred by taking **actions within the environment**.
- In the Cart-Pole game: relatively straightforward for the agent to learn that the left action corresponds directly to the cart moving leftward.
- More complex relationships → would be more difficult to learn and so would require more gameplay.
  - The left action in the state **s** captured in the last figure tends to correspond to a more vertically oriented pole in the next state s_{t+1}.

Markov Decision Processes

- ɣ (gamma) is a hyperparameter called the *discount factor* (also known as *decay*).
  - The same meaning as in *discounted cash flow*.

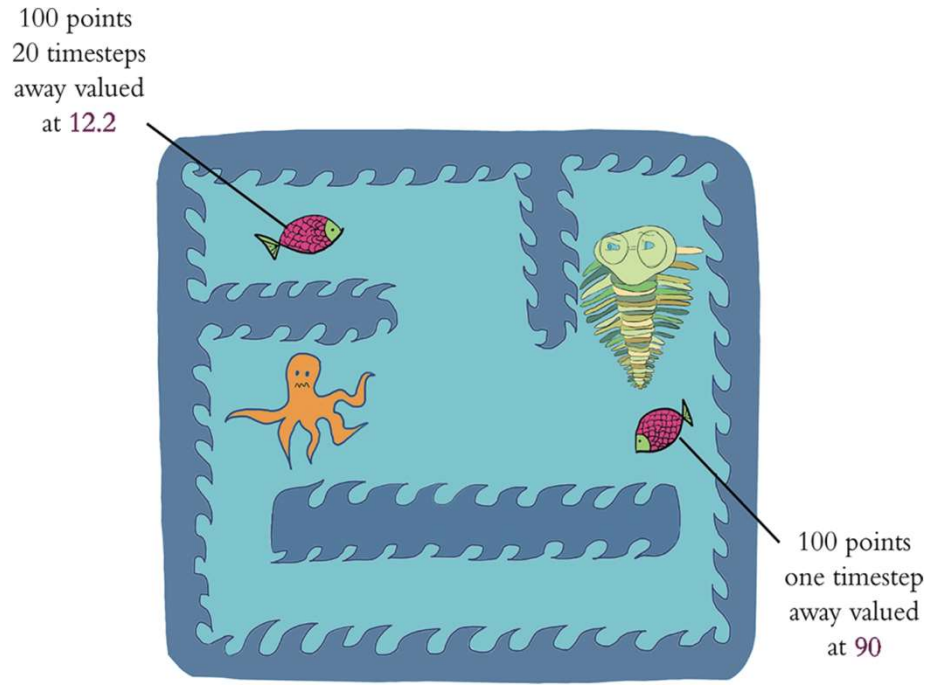
- Let's use a Pac-Man example to understand its use.
  - It explores two-dimensional surface, gaining reward points for collecting fruit and dying if he gets caught by one of the ghosts that's chasing him.

- When the agent considers the value of a prospective reward, **it should value a reward that can be attained immediately** more highly than an equivalent reward that would require more timesteps to attain.
  - 100 points for cherries that are only one pixel's distance away x 100 points for for cherries that are a distance of 20 pixels away.

Markov Decision Processes

- **Immediate reward is more valuable than some distant reward.**
- We don't know what can happen before reaching the distant reward: a ghost or some other hazard could get in Pac-Man's way.
- For $\gamma = 0.9 \rightarrow$ cherries one timestep away would be considered to be worth 90 points.
  - $100 \times \gamma^t = 100 \times 0.9^1 = 90$.
- For $\gamma = 0.9 \rightarrow$ cherries 20 timesteps away would be considered to be worth only 12.2 points.
  - $100 \times \gamma^t = 100 \times 0.9^{20} = 12.2$.

Markov Decision Processes



Based on the discount factor γ, in a Markov decision process more-distant reward is discounted relative to reward that's more immediately attainable. Using the Atari game Pac-Man to illustrate this concept (a green trilobite sitting in for Mr. Pac-Man himself), with $\gamma = 0.9$, cherries (or a fish!) only one timestep away are valued at 90 points, whereas cherries (a fish) 20 timesteps away are valued at 12.2 points.

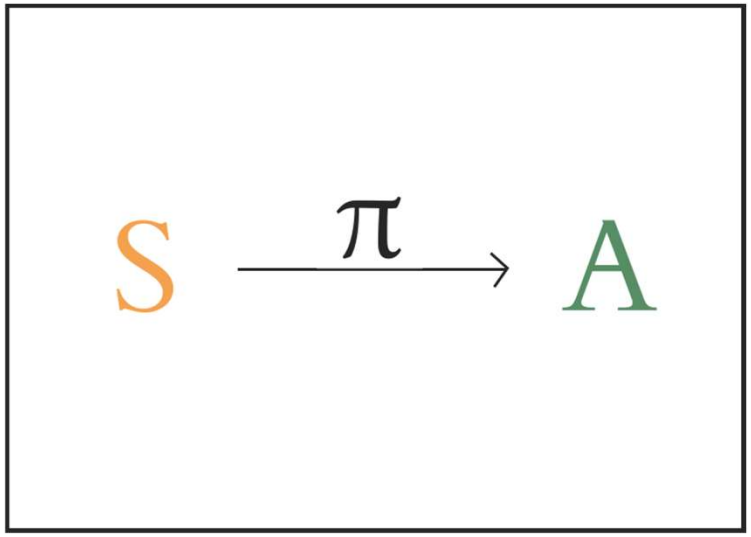
The Optimal Policy

- The ultimate objective with an MDP is to **find a function** that enables an agent to take an appropriate action **a** (from the set of all possible actions **A**) when it encounters any particular state **s** from the set of all possible environmental states **S**.

- In other words, we'd like our agent to learn a function that enables it to map **S** to **A**.

- Such a function is denoted by π and we call it the **policy function**.

The Optimal Policy



The policy function π enables an agent to map any state s (from the set of all possible states S) to an action a from the set of all possible actions A.

The Optimal Policy

- The high-level idea of the policy function π is this:

- Regardless of the particular circumstance the agent finds itself in, what is the **policy** it should follow that will enable it to maximize its reward?

$$J(\pi^*) = \max_{\pi} J(\pi) = \max_{\pi} \mathbb{E}\left[\sum_{t>0} \gamma^t \boldsymbol{r}_t\right]$$

The Optimal Policy

- $J(\pi)$ is called an objective function.
  - We can then apply machine learning/optimization techniques to maximize reward.
  - Our objective is to maximize reward → **Gradient Ascent**.
- π representes **any** policy function that maps **S** to **A**.
- π^* represents an optimal policy for mapping **S** to **A**.
  - $\pi^*(s)$ will return an action **a** that will lead to the agent attaining the **maximum** possible **discounted future reward.**

The Optimal Policy

- **Expected Discounted Future Reward** is defined by

$$\mathbb{E}\left[\sum_{t>0} \gamma^t r_t\right]$$

- $\sum_{\{t>0\}} \gamma^t r_t$ is the discounted future reward over all future timesteps (i.e., t > 0).
  - We multiply the reward that can be attained in any given future timestep (r_t) by the discount factor of that timestep (γ^t).
  - **One shall note that rewards are penalized.**

Essential Theory of Deep Q-Learning Networks

- We defined Reinforcement Learning as a **Markov Decision Process**:
  - We'd like our agent, when it encounters any given state **s** at any given timestep ***t*** to follow some optimal policy ***π****.*
  - This policy will enable it to select an action **a** that maximizes the discounted future reward it can obtain.

- Even with a simple problem (link Cart-Pole): **it is computationally intratable (or, at least, extremely computationally ineficiente)** to definitively calculate the maximum cumulative discounted future reward.

Essential Theory of Deep Q-Learning Networks

- There are way **too many possible future outcomes** to take into consideration.
  - Because of the combination of all possible future states **S** and all the possible actions **A** that could be taken in those future states.

- As a computational shortcut, we'll see the **Q-learning** approach for **estimating** what the optimal action **a** in a given situation might be.

Value Functions

- **Value functions** are a fundamental concept of Q-Learning.
- The value function is defined by $V^{\pi}(s)$:

$$V^{\pi}(s) = E_{\{s_0=s,\, \tau \sim \pi\}}[\sum_{t=0}^{T} \gamma_t r^t]$$

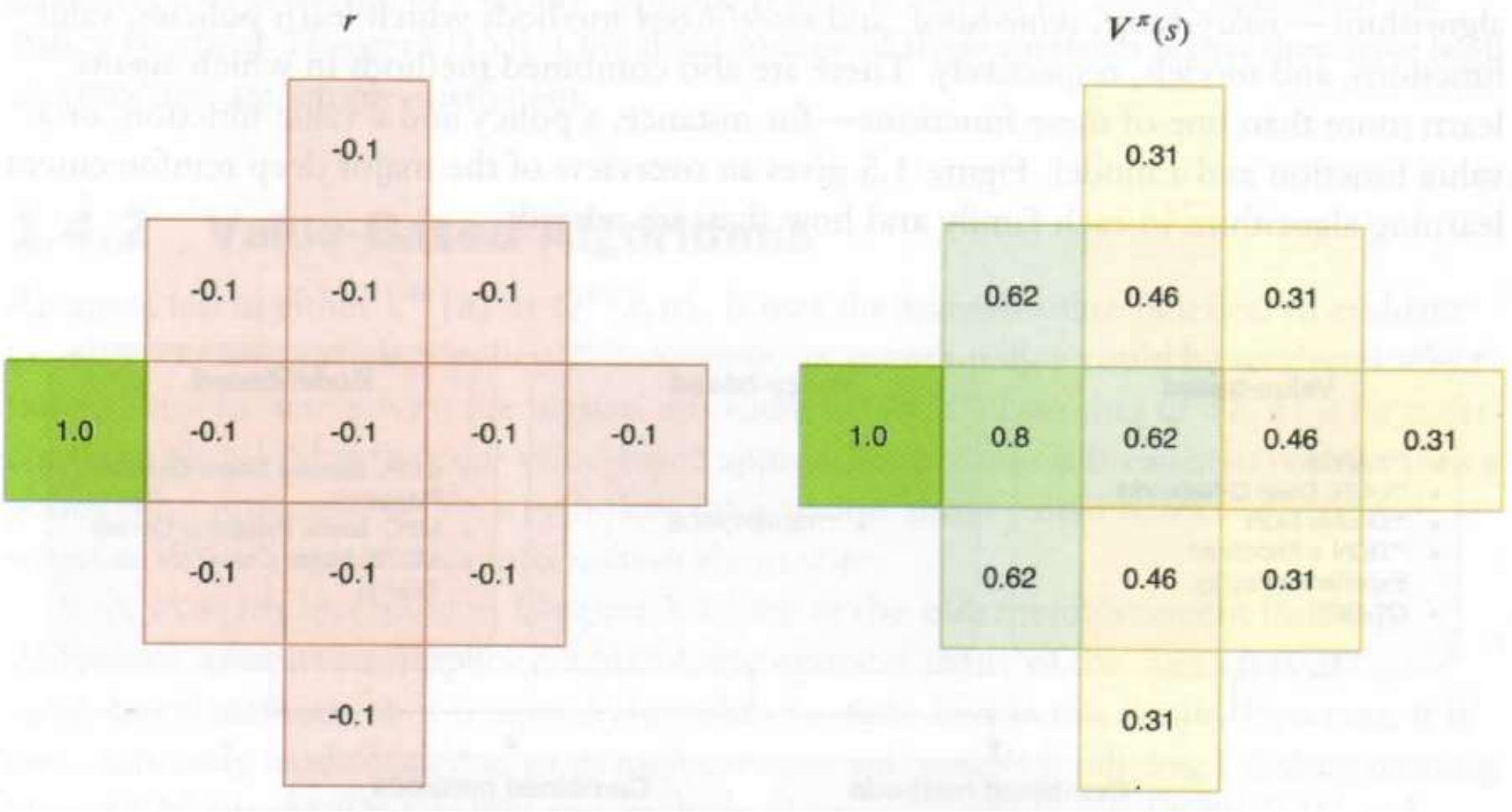
- It provides with an indication of how valuable a given state **s** is if our agent follows its policy from that state onward.

Value Functions

- We should also define the Q function.
- The value function is defined by $V^{\pi}(s)$.
- It provides with an indication of how valuable a given state **s**

$$Q^{\pi}(s, a) = E_{\{s_0=s,\, a_0=a, \tau\sim\pi\}}[\sum_{t=0}^{T} \gamma_t r^t]$$

Value Functions



Rewards **r** (left) and values V^{π}(**s**) (right) for each state **s** in a simple grid-world environment. The value of a state is calculated from the rewards with **γ** = 0.9 while using a policy **π** that always takes the shortest path to the goal state with **r** = +1.

Defining a DQN Agent

- Our code for defining a DQN agent that learns how to act in an environment - the Cart-Pole game - is provided within our *Cartpole DQN* Jupyter notebook.

- The most significant new addition to the list is `gym`, the Open AI Gym itself.

Deep Reinforcement Learning Algorithms

- In Reinforcement Learning, an **agent learns functions** to help it act and maximize the objective.

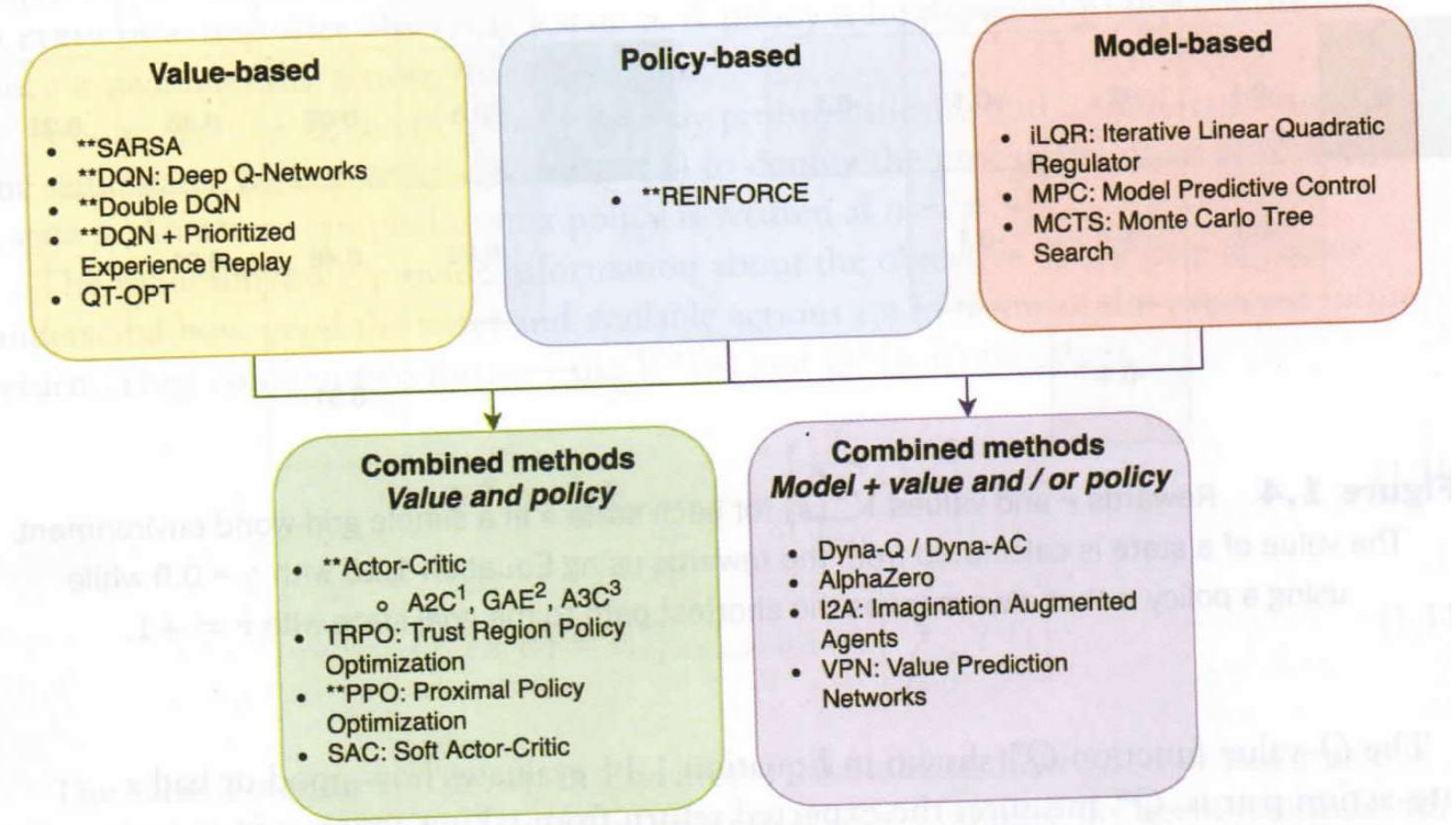
- We saw the three pimary learnable functions in reinforcement learning:
  - Policy $\boldsymbol{\pi}$;
  - Value functions $V^{\pi}(\mathbf{s})$ or $Q^{\pi}(\mathbf{s}, \mathbf{a})$; and
  - Environment model $\mathbf{P}(\mathbf{s'} \mid \mathbf{s}, \mathbf{a})$.

Deep Reinforcement Learning Algorithms

- Correspondigly, there are three major families of deep reinforcement learning algorithms:
  - **Policy-based**: learn policies;
  - **Value-based**: learn value functions; and
  - **Model-based**: learn models.

- There are also combined methods: agents learn **more than one** of these functions.
  - For instance, a policy and a value function, or a value function and a model.

Deep Reinforcement Learning Algorithms



**: discussed in this book
1. A2C: Advantage Actor-Critic
2. A3C: Asynchronous Advantage Actor-Critic
3. GAE: Actor-Critic with Generalized Advantage Estimation

LeCun's View of Reinforcement Learning Limitations

- The interest of Reinforcement Learning (RL) is to be able to train systems whose performance we can evaluate **without giving them the correct answer**.

- It is mostly used when the system needs to **produce actions**, for example, control a robot or play a game.

- We saw the stunning success of RL in the domain of games with AlphaGo from DeepMind, AlphaZero, its sucessor, and Elf OpenGo from Facebook.

LeCun's View of Reinforcement Learning Limitations

- Unfortunately, this learning paradigm, in its most famous form **needs a gigantic number of interactions (trial and error) to learn even simple tasks.**

LeCun's View of Reinforcement Learning Limitations

- Like supervised learning, this kind of learning requires **a lot of resources and many interactions to achieve a superhuman performance**.
- DeepMind trained a system to play classic Atari games (there are 80).
- To reach a correct level of play, it was necessary to spend the equivalent of **80 hours of training per match**.
  - A human needed only 15 minutes of play to achieve the same level.

LeCun's View of Reinforcement Learning Limitations

- However, **if we let the system train much longer, it reachs levels beyond human possibilities**.
- In fact, these 80 hours is the time taken by the machine if we let it play the game in real time.
- **But it can play the game much faster.**
  - And it can even play simultaneous matches.
- But this can only be done to games.
- **There's no way of making the clock tick faster when training a car to drive on the road.**

LeCun's View of Reinforcement Learning Limitations

- In real world, Reinforcement Learning, in its purest form is properly inapplicable.
- Suppose we want to make a vehicle learn to drive autonomously.
- It would need to drive for million of hours.
- **… and also would need to cause tens of thousands of accidents before learning to avoid it.**

LeCun's View of Reinforcement Learning Limitations

- If the vehicle falls from a Cliff, the system would say: "Ah, I must have made a mistake."
  - And then it would correct a little its a strategy.
- A second time, it would fall from the same cliff, maybe in a little diferente way.
  - And it would again correct itself a little.
- **The car would have to fall thousands of times before the system figured out how to avoid the fall.**
  - This is infeasible in real world!

LeCun's View of Reinforcement Learning Limitations

- What allows the majority of humans to learn to drive in a **few practical classes** and **with little supervision**?
  - Without even causing acidentes (for most people)...
- Supervised Learning and Reinforcement Learning are not enough.
- In his opinion, it remains to **invent a new paradigm** so that the **machine can equal human or animal learning**.

LeCun's View of Reinforcement Learning Limitations

- We could think of using the **simulation**.
- Another problem rises then: simulators should be sufficiently powerful and accurate → they should reflect precisely what happens in the reality.
- **In this way, once the model was trained in the simulator, we can transposed its capacity to the real world.**
- This not always possible!
- This problem **sim2real** (simulation to real world) is a rechercher domain very active nowadays.

LeCun's View of Reinforcement Learning Limitations

- A part of the scientifique community believes that this form of learning will be the **key to designing a human level AI**.
- David Silver, from AlphaGo, says that "Reinforcement learning is the very essence of intelligence".
- **LeCun says that he felt obligated to be the pessimist one:** he invoked the black forest cake (*forêt-noire*).
  - It's a chocolat cake made of alternating layers of sponge cake and cream, very imposing, coated with icing and often topped with a candied cherry.

LeCun's View of Reinforcement Learning Limitations

- In the conference, he said that if the intelligence is a black forest cake, then:
  - The sponge cake part represents the **self-supervised learning**, which is the principal learning mode for an animal and a human;
  - The icing part corresponds to the **supervised learning**;
  - The cherry on top of the cake is... **reinforcement learning**.

Referências

1. Capítulo 13 do livro *"Deep Learning Illustrated: A Visual, Interactive Guide to Artificial Intelligence"* de Jon Krohn, Grant Beyleveld e Aglaé Bassens.

2. Capítulo 3 do livro *"Neural Networks and Deep Learning: A Textbook"* de Charu C. Aggarwal.

3. Capítulo 9 do livro *"Quand la machine apprend: La révolution deus neurones artificiels et de l'apprentissage profond"* de Yann Le Cun.