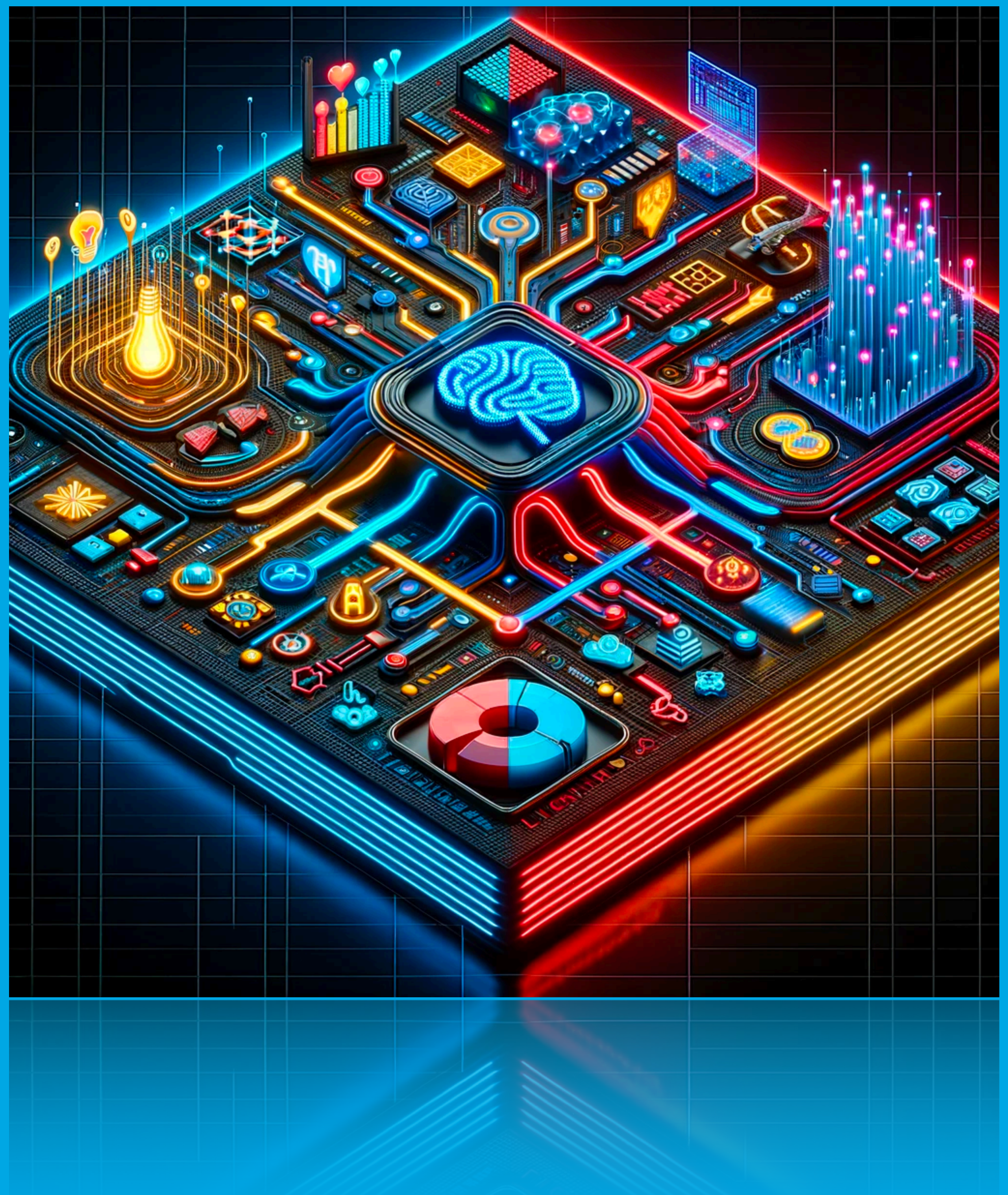
Reinforcement Learning for Finance

Dr. Yves J. Hilpisch

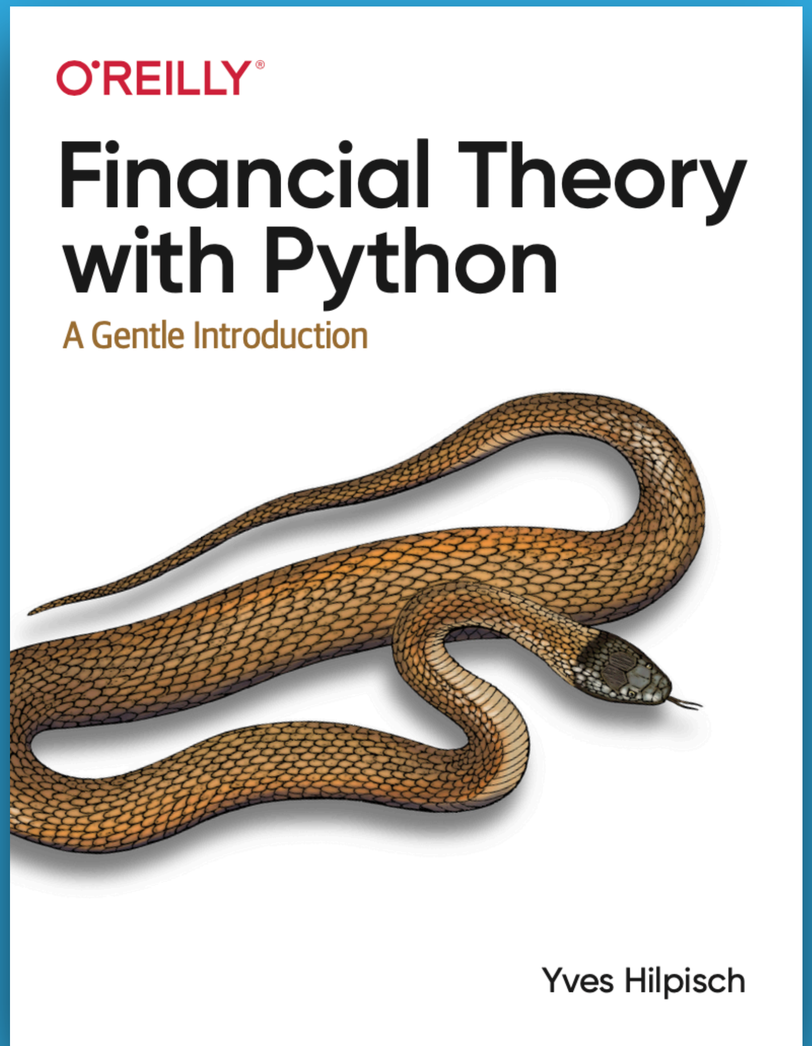
ODSC, London, September 2024

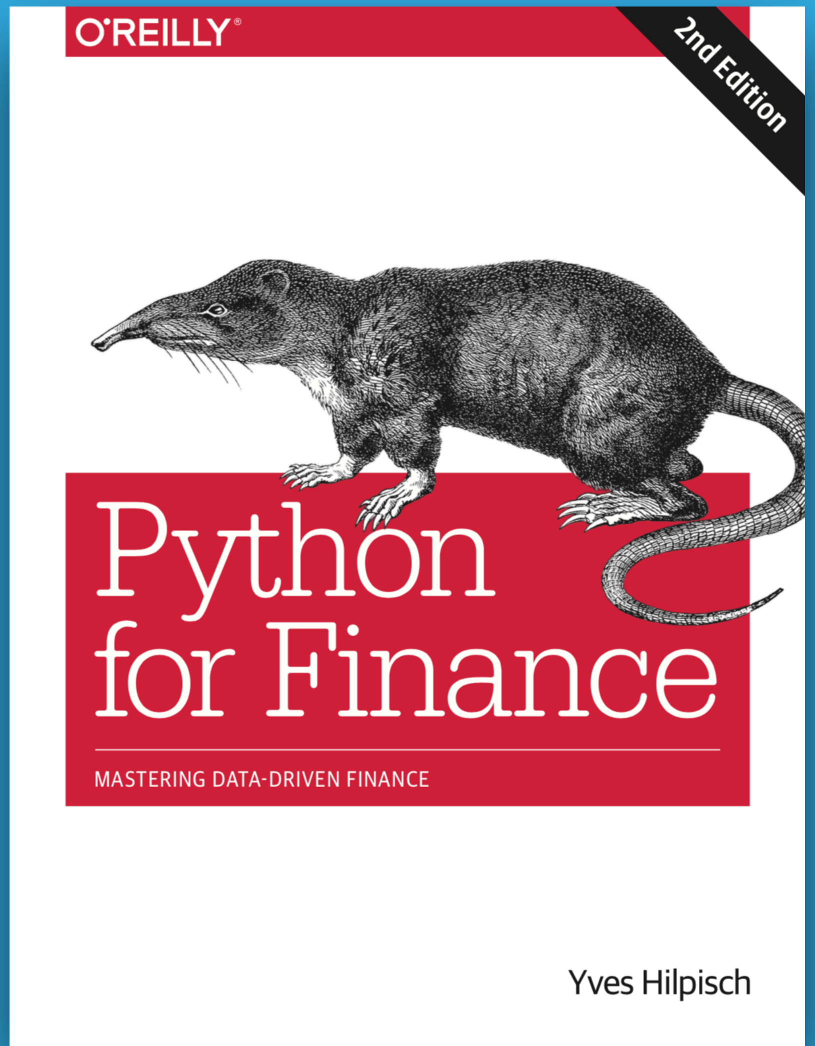


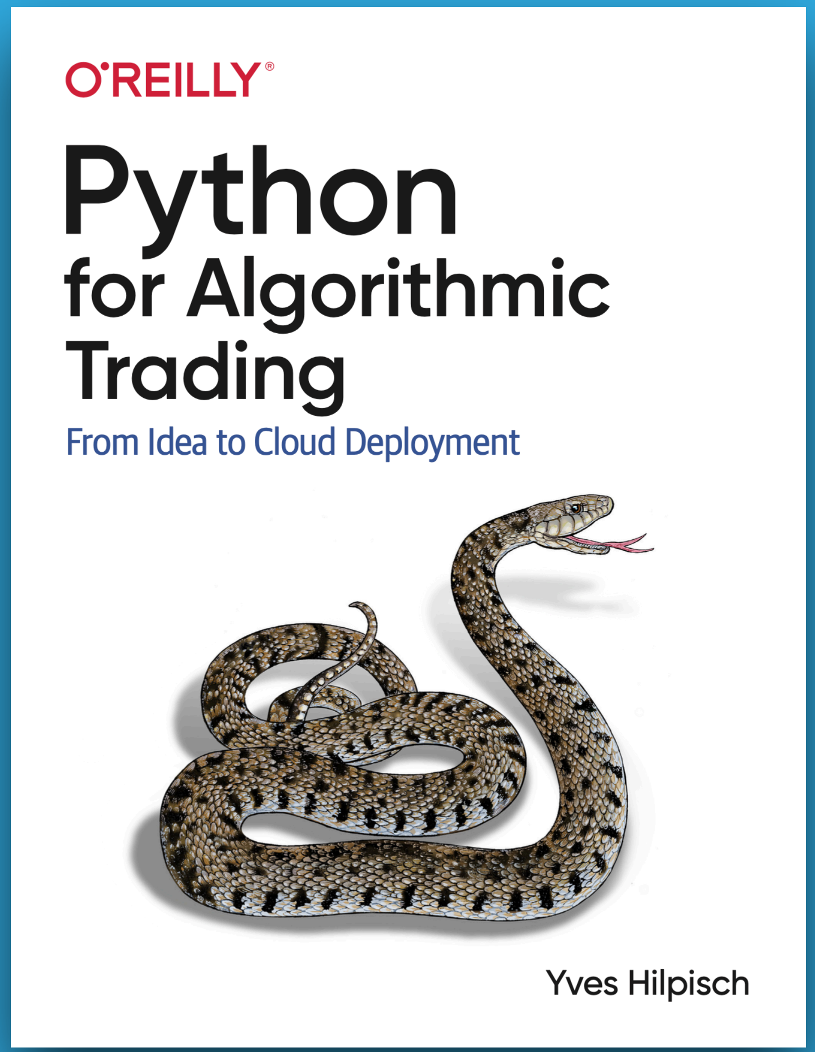
Introduction

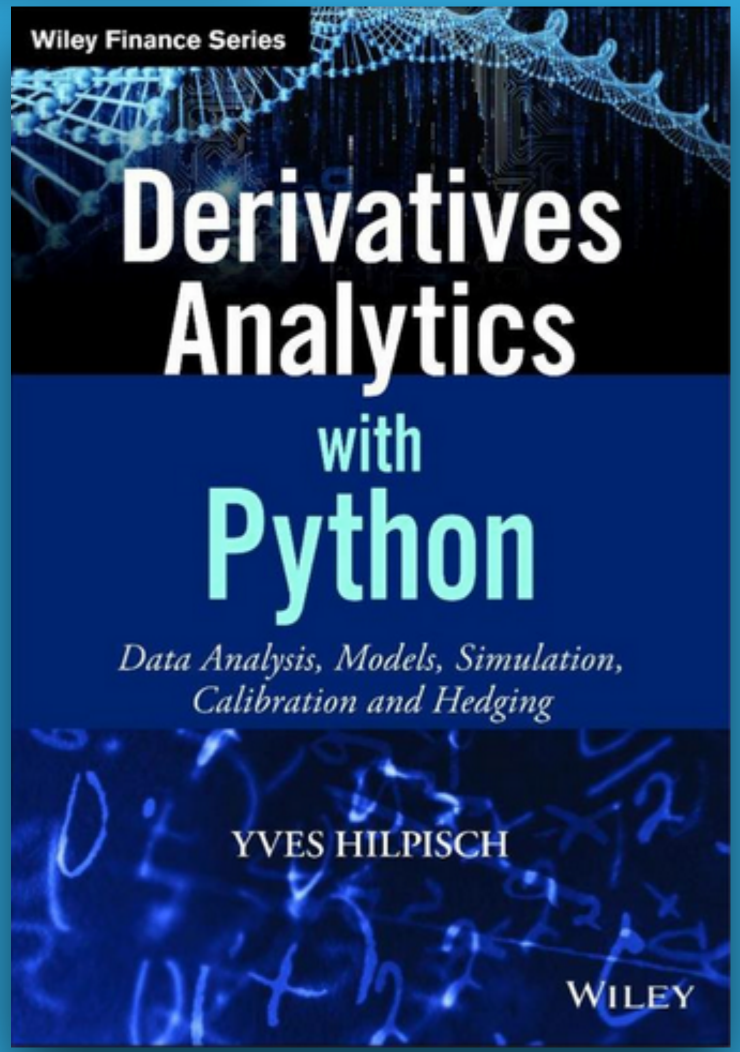
Python and AI for Finance

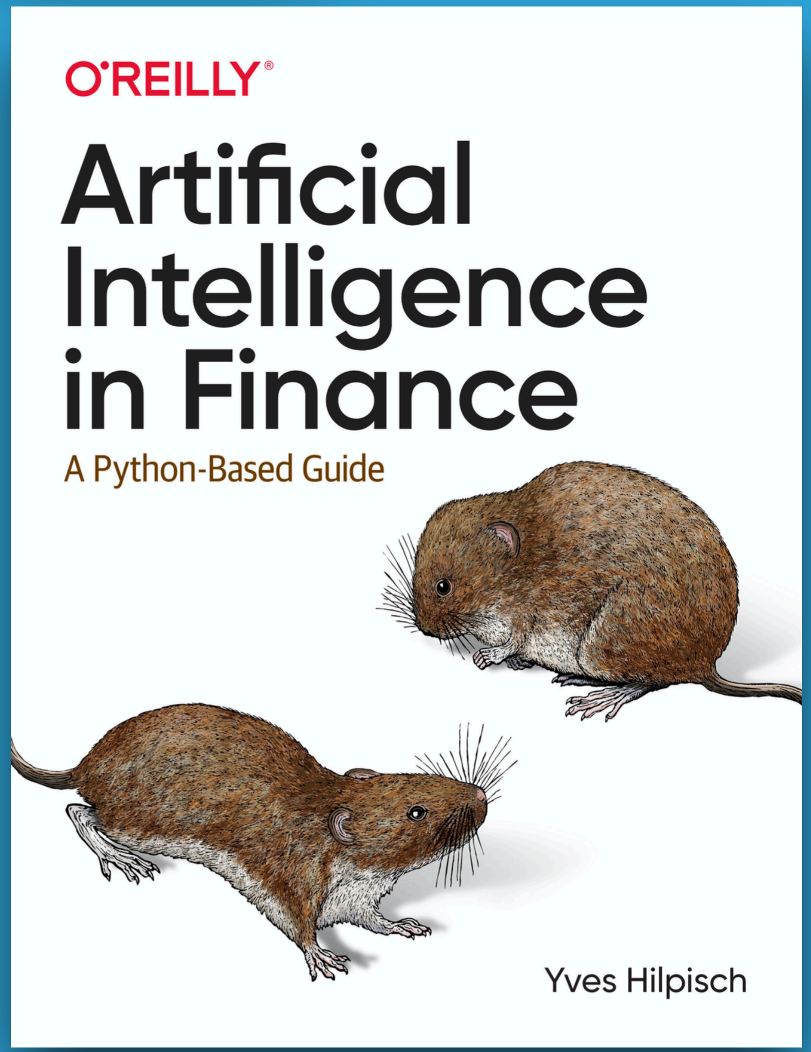
Since 2014 publishing about Python & AI for Quant Finance.

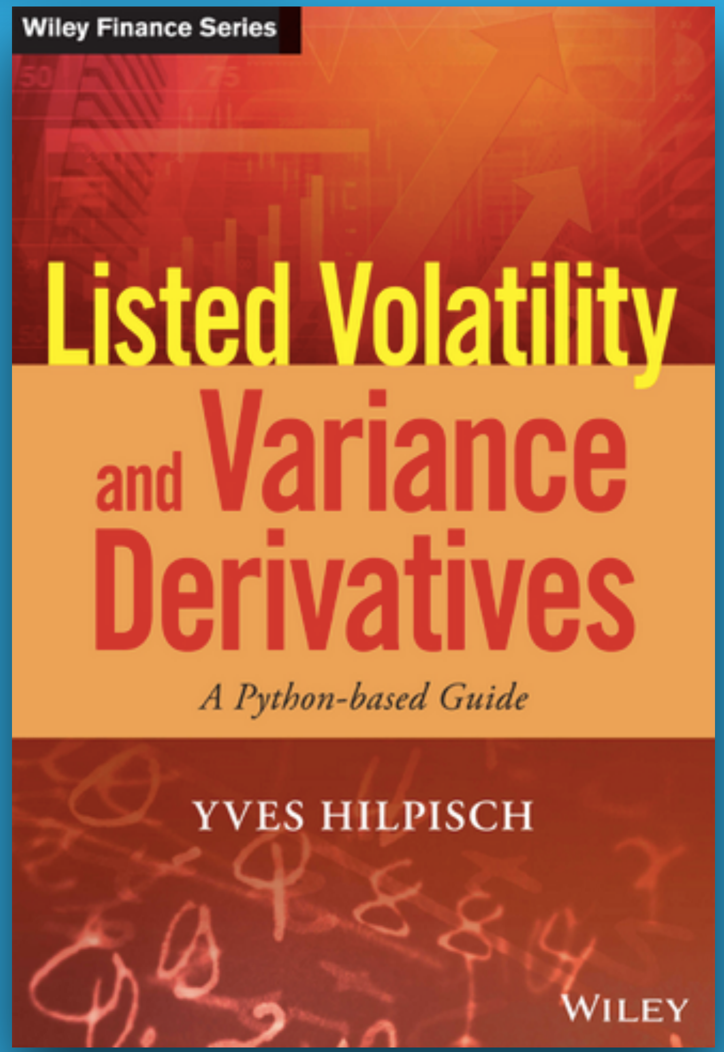












http://books.tpq.io

THE PYTHON QUANTS

CPF

Certificate in Python for Finance

(magna cum laude)

Peter Mueller, CFA

(born February 22, 1995)

June 19, 2023

Dr. Yves J. Hilpisch

CEO & Program Director

4-12 months program (live or self-paced)

330 hours of instruction → 2,000,000 words

7 Books → 2,750 pages PDF

500 Jupyter Notebooks → 50,000 LOC

350 PY Files → 35,000+ LOC

https://cpf.tpq.io

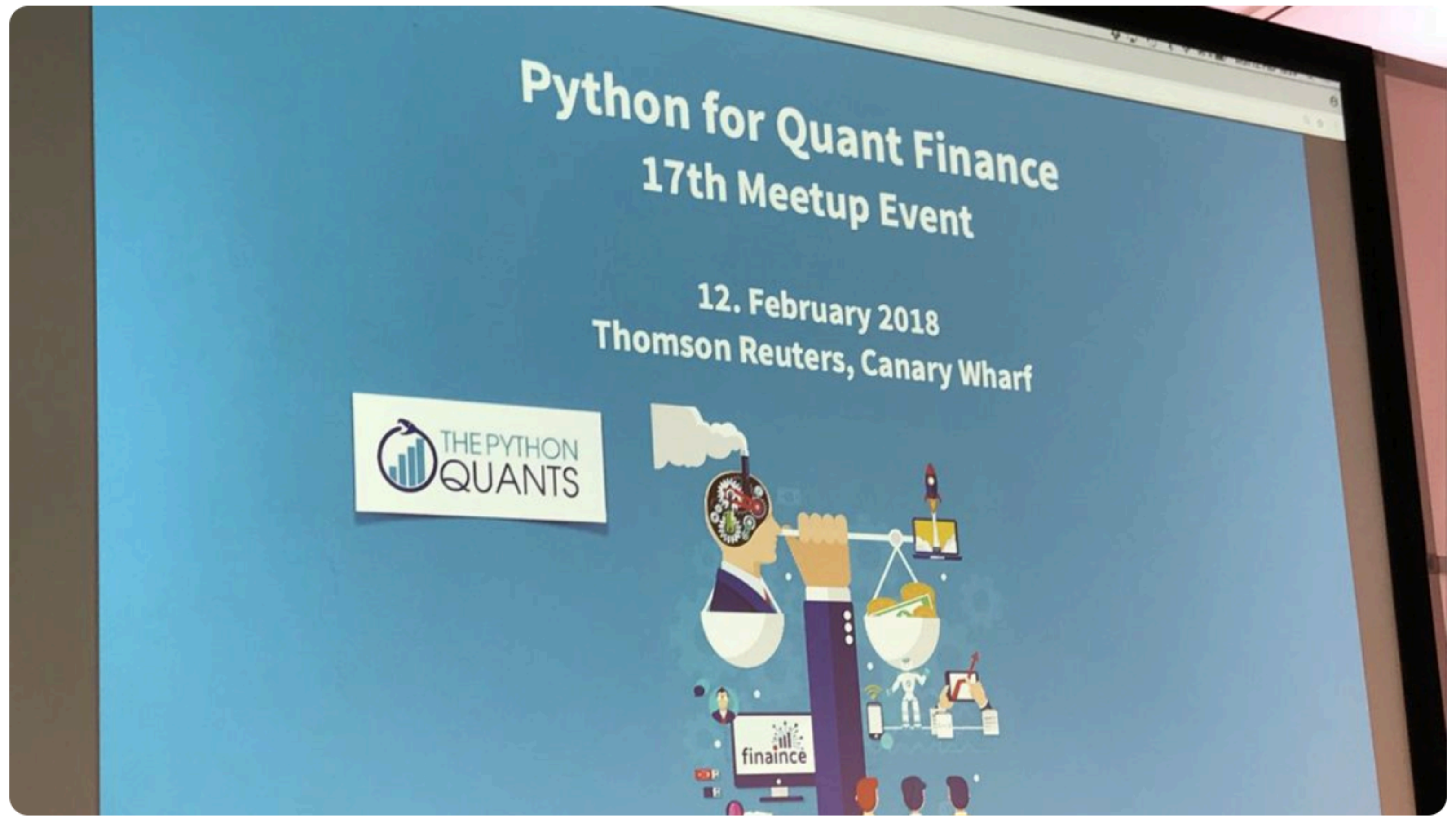
(pro)
quants@dev
~ $

Community of
professional & aspiring
quant developers &
quant researchers.

1,000+ Members
and growing.

Webinar series
“Reinforcement Learning
in Finance”

https://bit.ly/quants_dev



Python for Quant Finance

London, United Kingdom
3,416 members · Public group
Organized by **Yves H.** and **2 others**

Share:

About Events Members Photos Discussions More

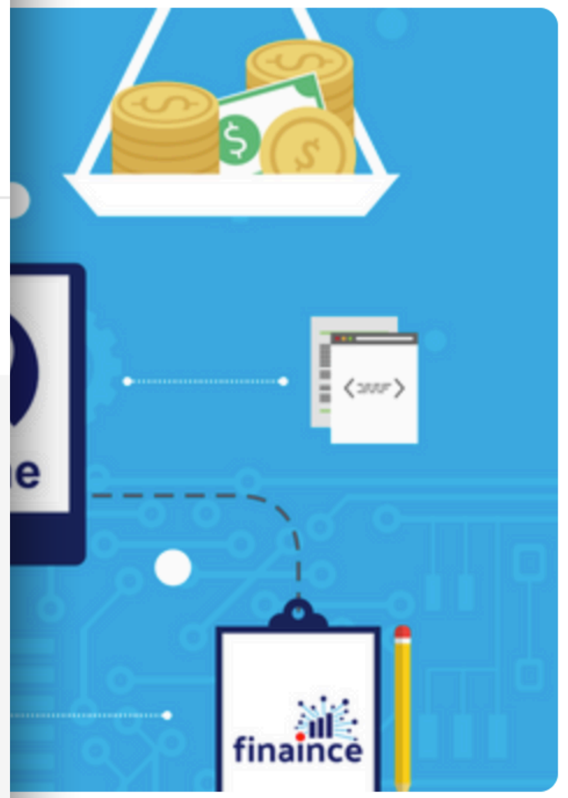
Join this group

What we're about

This group is about the use of Python & AI for Quantitative Financial Applications, Algorithmic Trading and Interactive Financial Analytics.

Organizers

Yves H. and **2 others**
Message



Artificial Intelligence in Finance & Algorithmic Trading

New York, NY
345 members · Public group
Organized by **Yves Hilpisch**

Share:

About Events Members Photos Discussions More

Manage group
Create event

What we're about

This Meetup group is concerned with data-driven and AI-first finance in general and algorithmic trading in particular. Its events cover the latest...

Organizer

Yves Hilpisch
Message

Dr. Yves J. Hilpisch is the founder and CEO of **The Python Quants** (http://tpq.io), a group focusing on the use of Python and open source technologies for financial data science, artificial intelligence, algorithmic trading, and computational finance. He is also the founder and CEO of **The AI Machine** (http://aimachine.io), a company focused on AI-powered algorithmic trading based on a proprietary strategy execution platform.

Yves has a Diploma in Business Administration, a Ph.D. in Mathematical Finance, and is Adjunct Professor for Computational Finance.

Yves is the author of six books (https://home.tpq.io/books):

* Reinforcement Learning for Finance (2024, O'Reilly)
* Finance with Python (2021, O'Reilly)
* Artificial Intelligence in Finance (2020, O'Reilly)
* Python for Algorithmic Trading (2020, O'Reilly)
* Python for Finance (2018, 2nd ed., O'Reilly)
* Listed Volatility and Variance Derivatives (2017, Wiley Finance)
* Derivatives Analytics with Python (2015, Wiley Finance)

Yves is the director of the online training program leading to the **Certificates in Python for Finance** (https://cpf.tpq.io). He also lectures on computational finance, reinforcement learning, and algorithmic trading at the **CQF Program** (http://cqf.com).

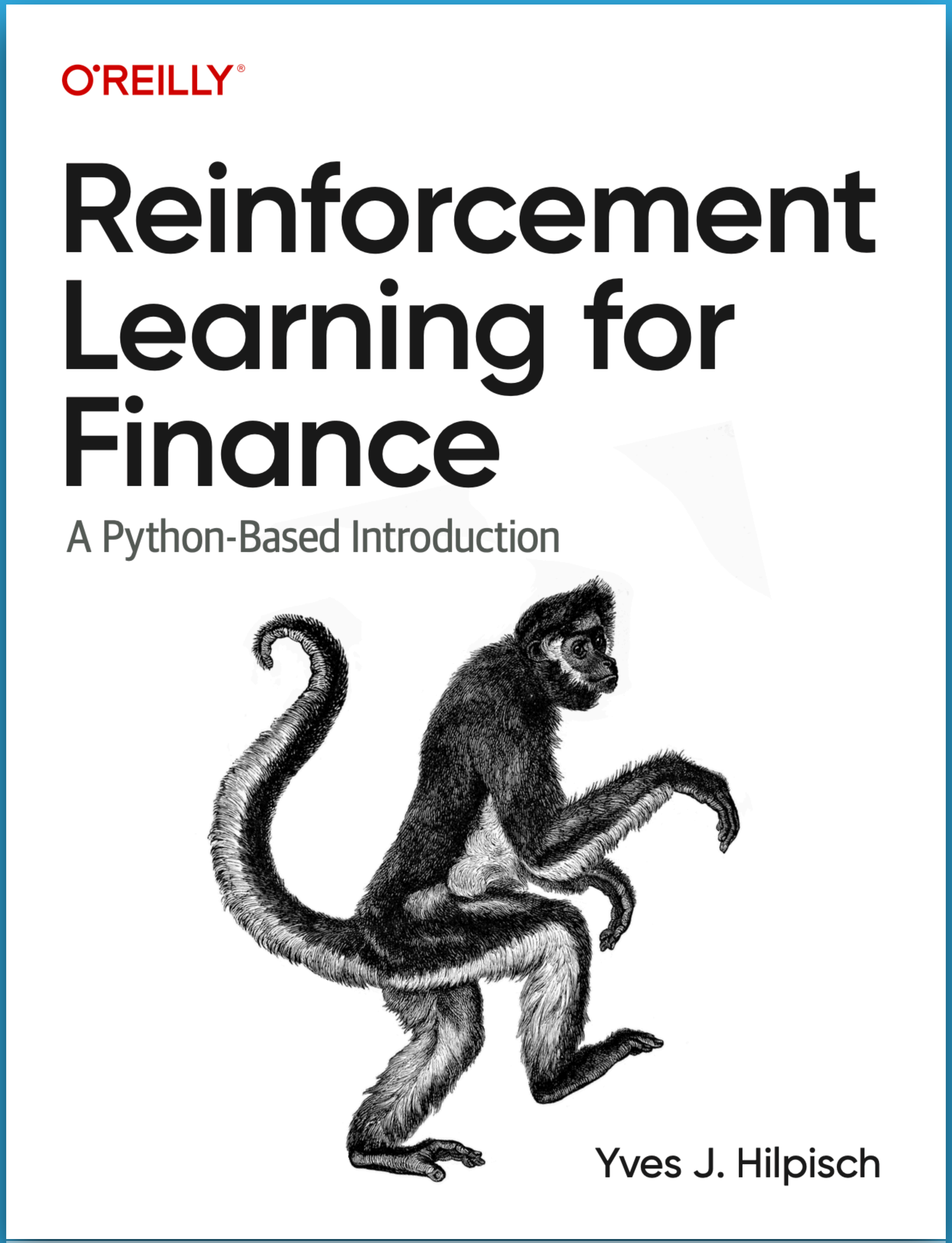
Yves is the originator of the financial analytics library **DX Analytics** (http://dx-analytics.com) and organizes Meetup group **events, conferences, and bootcamps** about Python, artificial intelligence and algorithmic trading in London (http://pqf.tpq.io) and New York (http://aifat.tpq.io). He has given **keynote speeches** at technology conferences in the United States, Europe, and Asia.

http://hilpisch.com

Overview

RL for Finance

A Python-based introduction with different applications.



http://books.tpq.io

Reinforcement Learning for Finance

The Basics
Learning through Interaction
Deep Q-Learning
Financial Q-Learning

Data Augmentation
Simulated Data
Generated Data

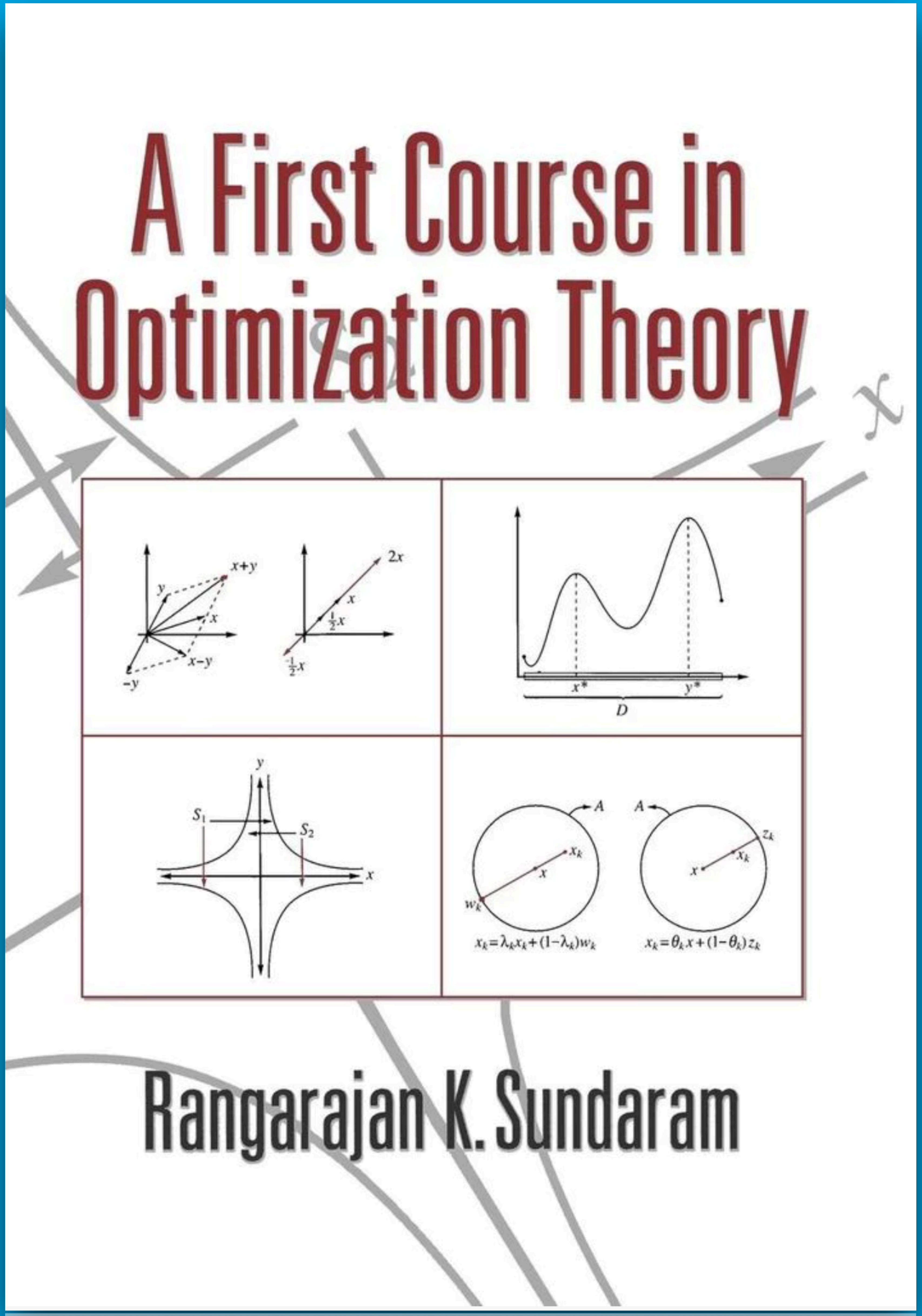
Applications
Algorithmic Trading
Dynamic Hedging
Dynamic Asset Allocation
Optimal Execution

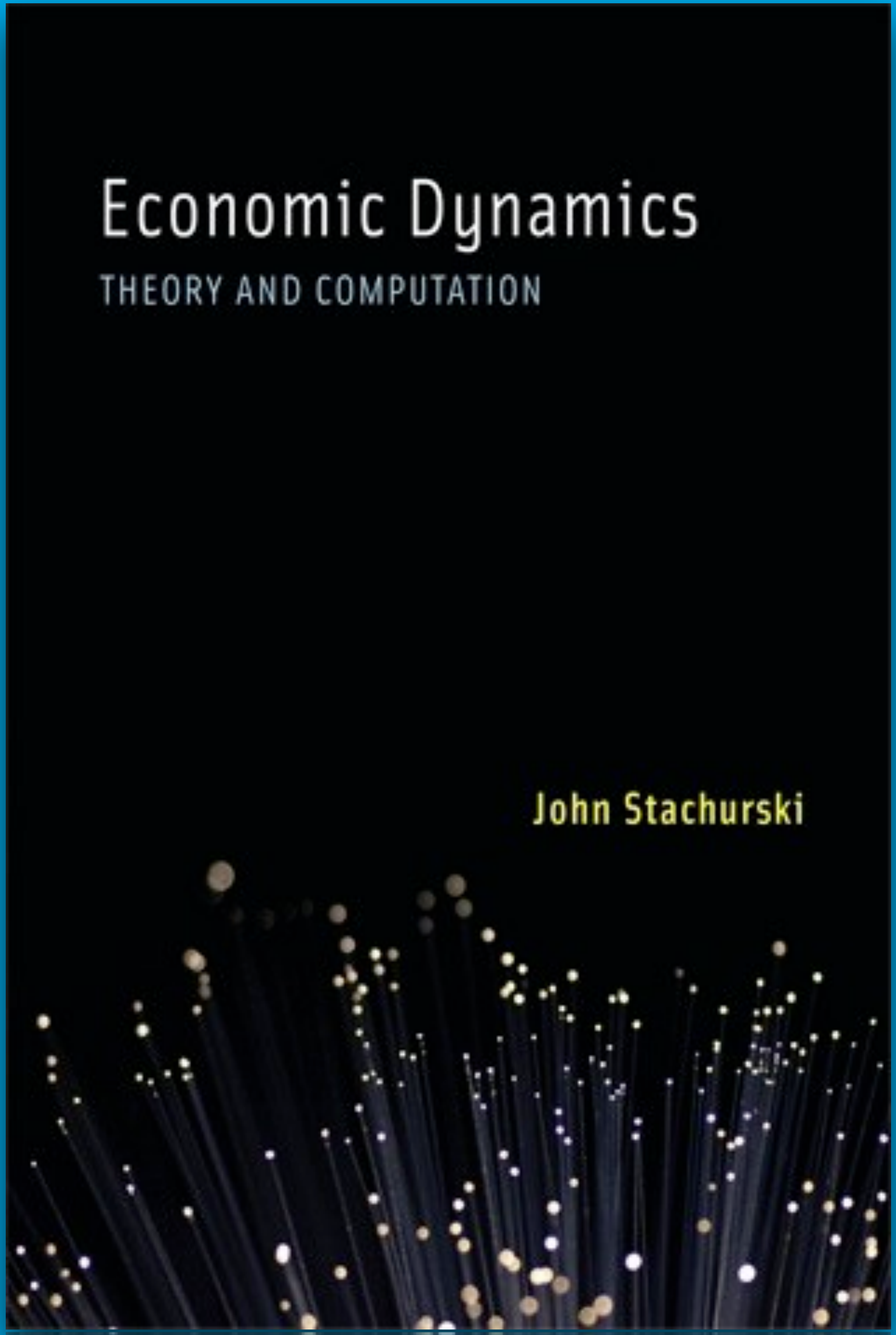
Auxiliary Topics
Deep Neural Networks
Optimization

Reinforcement Learning

https://bit.ly/odsc_ldn_2024

1) Dynamic Programming


A First Course in
Optimization Theory
Rangarajan K. Sundaram


Economic Dynamics
THEORY AND COMPUTATION
John Stachurski

A ***Finite Horizon (Markovian) Dynamic Programming Problem*** (FHDP) is defined by a tuple

$$\{S, A, T, (r_t, f_t, \Phi_t)_{t=0}^{T}\}$$

where

1. S is the *state space* of the problem, with generic element s.
2. A is the *action space* of the problem, with generic element a.
3. T, a positive integer, is the *horizon* of the problem.
4. For each $t \in \{0, 1, \ldots, T\}$
   A. $r_t : S \times A \to \mathbb{R}$ is the period-t *reward function*
   B. $f_t : S \times A \to S$ is the period-t *transition function*
   C. $\Phi_t : S \to P(A)$ is the period-t *feasible action correspondence*

See Sundaram (1996, pp. 268–269).

The objective is to choose a plan for taking actions at each point in time in order to maximize the sum of the per-period rewards over the horizon of the model, i.e. to solve

$$\text{Maximize} \quad \sum_{t=1}^{T} r_t(s_t, a_t)$$

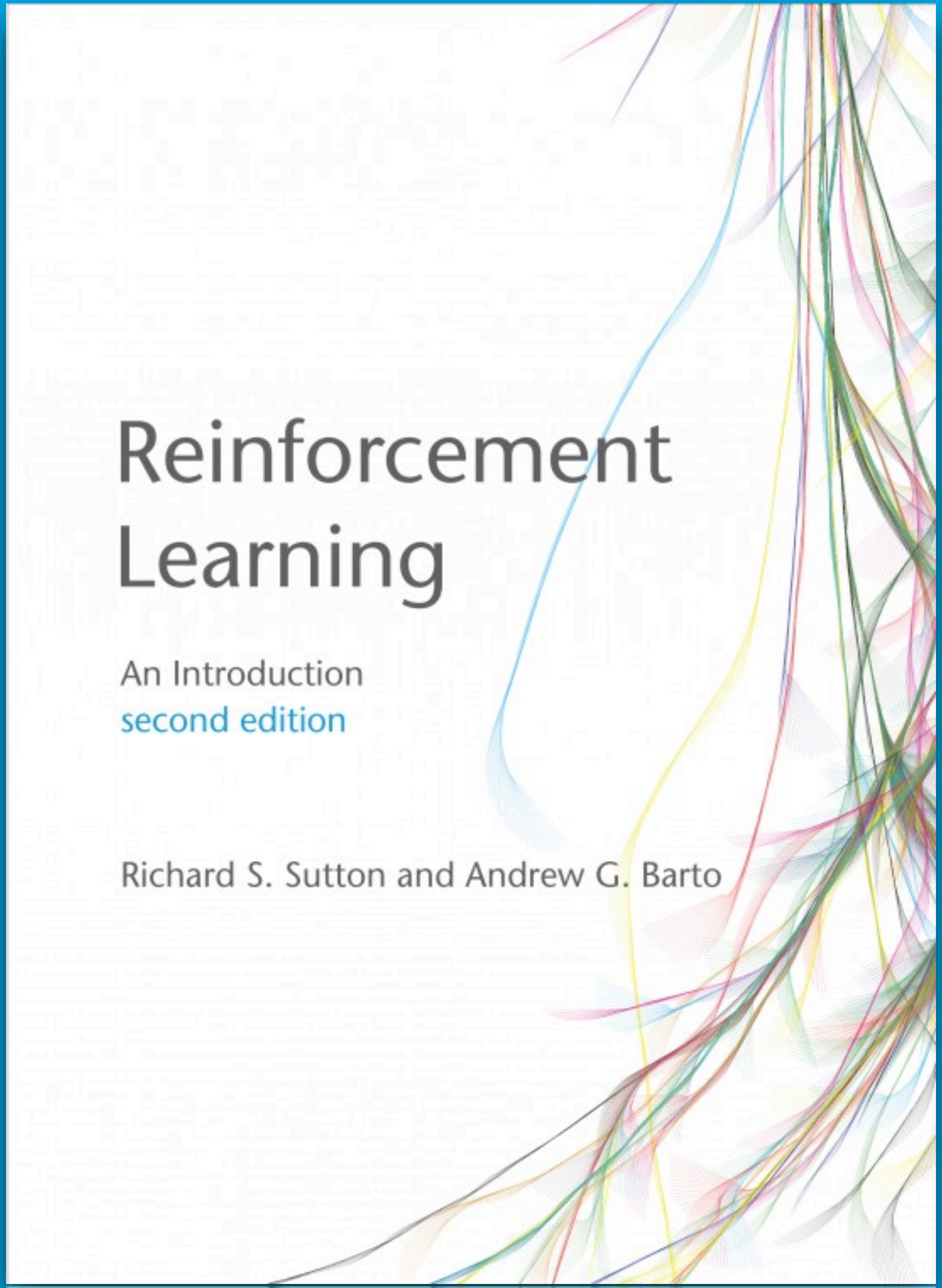
$$\text{subject to} \quad s_0 = s \in S$$

$$s_t = f_{t-1}(s_{t-1}, a_{t-1}), t = 1, \ldots, T$$

$$a_t \in \Phi_t(s_t), t = 1, \ldots, T$$

See Sundaram (1996, pp. 268–269).

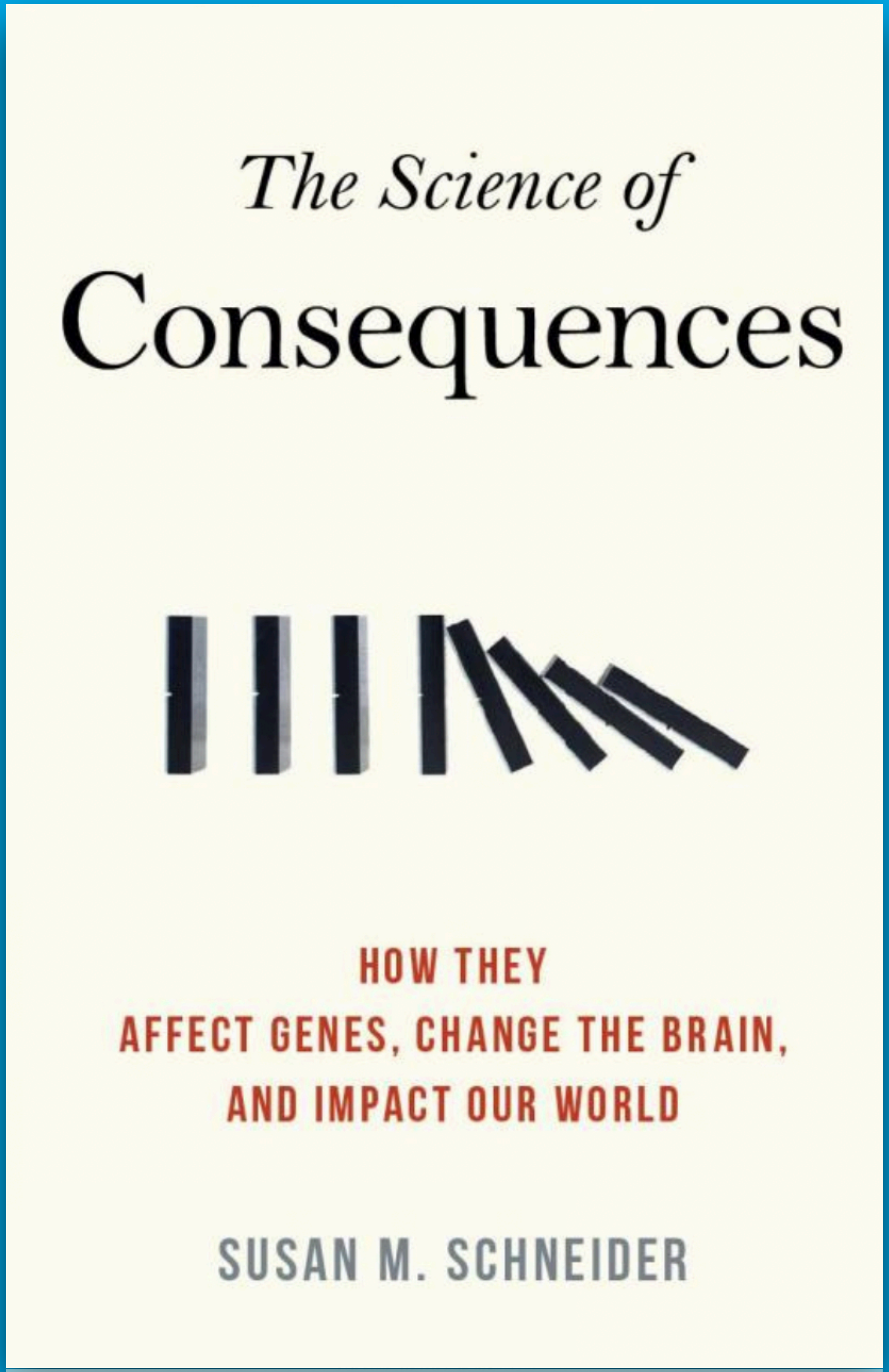
2) Basic Notions

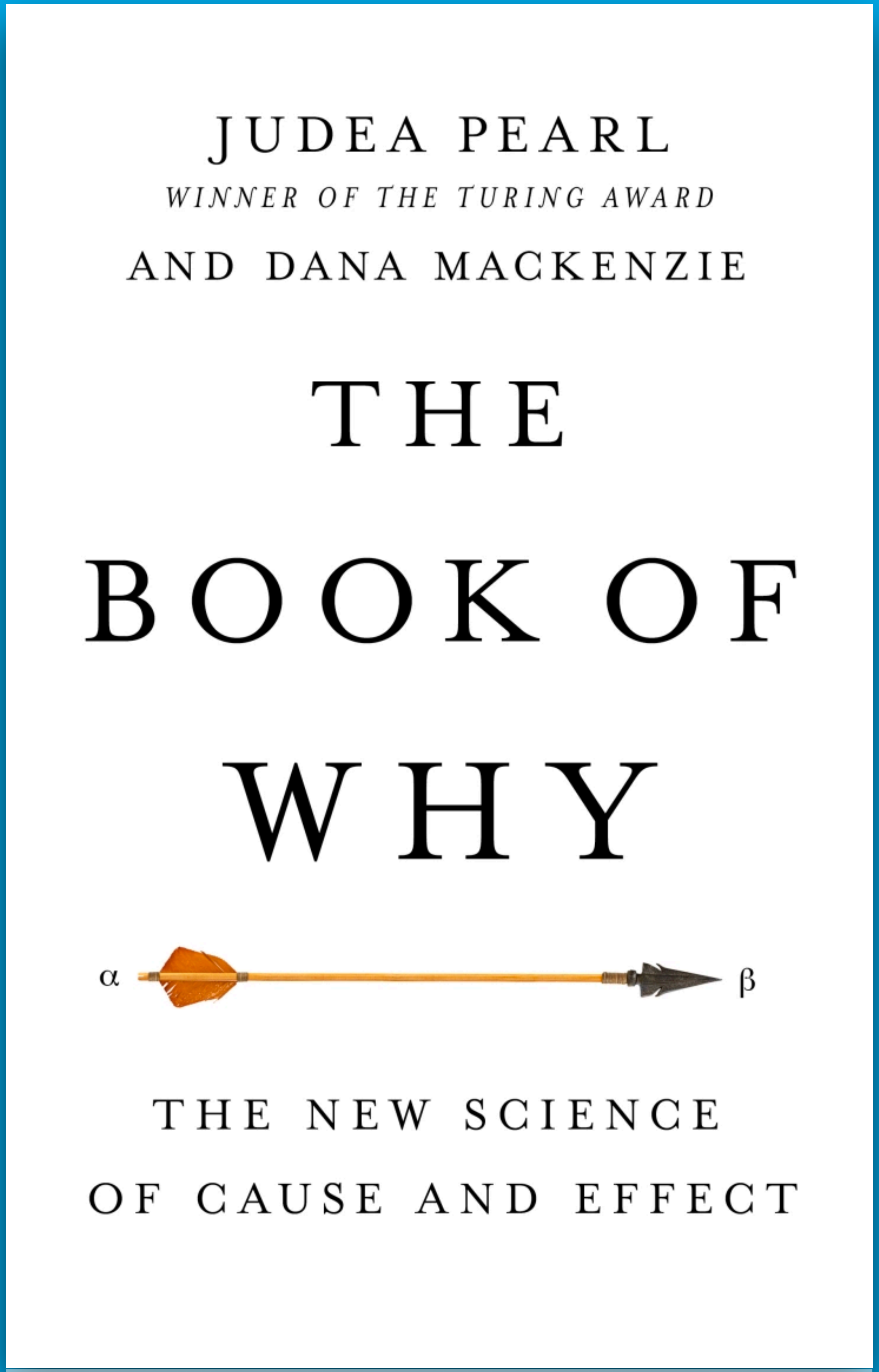


"Of all the forms of machine learning, reinforcement learning is the closest to the kind of learning that humans and other animals do, and many of the core algorithms of reinforcement learning were originally inspired by biological learning systems."

"The most important feature distinguishing reinforcement learning from other types of learning is that it uses training information that evaluates the actions taken rather than instructs by giving correct actions."

"Reinforcement learning is about learning from interaction how to behave in order to achieve a goal. The reinforcement learning agent and its environment interact over a sequence of discrete time steps."


The Science of
Consequences
HOW THEY
AFFECT GENES, CHANGE THE BRAIN,
AND IMPACT OUR WORLD
SUSAN M. SCHNEIDER


JUDEA PEARL
WINNER OF THE TURING AWARD
AND DANA MACKENZIE
THE
BOOK OF
WHY
α
β
THE NEW SCIENCE
OF CAUSE AND EFFECT

Environment
The ***environment*** defines the problem at hand. This can be a computer game to be played or a financial market to be traded in.

State
A ***state*** subsumes all relevant parameters that describe the current status of the environment. In a computer game this might be the whole screen with all its pixels. In a financial market, this might include current and historical price levels, financial indicators such as moving averages, macroeconomic variables, and so on.

Agent

The term ***agent*** subsumes all elements of the RL algorithm that interacts with the environment and that learns from these interactions. In a gaming context, the agent might represent a player playing the game. In a financial context, the agent could represent a trader placing bets on rising or falling markets.

Action

An agent can choose one ***action*** from a (limited) set of allowed actions. In a computer game, movements to the left or right might be allowed actions, while in a financial market going long or short could be admissible.

Step

Given an action of an agent, the state of the environment is updated. One such update is generally called a ***step***. The concept of a step is general enough to encompass both heterogeneous and homogeneous time intervals between two steps. While in computer games, real-time interaction with the game environment is simulated by rather short, homogeneous time intervals ("game clock"), a trading bot interacting with a financial market environment could take actions at longer, heterogeneous time intervals, for instance.

Reward
Depending on the action an agent chooses, a ***reward*** (or ***penalty***) is awarded. For a computer game, points are a typical reward. In a financial context, profit (or loss) is a standard reward.

Target
The ***target*** specifies what the agent tries to maximize. In a computer game, this in general is the score reached by the agent. For a financial trading bot, this might be the trading profit.

Policy
The ***policy*** defines which action an agent takes given a certain state of the environment. Given a certain state of a computer game, represented by all the pixels that make up the current scene, the policy might specify that the agent chooses “move right” as the action. A trading bot that observes three price increases in a row might decide, according to its policy, to short the market.

Episode
An ***episode*** is a set of steps from the initial state of the environment until success is achieved or failure is observed. In a game, from the start of the game until a win or loss. In the financial world, for example, from the beginning of the year to the end of the year or to bankruptcy.

3) Deep Q–Learning

Reward Function
The reward function R assigns to each state-action (S, A) pair a numerical reward.

$$R : S \times A \to \mathbb{R}$$

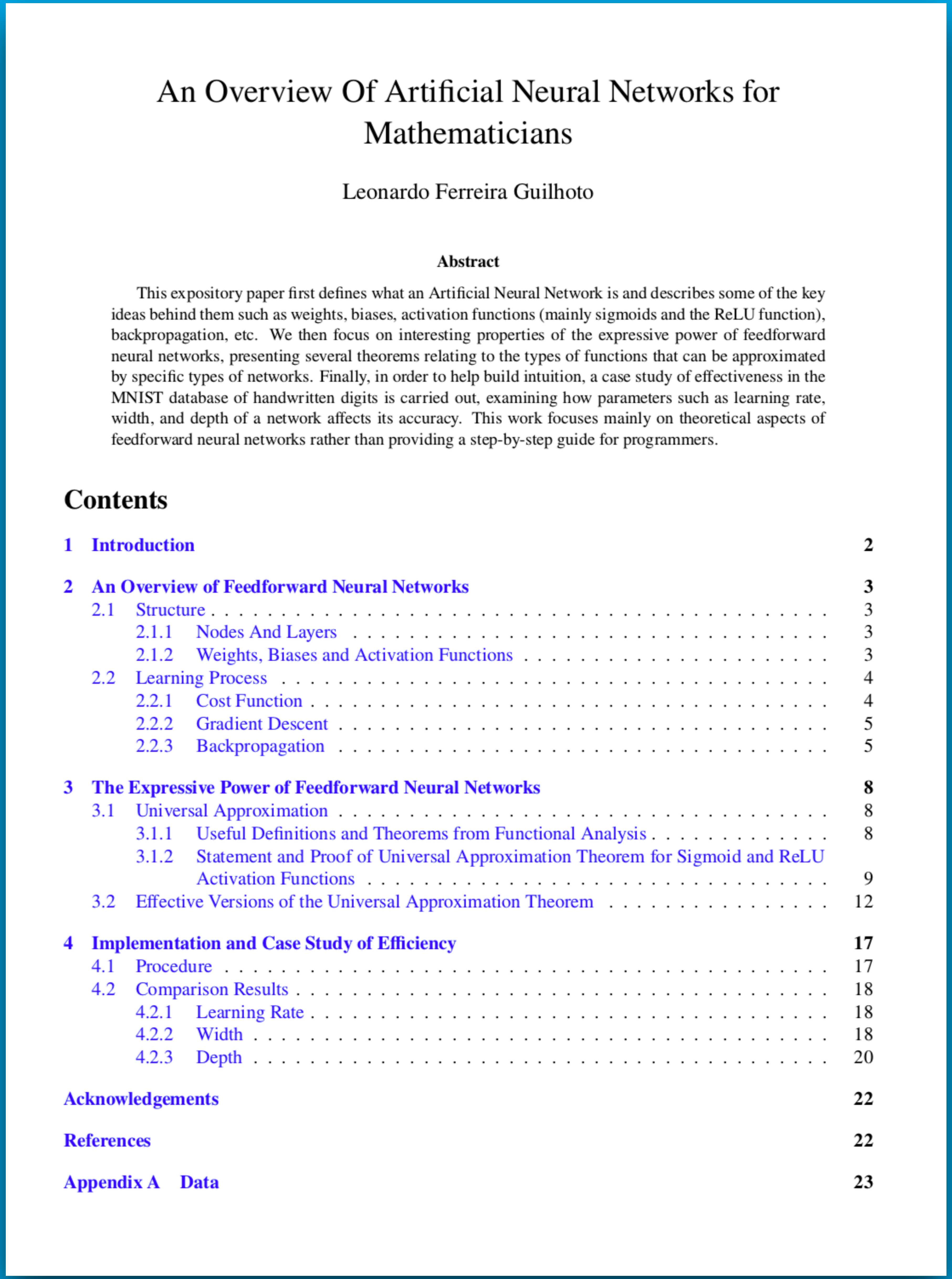
Action Policy
An action policy Q assigns to each state S and allowed action A a numerical value. The numerical value is composed of the **immediate reward** of taking action A and the **discounted delayed reward** – given an optimal action taken in the subsequent state.

$$Q : S \times A \to \mathbb{R},$$

$$Q\left(S_t, A_t\right) = R\left(S_t, A_t\right) + \gamma \cdot \max_a Q\left(S_{t+1}, a\right)$$

Representation
In general, the optimal action policy Q can not be specified in closed form (e.g. in the form of a table). Therefore, Q–learning relies in general on approximate representations for the optimal policy Q.

Neural Network
Due to the approximation capabilities of neural networks (“Universal Approximation Theorems”), neural networks are typically used to represent optimal action policies Q. Features are the parameters that describe the state of the environment. Labels are values attached to each allowed action.



An Overview Of Artificial Neural Networks for Mathematicians

Leonardo Ferreira Guilhoto

Abstract

This expository paper first defines what an Artificial Neural Network is and describes some of the key ideas behind them such as weights, biases, activation functions (mainly sigmoids and the ReLU function), backpropagation, etc. We then focus on interesting properties of the expressive power of feedforward neural networks, presenting several theorems relating to the types of functions that can be approximated by specific types of networks. Finally, in order to help build intuition, a case study of effectiveness in the MNIST database of handwritten digits is carried out, examining how parameters such as learning rate, width, and depth of a network affects its accuracy. This work focuses mainly on theoretical aspects of feedforward neural networks rather than providing a step-by-step guide for programmers.

Contents

1 Introduction 2

2 An Overview of Feedforward Neural Networks 3
2.1 Structure . . . 3
2.1.1 Nodes And Layers . . . 3
2.1.2 Weights, Biases and Activation Functions . . . 3
2.2 Learning Process . . . 4
2.2.1 Cost Function . . . 4
2.2.2 Gradient Descent . . . 5
2.2.3 Backpropagation . . . 5

3 The Expressive Power of Feedforward Neural Networks 8
3.1 Universal Approximation . . . 8
3.1.1 Useful Definitions and Theorems from Functional Analysis . . . 8
3.1.2 Statement and Proof of Universal Approximation Theorem for Sigmoid and ReLU Activation Functions . . . 9
3.2 Effective Versions of the Universal Approximation Theorem . . . 12

4 Implementation and Case Study of Efficiency 17
4.1 Procedure . . . 17
4.2 Comparison Results . . . 18
4.2.1 Learning Rate . . . 18
4.2.2 Width . . . 18
4.2.3 Depth . . . 20

Acknowledgements 22

References 22

Appendix A Data 23

“In the mathematical theory of artificial neural networks, the universal approximation theorem states that a feed-forward network with a single hidden layer containing a finite number of neurons can approximate continuous functions on compact subsets of R^n, under mild assumptions on the activation function. The theorem thus states that simple neural networks can represent a wide variety of interesting functions when given appropriate parameters; however, it does not touch upon the algorithmic learnability of those parameters.”
—https://en.wikipedia.org/wiki/Universal_approximation_theorem

Exploration
This refers to actions taken by an agent that are random in nature. The purpose is to explore random actions and their associated values beyond what the current optimal policy would dictate.

Exploitation
This refers to actions taken in accordance with the current optimal policy.

Replay
This refers to the (regular) updating of the optimal action policy given past and memorized experiences (by re-training the neural network).

gamma
The parameter `gamma` represents the discount factor by which delayed rewards are taken into account.

epsilon
The parameter `epsilon` defines the ratio with which the algorithm relies on exploration as compared to exploitation.

epsilon_decay
The parameter `epsilon_decay` specifies the rate at which `epsilon` is reduced.

The Python Quants GmbH

Dr. Yves J. Hilpisch
http://tpq.io | training@tpq.io
@dyjh