Reinforcement Learning in Finance—
From Playing Games to Algorithmic Trading

Dr. Yves J. Hilpisch

QUANTS@DEV WEBINAR

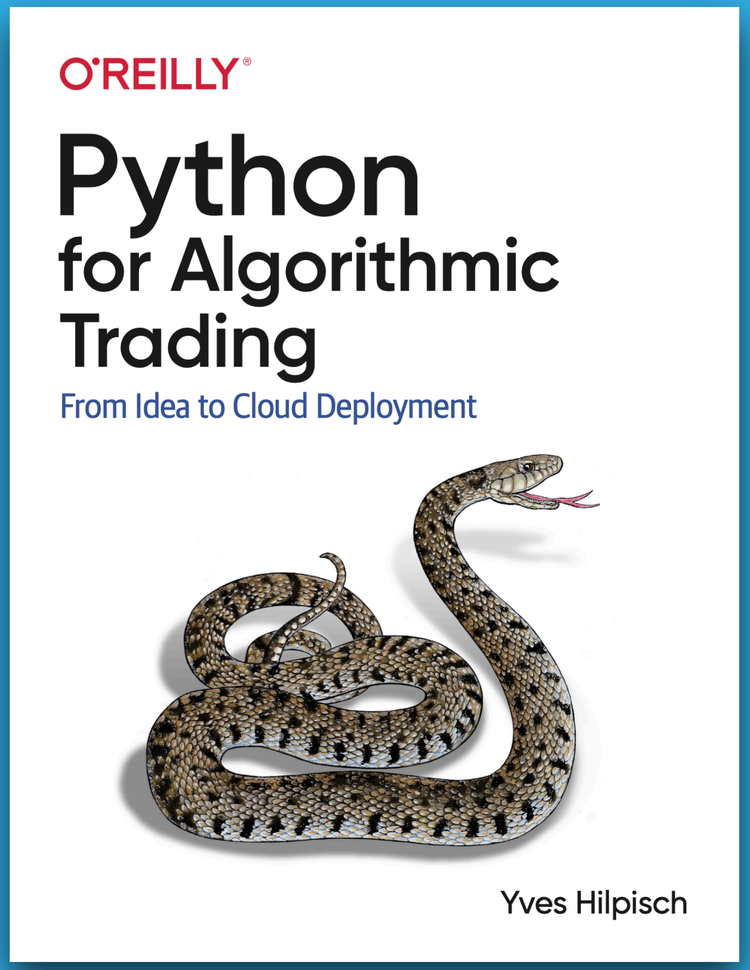
(pro)
quants@dev
~ $

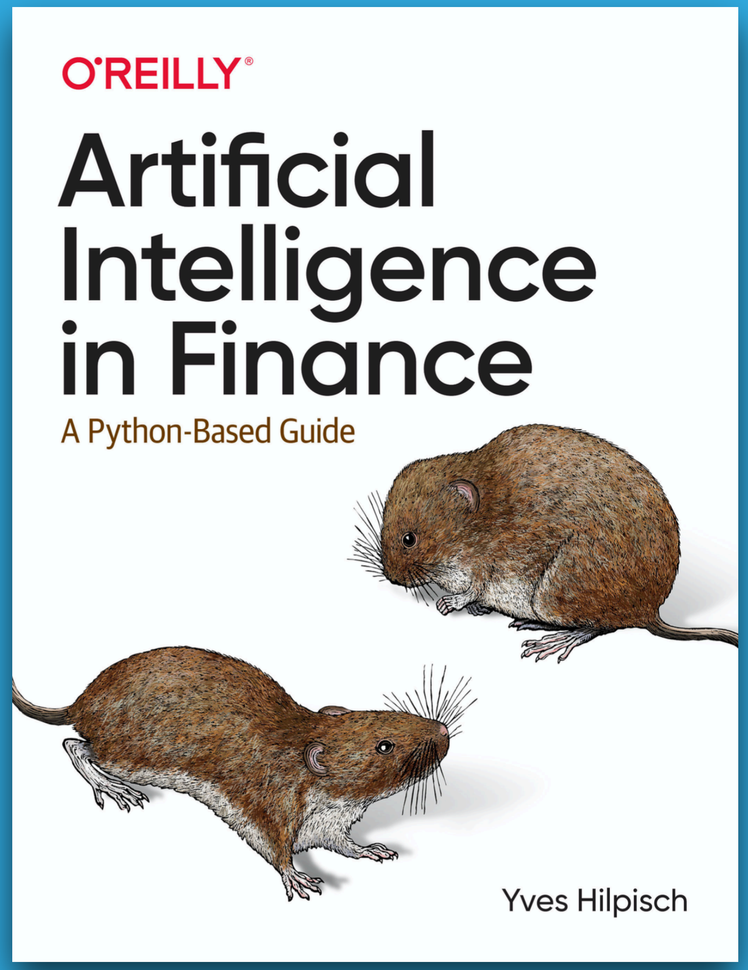
The AI Machine

THE PYTHON QUANTS

Introduction

Python & AI for Finance & Trading





http://books.tpq.io

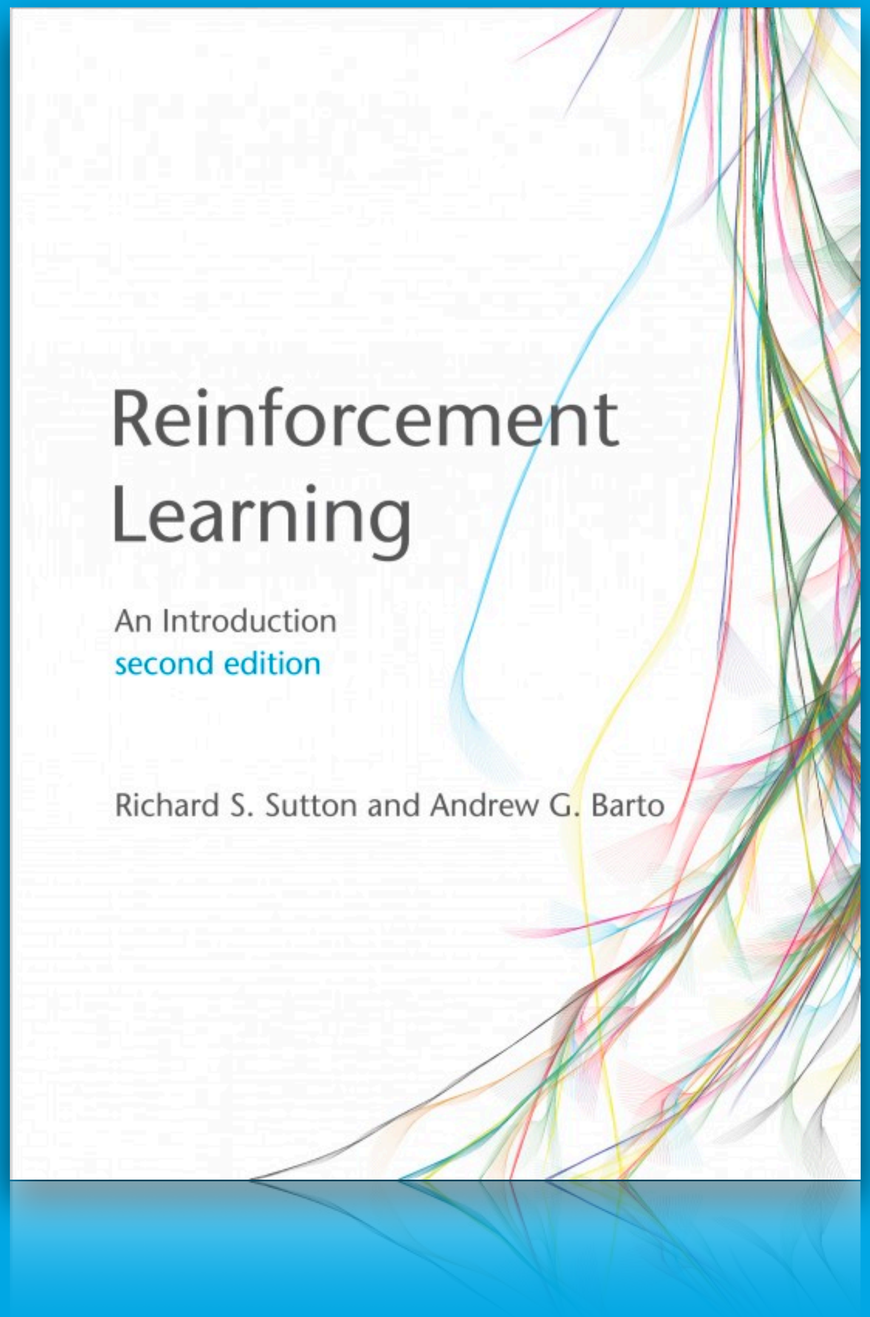
Github Repo with Code Resources:
https://github.com/yhilpisch/rlfinance

Google Colab:
https://colab.research.google.com/github/yhilpisch/rlfinance

Join the Discord Server:
https://discord.gg/uJPtp9Awaj

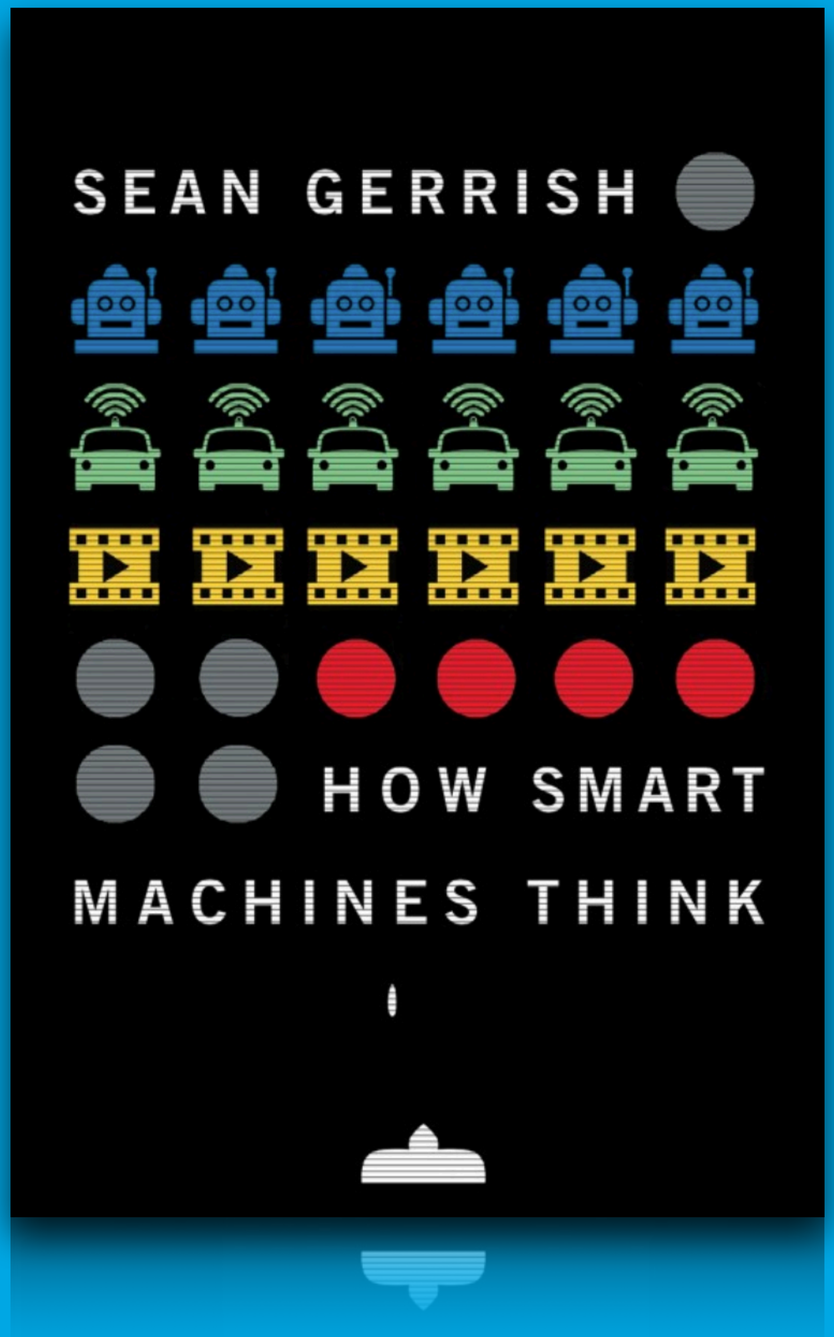
Follow on Twitter:
https://twitter.com/quants_dev

Reinforcement Learning —Success Stories



Reinforcement Lerning –An Introduction

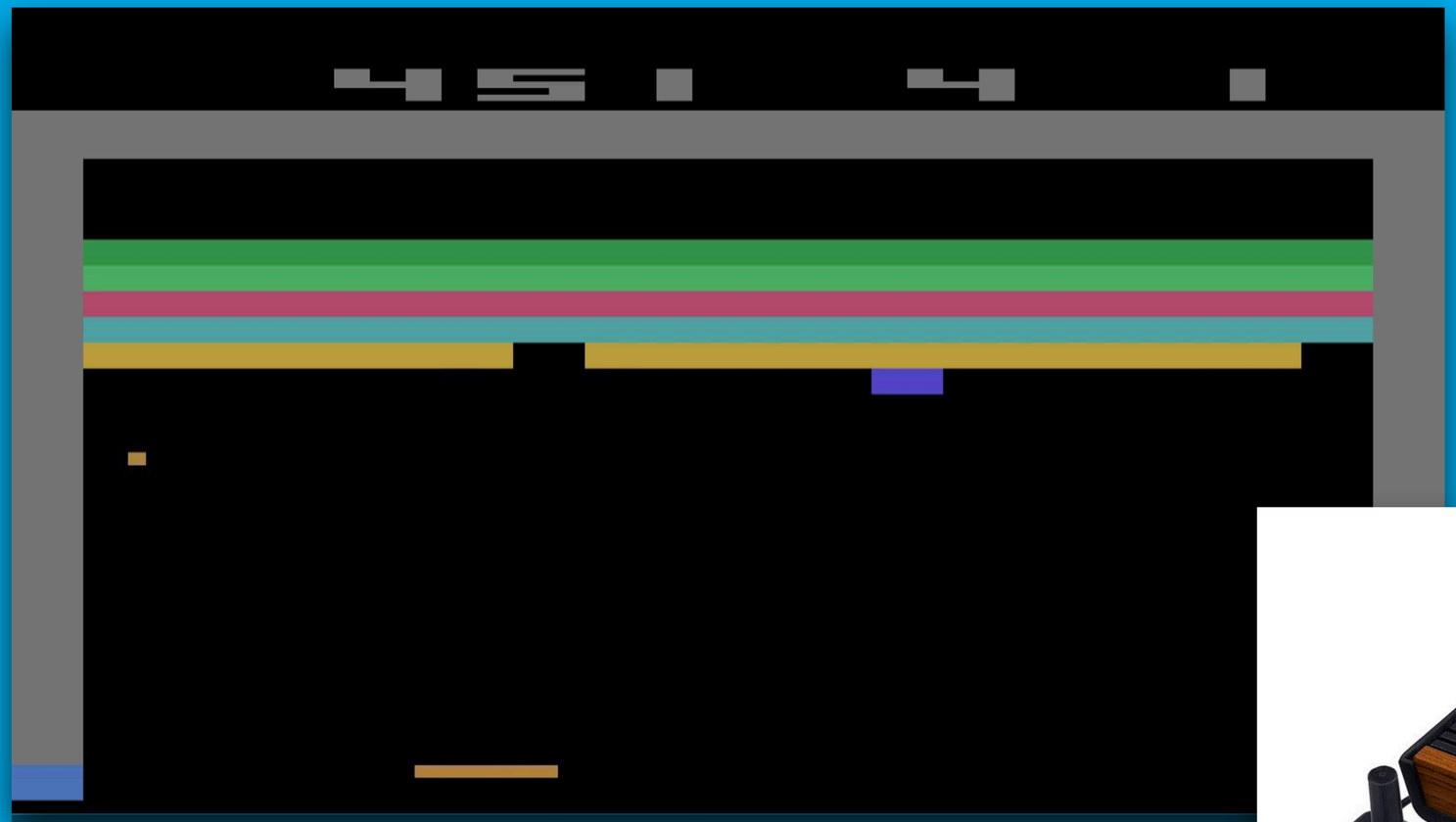
- Tabular Methods
- Finite Markov Decision Process
- Dynamic Programming
- Monte Carlo Simulation
- Temporal Difference Learning
- On– & Off Policy Approximations
- …
- ***(Deep) Q–Learning***



Success Stories about Deep Learning and Deep Reinforcement Learning:

- Self-Driving Cars
- Recommendation Engines
- Playing Atari Games
- Image Recognition & Classification
- Speech Recognition
- Playing the Game of Go

RL Success Stories —Atari Games and Reinforcement Learning


45 4


POWER
ON
OFF
COLOR
B-W
TV TYPE
VIDEO COMPUTER SYSTEM™
GAME SELECT
GAME RESET
ATARI®

“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 Learning Environment, with no adjustment of the architecture or learning algorithm. We find that it outperforms all previous approaches on six of the games and surpasses a human expert on three of them.”

Mnih, V. (2013): “Playing Atari with Deep Reinforcement Learning”. https://arxiv.org/pdf/1312.5602v1.pdf

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 Learning Environment, with no adjustment of the architecture or learning algorithm. We find that it outperforms all previous approaches on six of the games and surpasses a human expert on three of them.

1 Introduction

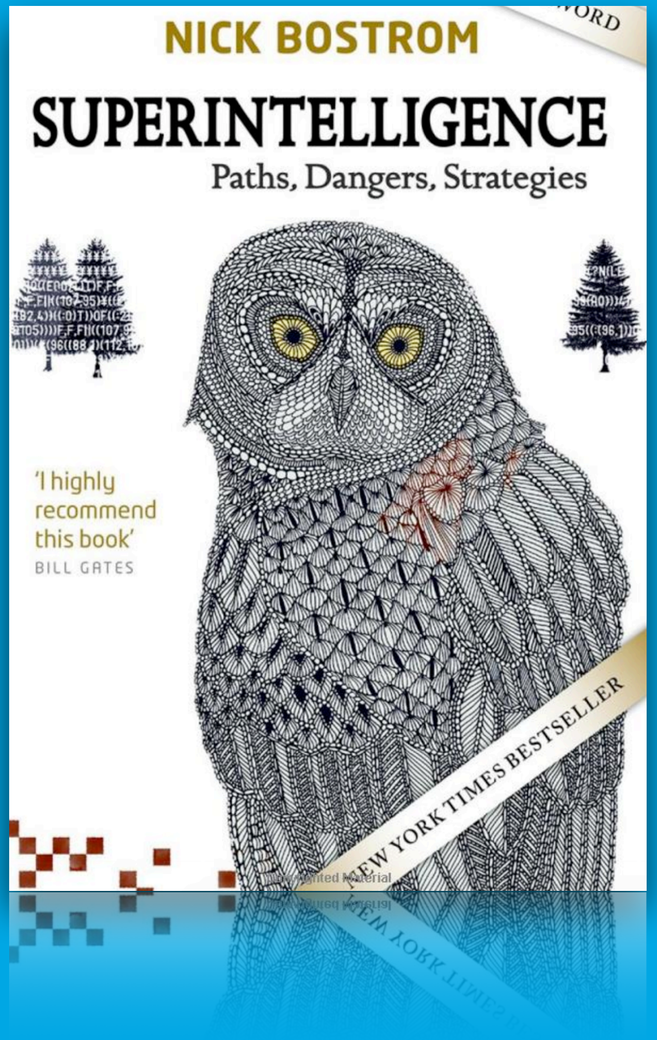
Learning to control agents directly from high-dimensional sensory inputs like vision and speech is one of the long-standing challenges of reinforcement learning (RL). Most successful RL applications that operate on these domains have relied on hand-crafted features combined with linear value functions or policy representations. Clearly, the performance of such systems heavily relies on the quality of the feature representation.

Recent advances in deep learning have made it possible to extract high-level features from raw sensory data, leading to breakthroughs in computer vision [11, 22, 16] and speech recognition [6, 7]. These methods utilise a range of neural network architectures, including convolutional networks, multilayer perceptrons, restricted Boltzmann machines and recurrent neural networks, and have exploited both supervised and unsupervised learning. It seems natural to ask whether similar techniques could also be beneficial for RL with sensory data.

However reinforcement learning presents several challenges from a deep learning perspective. Firstly, most successful deep learning applications to date have required large amounts of hand-labelled training data. RL algorithms, on the other hand, must be able to learn from a scalar reward signal that is frequently sparse, noisy and delayed. The delay between actions and resulting rewards, which can be thousands of timesteps long, seems particularly daunting when compared to the direct association between inputs and targets found in supervised learning. Another issue is that most deep learning algorithms assume the data samples to be independent, while in reinforcement learning one typically encounters sequences of highly correlated states. Furthermore, in RL the data distribution changes as the algorithm learns new behaviours, which can be problematic for deep learning methods that assume a fixed underlying distribution.

This paper demonstrates that a convolutional neural network can overcome these challenges to learn successful control policies from raw video data in complex RL environments. The network is trained with a variant of the Q-learning [26] algorithm, with stochastic gradient descent to update the weights. To alleviate the problems of correlated data and non-stationary distributions, we use

RL Success Stories —Go and AlphaGo



"Go-playing programs have been improving at a rate of about 1 dan/year in recent years. If this rate of improvement continues, they might beat the human world champion in about a decade."

Nick Bostrom (2014): Superintelligence.

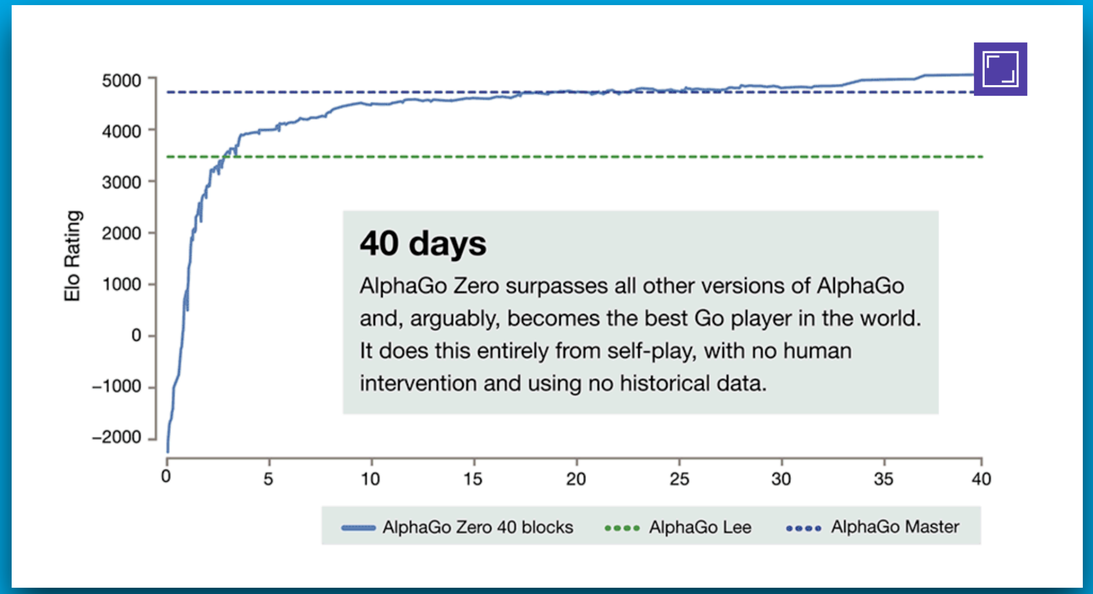
The story of AlphaGo so far

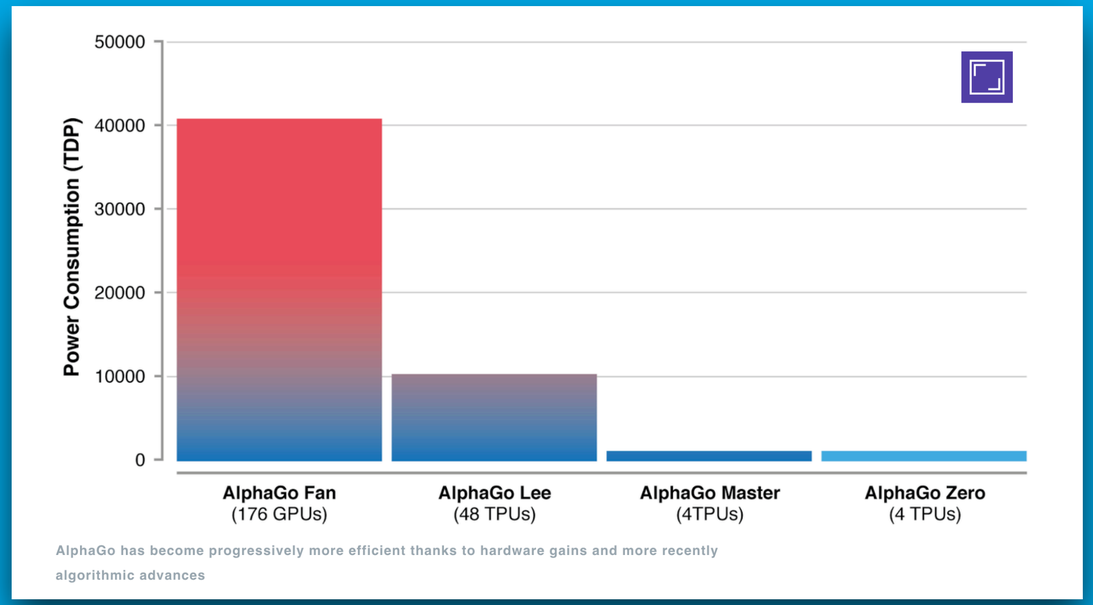
AlphaGo is the first computer program to defeat a professional human Go player, the first program to defeat a Go world champion, and arguably the strongest Go player in history.

AlphaGo's first formal match was against the reigning 3-times European Champion, Mr Fan Hui, in October 2015. Its 5-0 win was the first ever against a Go professional, and the results were published in full technical detail in the international journal, **Nature**. AlphaGo then went on to compete against legendary player Mr Lee Sedol, winner of 18 world titles and widely considered to be the greatest player of the past decade.

AlphaGo's 4-1 victory in Seoul, South Korea, in March 2016 was watched by over 200 million people worldwide. It was a landmark achievement that experts agreed was a decade ahead of its time, and earned AlphaGo a 9 dan professional ranking (the highest certification) - the first time a computer Go player had ever received the accolade.

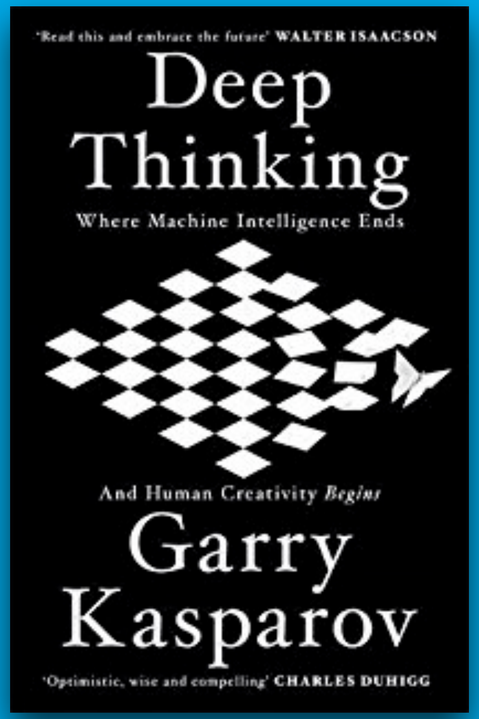
During the games, AlphaGo played a handful of **highly inventive winning moves,** several of which - including move 37 in game two - were so surprising they overturned hundreds of years of received wisdom, and have since been examined extensively by players of all levels. In the course of winning, AlphaGo somehow taught the world completely new knowledge about perhaps the most studied and contemplated game in history.





AlphaGo has become progressively more efficient thanks to hardware gains and more recently algorithmic advances

RL Success Stories
—Chess, Deep Blue & AlphaZero



"It was a pleasant day in Hamburg in June 6, 1985, … Each of my opponents, all thirty-two of them, was a computer. … it didn't come as much of a surprise, …, when I achieved

"Twelve years later I was in New York City fighting for my chess life. Against just one machine, a $10 million IBM supercomputer nicknamed 'Deep Blue'."

"Jump forward another 20 years to today, to 2017, and you can download any number of free chess apps for your phone that rival any human Grandmaster."

AlphaZero: Shedding new light on the grand games of chess, shogi and Go

"Traditional chess engines – including the world computer chess champion Stockfish and IBM's ground-breaking Deep Blue – rely on **thousands of rules and heuristics handcrafted by strong human players** that try to account for every eventuality in a game. …

AlphaZero takes a totally different approach, replacing these hand-crafted rules with a **deep neural network** and **general purpose algorithms** that know nothing about the game beyond the basic rules."

"The amount of **training** the network needs depends on the style and complexity of the game, taking **approximately 9 hours for chess**, 12 hours for shogi, and 13 days for Go."

"In Chess, for example, it searches **only 60 thousand positions** per second in chess, compared to roughly 60 million for Stockfish."

Source: http://deepmind.com

Reinforcement Learning —Basic Notions

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.

Reinforcement Learning —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

“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).

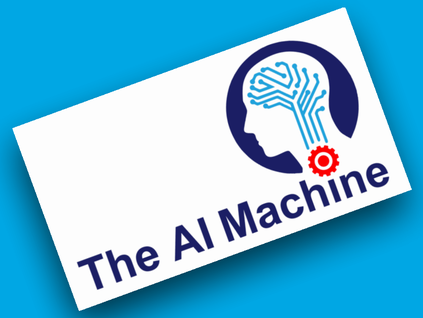
gamma
The parameter `gamma` represents the discount factor by which delayed rewards are taken into account.

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.

QUANTS@DEV

Dr. Yves J. Hilpisch
The Python Quants GmbH
tpq.io | qd@tpq.io | @dyjh
@quants_dev



Discord Server
https://discord.gg/uJPtp9Awaj

Twitter Account
https://twitter.com/quants_dev

(pro)
quants@dev
~ $