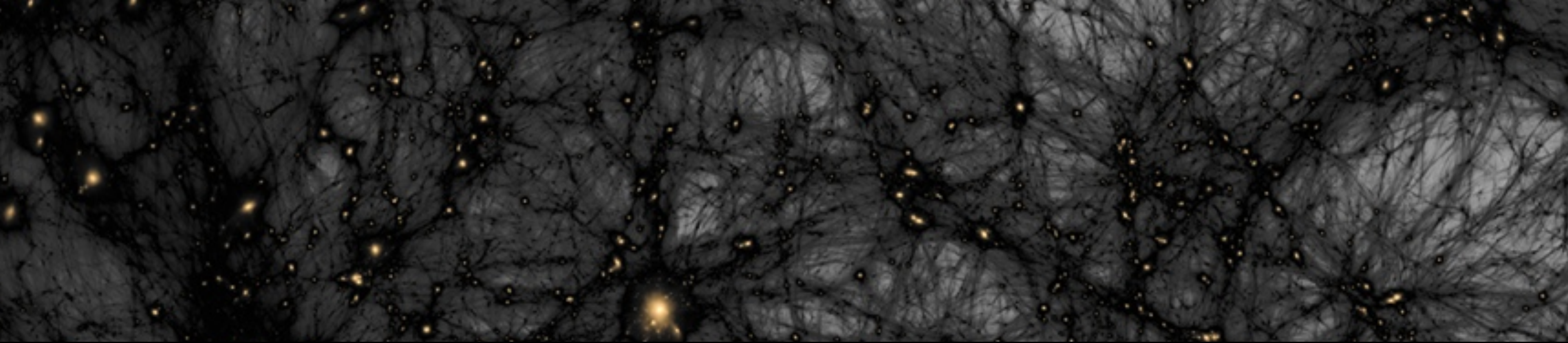


AN INTRODUCTION TO REINFORCEMENT LEARNING

Alkistis Pourtsidou
Queen Mary, University of London

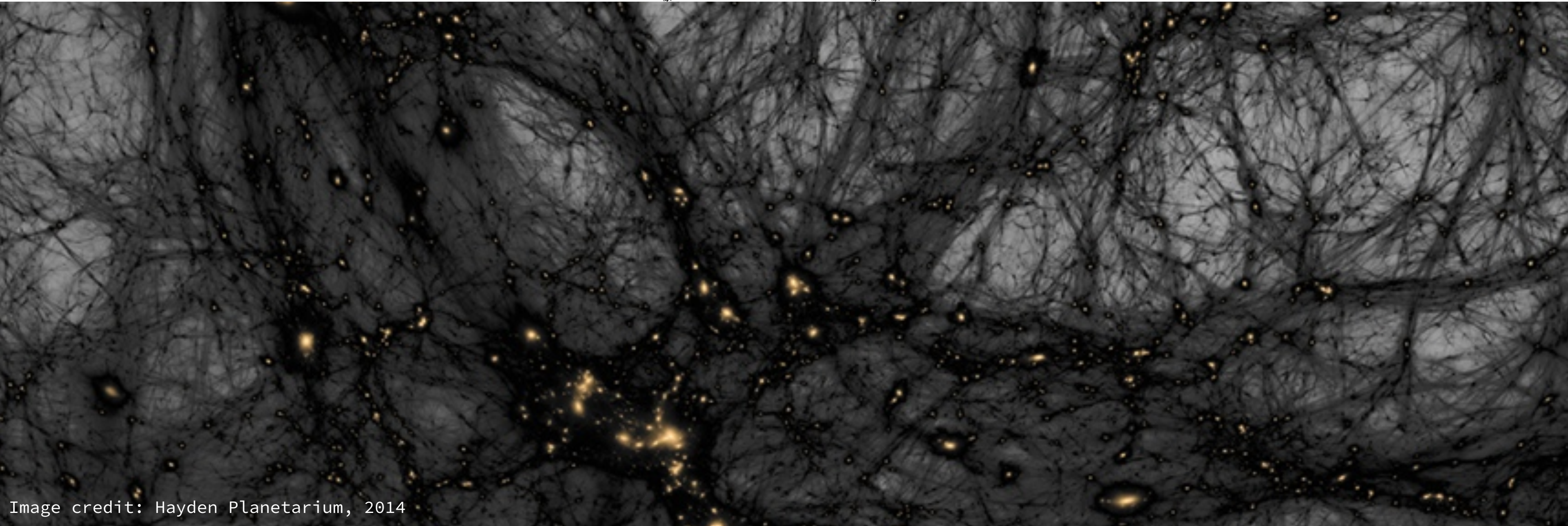


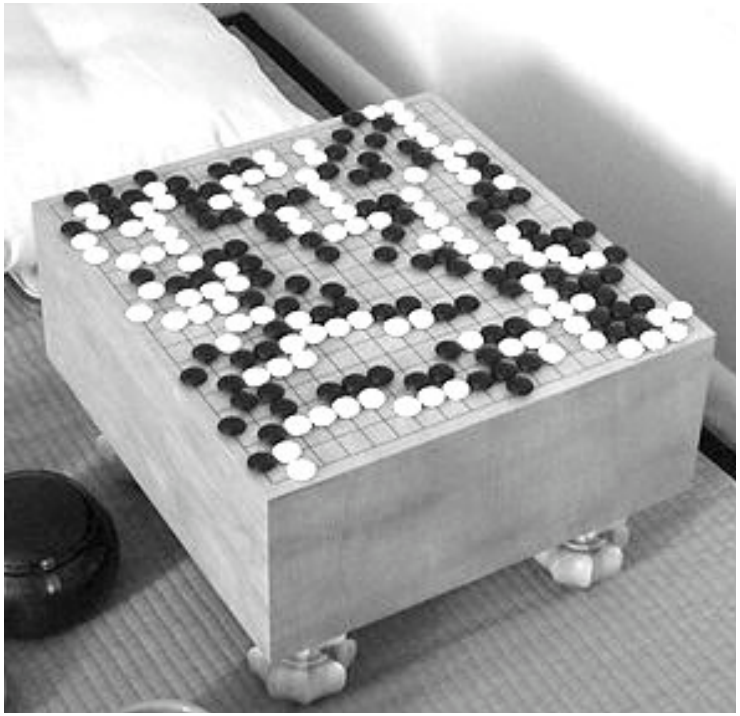
Image credit: Hayden Planetarium, 2014

REFERENCES AND TUTORIALS

This lecture is heavily based on the following resources:

- **Introduction to Reinforcement Learning** lecture course by D. Silver http://www0.cs.ucl.ac.uk/staff/d.silver/web/Teaching.html
- **Hands on Machine Learning with Scikit Learn and TensorFlow** by A. Geron, see Chapter 16 and Github repo https://github.com/ageron/handson-ml
- **Reinforcement Learning** by S. Sutton and A. G. Barto https://drive.google.com/file/d/1xeUDVGWGUUv1-ccUMAZHJLej2C7aAFWY/view

A BIT OF HISTORY



- Reinforcement Learning (RL) is one of the oldest Machine Learning fields (1950s)
- <u>Games revolution in 2013:</u> Researchers from the DeepMind startup built a system that could play any Atari game
- In 2016, their system beat the world champion of the Go game
- Wide range of applications today (games, robots, cars,...)
- DeepMind was bought by Google for half a billion dollars!

The many faces of RL

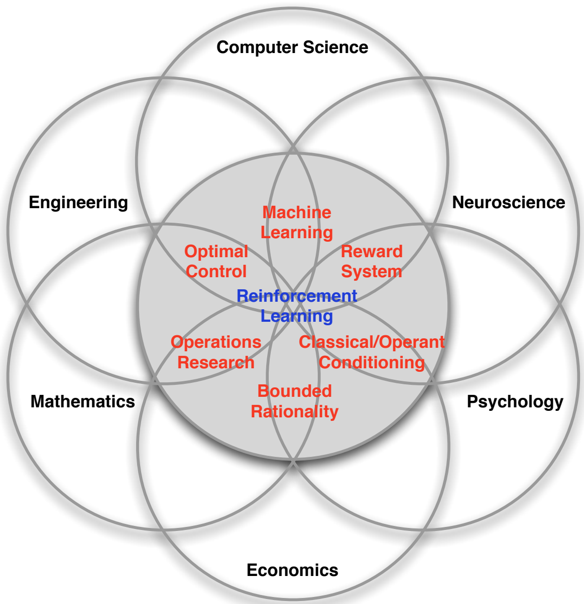


Image credit: D. Silver

- Sits at the intersection of many different fields
- The science of **decision making** is very general and fundamental
- **Goal:** understand optimal way to make decisions
- Basically same methods under different names in engineering, neuroscience, etc.

RL IS A BRANCH OF MACHINE LEARNING

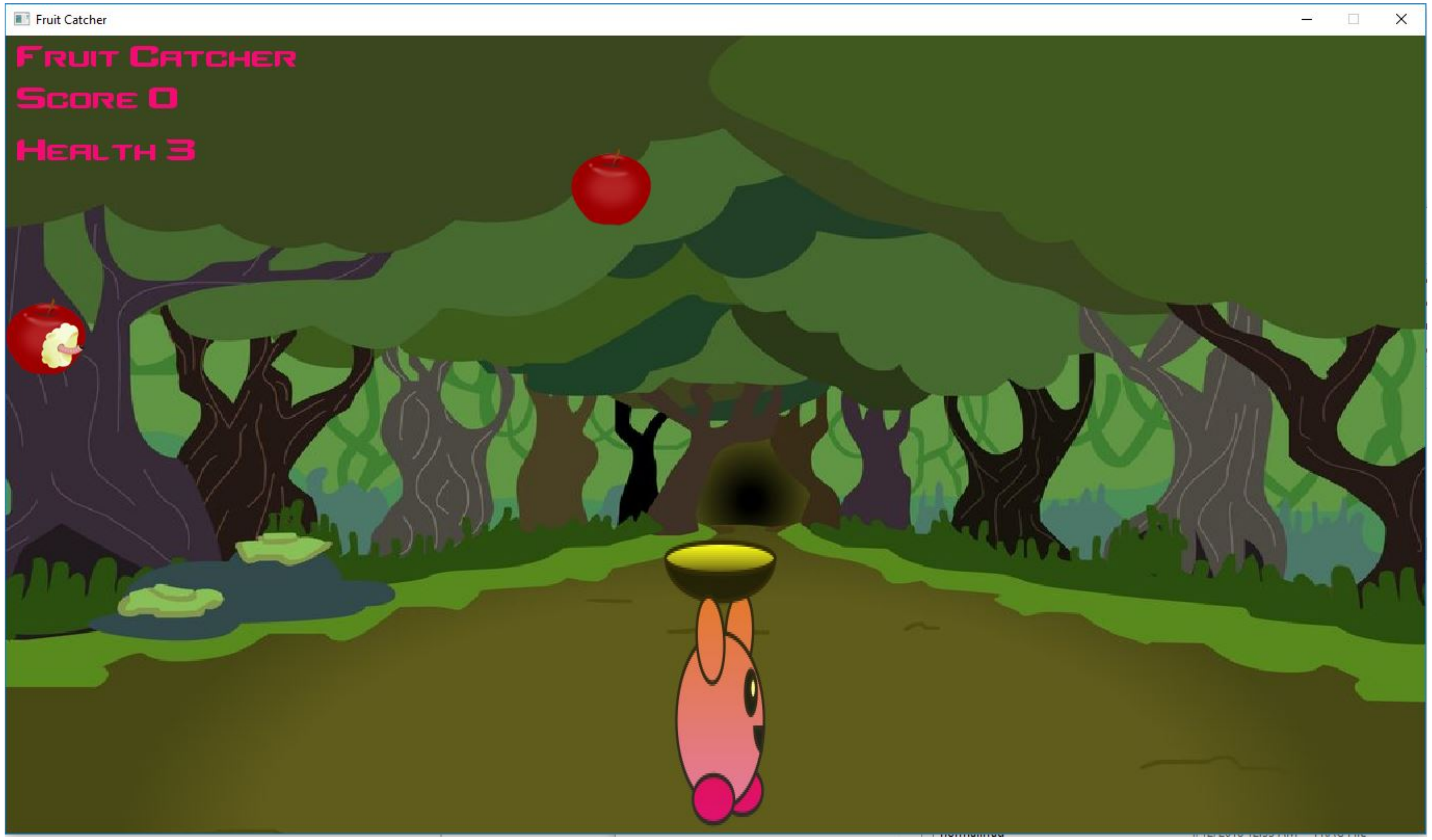
- In **supervised learning**, we have an input and a target value or class we want to predict
- In **unsupervised learning**, we only have an input and look for patterns in that input
- **Reinforcement learning**:
    - No supervisor, just *reward signal*
    - We train an **agent** to maximise a **reward** through interactions with an **environment**
    - Time matters (more about that later) - e.g. decisions unfold over time
    - System is dynamic, non IID (basically independent and “static”) data.

Real world examples of RL use

- Self-driving cars
- Manage investment portfolio - e.g. incoming stream of data, has to make decisions on what to invest
- Make a robot walk - room is the stream of data, falling over or crashing at the wall is bad!
- Control a power station - e.g. maximise power while respecting regulations/laws
- Learn to play computer games (better than humans) without even knowing the rules - *trial and error learning*!

REAL WORLD EXAMPLES OF RL USE

- Game example: Cather - catch the fruit before it reaches the floor
- We have the game **environment** (basically a game simulation), the **actions** (joystick movements) and the RL algorithm learns to play it
- See https://edersantana.github.io/articles/keras_rl/ for code example



Rewards

- A **reward** R_t is a scalar feedback signal: in simpler words, just a number
- Indicates how well an agent is doing at time-step t
- E.g. if you catch the fruit, $R_t = +1$. If not, $R_t = -1$
- The agent's job is to maximise cumulative (i.e. summed up) reward
- Reinforcement Learning is based on the reward hypothesis:

Definition (Reward Hypothesis)

All goals can be described by the maximisation of expected cumulative reward

Rewards

- A **reward** R_t is a scalar feedback signal: in simpler words, just a number
- Indicates how well an agent is doing at step t
- E.g. if you catch the fruit, $R_t = +1$. If not, $R_t = -1$
- The agent's job is to maximise cumulative (i.e. summed up) reward
- Reinforcement Learning is based on the reward hypothesis:

Definition (Reward Hypothesis)

All goals can be described by the maximisation of expected cumulative reward

Question: What if the goal is time based, e.g. "achieve X in the shortest amount of time". Any ideas on how we can define reward here?

Rewards examples

- **Self-driving car**
  - **+1** for following desired trajectory
  - **-50** for crashing! (large negative reward)
- **Robot walking**
  - **+1** for forward motion
  - **-50** for falling over!
- **Playing Atari games**
  - **+** for winning points
  - **-** for losing points

COMMON FRAMEWORK: SEQUENTIAL DECISION MAKING

- Goal: **select actions to maximise total future reward**
  - Actions may have long term consequences so need to think ahead
  - Reward may be delayed!
  - It may be better to sacrifice immediate reward to gain more long-term reward

- **Examples:**
  - An investment (may take months to mature)
  - Fueling a helicopter (to prevent a crash in several hours)

The Agent

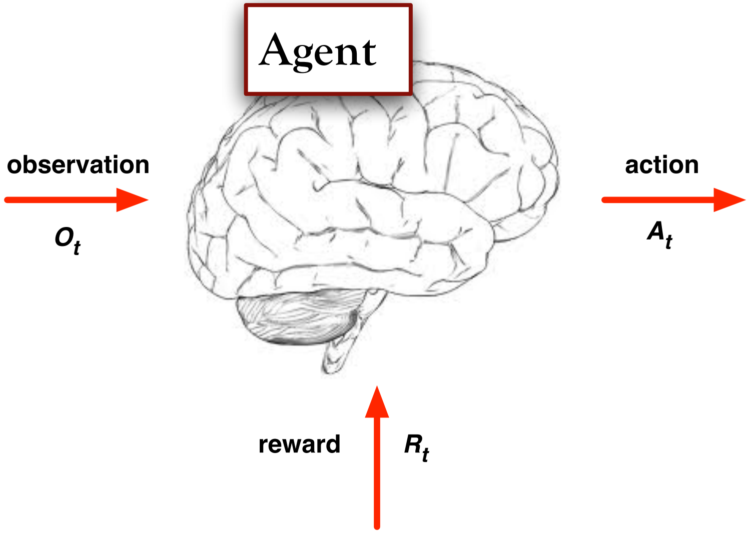
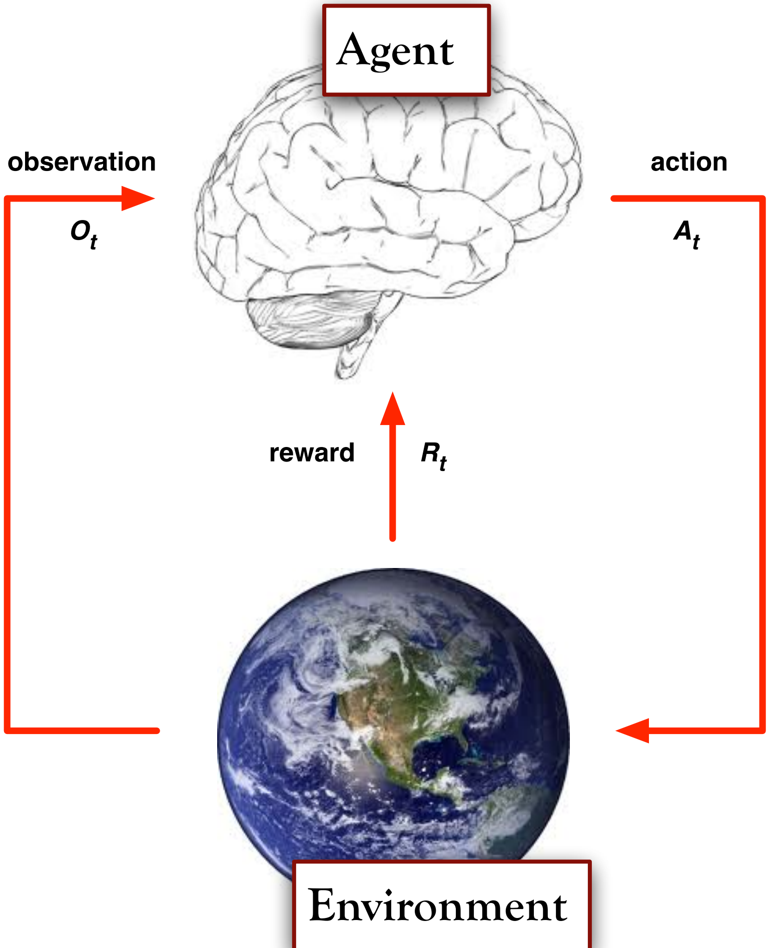


Image credit: D. Silver

- Via the RL algorithm, we are controlling the agent (e.g. robot with a camera)
- Every step: the agent sees a snapshot - *observation* - of what is happening in "its world"
- Gets *reward* signal
- Has to make a decision - *action*

The Environment

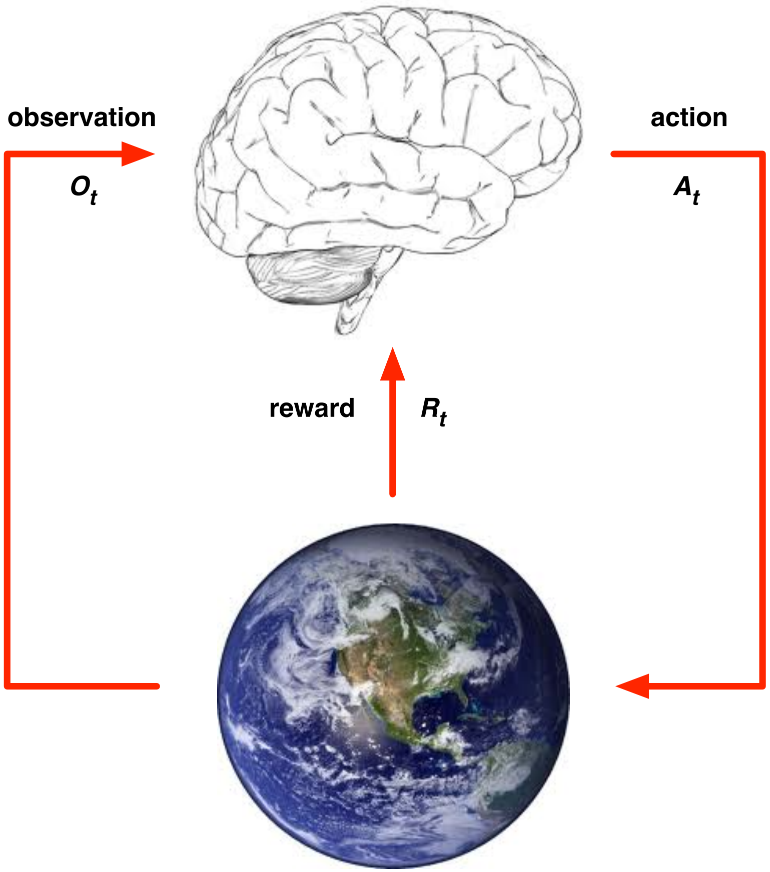
Based on slide by D. Silver



- At each step t the agent:
  - Executes action A_t
  - Receives observation O_t
  - Receives scalar reward R_t
- The environment:
  - Receives action A_t
  - Emits observation O_{t+1}
  - Emits scalar reward R_{t+1}
- t increments at env. step

Example: Atari game, generates the next screen (observation) and the score (reward).

The Environment



- At each step t the agent:
  - Executes action A_t
  - Receives observation O_t
  - Receives scalar reward R_t
- The environment:
  - Receives action A_t
  - Emits observation O_{t+1}
  - Emits scalar reward R_{t+1}
- t increments at env. step

Question: Does the agent (we) have control over the environment?

SUMMARY: AGENT AND ENVIRONMENT

- The environment defines a set of **actions** an agent can take
- The agent observes the current **state** of the environment, tries **actions** and learns a **policy**
- A policy is a *distribution over the possible actions* (given the state of the environment)

Agents and Environments Examples

- **Walking Robot example**
- Agent: the program controlling the robot
- Environment: the real world
- The agent observes the environment through a set of sensors (cameras, touch sensors,...)
- Its actions consist of sending signals to active motors
- + when it approaches the target destination
- - when it goes in the wrong direction or falls down

Agents and Environments Examples

- **Computer game example (e.g. Catcher, Go, PacMan)**
- Agent: the program controlling the game
- Environment: a simulation of the game - see e.g. PyGame learning environment https://pygame-learning-environment.readthedocs.io/en/latest/user/games/catcher.html
- Actions are the possible joystick positions (up, down, left, right, etc.)
- Rewards are game points

History and State

- **History:** the sequence of observations, actions, rewards

$$H_t = A_1, O_1, R_1, \ldots, A_t, O_t, R_t$$

I.e., all observable variables up to time t

- The algorithm we build is a mapping from *history* –> *picking the next action*
- The agent selects actions depending on the history
- The environment selects observations/rewards based on the history
- But going back to an enormous history all the time is not optimal
- A **state** captures the required information concisely - it's basically a summary of what we need to pick the next action

Agent state

$$H_t = A_1, O_1, R_1, ..., A_t, O_t, R_t$$

- A state is a function of the history:

$$S_t = f(H_t)$$

- For example, this function could just pick the last observation and only look at it, ignoring all previous observations:

Agent state

$$H_t = A_1, O_1, R_1, ..., A_t, O_t, R_t$$

- A state is a function of the history:

$$S_t = f(H_t)$$

- For example, this function could just pick the last observation and only look at it, ignoring all previous observations:

$$S_t = A_{t-1}, O_{t-1}, R_{t-1}$$

AGENT STATE

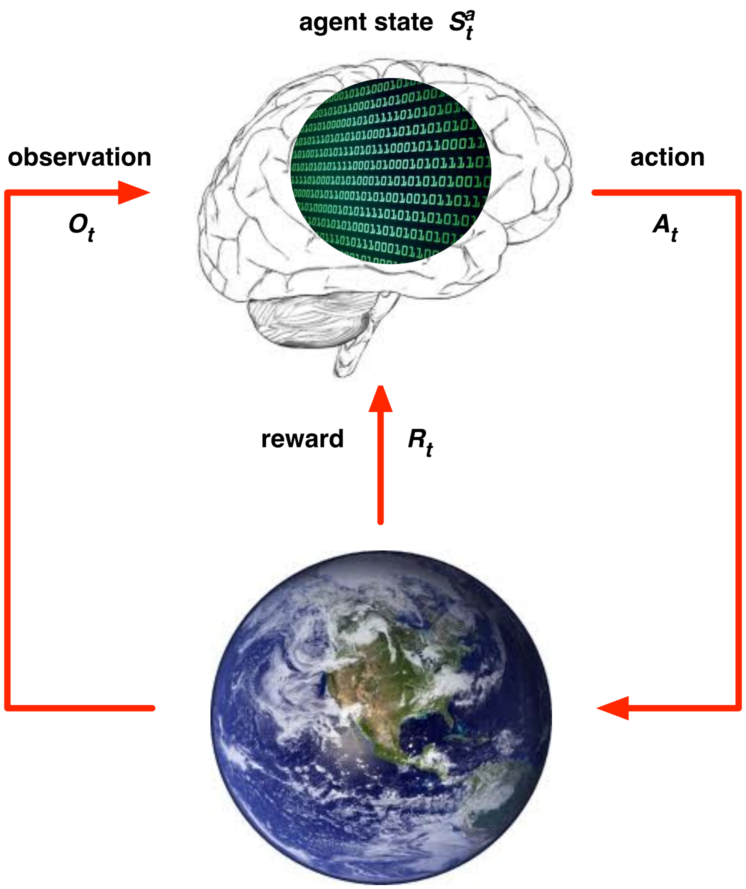


Image credit: D. Silver

- The **agent state** defines the information used by RL algorithms
- It is the agent's *internal representation*; the information the agent uses to pick the next action
- It can be any function of the history
- **Our goal is to build a model for picking actions**

MARKOV STATE

- An **information state** (**Markov state**) contains all useful information from the history

Definition

A state S_t is Markov if and only if

$$\mathbb{P}[S_{t+1} \mid S_t] = \mathbb{P}[S_{t+1} \mid S_1, ..., S_t]$$

- Current state is all that matters
- Future is independent of the rest of the history
- **Example:** self-driving car —> current position x and velocity v are enough, (x,v) before irrelevant!

TWO MAIN COMPONENTS OF AN RL AGENT

- **Policy:** a map from state to action: how the agent picks its actions, its behaviour function
  - E.g. deterministic policy $a = \pi(s)$
- **Value function:** Estimates how good each state or action is, how well we are doing in a particular situation –> **a prediction of future reward**
- Let's see how these two work in more detail…

POLICY

- The agent's behaviour
- It maps state to action
- **Formally:** a distribution over the possible actions the agent can take in the environment given the current state of the environment

$$\pi(a|s)$$

- **Goal:** a policy that leads to the maximum reward

Value function

- **Value function:** Prediction of expected (future) total reward given state *s*
- How good is a state for the agent to be in
- Depends on policy

$$\mathrm{v}_\pi(s) = \mathcal{E}_\pi[R_t + \gamma R_{t+1} + \gamma^2 R_{t+2} + ... | S_t = s]$$

γ

- Discount factor
- It is a measure of how far ahead in time we look, how much weight is given to future rewards

Value function

- **Value function:** Prediction of expected (future) total reward given state *s*
- How good is a state for the agent to be in
- Depends on policy

$$\mathrm{v}_\pi(s) = \mathcal{E}_\pi[R_t + \gamma R_{t+1} + \gamma^2 R_{t+2} + \ldots | S_t = s]$$

Question: What does γ close to 0 mean? What about γ=0.9? And why we usually see γ<1?

THE MAZE: REWARD, ACTIONS, STATES

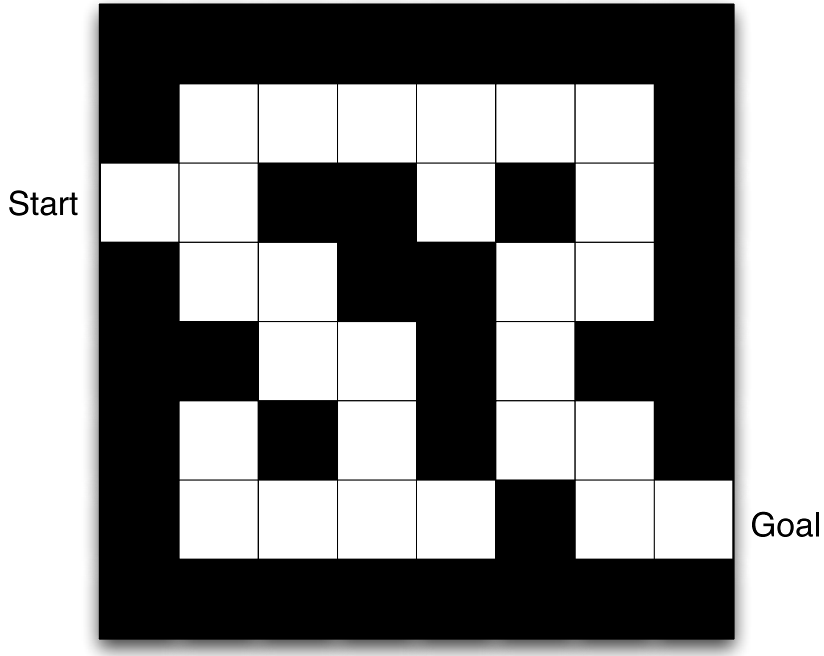


Image credit: D. Silver

- Reach the goal as quickly as possible
- R = -1 per time-step
- **Actions:** Up,Down,Left,Right
- **State:** the agent's location on the grid

THE MAZE: POLICY

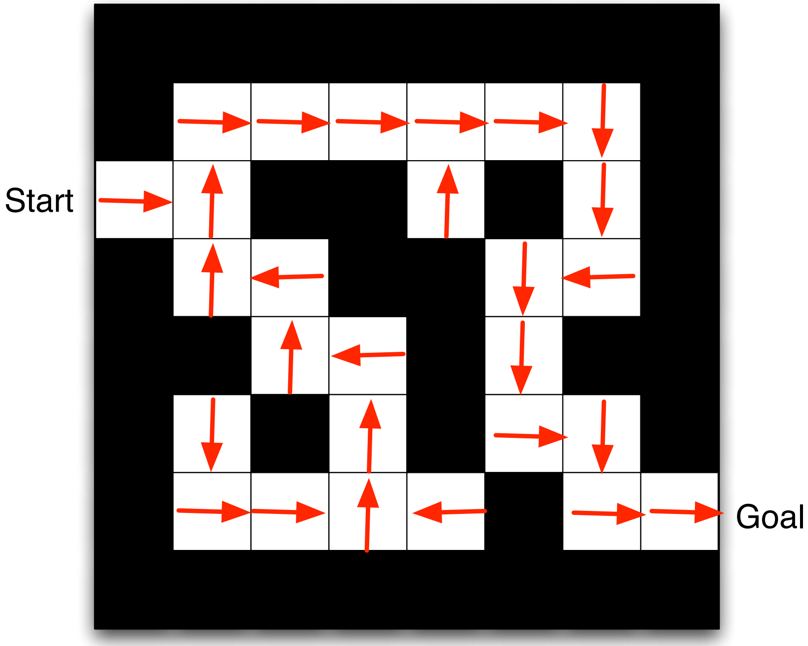


Image credit: D. Silver

- The arrows represent the policy for each state *s*
- What the agent will choose to do at each state (grid position)
- A mapping from state to action

$$a = \pi(s)$$

POLICY NETWORKS

- In a Deep RL agent, the policy is represented by a neural network with parameters θ
- We have: $\pi_\theta(a|s) = NN(s;\theta)$
- The neural network takes in the state as input and outputs the appropriate distribution over actions

THE MAZE: VALUE FUNCTION

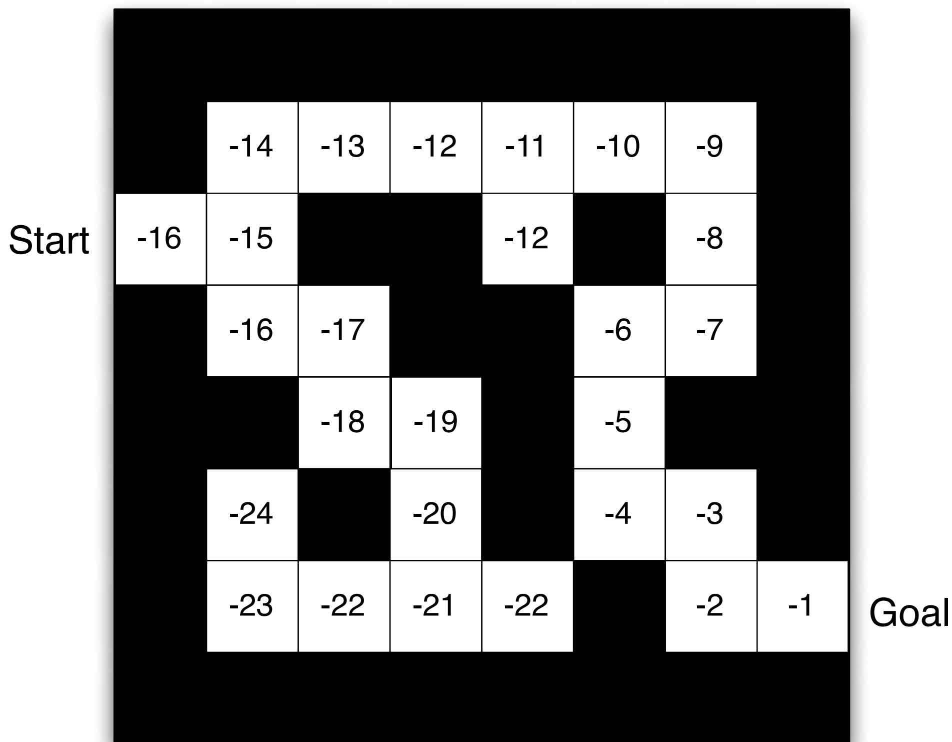


Image credit: D. Silver

- Just about to reach the goal: value function = -1 (the highest)
- Two steps away from the goal: value function = -2
- **Having these values means we can build an optimal policy**
- E.g. if we are at -15 we should go up and not left or down

LEARNING WITH POLICY GRADIENTS

- We need to teach the agent to maximise the expected reward following a policy
- Need to give our agent "intelligence" by making it learn from its experience in interacting with the environment.
- Reminder: actions determined by policy $\pi_\theta(a|s)$
- **Policy gradients** algorithms optimise the parameters (θ) of a policy by following the gradients toward higher rewards
- We will just illustrate this method with a simple example (see the references for the strict mathematical formalism)

LEARNING WITH POLICY GRADIENTS

- Consider a robotic vacuum cleaner whose goal (reward) is picking up as much dust as possible in 10 minutes
- Its policy could be to move forward with some probability **P** per second
- Or randomly rotate left or right with probability **1-P**
- The rotation angle would be a *random angle* between **-r** and **+r**
- Eventually, the robot will pick up all the dust.
- But how much can it pick up in 10 minutes?
- How would we train such a robot?

LEARNING WITH POLICY GRADIENTS

- Consider a robotic vacuum cleaner whose goal (reward) is picking up as much dust as possible in 10 minutes
- Its policy could be to move forward with some probability **P** per second
- Or randomly rotate left or right with probability **1-P**
- The rotation angle would be a *random angle* between **-r** and **+r**
- Eventually, the robot will pick up all the dust. But how much can it pick up in 10 minutes?
- How would we train such a robot?

Question: Which are the policy parameters in this example?

LEARNING WITH POLICY GRADIENTS

- There are 2 policy parameters we can tweak: **the probability P and the angle range r** (let's just think about **P** for simplicity)

- **Brute force approach:** Try out many different values, pick the combination that performs best. This is a *policy search* brute force example, but when the policy space is too large it's hopeless

LEARNING WITH POLICY GRADIENTS

- There are 2 policy parameters we can tweak: **the probability p and the angle range r** (let's just think about **p** for simplicity)

- **Brute force approach:** Try out many different values, pick the combination that performs best. This is a *policy search* brute force example, but when the policy space is too large it's hopeless

- **Optimisation techniques:** Slightly increase **P** and evaluate whether this increases the amount of dust picked up in 10 mins. If it does, then increase some more; if not, decrease. This is an example of **learning with policy gradients**.

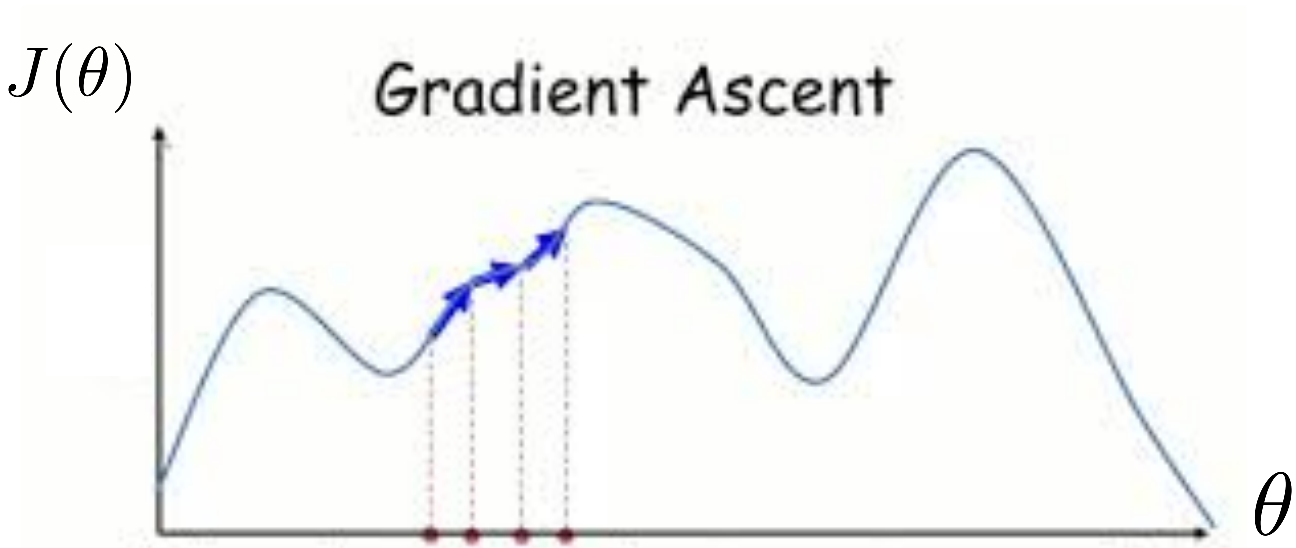
LEARNING WITH POLICY GRADIENTS

- There are 2 policy parameters we can tweak: **the probability p and the angle range r** (let's just think about **p** for simplicity)

- **Brute force approach:** Try out many different values, pick the combination that performs best. This is a *policy search* brute force example, but when the policy space is too large it's hopeless

- **Optimisation techniques:** Slightly increase p and evaluate whether this increases the amount of dust picked up in 10 mins. If it does, then increase some more; if not, decrease. This is an example of **learning with policy gradients**.

- A bit more formally/generally: Evaluate the gradients of the rewards with respect to the policy parameters, then tweak these parameters following the gradient toward higher rewards (*gradient ascent*)



$$Policy: \ \pi_\theta$$
$$Objective\ function\ : J(\theta)$$
$$Gradient: \ \nabla_\theta J(\theta)$$
$$Update: \theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$$

ENOUGH THEORY - SHOW US SOME CODE!

- I strongly suggest the "hello world" of RL, the cart-pole balancing!
- See, for example, https://github.com/ageron/handson-ml/blob/master/16_reinforcement_learning.ipynb