Co-funded by the
Erasmus+ Programme
of the European Union

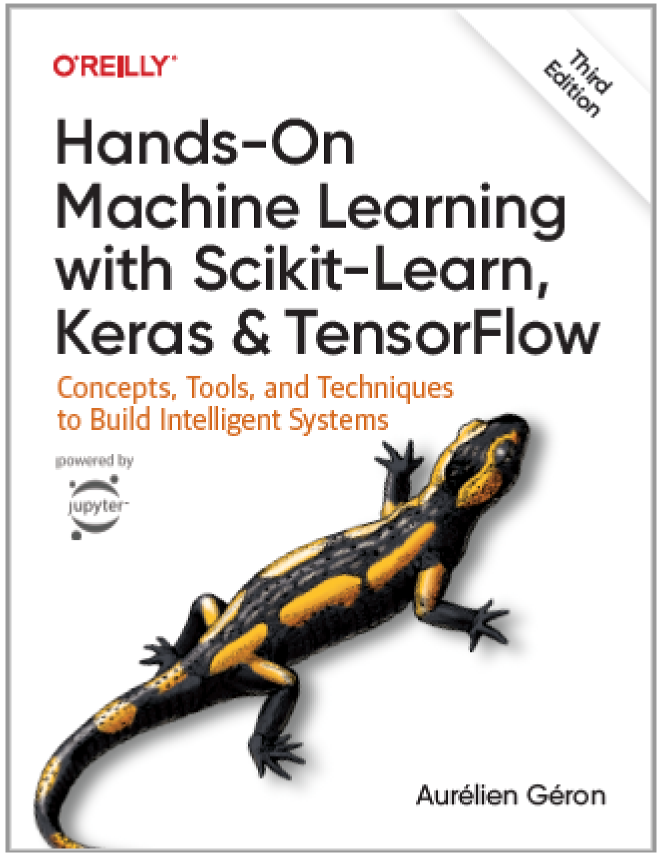
Reinforcement Learning

Prof. Gheith Abandah

Developing Curricula for Artificial Intelligence and Robotics (DeCAIR)
618535-EPP-1-2020-1-JO-EPPKA2-CBHE-JP

Reference

- Chapter 18: **Reinforcement Learning**


O'REILLY
Third Edition
Hands-On Machine Learning with Scikit-Learn, Keras & TensorFlow
Concepts, Tools, and Techniques to Build Intelligent Systems
powered by jupyter
Aurélien Géron

- Aurélien Géron, **Hands-On Machine Learning with Scikit-Learn, Keras and TensorFlow**, O'Reilly, 3rd Edition, 2022
  - Material: https://github.com/ageron/handson-ml3

Outline

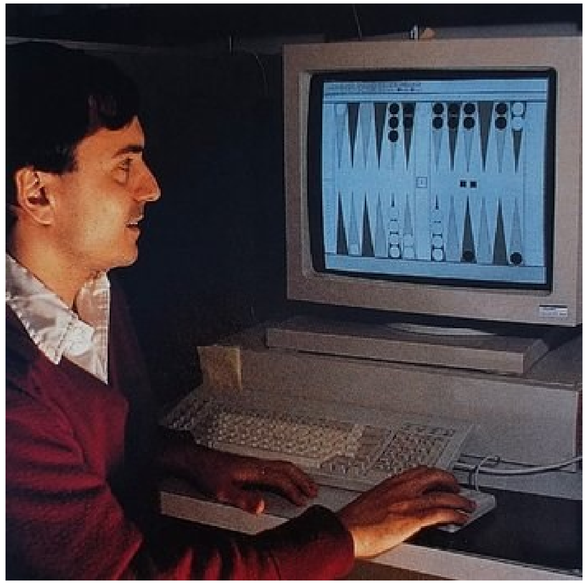
Introduction

- YouTube Video: **An introduction to Reinforcement Learning** from Arxiv Insights

https://youtu.be/JgvyzIkgxF0

1. Introduction – History

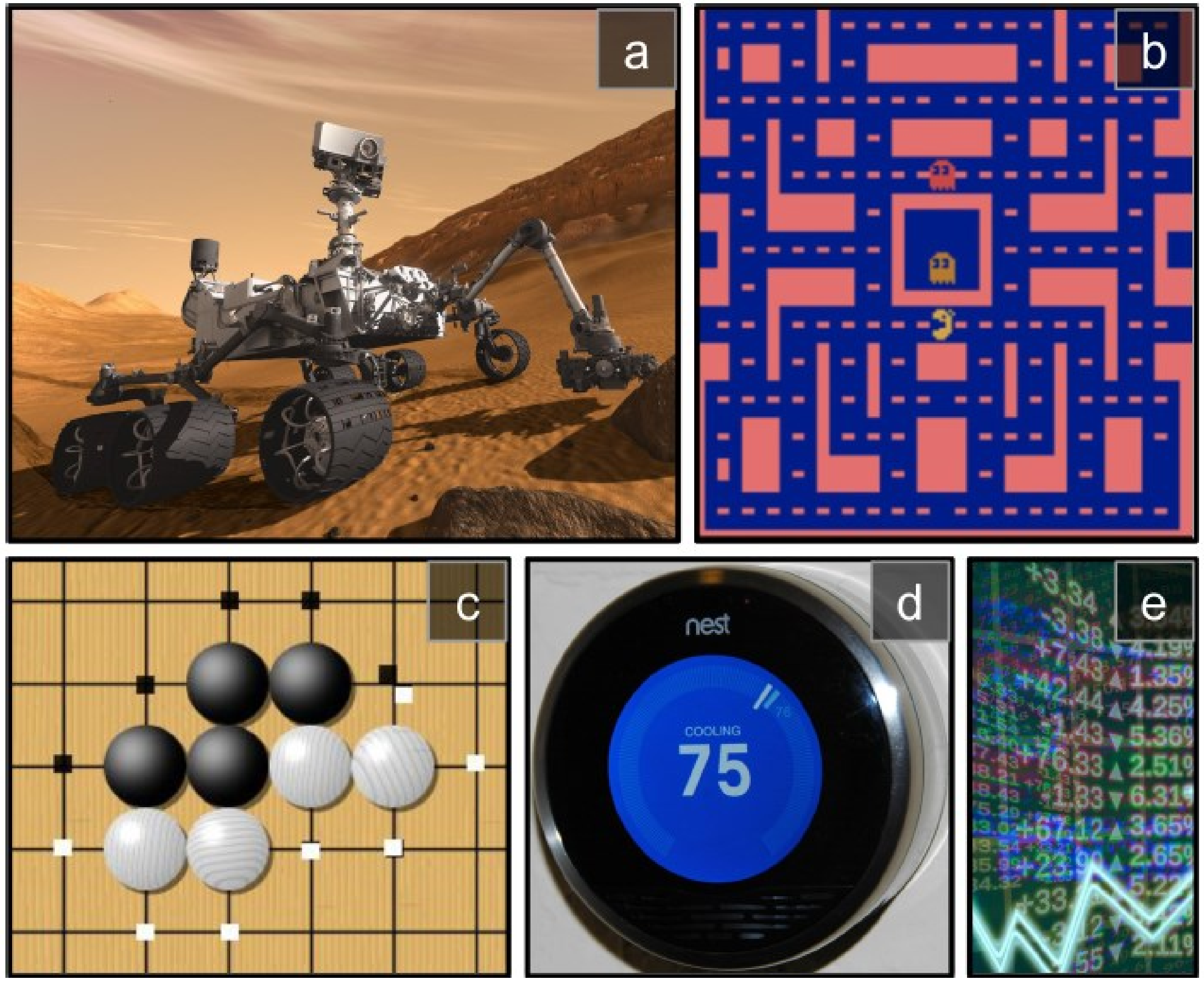
- RL started in **1950s**
- **1992**: IBM's TD-Gammon, a Backgammon playing program.
- **2013**: DeepMind demonstrated a system that learns to play Atari games from scratch.
- Use **deep learning** with raw pixels as inputs and without any prior knowledge of the rules of the games.
- **2014**: Google bought DeepMind for $500M.
- **2016**: AlphaGo beats Lee Sedol.



1. Introduction – Definition

- In Reinforcement Learning, a software **agent** makes **observations** and takes **actions** within an **environment**, and in return it receives **rewards**.
- Its objective is to learn to act in a way that will **maximize its expected long-term rewards**.
- In short, the agent acts in the environment and learns by trial and error to maximize its **pleasure** and minimize its **pain**.

1. Introduction – Examples

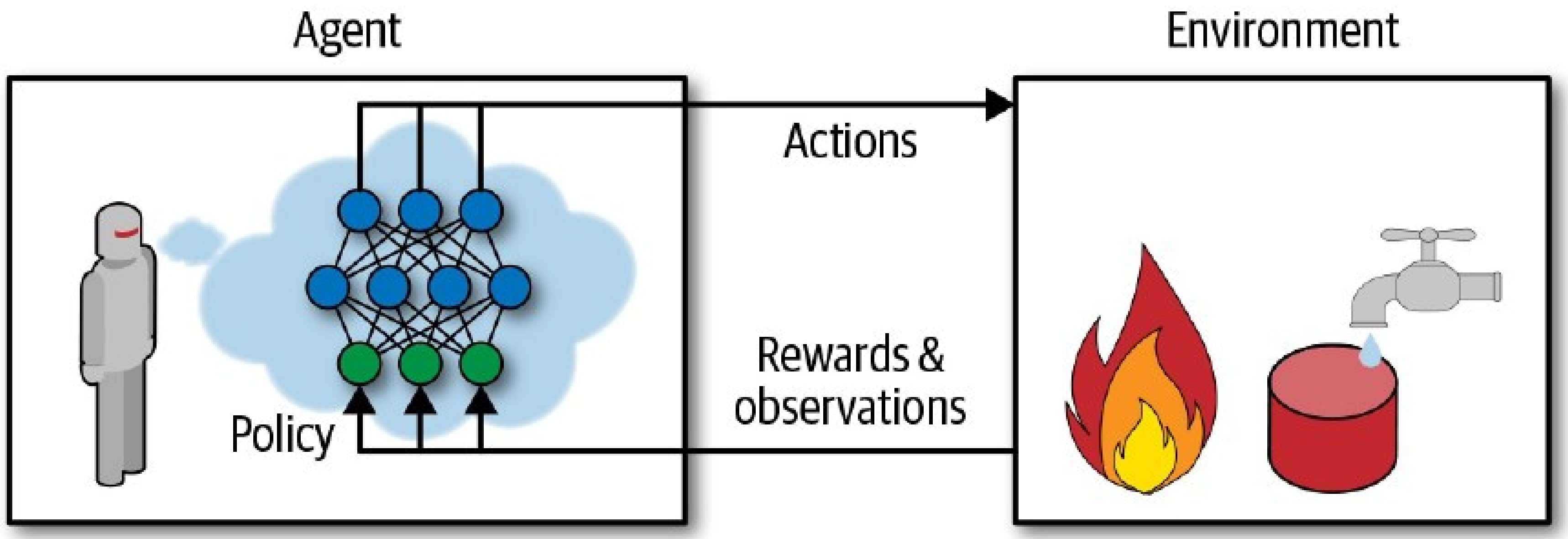


(a) Robotics
(b) Ms. Pac-Man
(c) Go player
(d) Thermostat
(e) Automatic trader

Outline

2. Policy Search

- The algorithm used by the software agent to determine its actions is called its **policy**.
- The policy can be **deterministic** or **stochastic**.
- **Policy search techniques**: Brute force, Genetic algorithm, Policy Gradient (PG), Q-Learning.



Outline

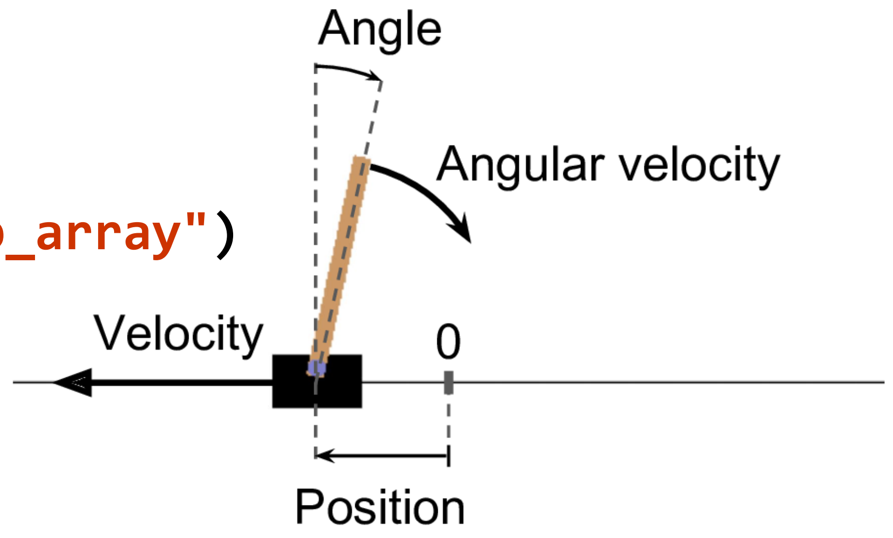
3. OpenAI Gym

- OpenAI Gym is a toolkit that provides **simulated environments** (Atari games, board games, 2D and 3D physical simulations, …).
- OpenAI is a nonprofit AI research company funded in part by Elon Musk. Got $1 billion investment from Microsoft.

```
$ pip3 install --upgrade gym
```

```
>>> import gym
>>> env = gym.make("CartPole-v1", render_mode="rgb_array")
>>> obs, info = env.reset()
>>> obs
array([-0.012586, -0.001566, 0.042077, -0.001805])
```

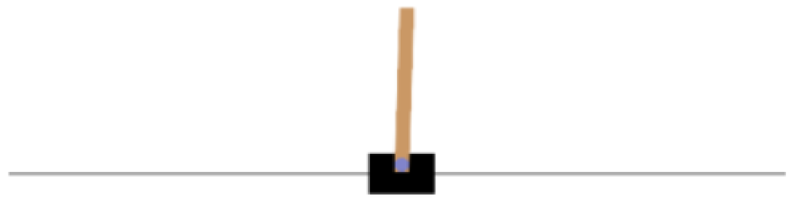
Cart position, cart velocity, pole angle, pole velocity



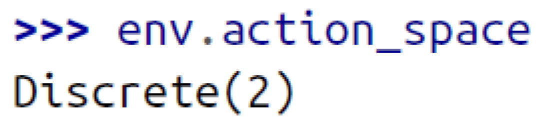
3. OpenAI Gym

- `render()` can return the rendered image as a NumPy array.

```
>>> img = env.render()
>>> img.shape # height, width, channels (3 = RGB)
(400, 600, 3)
```



3. Balancing the pole



```
>>> env.action_space
Discrete(2)
```

The possible actions are integers 0 and 1, which represent accelerating left (0) or right (1).

```
>>> action = 1 # accelerate right
>>> obs, reward, done, truncated, info = env.step(action)
>>> obs
array([-0.012617, 0.192928, 0.042041, -0.280921])
>>> reward
1.0
>>> done, truncated
False, False
>>> info
{}
```

3. Balancing the pole

```
def basic_policy(obs):
        angle = obs[2]
        return 0 if angle < 0 else 1

totals = []
for episode in range(500):
    episode_rewards = 0
    obs, info = env.reset()
    for step in range(200):
        action = basic_policy(obs)
        obs, reward, done, truncated, info = env.step(action)
        episode_rewards += reward
        if done or truncated:
             break
    totals.append(episode_rewards)
```

Accelerates left when the pole is leaning left and accelerates right when the pole is leaning right.

3. Balancing the pole

- Even with 500 tries, this policy never managed to keep the pole upright for more than 63 consecutive steps.

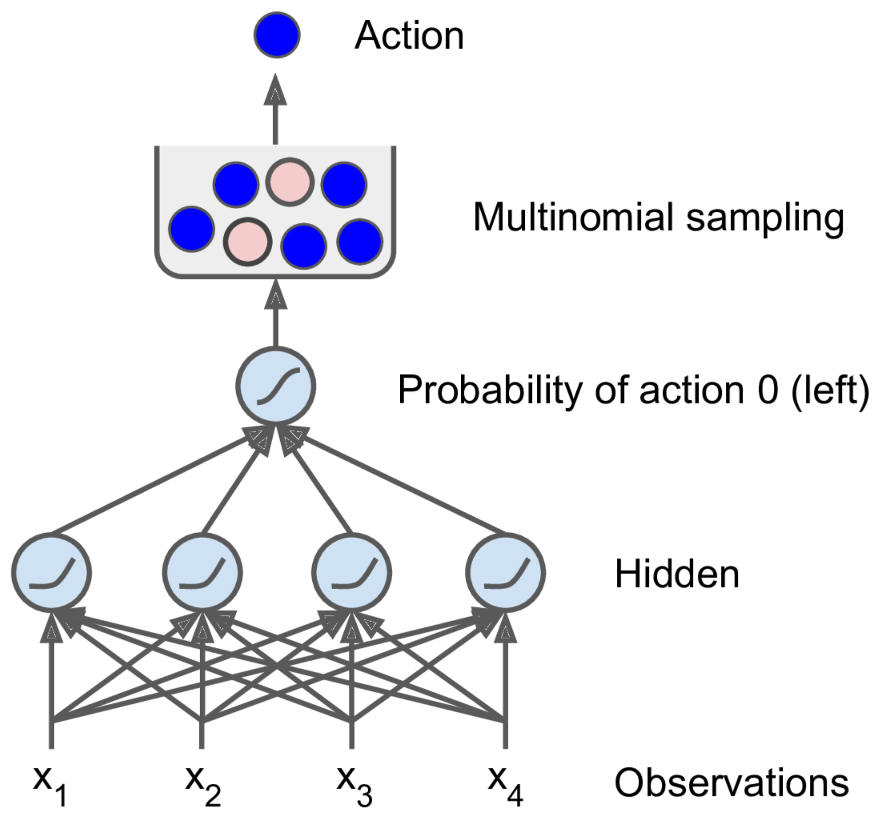
```
>>> import numpy as np
>>> np.mean(totals), np.std(totals), np.min(totals), np.max(totals)
(41.698, 8.389445512070509, 24.0, 63.0)
```

Outline

1. Introduction
2. Policy Search
3. OpenAI Gym
4. Neural Network Policies
5. Exercises

4. Neural Network Policies

- Takes an **observation as input**, and **outputs** the **probability** for **each action**
- We **select an action** randomly, according to the estimated probabilities.
- **Explore and exploit**



4. Neural Network Policy in Keras

```
# Building a polity network is easy
import tensorflow as tf
from tensorflow import keras

n_inputs = 4 # == env.observation_space.shape[0]

model = keras.Sequential([
      layers.Dense(5, activation="relu",
          input_shape=[n_inputs]),
      layers.Dense(1, activation="sigmoid"),
])
# Training it is something else
```

Exercises

18.1. How would you define Reinforcement Learning? How is it different from regular supervised or unsupervised learning?

18.2. Can you think of three possible applications of RL that were not mentioned in this chapter? For each of them, what is the environment? What is the agent? What are some possible actions? What are the rewards?

Summary