Bertinoro International Spring School 2026

Introduction to Reinforcement Learning

Mirco Musolesi

mircomusolesi@acm.org

Introduction to Reinforcement Learning

- Key idea: a natural way of thinking about learning is learning through interaction with the external world.
- Learning from interaction is a foundational idea underlying nearly all theories of learning and intelligence.
- Reinforcement learning is learning what to do - how to map situations to actions - so as to maximise a numerical reward.
  - *Goal-directed* learning from interaction.
- The learner is not told which actions to take, but instead it must discover which actions yield the most reward by trying them.

Mirco Musolesi

VOL. LIX. NO. 236.] [October, 1950

MIND

A QUARTERLY REVIEW

OF

PSYCHOLOGY AND PHILOSOPHY

I.—COMPUTING MACHINERY AND INTELLIGENCE

BY A. M. TURING

1. *The Imitation Game.*

I PROPOSE to consider the question, 'Can machines think?' This should begin with definitions of the meaning of the terms 'machine' and 'think'. The definitions might be framed so as to reflect so far as possible the normal use of the words, but this attitude is dangerous. If the meaning of the words 'machine' and 'think' are to be found by examining how they are commonly used it is difficult to escape the conclusion that the meaning and the answer to the question, 'Can machines think?' is to be sought in a statistical survey such as a Gallup poll. But this is absurd. Instead of attempting such a definition I shall replace the question by another, which is closely related to it and is expressed in relatively unambiguous words.

The new form of the problem can be described in terms of a game which we call the 'imitation game'. It is played with three people, a man (A), a woman (B), and an interrogator (C) who may be of either sex. The interrogator stays in a room apart from the other two. The object of the game for the interrogator is to determine which of the other two is the man and which is the woman. He knows them by labels X and Y, and at the end of the game he says either 'X is A and Y is B' or 'X is B and Y is A'. The interrogator is allowed to put questions to A and B thus:

C: Will X please tell me the length of his or her hair?

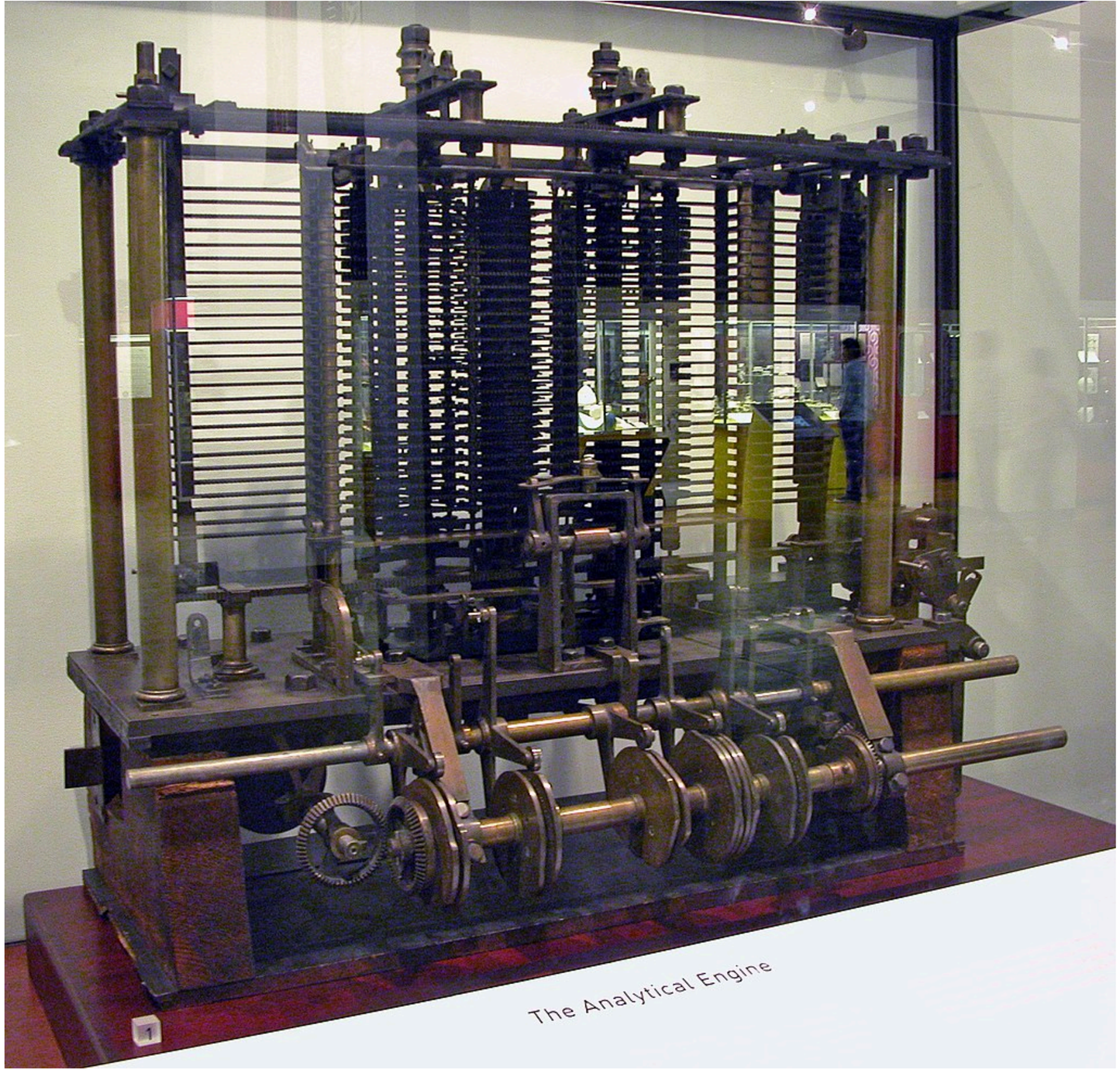
course only a finite part can have been used at any one time. Likewise only a finite amount can have been constructed, but we can imagine more and more being added as required. Such computers have special theoretical interest and will be called infinitive capacity computers.

The idea of a digital computer is an old one. Charles Babbage, Lucasian Professor of Mathematics at Cambridge from 1828 to 1839, planned such a machine, called the Analytical Engine, but it was never completed. Although Babbage had all the essential ideas, his machine was not at that time such a very attractive prospect. The speed which would have been available would be definitely faster than a human computer but something like 100 times slower than the Manchester machine, itself one of the slower of the modem machines. The storage was to be purely mechanical, using wheels and cards.

The fact that Babbage's Analytical Engine was to be entirely mechanical will help us to rid ourselves of a superstition. Importance is often attached to the fact that modem digital computers are electrical, and that the nervous system also is electrical. Since Babbage's machine was not electrical, and since all digital computers are in a sense equivalent, we see that this use of electricity cannot be of theoretical importance. Of course electricity usually comes in where fast signalling is concerned, so that it is not surprising that we find it in both these connections. In the nervous system chemical phenomena are at least as important as electrical. In certain computers the storage system is mainly acoustic. The feature of using electricity is thus seen to be only a very superficial similarity. If we wish to find such similarities we should look rather for mathematical analogies of function.

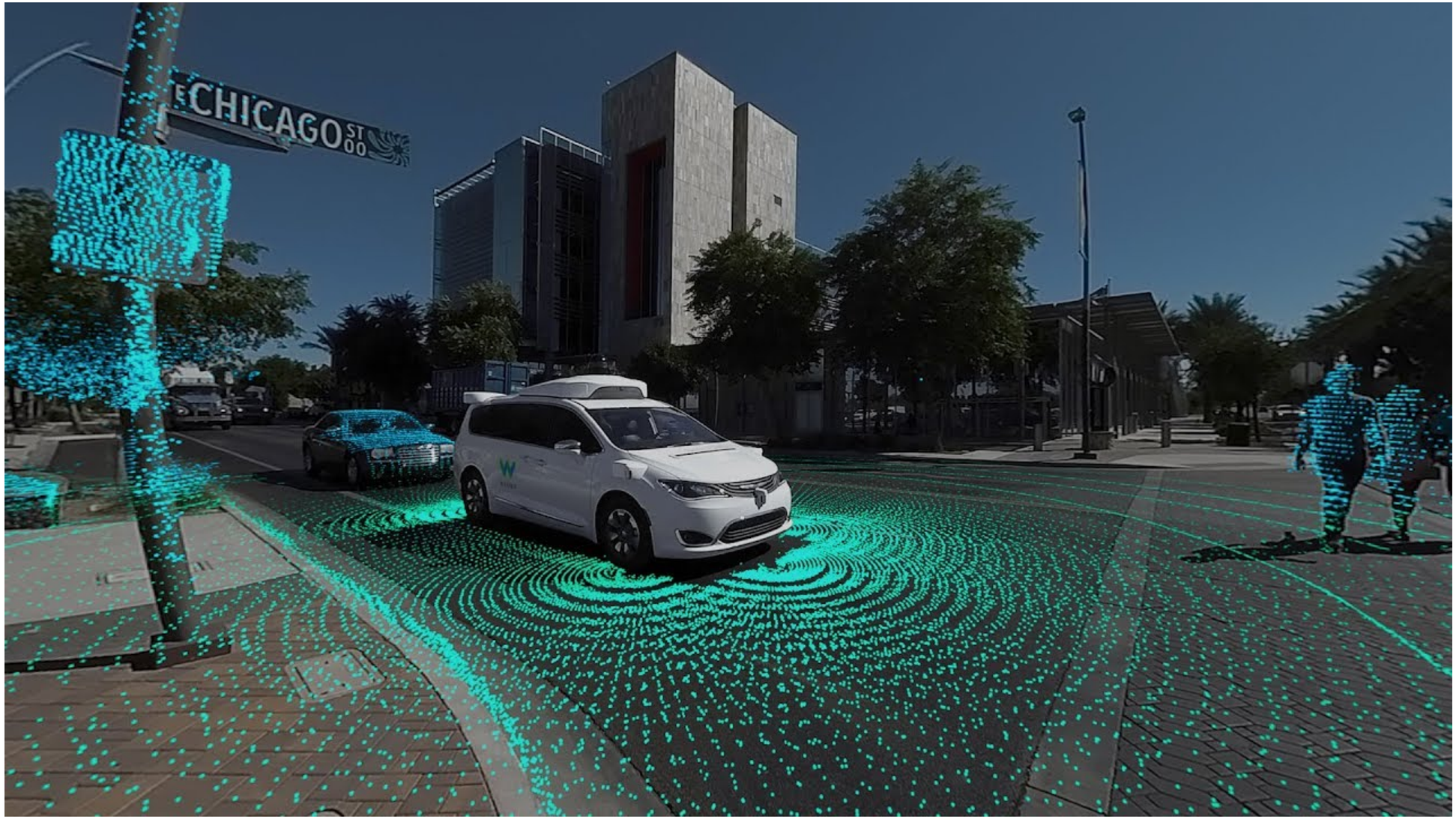
5. *Universality of Digital Computers.*

The digital computers considered in the last section may be classified amongst the 'discrete state machines'. These are the machines which move by sudden jumps or clicks from one quite definite state to another. These states are sufficiently different for the possibility of confusion between them to be ignored. Strictly speaking there are no such machines. Everything really moves continuously. But there are many kinds of machine which can profitably be *thought of* as being discrete state machines. For instance in considering the switches for a lighting system it is a convenient fiction that each switch must be definitely on or definitely off. There must be intermediate positions, but for most purposes we can forget about them. As an example of a discrete state machine we might



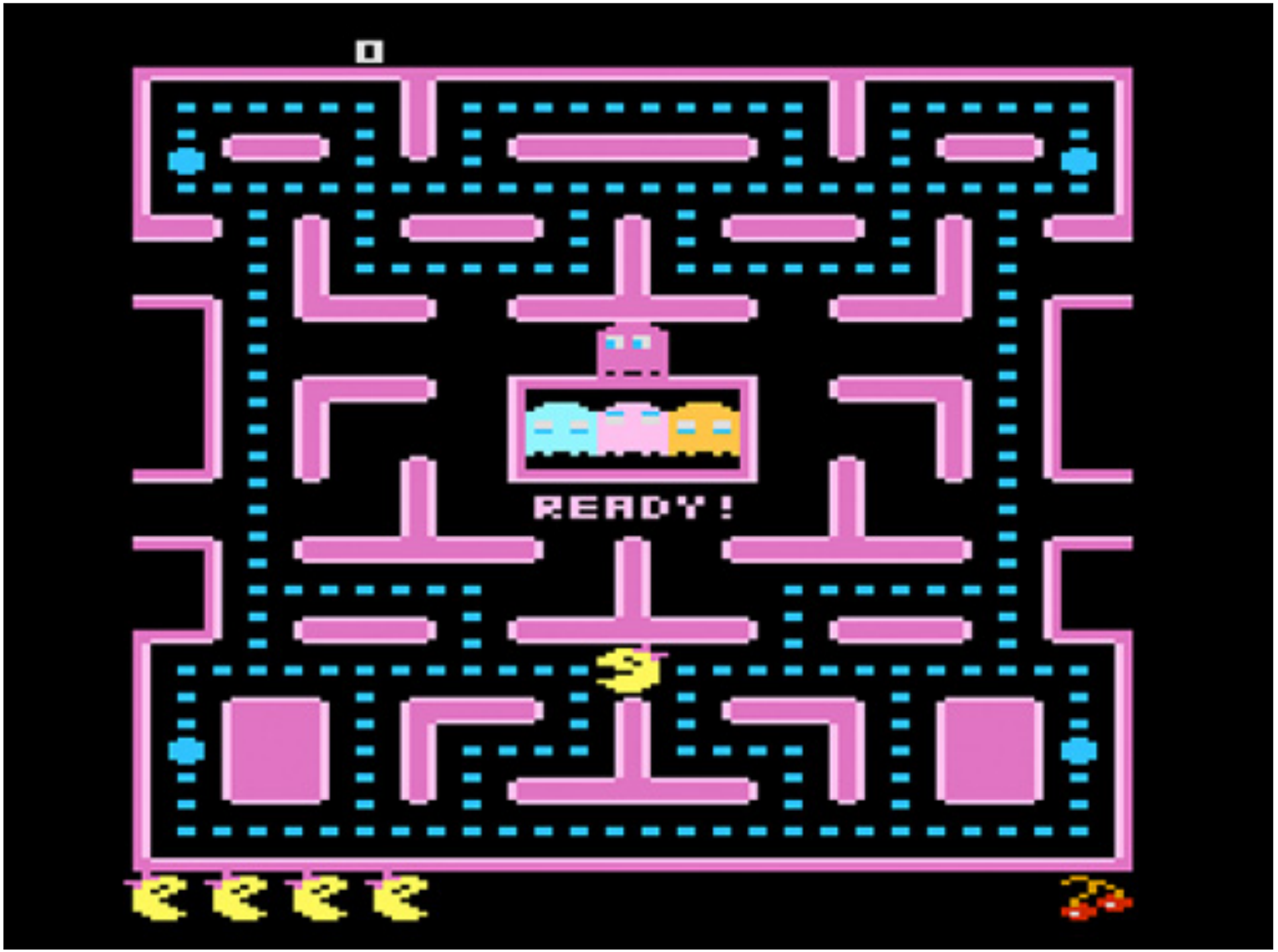
Credit: Wikipedia


WAYMO



https://www.youtube.com/watch?v=B8R148hFxPw


WAYVE


0
READY!

Playing Atari with Deep Reinforcement Learning

Volodymyr Mnih **Koray Kavukcuoglu** **David Silver** **Alex Graves** **Ioannis Antonoglou**

Daan Wierstra **Martin Riedmiller**

DeepMind Technologies

{vlad,koray,david,alex.graves,ioannis,daan,martin.riedmiller} @ deepmind.com

Abstract

We present the first deep learning model to successfully learn control policies directly from high-dimensional sensory input using reinforcement learning. The model is a convolutional neural network, trained with a variant of Q-learning, whose input is raw pixels and whose output is a value function estimating future rewards. We apply our method to seven Atari 2600 games from the Arcade Learn

ARTICLE

doi:10.1038/nature16961

Mastering the game of Go with deep neural networks and tree search

David Silver[1]*, Aja Huang[1]*, Chris J. Maddison[1], Arthur Guez[1], Laurent Sifre[1], George van den Driessche[1], Julian Schrittwieser[1], Ioannis Antonoglou[1], Veda Panneershelvam[1], Marc Lanctot[1], Sander Dieleman[1], Dominik Grewe[1], John Nham[2], Nal Kalchbrenner[1], Ilya Sutskever[2], Timothy Lillicrap[1], Madeleine Leach[1], Koray Kavukcuoglu[1], Thore Graepel[1] & Demis Hassabis[1]

The game of Go has long been viewed as the most challenging of classic games for artificial intelligence owing to its enormous search space and the difficulty of evaluating board positions and moves. Here we introduce a new approach to computer Go that uses 'value networks' to evaluate board positions and 'policy networks' to select moves. These deep neural networks are trained by a novel combination of supervised learning from human expert games, and reinforcement learning from games of self-play. Without any lookahead search, the neural networks play Go at the level of state-

ARTICLE

doi:10.1038/nature24270

Mastering the game of Go without human knowledge

David Silver[1]*, Julian Schrittwieser[1]*, Karen Simonyan[1]*, Ioannis Antonoglou[1], Aja Huang[1], Arthur Guez[1], Thomas Hubert[1], Lucas Baker[1], Matthew Lai[1], Adrian Bolton[1], Yutian Chen[1], Timothy Lillicrap[1], Fan Hui[1], Laurent Sifre[1], George van den Driessche[1], Thore Graepel[1] & Demis Hassabis[1]

A long-standing goal of artificial intelligence is an algorithm that learns, *tabula rasa*, superhuman proficiency in challenging domains. Recently, AlphaGo became the first program to defeat a world champion in the game of Go. The tree search in AlphaGo evaluated positions and selected moves using deep neural networks. These neural networks were trained by supervised learning from human expert moves, and by reinforcement learning from self-play. Here we introduce an algorithm based solely on reinforcement learning, without human data, guidance or domain knowledge beyond game rules. AlphaGo becomes its own teacher: a neural network is trained to predict AlphaGo's own move selections and also the winner of AlphaGo's games. This neural network improves the strength of the tree search, resulting in higher quality

https://www.youtube.com/watch?v=6EQAsrfUlyo

RESEARCH

COMPUTER SCIENCE

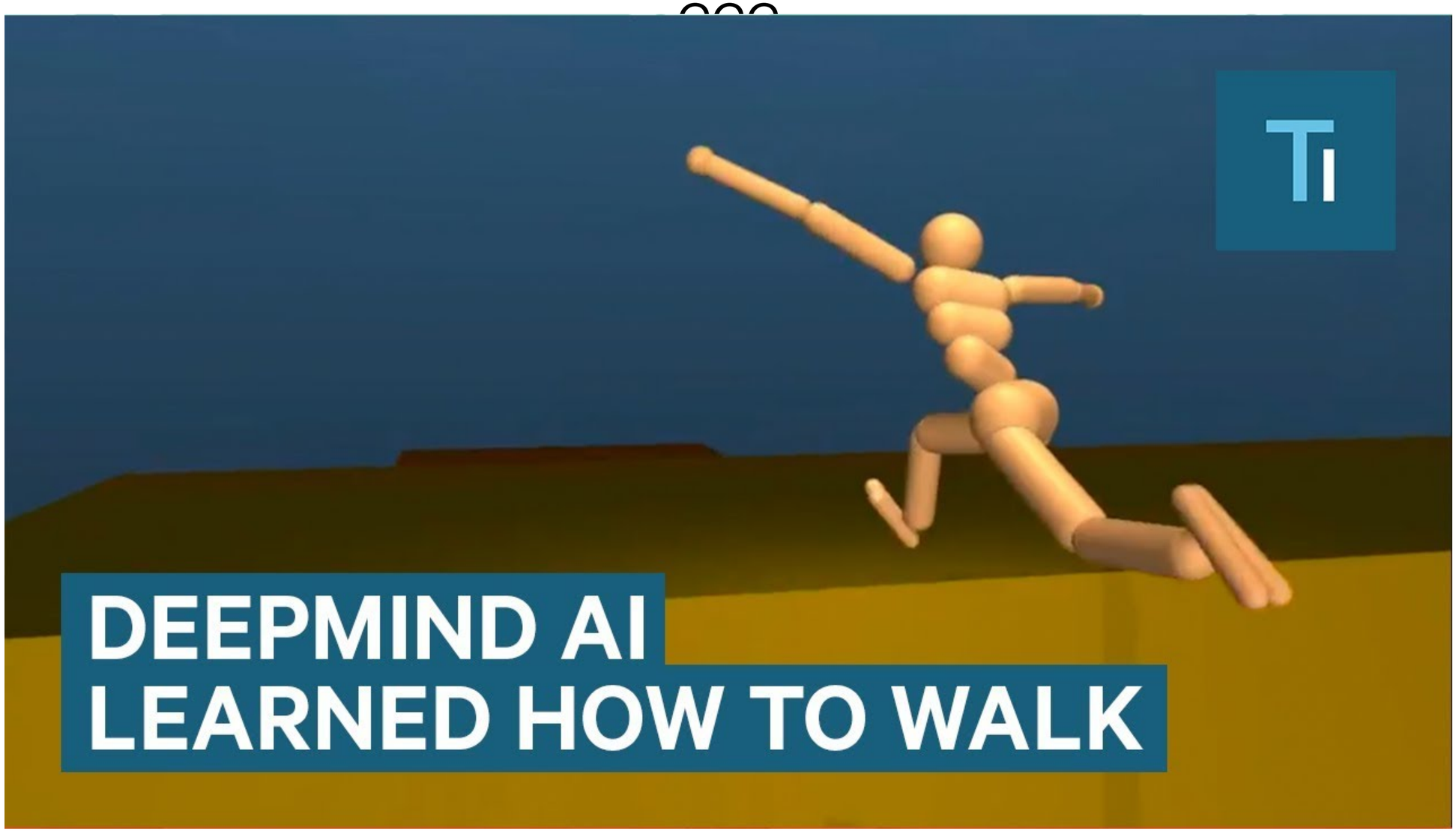
DeepStack: Expert-level artificial intelligence in heads-up no-limit poker

Matej Moravčík,[1,2]* Martin Schmid,[1,2]* Neil Burch,[1] Viliam Lisý,[1,3] Dustin Morrill,[1] Nolan Bard,[1] Trevor Davis,[1] Kevin Waugh,[1] Michael Johanson,[1] Michael Bowling[1]†

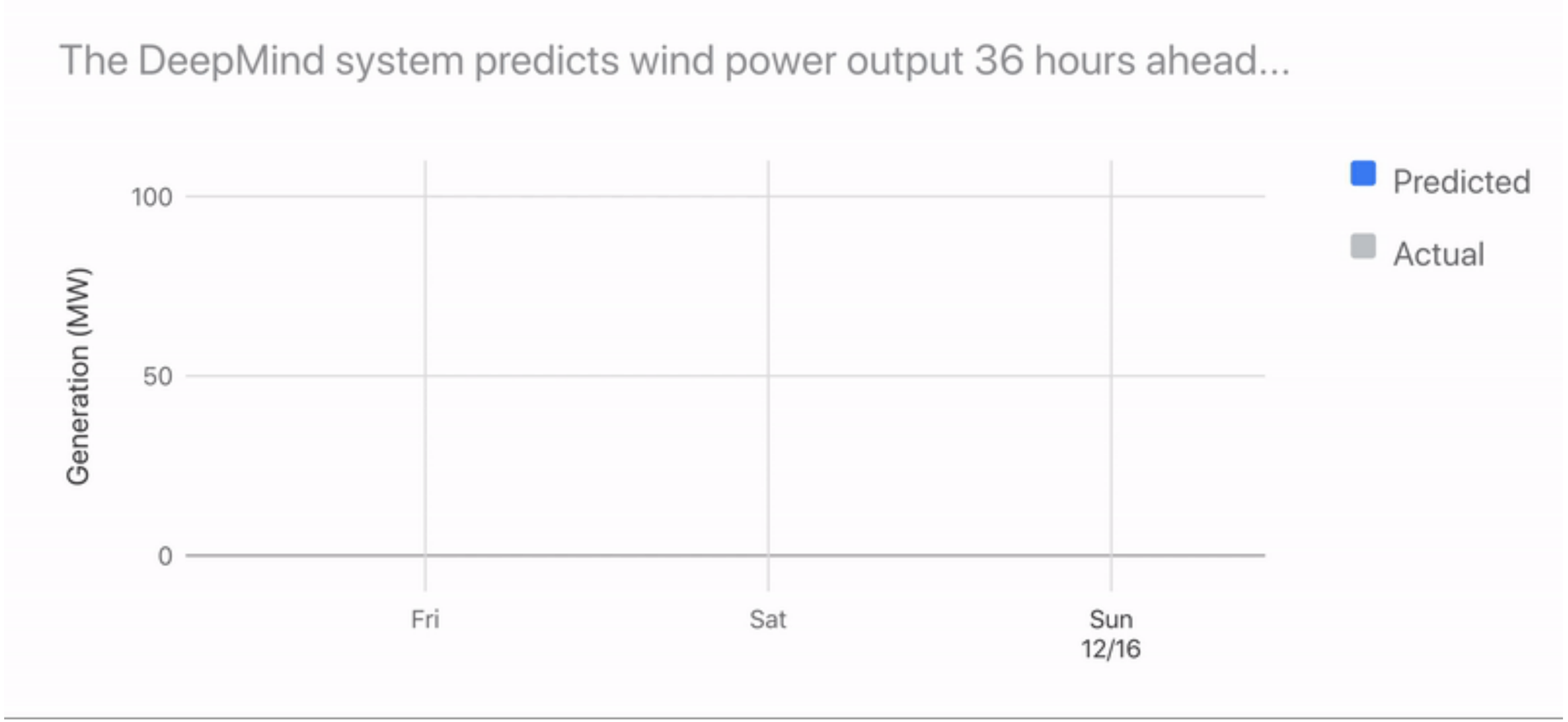
Artificial intelligence has seen several breakthroughs in recent years, with games often serving as milestones. A common feature of these games is that players have perfect information. Poker, the quintessential game of imperfect information, is a long-standing challenge problem in artificial intelligence. We introduce DeepStack, an algorithm for imperfect-information settings. It combines recursive reasoning to handle information asymmetry, decomposition to focus computation on the relevant decision, and a form of intuition that is automatically learned from self-play using deep learning. In a study involving 44,000 hands of poker, DeepStack defeated, with statistical significance, professional poker players in heads-up no-limit Texas hold'em. The approach is theoretically sound and is shown to produce strategies that are more difficult to exploit than prior approaches.

Likely as a result of this loss of information, such programs are behind expert human play. In 2015, the computer program Claudico lost to a team of professional poker players by a margin of 91 milli–big blinds per game (mbb/g) (*19*), which is a "huge margin of victory" (*20*). Furthermore, it was recently shown that abstraction-based programs from the Annual Computer Poker Competition have massive flaws (*21*). Four such programs (including top programs from the 2016 competition) were evaluated using a local best response technique that produces an approximate lower bound on how much a strategy can lose. All four abstraction-based programs are beatable by more than 3000 mbb/g, whereas simply folding each game results in 750 mbb/g.

DeepStack takes a fundamentally different approach. It continues to use the recursive reasoning of CFR to handle information asymmetry. However, it does not compute and store a complete strategy prior to play and so has no need for explicit abstraction. Instead, it considers each particular situation as it arises during play, but not in isolation. It avoids reasoning about the


DEEPMIND AI
LEARNED HOW TO WALK

Mirco Musolesi



Source: https://deepmind.com/blog/article/machine-learning-can-boost-value-wind-energy

RL in Android

- RL used in Android for:
  - Adaptive battery:
    - It is used to learn and anticipate future battery use
  - Adaptive brightness of the video:
    - Algorithm learns preferences in terms of brightness from the user

The Brave New World

Sparks of Artificial General Intelligence: Early experiments with GPT-4

Sébastien Bubeck Varun Chandrasekaran Ronen Eldan Johannes Gehrke
Eric Horvitz Ece Kamar Peter Lee Yin Tat Lee Yuanzhi Li Scott Lundberg
Harsha Nori Hamid Palangi Marco Tulio Ribeiro Yi Zhang

Microsoft Research

Abstract

Artificial intelligence (AI) researchers have been developing and refining large language models (LLMs) that exhibit remarkable capabilities across a variety of domains and tasks, challenging our understanding of learning and cognition. The latest model developed by OpenAI, GPT-4 [Ope23], was trained using an unprecedented scale of compute and data. In this paper, we report on our investigation of an early version of GPT-4, when it was still in active development by OpenAI. We contend that (this early version of) GPT-4 is part of a new cohort of LLMs (along with ChatGPT and Google's PaLM for example) that exhibit more general intelligence than previous AI models. We discuss the rising capabilities and implications of these models. We demonstrate that, beyond its mastery of language, GPT-4 can solve novel and difficult tasks that span mathematics, coding, vision, medicine, law, psychology and more, without needing any special prompting. Moreover, in all of these tasks, GPT-4's performance is strikingly close to human-level performance, and often vastly surpasses prior models such as ChatGPT. Given the breadth and depth of GPT-4's capabilities, we believe that it could reasonably be viewed as an early (yet still incomplete) version of an artificial general intelligence (AGI) system. In our exploration of GPT-4, we put special emphasis on discovering its limitations, and we discuss the challenges ahead for advancing towards deeper and more comprehensive versions of AGI, including the possible need for pursuing a new paradigm that moves beyond next-word prediction. We conclude with reflections on societal influences of the recent technological leap and future research directions.

Reinforcement Learning for Generative AI

ChatGPT

Llama 4

Gemini

ANTHROP\C

Talk to Claude

Deep Reinforcement Learning from Human Preferences

Paul F Christiano
OpenAI
paul@openai.com

Jan Leike
DeepMind
leike@google.com

Tom B Brown
Google Brain*
tombbrown@google.com

Miljan Martic
DeepMind
miljanm@google.com

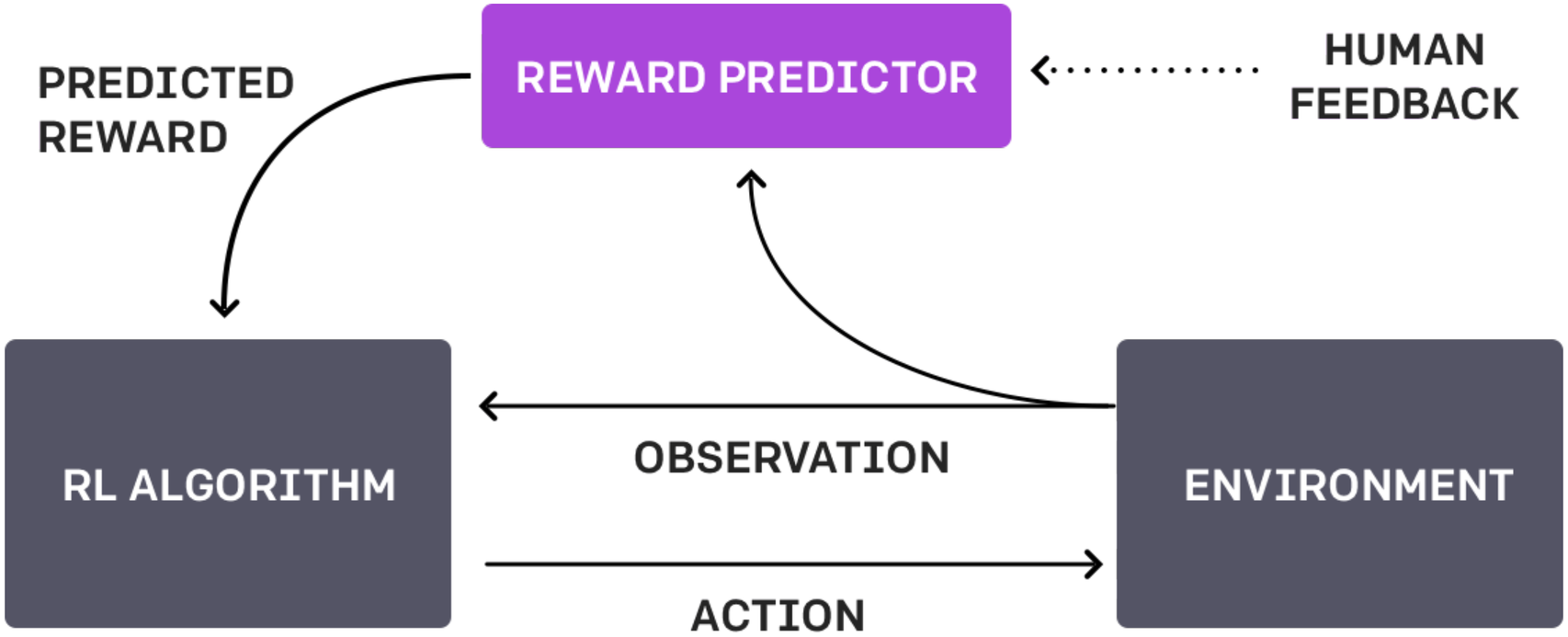
Shane Legg
DeepMind
legg@google.com

Dario Amodei
OpenAI
damodei@openai.com

Abstract

For sophisticated reinforcement learning (RL) systems to interact usefully with real-world environments, we need to communicate complex goals to these systems. In this work, we explore goals defined in terms of (non-expert) human preferences between pairs of trajectory segments. We show that this approach can effectively solve complex RL tasks without access to the reward function, including Atari games and simulated robot locomotion, while providing feedback on less than 1% of our agent's interactions with the environment. This reduces the cost of human oversight far enough that it can be practically applied to state-of-the-art RL systems. To demonstrate the flexibility of our approach, we show that we can successfully train complex novel behaviors with about an hour of human time. These behaviors and environments are considerably more complex than any which have been previously learned from human feedback.

Reinforcement Learning from Human Feedback (RLHF)



Source: OpenAI

Reinforcement Learning for Generative AI: State of the Art, Opportunities and Open Research Challenges

Giorgio Franceschelli GIORGIO.FRANCESCHELLI@UNIBO.IT
Department of Computer Science and Engineering, University of Bologna, Bologna, Italy

Mirco Musolesi M.MUSOLESI@UCL.AC.UK
Department of Computer Science, University College London, London, United Kingdom
Department of Computer Science and Engineering, University of Bologna, Bologna, Italy

Abstract

Generative Artificial Intelligence (AI) is one of the most exciting developments in Computer Science of the last decade. At the same time, Reinforcement Learning (RL) has emerged as a very successful paradigm for a variety of machine learning tasks. In this survey, we discuss the state of the art, opportunities and open research questions in applying RL to generative AI. In particular, we will discuss three types of applications, namely, RL as an alternative way for generation without specified objectives; as a way for generating outputs while concurrently maximizing an objective function; and, finally, as a way of embedding desired characteristics, which cannot be easily captured by means of an objective function, into the generative process. We conclude the survey with an in-depth discussion of the opportunities and challenges in this fascinating emerging area.

“Generative” Agents

Generative Agents: Interactive Simulacra of Human Behavior

Joon Sung Park
Stanford University
Stanford, USA
joonspk@stanford.edu

Joseph C. O’Brien
Stanford University
Stanford, USA
jobrien3@stanford.edu

Carrie J. Cai
Google Research
Mountain View, CA, USA
cjcai@google.com

Meredith Ringel Morris
Google DeepMind
Seattle, WA, USA
merrie@google.com

Percy Liang
Stanford University
Stanford, USA
pliang@cs.stanford.edu

Michael S. Bernstein
Stanford University
Stanford, USA
msb@cs.stanford.edu

Figure 1: Generative agents are believable simulacra of human behavior for interactive applications. In this work, we demonstrate generative agents by populating a sandbox environment, reminiscent of The Sims, with twenty-five agents. Users can observe and intervene as agents plan their days, share news, form relationships, and coordinate group activities.

ABSTRACT

Believable proxies of human behavior can empower interactive applications ranging from immersive environments to rehearsal spaces for interpersonal communication to prototyping tools. In

authors write; they form opinions, notice each other, and initiate conversations; they remember and reflect on days past as they plan the next day. To enable generative agents, we describe an architecture that extends a large language model to store a complete record of the agent’s experiences using natural language, synthesize those

2025 Top 10 Strategic Technology Trends

Source: Gartner

Gartner

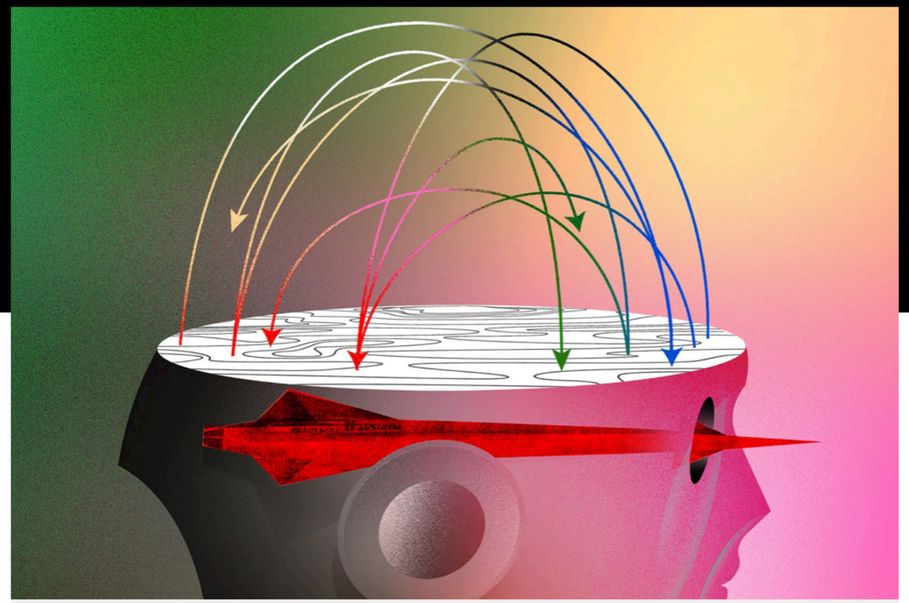
ANALYSIS

AI Has Entered the Situation Room

Data lets us see with unprecedented clarity—but reaping its benefits requires changing how foreign policy is made.

JUNE 19, 2023, 11:00 PM

By Stanley McChrystal, a retired four-star U.S. Army general and an advisor to Rhombus Power, and Anshu Roy, the founder and CEO of Rhombus Power.



BRIAN STAUFFER ILLUSTRATION FOR FOREIGN POLICY

Examples of Problems

Credit: DepositPhoto

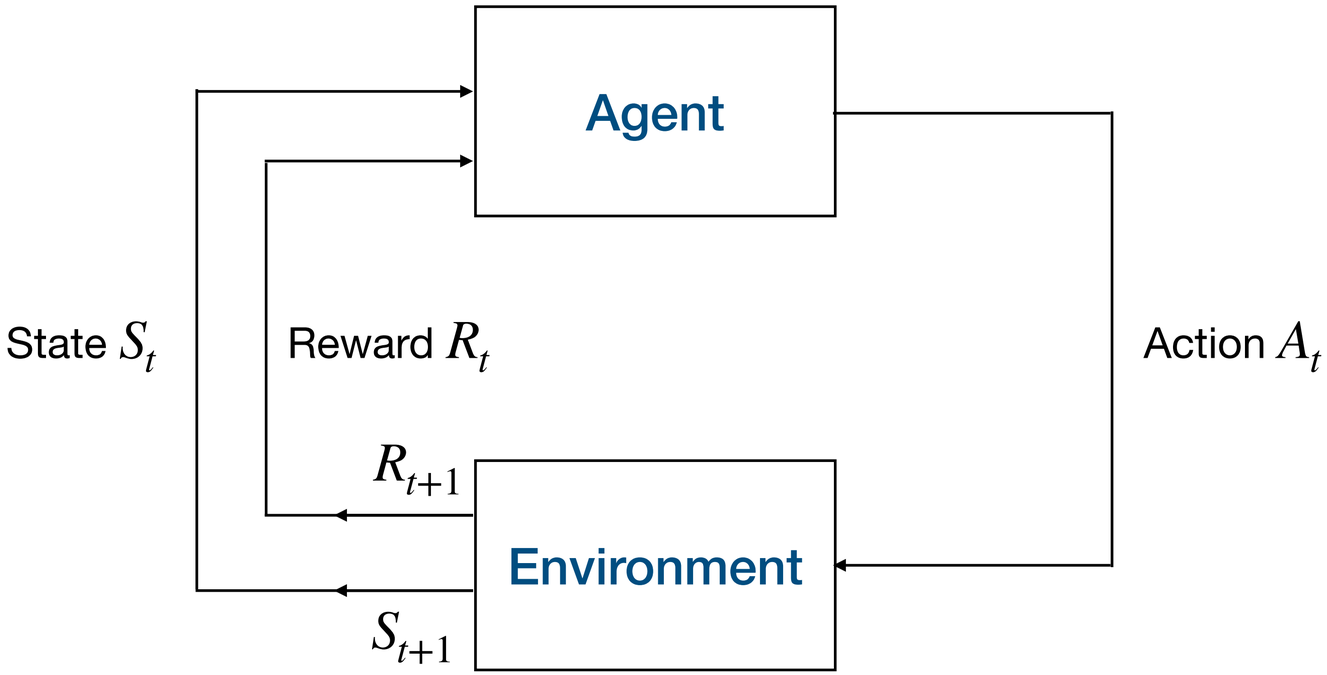
Examples of Problems

Finite Markov Decision Processes

- Markov Decision Processes (MDPs) are a mathematically idealised formulation of Reinforcement Learning for which precise theoretical statements can be made.
  - Tension between breadth of applicability and mathematical tractability.
  - MDPs provide a way for framing the problem of learning from experience, and, more specifically, from interacting with an environment.

Markov Decision Processes: Definitions

- Two entities:
  - **Agent**: learner and decision maker.
  - **Environment**: everything else outside the agent.
- The agent interacts with the environment selecting **actions**.
- The environment changes following actions of the agent.


Agent
State S_t
Reward R_t
Action A_t
R_{t+1}
Environment
S_{t+1}

Markov Decision Processes: Definitions

- The agent and the environment interact at each discrete step of a sequence $t = 0,1,2,3,...$
- At each time step t, the agent receives some representation of the environment **state** $S_t \in \mathcal{S}$ where $\mathcal{S}$ is the set of the states.
- On that basis, an agent selects an **action** $A_t \in \mathcal{A}(S_t)$ where $\mathcal{A}(S_t)$ is the set of the actions that can be taken in state S_t.
- At time $t + 1$ as a consequence of its action the agent receives a **reward** $R_{t+1} \in \mathcal{R}$, where $\mathcal{R}$ is the set of rewards (expressed as real numbers).

Goals and Rewards

- The goal of the agent is formalised in terms of the reward it receives.
- At each time step, the reward is a simple number $R_t \in \mathbb{R}$.
- Informally, the agent's goal is to maximise the total amount it receives.
- The agent should not maximise the immediate reward, but the *cumulative reward*.

The “Reward Hypothesis”

- We can formalise the goal of an agent by stating the “reward hypothesis”:

All of what we mean by goals and purposes can be well thought of as the maximisation of the expected value of the cumulative sum of a received scalar signal (reward).

Expected Returns

- We will now try to conceptualise the idea of **cumulative rewards** more formally.
- An agent receives a sequence of rewards $R_{t+1}, R_{t+2}, R_{t+3}, \ldots$
- In order to define cumulative rewards, we introduce the concept of **expected return** G_t, which is a function of the reward sequence.

Episodic Tasks and Continuing Tasks

- Typically, we identify two cases: episodic tasks and continuing tasks.
- An **episodic task** is one in which we can identify a final step of the sequence of rewards , i.e., in which the interaction between the agent and the environment can be broken into sub-sequences that we call **episodes** (such a play of a game, repeated tasks, etc.).
  - Each episode ends in terminal state after T steps, followed by a reset to a standard starting state or to a sample of a distribution of starting states.
  - The next episode is completely independent from the previous one.
- A **continuing task** is one in which it is not possible to identify a final state (e.g., on-going process control or robots with a long-lifespan).

Expected Return for Episodic Tasks and Continuing Tasks

- In the case of *episodic tasks* the expected return associated to the selection of an action A_t is the sum of rewards defined as follows:

$$G_t \doteq R_{t+1} + R_{t+2} + R_{t+3} + \ldots + R_T$$

- In the case of *continuing tasks* the expected return associated to the selection of an action A_t is defined as follows:

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

where γ is the discount rate, with $0 \leq \gamma \leq 1$.

Why Discounting?

- The definition of expected return that we used for episodic tasks would be problematic for continuing tasks: the expected return of time of termination T would be equal to ∞ in some cases, such as when the reward is equal to 1 at each time step.

- The discount rate determines the present value of future rewards: a reward received k time steps in the future is worth γ^{k-1} what it would be worth if it were received immediately.

Relation between Returns at Successive Time Steps

- Returns at successive time steps are related to each others as follows:

$$
\begin{aligned}
G_t &\doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 R_{t+4} + \dots \\
&= R_{t+1} + \gamma(R_{t+2} + \gamma R_{t+3} + \gamma^2 R_{t+4} + \dots) \\
&= R_{t+1} + \gamma G_{t+1}
\end{aligned}
$$

Policies and Value Functions

- Almost all reinforcement learning algorithms involve estimating value functions, i.e., functions of states (or of state-action pairs) that estimate how good it is for the agent to be in a given state (or how good it is to perform a given action in a given state).

- A policy is used to model the behaviour of the agent based on the previous experience and the rewards (and consequently the expected returns) an agent received in the past.

Definition of Policy

- Formally, a *policy* is a mapping from states to probabilities of each possible action.
- If the agent is following policy π at time t, then $\pi(a \mid s)$ is the probability that $A_t = a$ if $S_t = s$.

Definition of State-Value Function

- The value function of a state s under a policy π, denoted $v_\pi(s)$ is the expected return when starting in s and following π thereafter.

- For MDPs, we can define the *state-value function* v_π for policy π formally as:

$$v_\pi \doteq E_\pi[G_t \,|\, S_t = s] = E_\pi[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \,|\, S_t = s]$$

for all $s \in \mathcal{S}$

where $E_\pi[\,.\,]$ denotes the expected value of a random variable given that the agent follows π and t is any time step. The value of the terminal state is 0.

Definition of Action-Value Function

- Similarly, we define the *action-value function*, i.e., the value of taking action a in state s under a policy π, denoted $q_\pi(s, a)$, as the expected return starting from s, taking the action a, and thereafter following policy π:

$$q_\pi(s, a) \doteq E_\pi[G_t \,|\, S_t = s, A_t = a] = E_\pi[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \,|\, S_t = s, A_t = a]$$

Choosing the Rewards

- When we model a real system as a Reinforcement Learning problem, the hardest problem is to select the right rewards.
- Typically, we use negative values for actions that do not help us in reaching our goal and positive if they do (and sometimes we set the values to 0 if they do not help us in reaching the goal).
- An alternative is to set the values of rewards to a negative number until we reach our goal (and we set the value to 0 when we reach our goal).

Choosing the Rewards

- It is very important to keep in mind that we should not “reward” the intermediate steps or the single actions.
- We are not “teaching” the agent how to execute an intermediate step, but *how to reach the final goal*. If we do so, the agent will learn how to reach the intermediate step, e.g., how to execute a sub-task.
- The reward should tell the agent if the current step is a step forward towards the final goal or not.

Example of Rewards

Credit: Shutterstock

Maze -> Rewards: -1 for no exit 0 for exit

Examples of Rewards

Chess -> Rewards: 1 for victory, -1 for defeat

Choosing the Rewards

- Sometimes it's not possible to know the reward until the end of an episode. The typical example is a board game (chess, go, etc.).
- This is usually called *credit assignment problem,* i.e., the problem of assigning a reward to each step.
- In that case the reward might be assigned at the end of a Montecarlo rollout for example (stochastic estimate of the reward).
- For example if the game is successful we can use +1 as reward for all the steps that leads to the victory (or -1 otherwise).

Steps Toward Artificial Intelligence*

MARVIN MINSKY†, MEMBER, IRE

The work toward attaining "artificial intelligence" is the center of considerable computer research, design, and application. The field is in its starting transient, characterized by many varied and independent efforts. Marvin Minsky has been requested to draw this work together into a coherent summary, supplement it with appropriate explanatory or theoretical noncomputer information, and introduce his assessment of the state-of-the-art. This paper emphasizes the class of activities in which a general purpose computer, complete with a library of basic programs, is further programmed to perform operations leading to ever higher-level information processing functions such as learning and problem solving. This informative article will be of real interest to both the general PROCEEDINGS reader and the computer specialist.—*The Guest Editor*

***Summary*—The problems of heuristic programming—of making computers solve really difficult problems—are divided into five main areas: Search, Pattern-Recognition, Learning, Planning, and Induction.**

find only a few machines (mostly "general-purpose" computers, programmed for the moment to behave according to some specification) doing things that might

Marvin Minsky. Steps Toward Artificial Intelligence. Proceedings of the IRE. Volume 49. Issue 1. January 1961.

Example of Rewards

- In Go or Chess, the reward will be 1 for winning or -1 losing for the terminal state (i.e., the state at time T), but we will know the result of the game only at the end.
- Therefore, the reward can be assigned only at the end of an episode.
- In Go or Chess, we can for example assign 1 or -1 to each step in case of victory or loss at the end of the episode after a Montecarlo playout/rollout.

How to Estimate the State-Value (Action-Value) Functions

- If the behaviour of the MDP is known (i.e., the transitions probabilities between all the states are known), the state function or the action-state function can be estimated by considering all the possible moves.
- This is not possible when:
  - The transitions probabilities are not known.
  - The system is very complex (for example a board game has a very large number of potential game configurations).

How to Estimate the State-Value (Action-Value) Functions: Monte-Carlo Methods

- Alternatively, the state-value function v_π and the action-value function q_π can be estimated through experience.
- One possibility is to keep average values of the actual returns that have followed a certain state (or a certain action) while following a policy π. These values will converge to the actual state-value function v_π and the action-value function q_π asymptotically.
- These methods based on averaging sample returns are referred to as *Monte Carlo methods*.

How to Estimate the State-Value (Action-Value) Functions: Monte-Carlo Methods

- Monte Carlo methods are not appropriate in case the number of states is very large.
- In this case, it is not practical to keep separate averages for each state individually.
- Instead, v_π and q_π are maintained as parametrised functions with the number of parameters << number of states.
- Various function approximators of different complexity are possible.
  - Artificial neural networks are a possible option as function approximators -> Deep Reinforcement Learning

Optimal Policies and Optimal Value Functions

- Solving a reinforcement learning is roughly equivalent to finding a policy that maximises the amount of reward over the long run.

- In finite MDPs there is always at least one policy that is better or equal to all the other policies: this is called the *optimal policy*.

- Although there may be more than one, we denote all the optimal policies with π_*. They are characterised by the same value function v_* defined as

$$v_*(s) \doteq \max_{\pi} v_\pi(s)$$

for all $s \in \mathcal{S}$.

Optimal Policies and Optimal Value Functions

- Optimal policies also shares the same optimal action-value function q_*, which is defined as

$$q_* \doteq \max_{\pi} q_\pi(s, a)$$

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

- We can write q_* in terms of v_* as follows:

$$q_*(s, a) = E[R_{t+1} + \gamma v_*(S_{t+1}) \,|\, S_t = s, A_t = a].$$

Difference between Reinforcement Learning and Supervised Learning

- *Supervised learning* is learning from a set of labeled examples.
- In interactive problems, it is hard to obtain labels in the first place.
- In “unknown” situations, agents have to learn from their experience. In these situations, reinforcement learning is most beneficial.

Difference between Reinforcement Learning and Unsupervised Learning

- Unsupervised learning is learning from datasets containing unlabelled data.
- You might think that reinforcement learning is a type of unsupervised learning, because it does not rely on examples (labels) of correct behaviour and instead explores and learns it.
- But this is not the case: in reinforcement learning the goal is to maximise a reward signal instead of trying to find a hidden structure.
- For this reason, reinforcement learning is usually considered a third paradigm in addition to supervised and unsupervised learning.

References

- The notation and definitions of these slides are taken (with small variations) from:

Richard S. Sutton and Andrew G. Barto. Reinforcement Learning. An Introduction. Second Edition. MIT Press. 2018.