

Artificial Intelligence in Finance

Workshop at Texas State University
McCoy College of Business, October 2023

Dr. Yves J. Hilpisch



Introduction

CPF Program



<https://cpf.tpq.io>



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**

**6 Books
→ 2,500 pages
PDF**

**500 Jupyter
Notebooks
→ 50,000 LOC**

**350 PY Files
→ 35,000+ LOC**

<https://cpf.tpq.io>

Program Overview

A comprehensive set of classes and modules to master Python for Finance.

330+ hours

Foundations

Applications

PFF Bootcamp
(9 hours)

**Natural Language
Processing**
(3 hours)

RL for Finance
(15 hours)

AT Bootcamp
(12 hours)

Financial Packages
(6 hours)

Crypto Basics
(11 hours)

Python for Databases
(9 hours)

AI in Finance
(18 hours)

**Python for Algo
Trading Addon**
(22 hours)

**Listed Vol & Var
Derivatives**
(9 hours)

Linux Basics
(12 hours)

Python & Excel
(6 hours)

AI in Finance Book
(18 hours)

**Python for Algo
Trading Practice**
(8 hours)

**Python for Comp
Finance DX**
(12 hours)

Mathematics Basics
(45 hours)

Financial Data Science
(16 hours)

**Python for Asset
Management**
(25 hours)

**Python for Algo
Trading Core**
(15 hours)

**Python for Comp
Finance Core**
(17 hours)

**Python for Finance
Basics**
(35 hours)

Finance with Python
(9 hours)

Python in Finance

Hi Explain why Python has become the No. 1 programming language in finance. Provide 7 short bullet points. 

 Python's popularity in finance can be attributed to various reasons:   

1. **Easy to Learn:** Python has a simple syntax that closely resembles English, which makes it easier to learn and read compared to other programming languages.
2. **Versatile Libraries:** Python has a wide range of powerful libraries like Pandas, NumPy, and scikit-learn that are highly useful for data analysis, mathematical computations, machine learning, and much more.
3. **Data Handling Capability:** Python can easily handle large datasets, making it ideal for finance where organizations often deal with vast amounts of data.
4. **Integration:** Python can easily integrate with other languages and platforms, which allows organizations to leverage existing systems and software.
5. **Support for Algorithms and Modelling:** Python has strong support for complex financial and statistical models, making it an excellent choice for quantitative finance, algorithmic trading, and risk management.
6. **Community and Support:** Python has a very active community, which means that help is readily available and new libraries, tools, and methods are continuously being developed.
7. **Flexibility:** Python is highly flexible and versatile, which means it can be used for a wide range of tasks in finance, from data analysis and visualization to machine learning and artificial intelligence.

Python in Hedge Funds

Hi What do hedge funds primarily use Python for? Provide 7 short bullet points. 

 Sure, Python is widely used in hedge funds for a variety of tasks including:   

1. **Data Analysis:** Python's Pandas library is used for analyzing and manipulating data. This is essential in hedge funds where large volumes of market and financial data need to be processed and analyzed regularly.
2. **Quantitative Modelling:** Hedge funds use Python to build quantitative models for predicting market movements. Libraries like NumPy and SciPy are used for statistical analysis and modeling.
3. **Algorithmic Trading:** Python is used to create complex trading algorithms. These algorithms are then used to automate trading, making decisions based on predetermined conditions.
4. **Risk Management:** Python is used to calculate and manage risk. This includes analyzing potential losses and creating strategies to mitigate those risks.
5. **Data Visualization:** With libraries like Matplotlib and Seaborn, Python is used to visualize data, which can help in better understanding market trends and making informed investment decisions.
6. **Machine Learning:** Hedge funds use Python for developing machine learning models to make predictions and find patterns in data. Python's scikit-learn library is commonly used for this purpose.
7. **Web Scraping:** Hedge funds use Python to scrape data from various online sources. This can be helpful in gaining insights from data not available in traditional datasets. Libraries like BeautifulSoup and Scrapy are used for this purpose.

HI

What are the benefits of Python compared to other programming languages in algorithmic trading? Provide 7 short bullet points.



Python provides numerous advantages in algorithmic trading compared to other programming languages:



1. **Ease of Use:** Python's syntax is easy to learn and read, which allows algorithmic traders to quickly write and modify trading algorithms.
2. **Versatility:** Python has libraries and packages that cover almost every aspect of algorithmic trading, including data analysis (Pandas), mathematical operations (NumPy), machine learning (scikit-learn), and backtesting trading strategies (Zipline).
3. **Speed of Development:** Python's simplicity allows rapid development and testing of complex algorithmic trading strategies, which is a key factor in a field where speed is crucial.
4. **Integration Capabilities:** Python easily integrates with other systems, APIs, and databases. This enables traders to collect, store, and use data from various sources without much hassle.
5. **Open-Source Nature:** Python is open-source, which means there's a strong community support, continuous updates, and vast resources for learning and problem-solving.
6. **Support for Advanced Strategies:** Python's advanced libraries enable the use of high-level mathematical and statistical operations, machine learning models, and artificial intelligence in developing sophisticated trading strategies.
7. **Real-Time Data Handling:** Python's data handling capabilities make it possible to efficiently deal with real-time data, which is critical for algorithmic trading where decisions need to be made quickly based on current market conditions.

Python for Algorithmic Trading

Python for Computational Finance

HI

What makes Python beneficial for computationally demanding applications in financial engineering and computational finance? Provide 7 short bullet points.

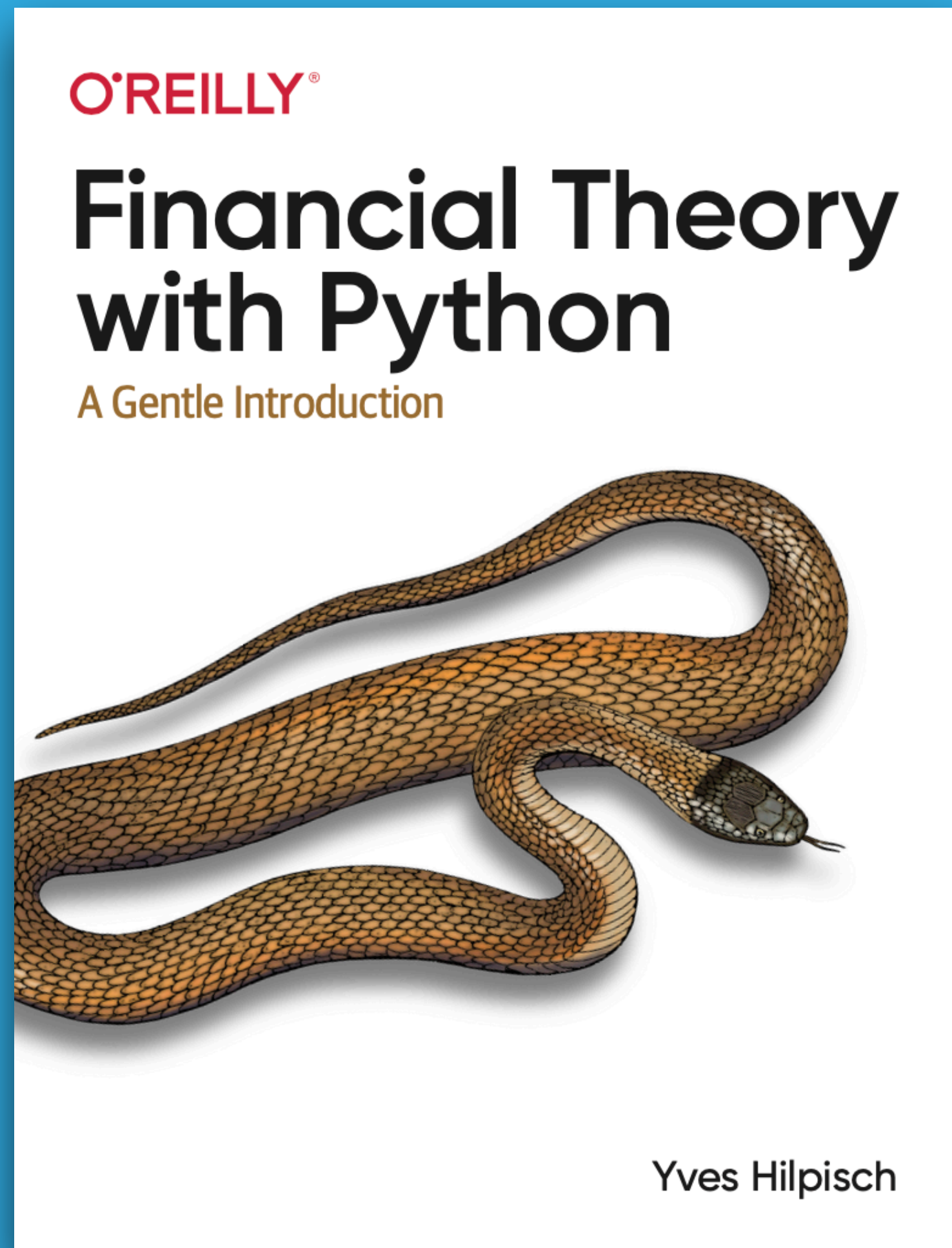


Python is highly beneficial for computationally demanding applications in financial engineering and computational finance for the following reasons:

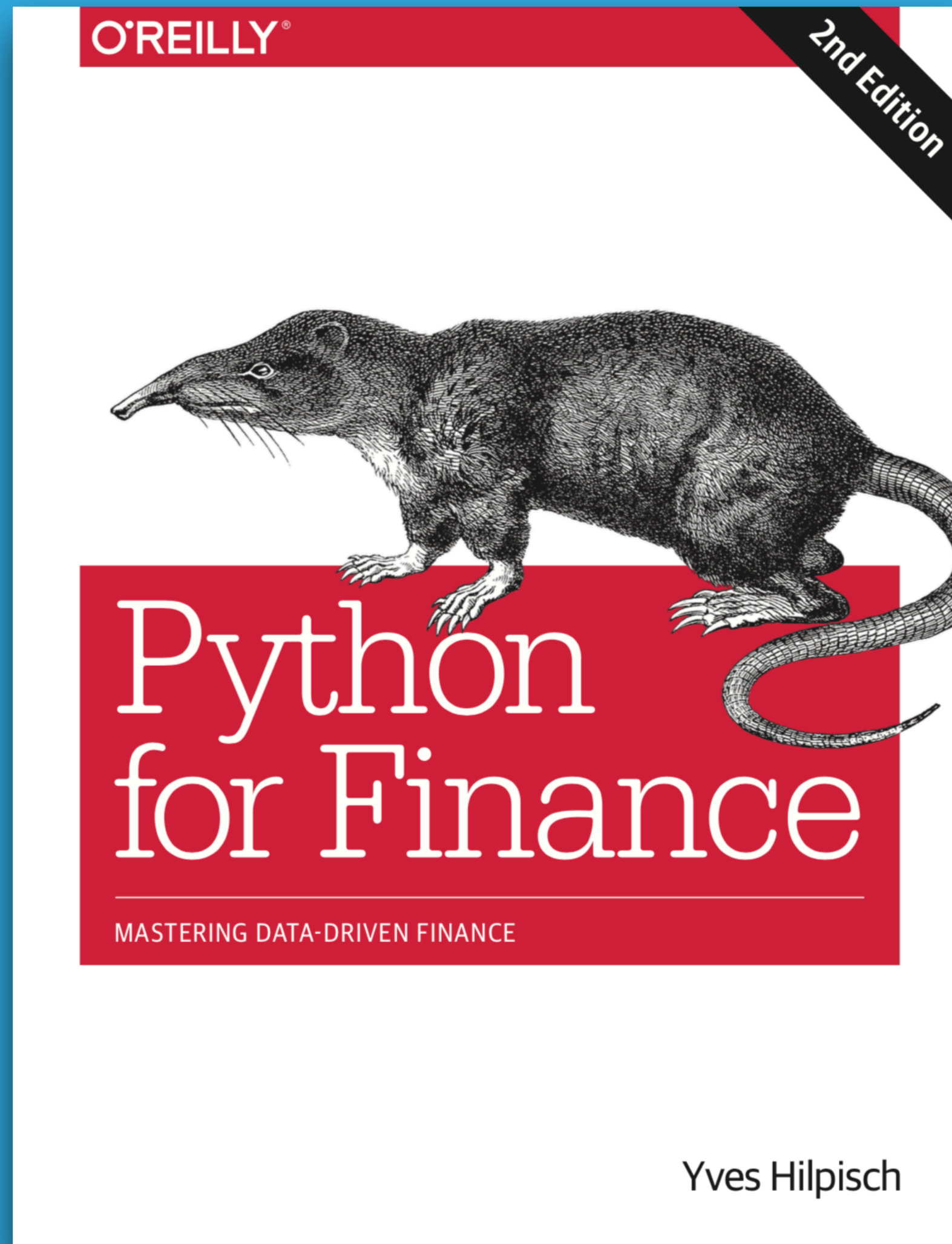


1. **Efficient Libraries:** Python offers numerous libraries, such as NumPy and Pandas, that have been optimized for high-performance mathematical and statistical computations, which are essential in financial engineering.
2. **Parallel Processing:** Python supports parallel processing, allowing computationally demanding tasks to be distributed and run simultaneously on multiple cores or machines.
3. **Scalability:** Python's capability to handle large datasets makes it ideal for computational finance, where extensive amounts of data are the norm.
4. **Machine Learning and AI:** Libraries like TensorFlow, PyTorch, and scikit-learn enable the implementation of machine learning models for complex financial predictions, risk analysis, and decision-making.
5. **Flexible and Dynamic:** Python is an interpreted language, meaning it allows for flexibility and dynamic changes, which are often necessary when adapting to changing financial scenarios.
6. **Interoperability:** Python can interact with other programming languages such as C++ or Java, which is beneficial when integrating with existing systems that use these languages for high-performance computing.
7. **Community Support:** Python's large and active community continuously improves its capabilities, ensuring the language stays up-to-date with the latest techniques in computational finance.

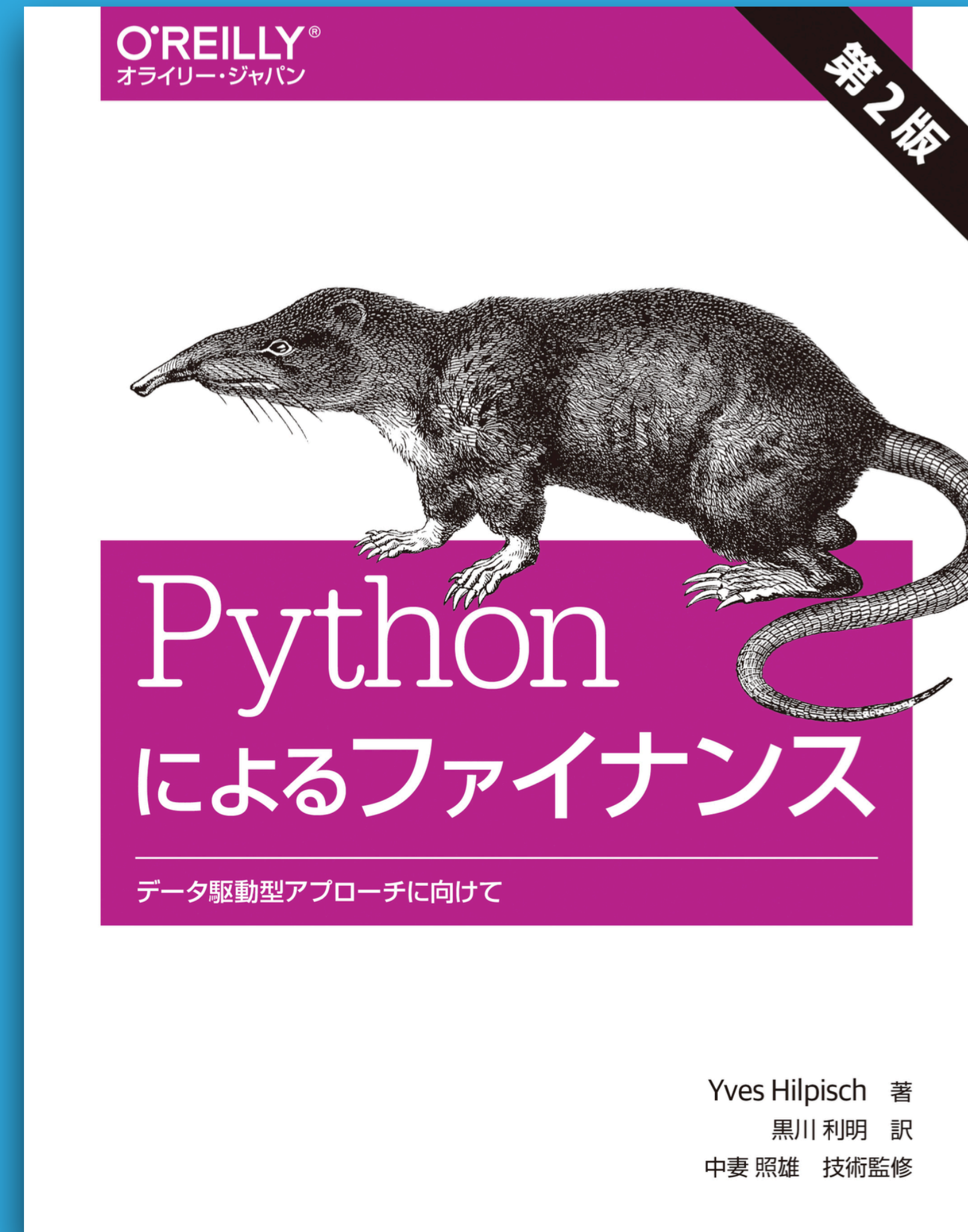
Financial Theory with Python – A Gentle Introduction



Python for Finance



Yves Hilpisch

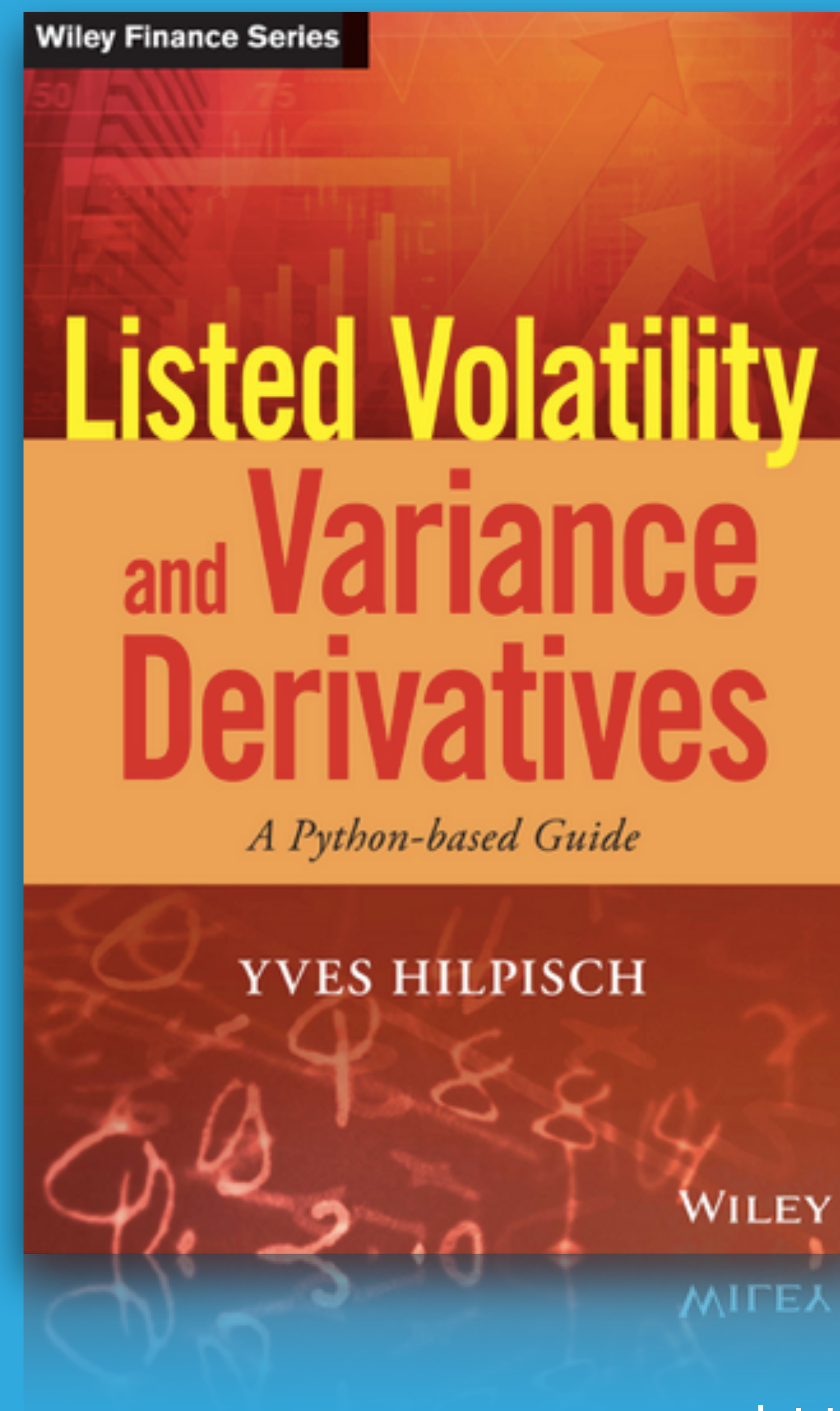
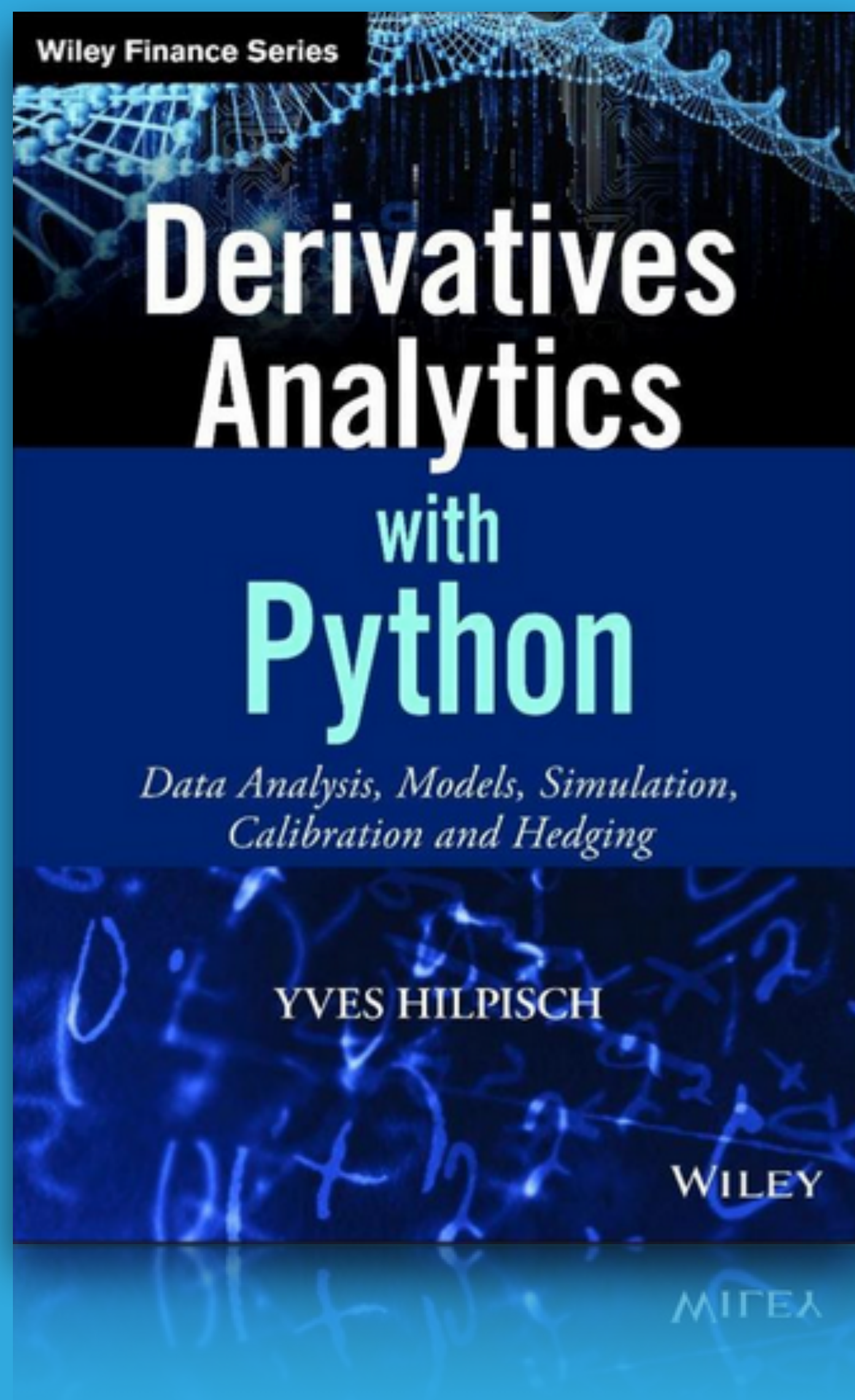


中妻 照雄 技術監修

黒川 利明 訳

<http://books.tpq.io>

Quant Finance with Python

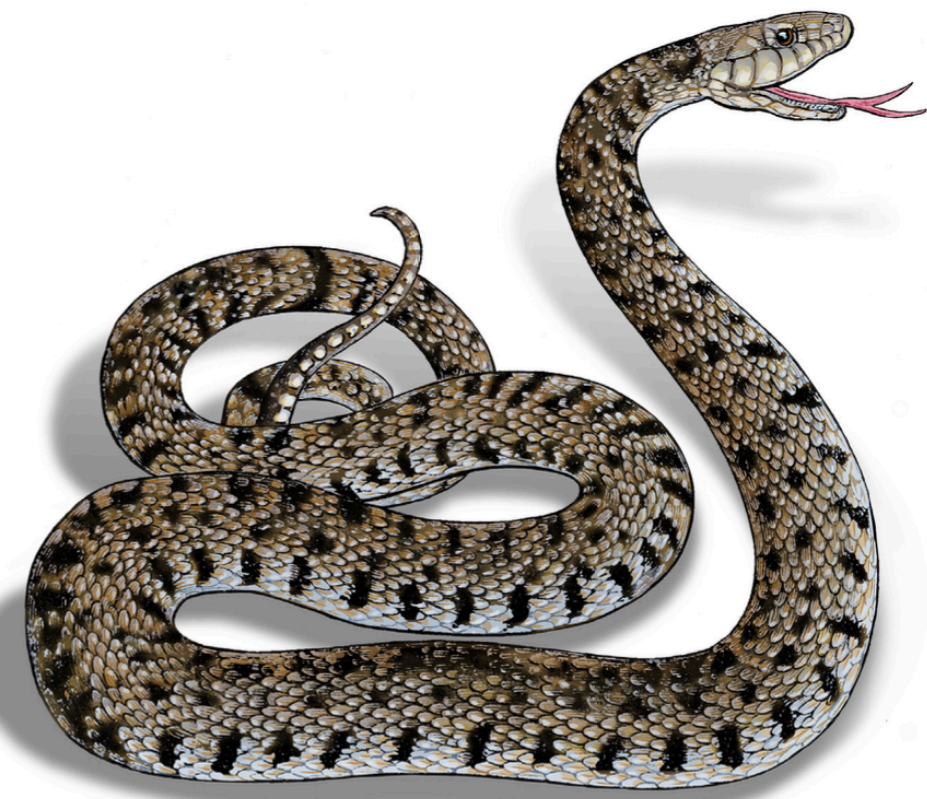


Python & AI for Finance & Trading

O'REILLY®

Python for Algorithmic Trading

From Idea to Cloud Deployment



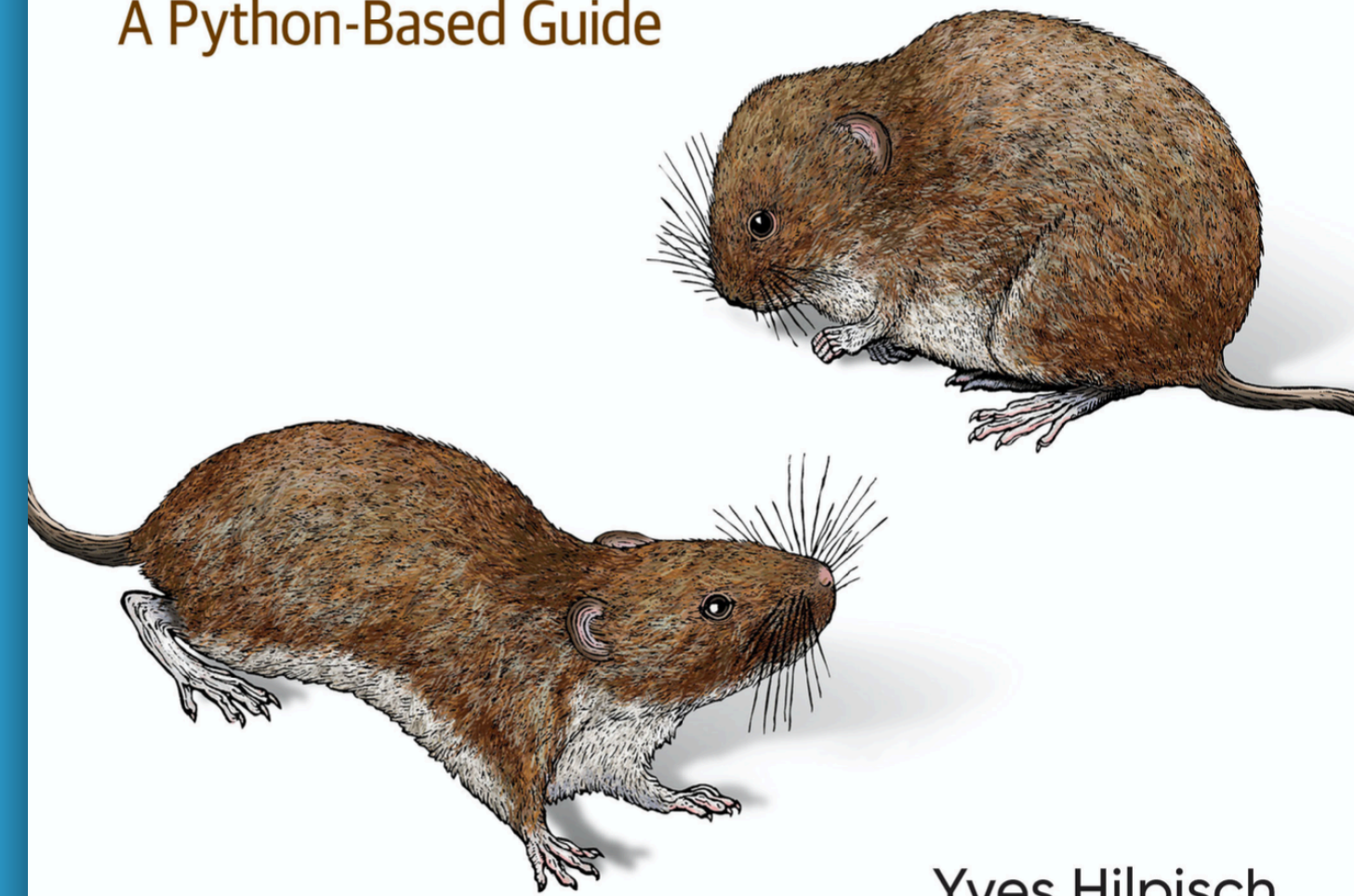
Yves Hilpisch

Yves Hilpisch

O'REILLY®

Artificial Intelligence in Finance

A Python-Based Guide



Yves Hilpisch

Yves Hilpisch

TL;DR: Yves – The Python Quant

Dr. Yves J. Hilpisch is founder and CEO of **The Python Quants** (<http://tpq.io>), the company behind the **Certificate in Python for Finance (CPF) Program** (<https://cpf.tpq.io>). He is also the founder and CEO of **The AI Machine** (<http://aimachine.io>), a company focused on AI-powered algorithmic trading.

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://books.tpq.io>):

- * Financial Theory 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 **CPF Program**. He also lectures on computational finance, machine 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.

Agenda

DAY 1

- **AI Wars | State of the Art**
- **Quantitative Finance with ChatGPT**
 - Discovering Research
 - Generating Quizzes
 - Coding Quant Finance Models
 - Financial Data Science
- **Group Discussion**
 - Benefits of LLMs
 - Problems from LLMs
- **Overview of ML/DL/RL/NLP/LLM**
- **Simple Financial Examples**
 - Stocks Clustering
 - Stock Return Prediction
- **ML Exercise (scikit-learn)**
- **Advanced Financial Examples**
 - Options Pricing
 - Credit Score Prediction
- **Financial Data APIs**
 - Historical Price Data
 - Historical Fundamental Data
 - Other Financial Data

DAY 2

- **Generating Financial Data**
 - Noisy Data (Historical Data + Noise)
 - Simulated Data (Vasicek Model)
 - Generated Data (GANs)
- **Advanced Financial Example**
 - Reinforcement Learning for Delta Hedging
- **Basics of Natural Language Processing**
 - Processing Text with Python
 - Using nltk, tokenization, summarization
- **NLP Exercise (nltk)**
- **Basics of Large Language Models**
 - transformers, self-attention
 - positional encoding, tokens, embeddings
- **GPT API usage with own texts**
- **Review and Discussion**

DISCLAIMER

This workshop is not only in part about ChatGPT and other LLMs.

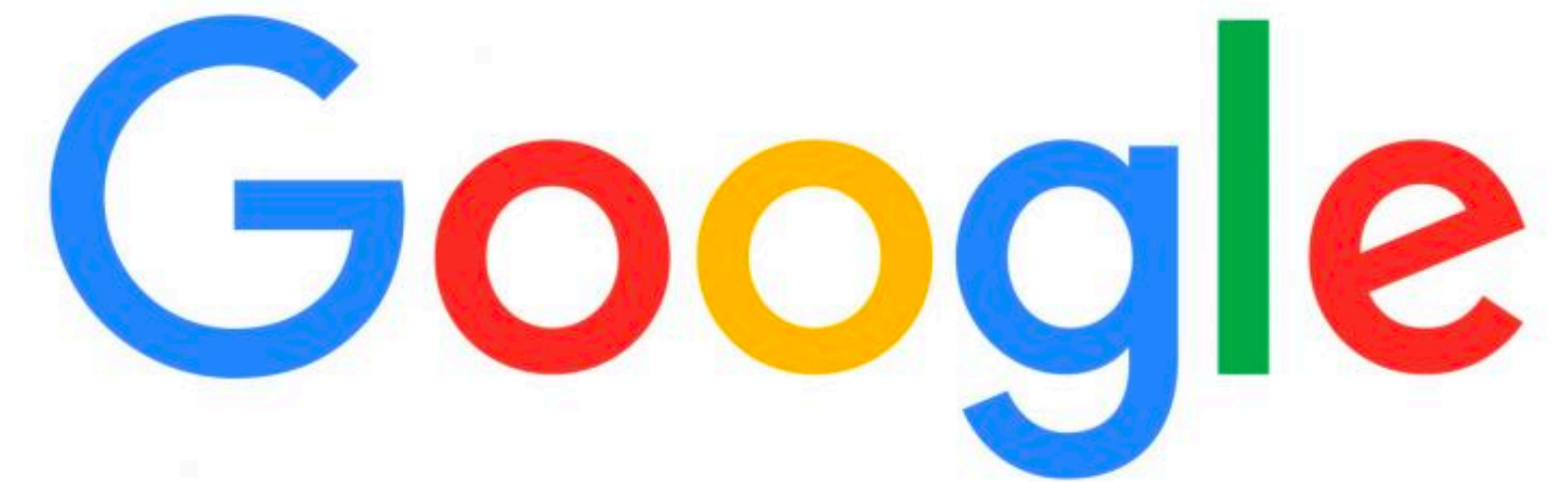
Parts of the content itself have been created with the help of tools like ChatGPT or Perplexity.ai. Content from such tools and models might have been used verbatim or in adjusted form (without any detailed attribution).

[Personal Note: GPT seems to have been trained on basically all my publicly available content without any attribution provided for answers based on that content.]

All financial examples represent technical illustrations only and do not represent financial or investment advice.

The code comes without representations or warranties, to the extent permitted by applicable law.

AI Wars State of the Art



"Generative AI's impact on productivity could add trillions of dollars in value to the global economy. Our latest research estimates that generative AI could add the equivalent of \$2.6 trillion to \$4.4 trillion annually across the 63 use cases we analyzed – by comparison, the United Kingdom's entire GDP in 2021 was \$3.1 trillion." McKinsey (2023): The economic potential of generative AI.



Business Schools Grapple With How To Teach Artificial Intelligence

MBA programs want to better prepare students for a world undergoing rapid transformation fueled by generative AI.



Photographer: Inga Kjer/Photothek

By Jo Constantz

October 19, 2023 at 4:30 PM GMT+2

October 19, 2023 at 4:30 PM GMT+2

By Jo Constantz

Copyright © 2023 by The New York Times Company

Dan Wang, a professor at Columbia Business School, says today's AI courses must address the technology's limits and opportunities. "The goal actually is not to advocate for or promote the use of AI tools, but rather for students to see, experience and understand the benefits, but importantly [also] the constraints," Wang says. ...

AI also will factor into how businesses are structured and how they operate. Organizationally, it will grow to become a part of everyday communication, from email, memos, reports and marketing copy to product development through design, software engineering and other processes. Understanding this, too, will be as essential for future executives as mastering public speaking and learning how to lead a team.

Quantitative Finance with ChatGPT

- Discovering Research
- Generating Quizzes
- Coding Quant Finance Models
- Financial Data Science

Group Discussion

- Benefits of LLMs
- Problems of LLMs

Statistical & Machine Learning

Mathematics.
*Function generates
data.*

$$f(x) = 2 + \frac{1}{2}x$$

$$y_i = f(x_i), i = 1, 2, \dots, n$$

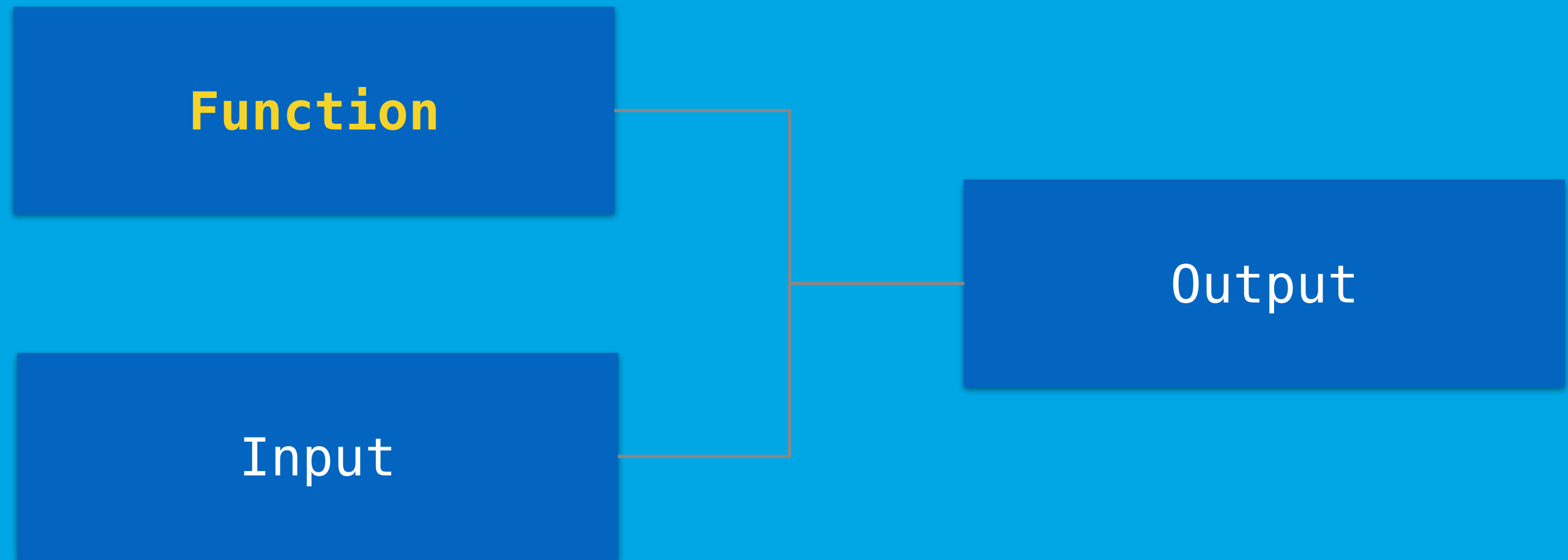
Statistics.
*Data generates
function.*

$$(y_i, x_i)_{i=1}^n$$

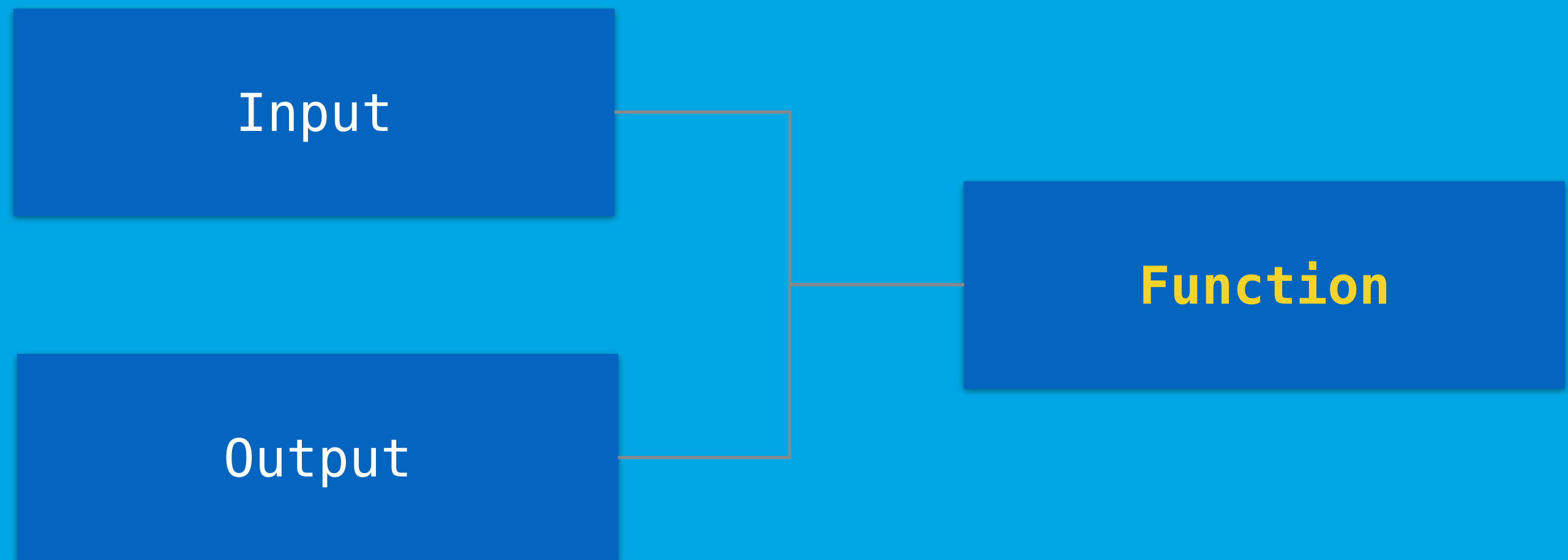
$$\hat{f}(x) = \alpha + \beta x \approx y$$

$$\alpha, \beta = ?, ?$$

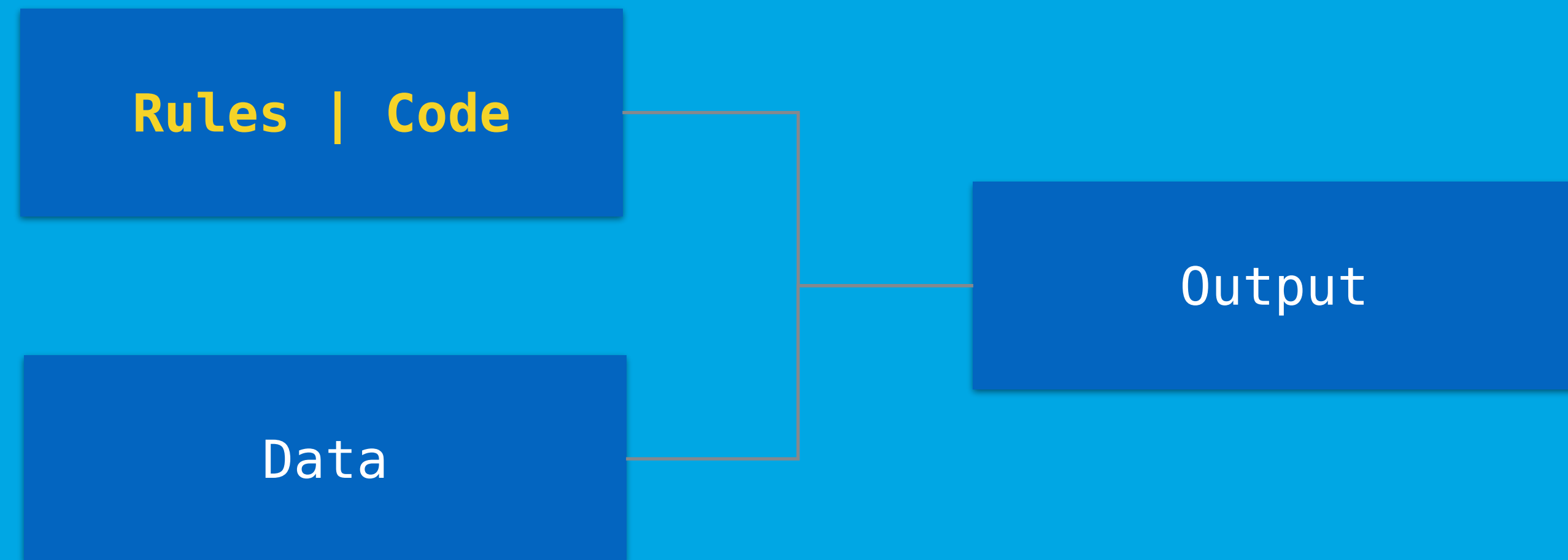
Mathematics.



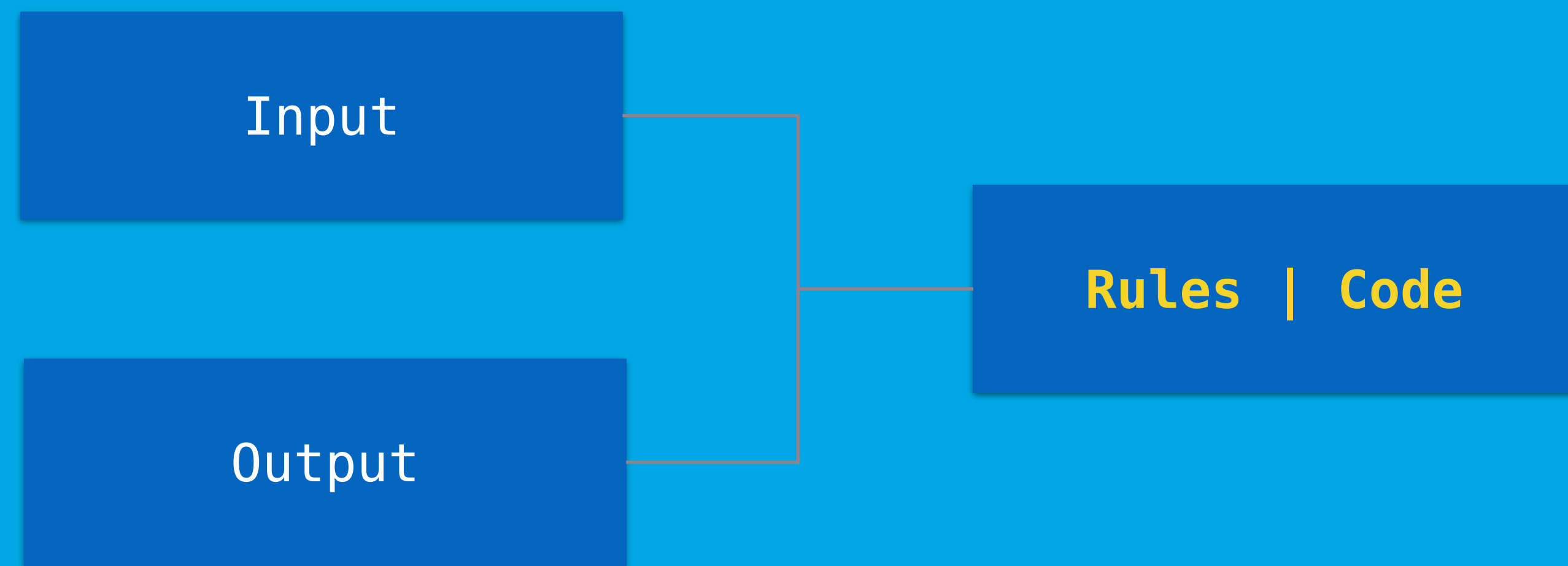
Statistics.



Programming.



Machine Learning.



Types of Learning

Unsupervised learning (UL)

These are algorithms that learn from a given sample data set of features (input) values only. They are supposed to learn about the input data set, given, for example, some guiding parameters. Clustering algorithms fall into that category. In a financial context, such algorithms might cluster stocks into certain groups.

Supervised learning (SL)

These are algorithms that learn from a given sample data set of features (input) and labels (output) values. Examples are OLS regression and neural networks. The purpose of supervised learning is to learn the relationship between the input and output values. In finance, such algorithms might be trained to predict whether a potential debtor is credit-worthy or not.

```
IPython: Program/cert
IPython: Program/cert (python3.7)

In [1]: import numpy as np

In [2]: features = np.array(((0, 0, 1, 1), (0, 1, 0, 1))).T

In [3]: features
Out[3]:
array([[0, 0],
       [0, 1],
       [1, 0],
       [1, 1]])

In [4]: labels = features[:, 0] | features[:, 1]

In [5]: labels
Out[5]: array([0, 1, 1, 1])

In [6]: from sklearn.naive_bayes import GaussianNB

In [7]: model = GaussianNB()

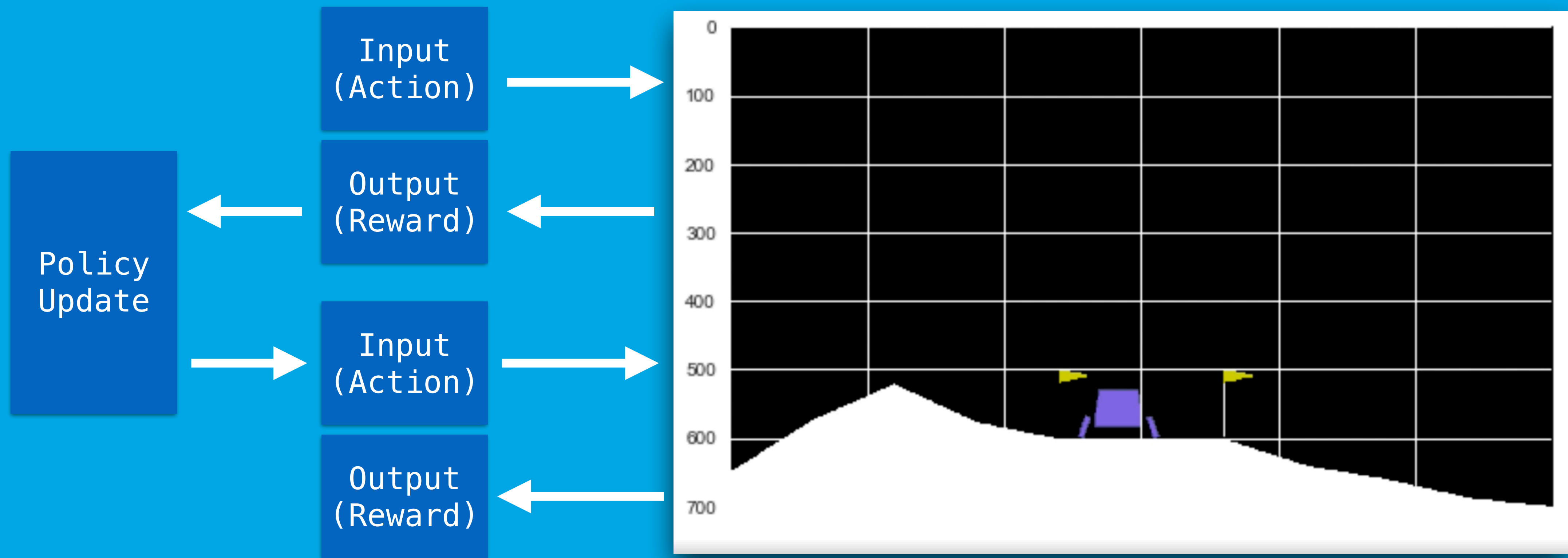
In [8]: model.fit(features, labels)
Out[8]: GaussianNB()

In [9]: model.predict(features)
Out[9]: array([0, 1, 1, 1])

In [10]:
```


Reinforcement learning (RL)

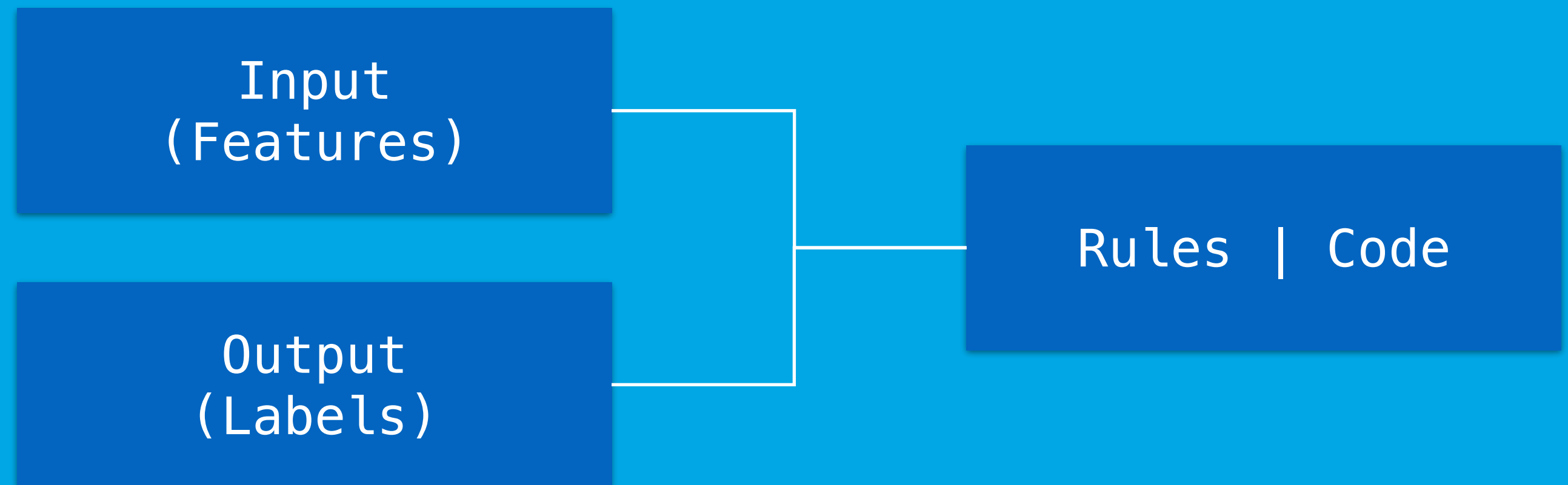
These are algorithms that learn from trial and error by receiving a reward for taking an action. They update an optimal action policy according to what rewards and punishments they receive. Such algorithms are, for example, used for environments where actions need to be taken continuously and rewards are received immediately, such as in a computer game.



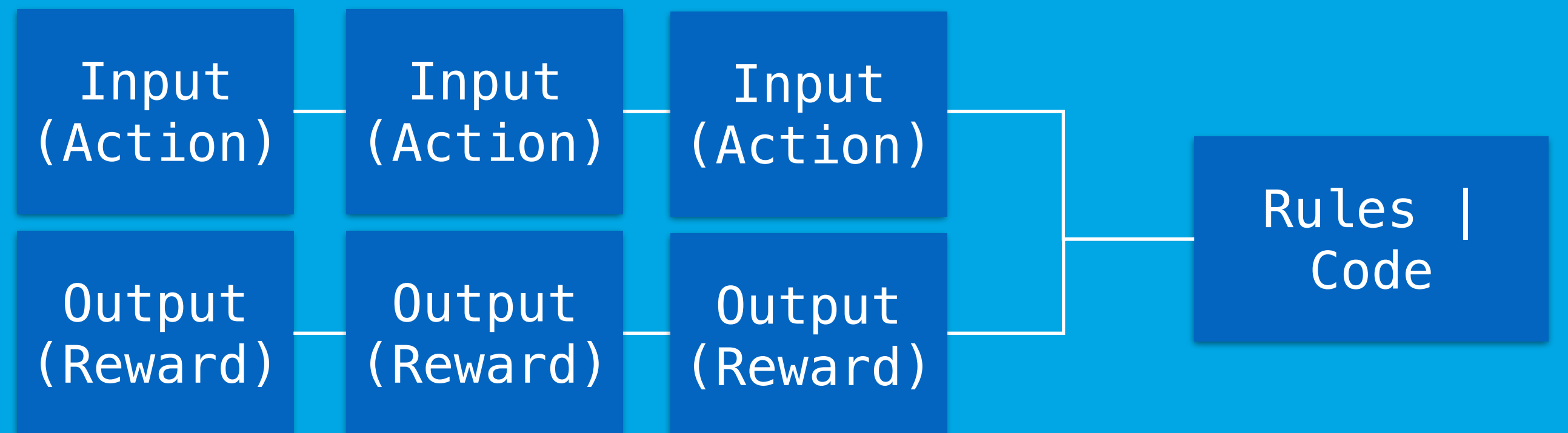
Unsupervised Learning.



Supervised Learning.



Reinforcement Learning.



Simple Financial Examples

- Stock Clustering
- Stock Price Prediction

ML Exercise (scikit-learn)

- Efficient Markets
- Stock Price Prediction ...
 - ... for a Random Walk
 - ... for a Financial Time Series

Advanced Financial Examples

- Option Pricing with DNNs
- Credit Scoring with DNNs

Financial Data APIs
—“What has been.”

Historical Prices and Fundame x +

← → ↺ eodhd.com 🔍 📄 ⭐ ⚙️ ⬇️ 🖨️ 🌐 ⋮

AAPL 172.88 TSLA 211.99 AMZN 125.17 BTC-USD 29879.1133 ETH-USD 1631.2738 EURUSD 1.0593

 EODHD
APIs

Data Bundles

Documentation

Academy

Blog

Pricing

Login

Registration

30+ years of data

Financial Data APIs

1,015,940,616 data points and counting
as of Oct 22, 2023

70+ stock exchanges all over the world

30+ years of data | 150,000+ tickers | 20000+ ETFs | 600+ indices | 1100+ forex pairs

FUNDAMENTALS

END OF DAY

REAL-TIME

DELAYED

TECHNICAL

INTRADAY

SCREENER

ALTERNATIVE

ECONOMIC

DELISTED

FOREX PAIRS

SPREADSHEETS

🔍 AAPL

Free trial version →

Feel free to compare our data sources with any other data provider

 Nasdaq

 Cboe

 BENZINGA

 ALTOVA

 U.S. SECURITIES AND
EXCHANGE COMMISSION

 ASX

 INEIO

 quotemedia

 London
Stock Exchange Group

 NASDAQ OMX

 S&P Global

 JS



Generating Data
—“What could have been.”

Generative Adversarial Nets

Ian J. Goodfellow, Jean Pouget-Abadie*, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, Yoshua Bengio†
Département d’informatique et de recherche opérationnelle
Université de Montréal
Montréal, QC H3C 3J7

Abstract

We propose a new framework for estimating generative models via an adversarial process, in which we simultaneously train two models: a generative model G that captures the data distribution, and a discriminative model D that estimates the probability that a sample came from the training data rather than G . The training procedure for G is to maximize the probability of D making a mistake. This framework corresponds to a minimax two-player game. In the space of arbitrary functions G and D , a unique solution exists, with G recovering the training data distribution and D equal to $\frac{1}{2}$ everywhere. In the case where G and D are defined by multilayer perceptrons, the entire system can be trained with backpropagation. There is no need for any Markov chains or unrolled approximate inference networks during either training or generation of samples. Experiments demonstrate the potential of the framework through qualitative and quantitative evaluation of the generated samples.

1 Introduction

The promise of deep learning is to discover rich, hierarchical models [2] that represent probability distributions over the kinds of data encountered in artificial intelligence applications, such as natural images, audio waveforms containing speech, and symbols in natural language corpora. So far, the most striking successes in deep learning have involved discriminative models, usually those that map a high-dimensional, rich sensory input to a class label [14, 22]. These striking successes have primarily been based on the backpropagation and dropout algorithms, using piecewise linear units [19, 9, 10] which have a particularly well-behaved gradient. Deep *generative* models have had less of an impact, due to the difficulty of approximating many intractable probabilistic computations that arise in maximum likelihood estimation and related strategies, and due to difficulty of leveraging the benefits of piecewise linear units in the generative context. We propose a new generative model estimation procedure that sidesteps these difficulties.¹

In the proposed *adversarial nets* framework, the generative model is pitted against an adversary: a discriminative model that learns to determine whether a sample is from the model distribution or the data distribution. The generative model can be thought of as analogous to a team of counterfeiters, trying to produce fake currency and use it without detection, while the discriminative model is analogous to the police, trying to detect the counterfeit currency. Competition in this game drives both teams to improve their methods until the counterfeits are indistinguishable from the genuine articles.

*Jean Pouget-Abadie is visiting Université de Montréal from Ecole Polytechnique.

†Sherjil Ozair is visiting Université de Montréal from Indian Institute of Technology Delhi

‡Yoshua Bengio is a CIFAR Senior Fellow.

¹All code and hyperparameters available at <http://www.github.com/goodfeli/adversarial>

Time-series Generative Adversarial Networks

Jinsung Yoon*
University of California, Los Angeles, USA
jsyoon0823@g.ucla.edu

Daniel Jarrett*
University of Cambridge, UK
daniel.jarrett@maths.cam.ac.uk

Mihaela van der Schaar
University of Cambridge, UK
University of California, Los Angeles, USA
Alan Turing Institute, UK
mv472@cam.ac.uk, mihaela@ee.ucla.edu

Abstract

A good generative model for time-series data should preserve *temporal dynamics*, in the sense that new sequences respect the original relationships between variables across time. Existing methods that bring generative adversarial networks (GANs) into the sequential setting do not adequately attend to the temporal correlations unique to time-series data. At the same time, supervised models for sequence prediction—which allow finer control over network dynamics—are inherently deterministic. We propose a novel framework for generating realistic time-series data that combines the flexibility of the unsupervised paradigm with the control afforded by supervised training. Through a learned embedding space jointly optimized with both supervised and adversarial objectives, we encourage the network to adhere to the dynamics of the training data during sampling. Empirically, we evaluate the ability of our method to generate realistic samples using a variety of real and synthetic time-series datasets. Qualitatively and quantitatively, we find that the proposed framework consistently and significantly outperforms state-of-the-art benchmarks with respect to measures of similarity and predictive ability.

1 Introduction

What is a good generative model for time-series data? The temporal setting poses a unique challenge to generative modeling. A model is not only tasked with capturing the distributions of features *within* each time point, it should also capture the potentially complex dynamics of those variables *across* time. Specifically, in modeling multivariate sequential data $\mathbf{x}_{1:T} = (\mathbf{x}_1, \dots, \mathbf{x}_T)$, we wish to accurately capture the conditional distribution $p(\mathbf{x}_t | \mathbf{x}_{1:t-1})$ of temporal transitions as well.

On the one hand, a great deal of work has focused on improving the temporal dynamics of autoregressive models for sequence prediction. These primarily tackle the problem of compounding errors during multi-step sampling, introducing various training-time modifications to more accurately reflect testing-time conditions [1, 2, 3]. Autoregressive models explicitly factor the distribution of sequences into a product of conditionals $\prod_t p(\mathbf{x}_t | \mathbf{x}_{1:t-1})$. However, while useful in the context of forecasting, this approach is fundamentally deterministic, and is not truly *generative* in the sense that new sequences can be randomly sampled from them without external conditioning. On the other hand, a separate line of work has focused on directly applying the generative adversarial network (GAN) framework to sequential data, primarily by instantiating recurrent networks for the roles of generator and discriminator [4, 5, 6]. While straightforward, the adversarial objective seeks to model $p(\mathbf{x}_{1:T})$ directly, without leveraging the autoregressive prior. Importantly, simply summing

* indicates equal contribution

RL for Delta Hedging

“Goldman Sachs Group Inc. is expanding its use of a technology that leverages artificial intelligence in the hopes that it will make it easier for clients to plan complex derivatives trades.

After already using the software to shake up the worlds of equities and foreign exchange options, the firm in recent weeks began allowing clients to use its visual structuring product for credit derivatives. It’s aiming to offer the service for rates trading in the first half of next year, Chris Churchman, who runs Goldman’s digital trading platform Marquee, said in an interview.

The product, which Goldman began offering last year, helps institutional clients with price discovery and trading ideas. It also assesses the risks of a given trade and can gauge the chances that the trade will pay off.”

Source: “Money Stuff” [Newsletter](#) by Matt Levine from 25. October 2023

In the Black-Scholes-Merton (1973) model, delta hedging is a strategy used to eliminate the risk associated with holding an option by continuously adjusting the position in the underlying asset. The main principle behind the model is to hedge the option by buying and selling the underlying asset in a specific way to eliminate risk. This type of hedging is called "continuously revised delta hedging" and is the basis of more complicated hedging strategies such as those engaged in by investment banks and hedge funds ¹.

Delta is a Greek letter used to represent the sensitivity of an option's price to changes in the price of the underlying asset. In the Black-Scholes-Merton model, the delta of a European-style call option can be calculated using the following formula:

$$\Delta = \frac{\partial C}{\partial S}$$

Where:

- Δ is the delta of the call option
- C is the price of the call option
- S is the price of the underlying asset

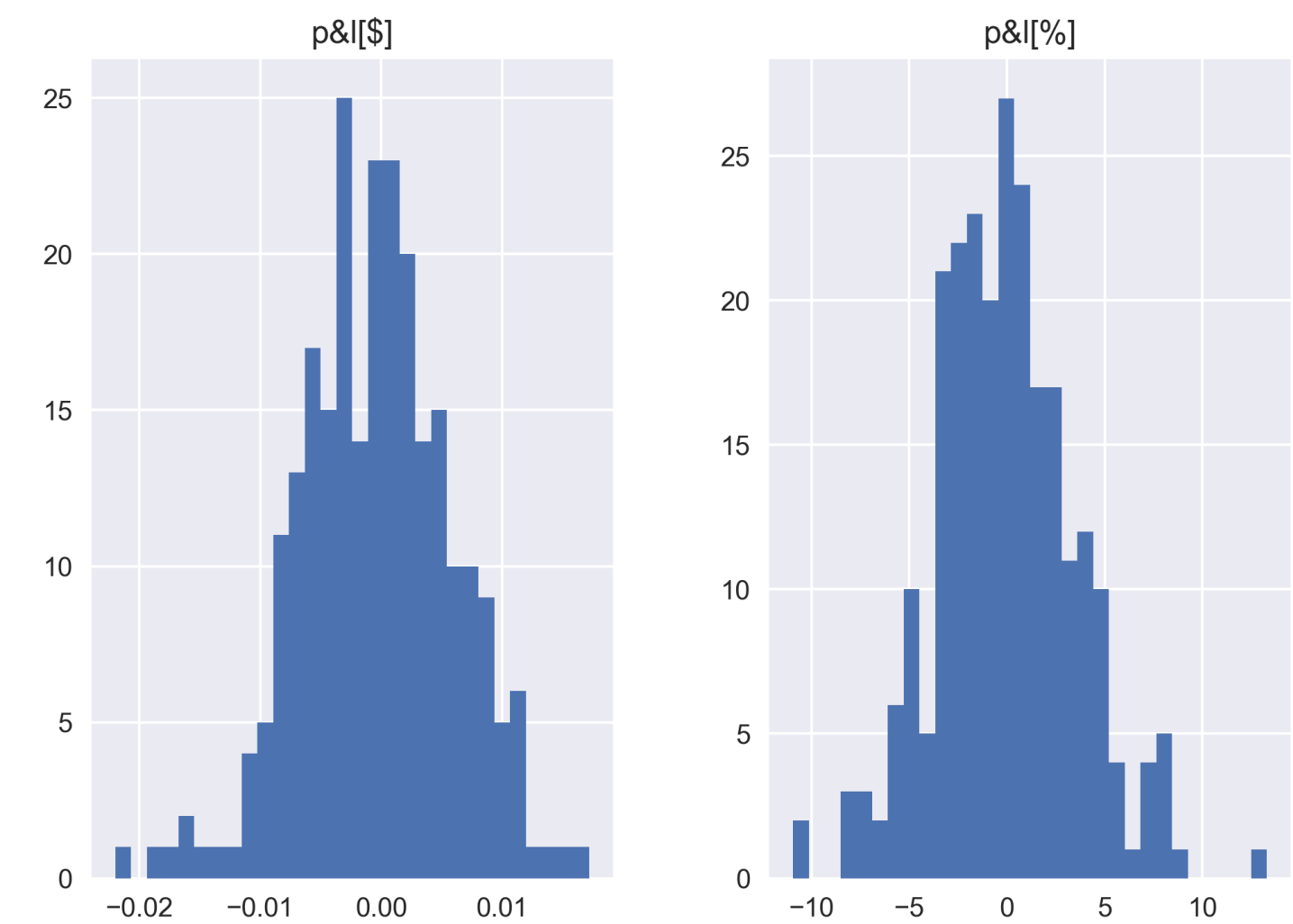
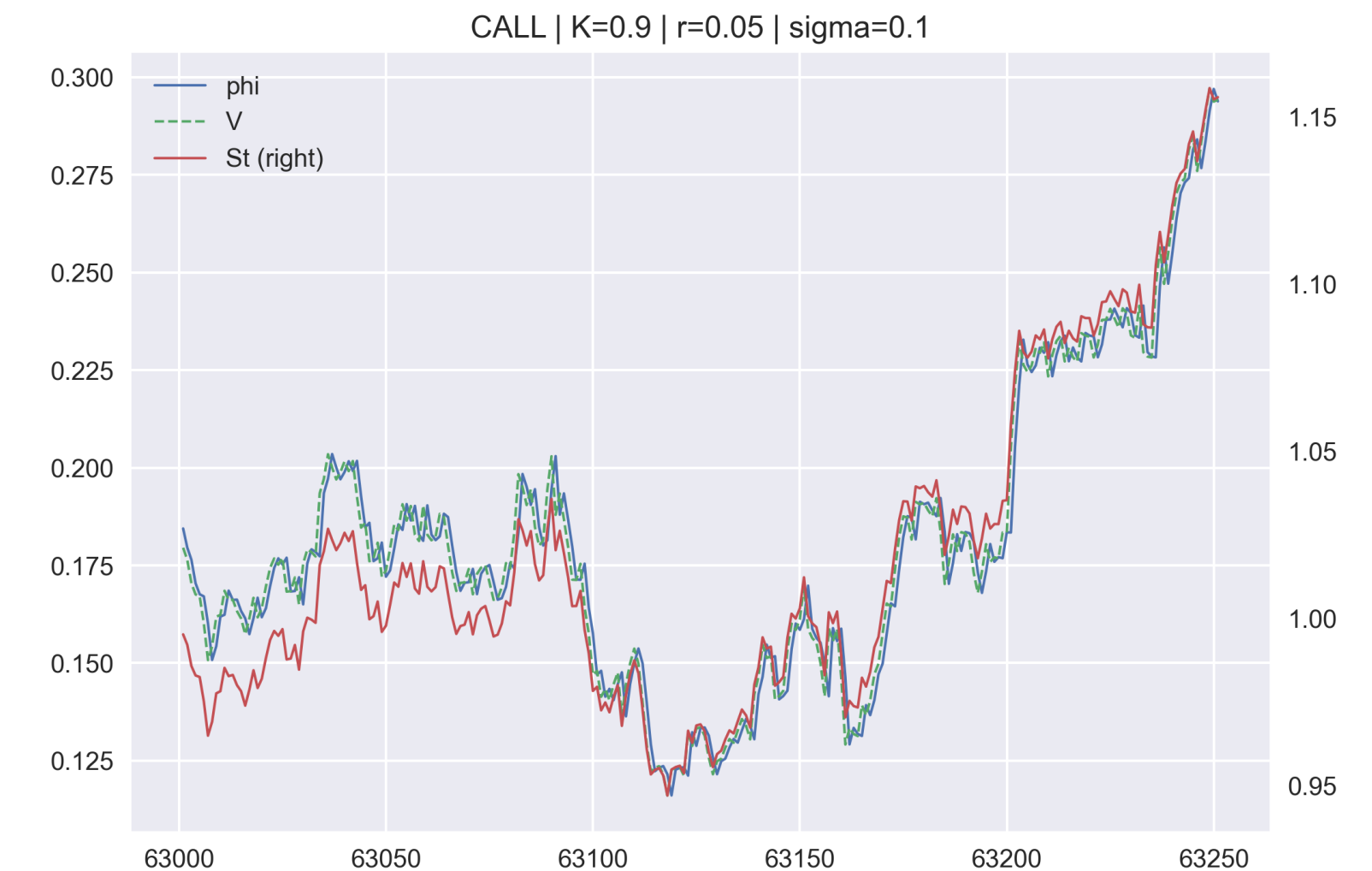
The delta of a European-style put option can be calculated using the following formula:

$$\Delta = \frac{\partial P}{\partial S}$$

Where:

- Δ is the delta of the put option
- P is the price of the put option
- S is the price of the underlying asset

Delta hedging involves adjusting the position in the underlying asset to maintain a delta-neutral position. A delta-neutral position is one where the change in the value of the option is offset by the change in the value of the underlying asset. This can be achieved by buying or selling the underlying asset as the price changes, based on the delta of the option.



Natural Language Processing

Here's a concise bullet-point summary of major techniques used in Natural Language Processing (NLP) with Python:

- **Tokenization:**
 - Splitting text into words, sentences, or other units.
 - Common libraries: NLTK, spaCy.
- **Part-of-Speech Tagging:**
 - Assigning grammatical categories (e.g., noun, verb) to tokens.
 - Common libraries: NLTK, spaCy.
- **Named Entity Recognition (NER):**
 - Identifying and classifying named entities (e.g., person names, organizations) in text.
 - Common libraries: spaCy, StanfordNLP.
- **Stemming and Lemmatization:**
 - Reducing words to their base or root form.
 - Stemming: Truncating words (e.g., "running" to "run").
 - Lemmatization: Transforming words to their dictionary form (e.g., "went" to "go").
 - Common libraries: NLTK, spaCy.

- **Stopword Removal:**
 - Removing common words (e.g., "and", "the") that may not add significant meaning in analysis.
 - Common libraries: NLTK, spaCy.
- **Word Embeddings:**
 - Representing words as dense vectors capturing semantic meaning.
 - Techniques: Word2Vec, GloVe, FastText.
 - Common libraries: Gensim, spaCy.
- **Topic Modeling:**
 - Discovering topics in large volumes of text.
 - Techniques: Latent Dirichlet Allocation (LDA), Non-Negative Matrix Factorization (NMF).
 - Common libraries: Gensim, scikit-learn.
- **Sentiment Analysis:**
 - Determining the sentiment or emotion expressed in text.
 - Common libraries: TextBlob, VADER, spaCy.
- **Syntax Parsing:**
 - Analyzing the grammatical structure of sentences.
 - Common libraries: spaCy, StanfordNLP.

- **Machine Translation:**
 - Translating text from one language to another.
 - Common libraries: OpenNMT, T2T (Tensor2Tensor).
- **Seq2Seq Models:**
 - Using recurrent neural networks (RNNs) for tasks like machine translation and text summarization.
 - Common libraries: TensorFlow, PyTorch.
- **Attention Mechanisms and Transformers:**
 - Advanced model architectures for handling long-range dependencies in text.
 - Techniques: BERT, GPT, T5.
 - Common libraries: HuggingFace's Transformers.
- **Text Classification:**
 - Categorizing text into predefined categories or labels.
 - Common libraries: scikit-learn, TensorFlow, PyTorch.
- **Regular Expressions:**
 - Pattern matching and text extraction.
 - Common library: re module in Python.
- **Dependency Parsing:**
 - Analyzing the grammatical relationships between words in a sentence.
 - Common libraries: spaCy, StanfordNLP.

Large Language Models

Attention Is All You Need

Ashish Vaswani*
Google Brain
avaswani@google.com

Noam Shazeer*
Google Brain
noam@google.com

Niki Parmar*
Google Research
nikip@google.com

Jakob Uszkoreit*
Google Research
usz@google.com

Llion Jones*
Google Research
llion@google.com

Aidan N. Gomez* †
University of Toronto
aidan@cs.toronto.edu

Lukasz Kaiser*
Google Brain
lukaszkaiser@google.com

Illia Polosukhin* ‡
illia.polosukhin@gmail.com

Abstract

The dominant sequence transduction models are based on complex recurrent or convolutional neural networks that include an encoder and a decoder. The best performing models also connect the encoder and decoder through an attention mechanism. We propose a new simple network architecture, the Transformer, based solely on attention mechanisms, dispensing with recurrence and convolutions entirely. Experiments on two machine translation tasks show these models to be superior in quality while being more parallelizable and requiring significantly less time to train. Our model achieves 28.4 BLEU on the WMT 2014 English-to-German translation task, improving over the existing best results, including ensembles, by over 2 BLEU. On the WMT 2014 English-to-French translation task, our model establishes a new single-model state-of-the-art BLEU score of 41.0 after training for 3.5 days on eight GPUs, a small fraction of the training costs of the best models from the literature.

1 Introduction

Recurrent neural networks, long short-term memory [12] and gated recurrent [7] neural networks in particular, have been firmly established as state of the art approaches in sequence modeling and transduction problems such as language modeling and machine translation [29, 2, 5]. Numerous efforts have since continued to push the boundaries of recurrent language models and encoder-decoder architectures [31, 21, 13].

*Equal contribution. Listing order is random. Jakob proposed replacing RNNs with self-attention and started the effort to evaluate this idea. Ashish, with Illia, designed and implemented the first Transformer models and has been crucially involved in every aspect of this work. Noam proposed scaled dot-product attention, multi-head attention and the parameter-free position representation and became the other person involved in nearly every detail. Niki designed, implemented, tuned and evaluated countless model variants in our original codebase and tensor2tensor. Llion also experimented with novel model variants, was responsible for our initial codebase, and efficient inference and visualizations. Lukasz and Aidan spent countless long days designing various parts of and implementing tensor2tensor, replacing our earlier codebase, greatly improving results and massively accelerating our research.

†Work performed while at Google Brain.

‡Work performed while at Google Research.

31st Conference on Neural Information Processing Systems (NIPS 2017), Long Beach, CA, USA.

Minds and Machines (2020) 30:681–694
<https://doi.org/10.1007/s11023-020-09548-1>

COMMENTARY



GPT-3: Its Nature, Scope, Limits, and Consequences

Luciano Floridi^{1,2} · Massimo Chiriatti³

Published online: 1 November 2020
© The Author(s) 2020

Abstract

In this commentary, we discuss the nature of reversible and irreversible questions, that is, questions that may enable one to identify the nature of the source of their answers. We then introduce GPT-3, a third-generation, autoregressive language model that uses deep learning to produce human-like texts, and use the previous distinction to analyse it. We expand the analysis to present three tests based on mathematical, semantic (that is, the Turing Test), and ethical questions and show that GPT-3 is not designed to pass any of them. This is a reminder that GPT-3 does not do what it is not supposed to do, and that any interpretation of GPT-3 as the beginning of the emergence of a general form of artificial intelligence is merely uninformed science fiction. We conclude by outlining some of the significant consequences of the industrialisation of automatic and cheap production of good, semantic artefacts.

Keywords Automation · Artificial Intelligence · GPT-3 · Irreversibility · Semantics · Turing Test

1 Introduction

Who mowed the lawn, Ambrogio (a robotic lawn mower)¹ or Alice? We know that the two are different in everything: bodily, “cognitively” (in terms of internal information processes), and “behaviourally” (in terms of external actions). And yet it is impossible to infer, with full certainty, from the mowed lawn who mowed it. Irreversibility and reversibility are not a new idea (Perumalla 2014). They find applications in many fields, especially computing and physics. In

¹ This is a real example, see <https://www.ambrogiorobot.com/en>. Disclosure: LF owns one.

✉ Luciano Floridi
luciano.floridi@oii.ox.ac.uk

¹ Oxford Internet Institute, 1 St Giles', Oxford OX1 3JS, UK

² The Alan Turing Institute, British Library, 96 Euston Rd, London NW1 2DB, UK

³ IBM Italia, University Programs Leader - CTO Blockchain & Digital Currencies, Rome, Italy



Large Language Models (LLMs) like OpenAI's GPT (Generative Pre-trained Transformer) series, BERT (Bidirectional Encoder Representations from Transformers), and others have revolutionized the field of NLP. Here's a concise bullet-point summary of the major techniques and components of LLMs:

- **Transformers Architecture:**
 - Introduced in the "Attention Is All You Need" paper.
 - Consists of encoder and decoder stacks (though LLMs like GPT only use the decoder stack).
- **Attention Mechanisms:**
 - Scaled Dot-Product Attention: Computes attention scores based on the dot product of query and key vectors.
 - Multi-Head Attention: Parallel attention layers that capture different types of information.
- **Positional Encoding:**
 - Since transformers don't have a built-in notion of sequence order, positional encodings are added to embeddings to provide positional information.

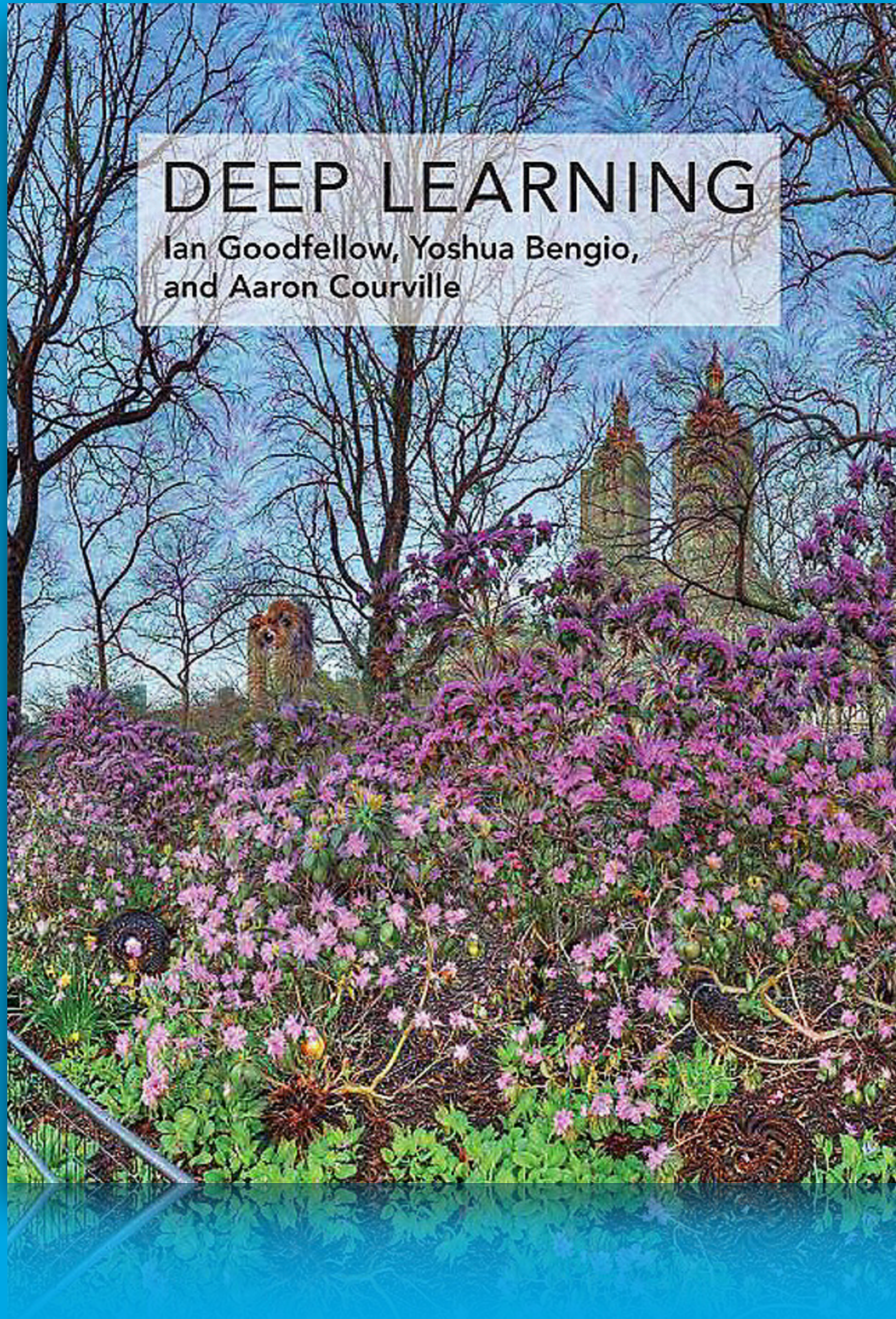
- **Embedding Layers:**
 - Convert input tokens into continuous vectors.
 - Often initialized with pre-trained embeddings like Word2Vec or GloVe, though they can be trained from scratch.
- **Layer Normalization:**
 - Normalizes the activations of each feature within a layer.
 - Helps stabilize and accelerate training.
- **Feed-Forward Neural Networks:**
 - Present in each transformer block.
 - Consists of two linear transformations with a ReLU activation in between.
- **Residual Connections:**
 - Helps in preventing the vanishing gradient problem.
 - Allows deeper models by adding the original input to the output of each sub-layer.
- **Masking:**
 - Padding Mask: Ensures the model doesn't attend to padding tokens.
 - Look-ahead Mask: Used in decoders to ensure that a position doesn't attend to future positions.

- **Pre-training and Fine-tuning:**
 - LLMs are often pre-trained on large corpora to learn language representations and then fine-tuned on specific tasks.
- **Bidirectional Context:**
 - Models like BERT are trained to consider both left and right context, making them deeply bidirectional.
- **Adaptive Learning Rates:**
 - Techniques like Adam optimizer are commonly used with learning rate warm-up and decay.
- **Regularization Techniques:**
 - Dropout: Used in various parts of the model, including attention weights.
 - Layer-wise Recurrent Mechanism: Some models use mechanisms where layers can pass information to the same layer in the next time step.

- **Parameter Sharing:**
 - In models like GPT, weights are shared between the token embedding layer and the final softmax layer.
- **Large-scale Training:**
 - LLMs have hundreds of millions to billions of parameters.
 - Training requires distributed computing and advanced techniques to handle such large models.
- **Prompt Engineering and Few-shot Learning:**
 - Techniques to guide LLMs in generating specific outputs or performing tasks with limited examples.
- **Knowledge Integration:**
 - Some LLMs are trained with external knowledge bases or facts to enhance their knowledge.

This list provides an overview of the major techniques and components of LLMs. The field is rapidly evolving, with new techniques and architectures emerging regularly.

APPENDIX: Dense Neural Networks



“A major source of difficulty in many real-world artificial intelligence applications is that many of the factors of variation influence every single piece of data we are able to observe.”

“Deep learning solves this central problem in representation learning by introducing representations that are expressed in terms of other, simpler representations. Deep learning enables the computer to build complex concepts out of simpler concepts.”

“The quintessential example of a deep learning model is the feedforward deep network, or multilayer perceptron (MLP). A multilayer perceptron is just a mathematical function mapping some set of input values to output values. The function is formed by composing many simpler functions. We can think of each application of a different mathematical function as providing a new representation of the input.”

Neural Network (Simple)
0 Hidden Layers



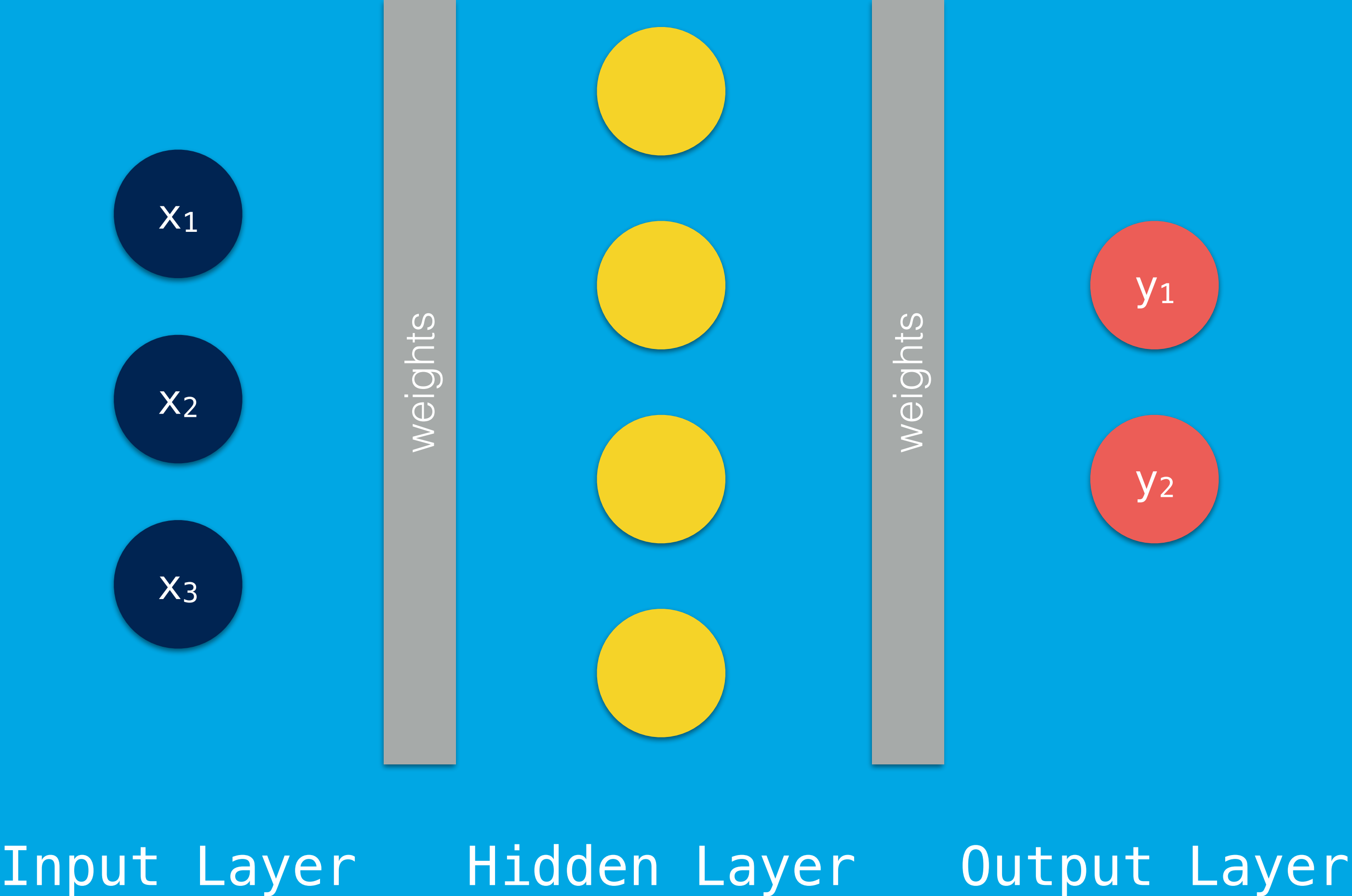
Input Layer



Output Layer

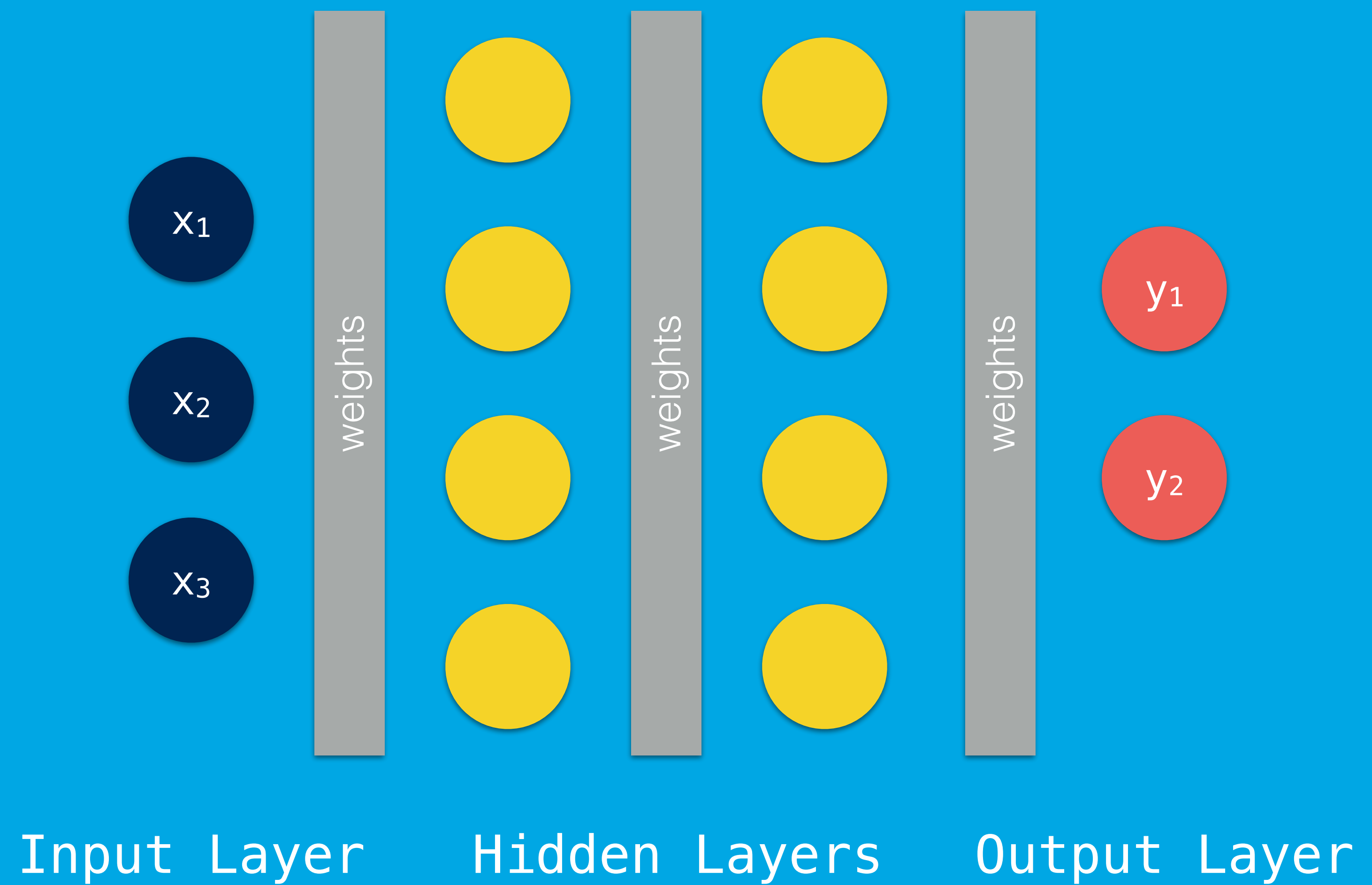
Neural Network (Shallow)

1 Hidden Layer



Neural Network (Deep)

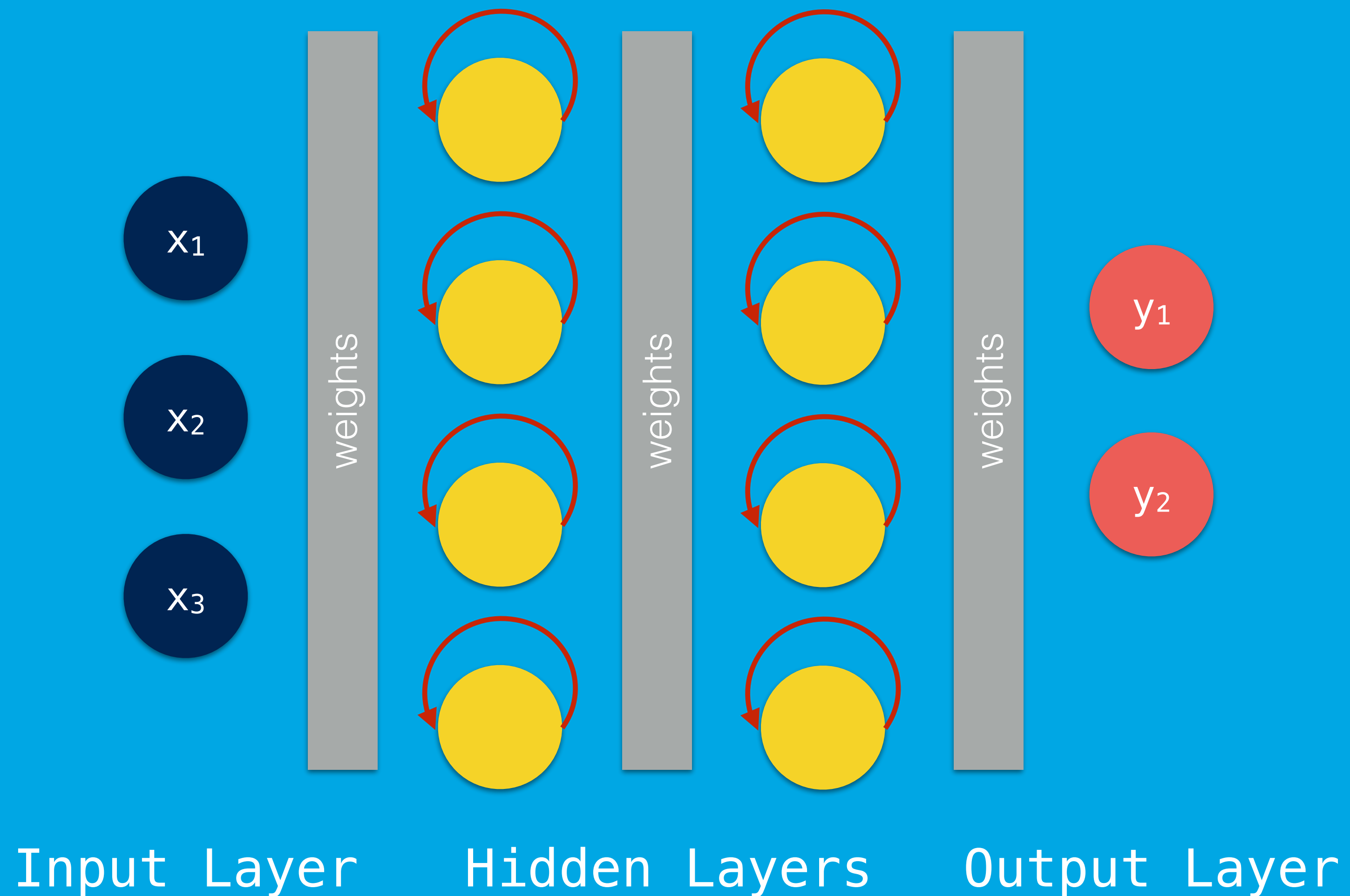
2+ Hidden Layers



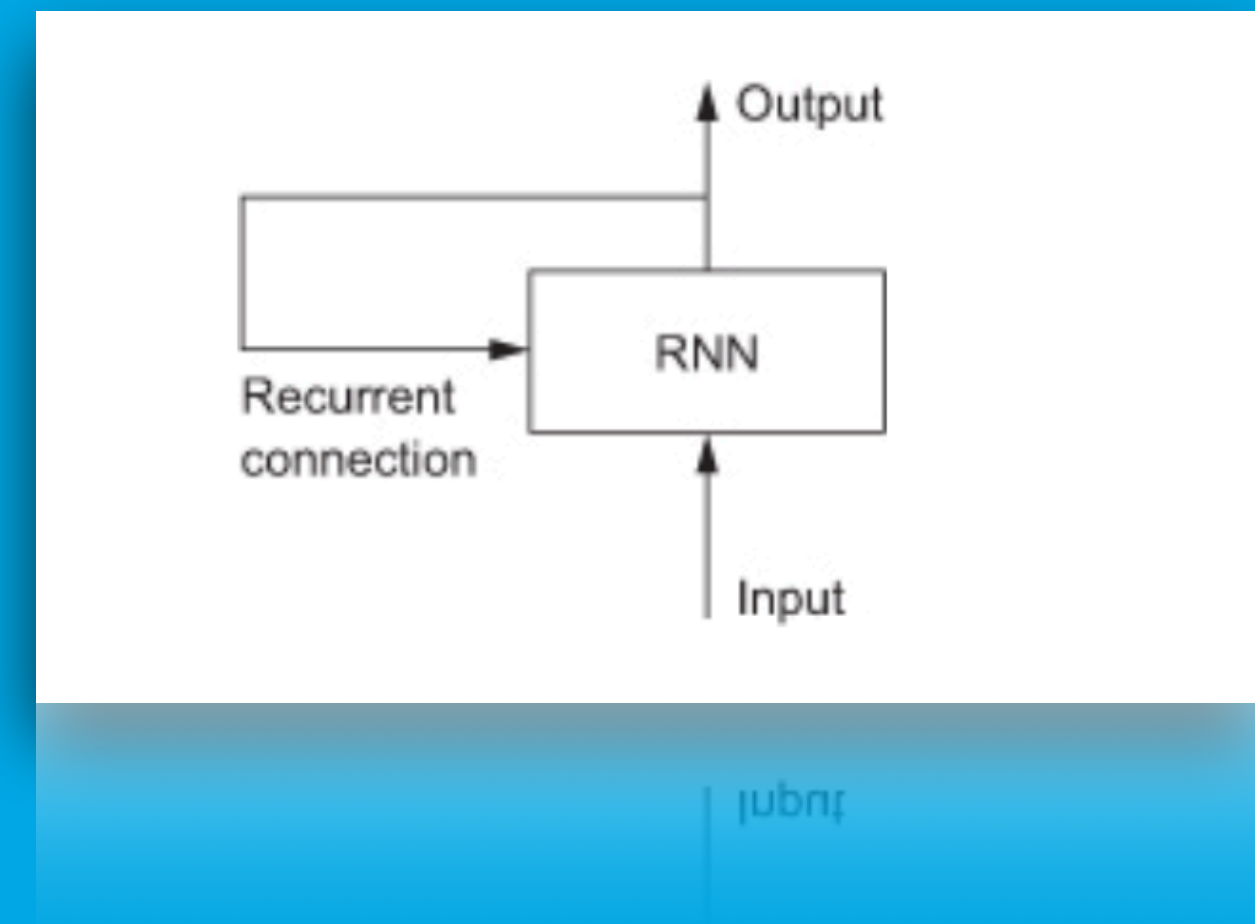
Recurrent Neural Networks

Recurrent Neural Network (Deep)

Remembering the Output/State



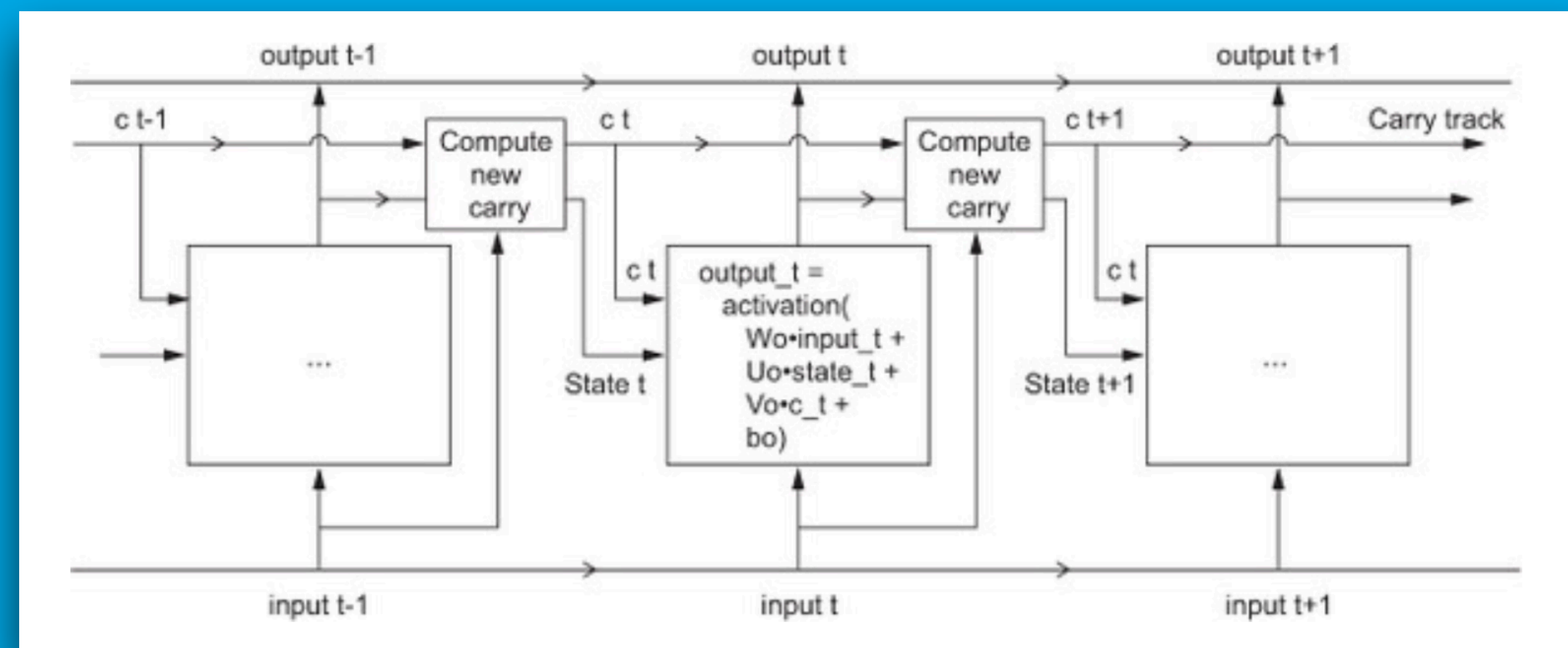
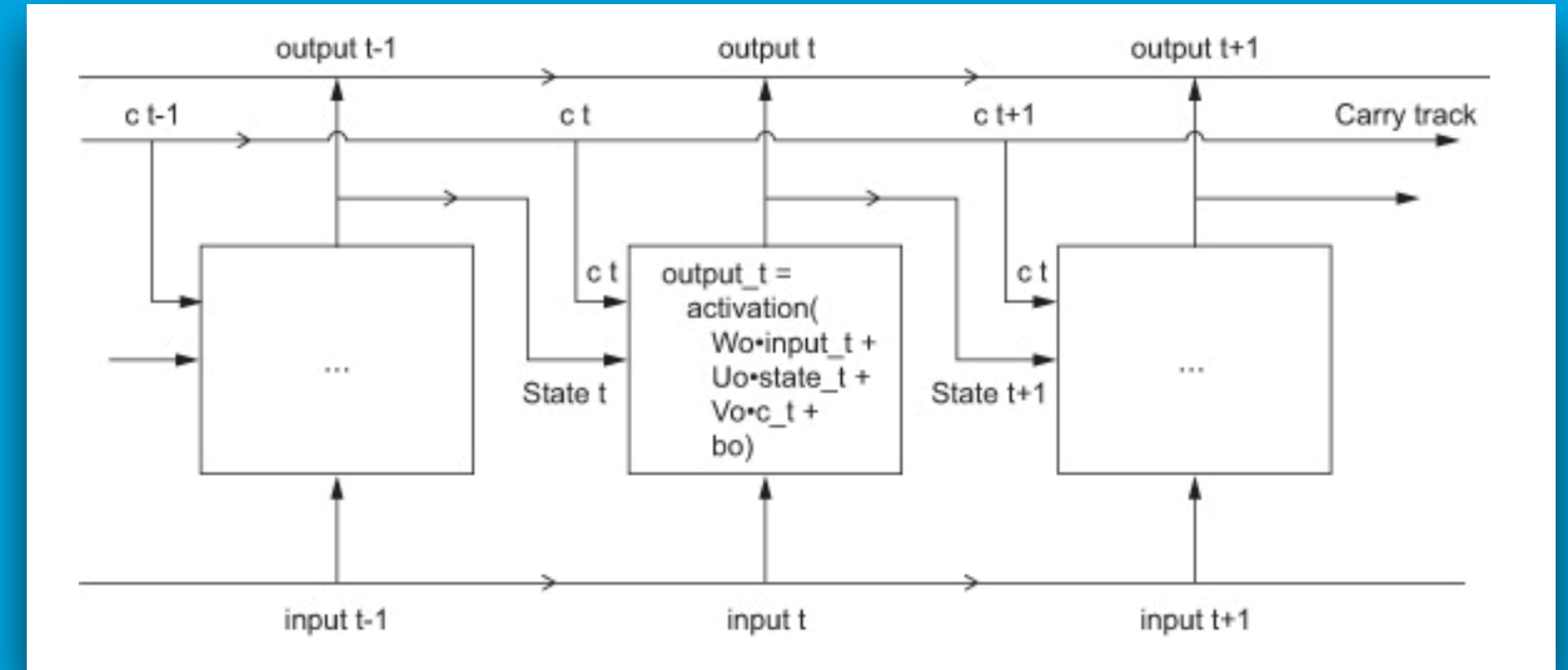
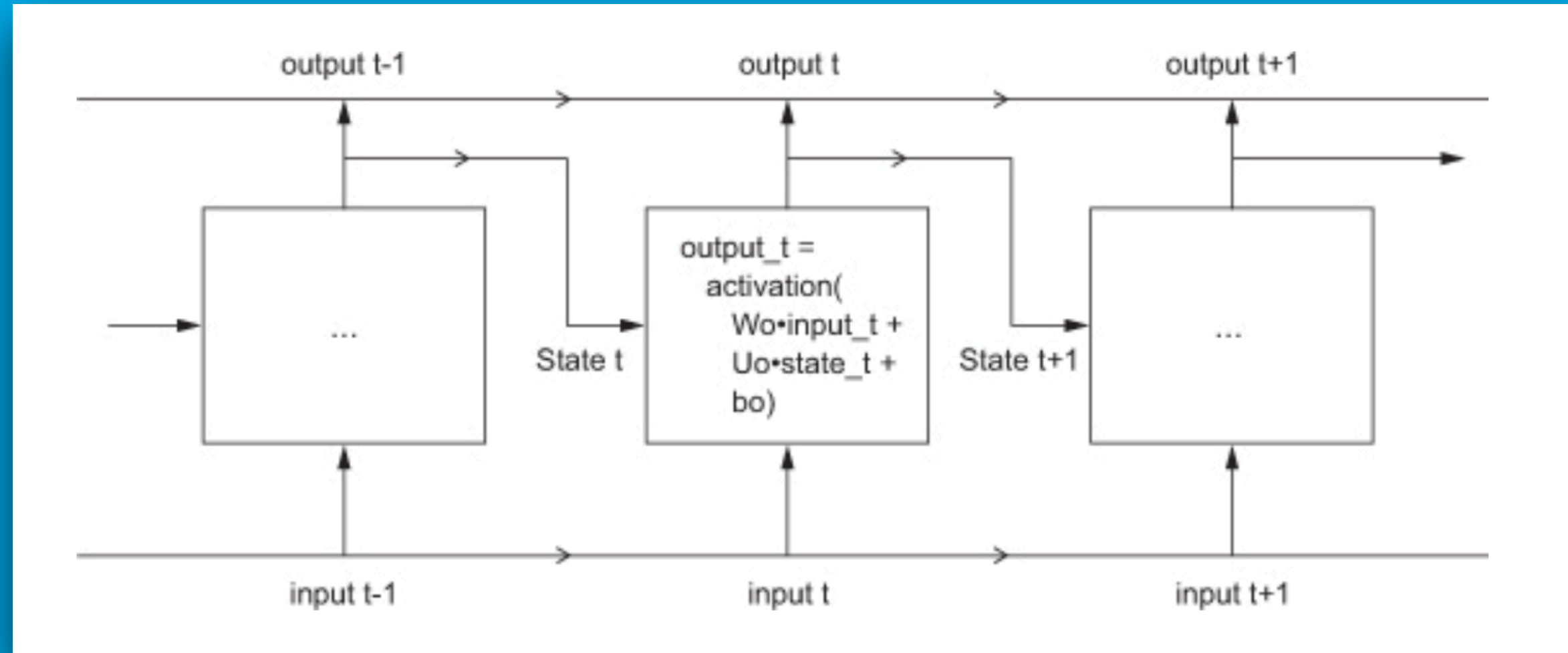
```
state_t = 0
for input_t in input_sequence:
    output_t = f(input_t, state_t)
    state_t = output_t
```



Source: Chollet (2017)

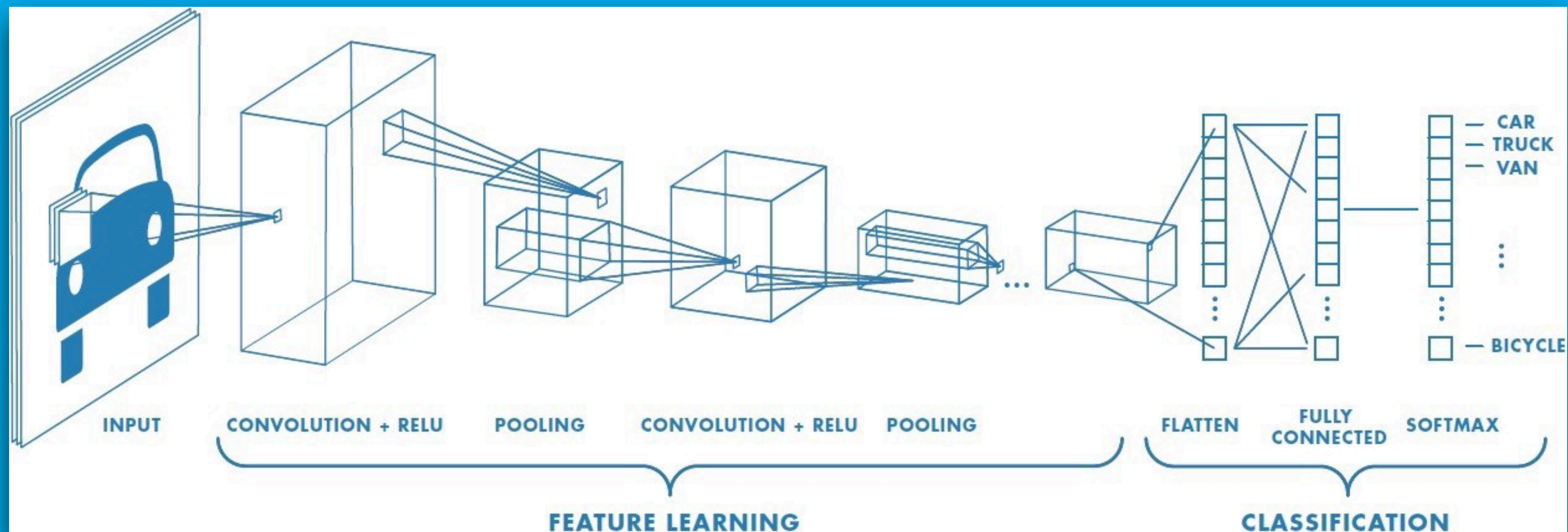
Long Short Term Memory (LSTM)

Adding a Carry Track



Source: Chollet (2017)

Convolutional Neural Networks



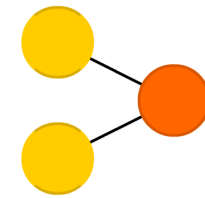
Overview of Neural Networks

A mostly complete chart of Neural Networks

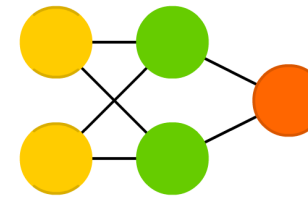
©2019 Fjodor van Veen & Stefan Leijnen asimovinstitute.org

- Input Cell
- Backfed Input Cell
- Noisy Input Cell
- Hidden Cell
- Probablistic Hidden Cell
- Spiking Hidden Cell
- Capsule Cell
- Output Cell
- Match Input Output Cell
- Recurrent Cell
- Memory Cell
- Gated Memory Cell
- Kernel
- Convolution or Pool

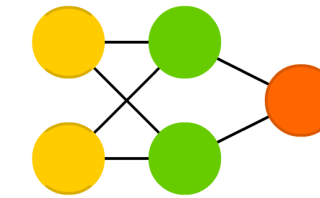
Perceptron (P)



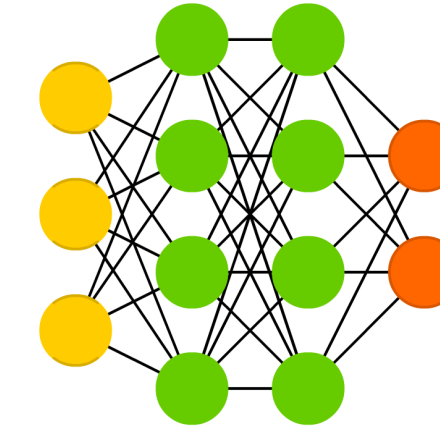
Feed Forward (FF)



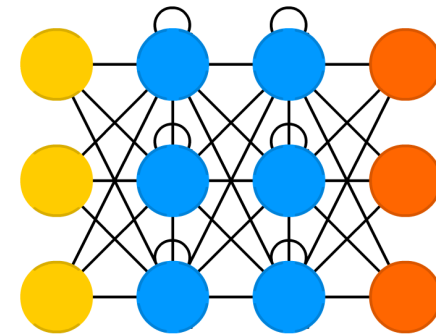
Radial Basis Network (RBF)



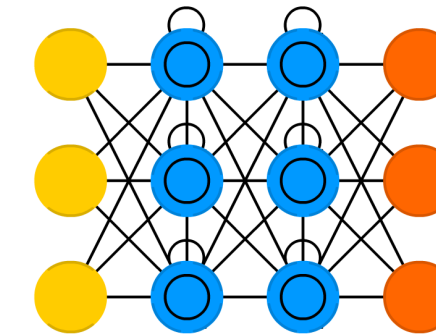
Deep Feed Forward (DFF)



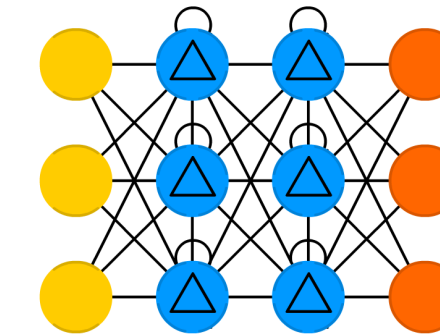
Recurrent Neural Network (RNN)



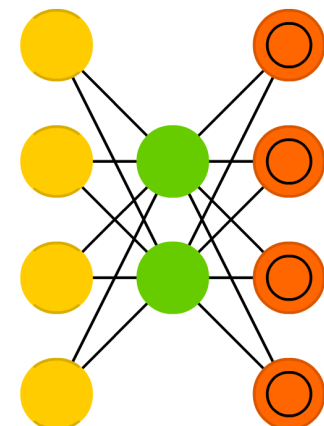
Long / Short Term Memory (LSTM)



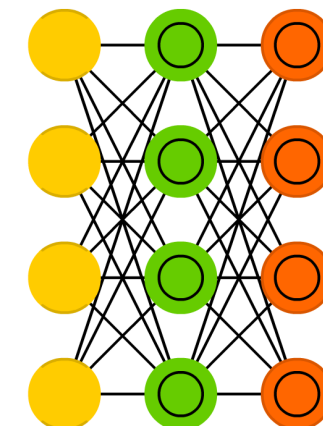
Gated Recurrent Unit (GRU)



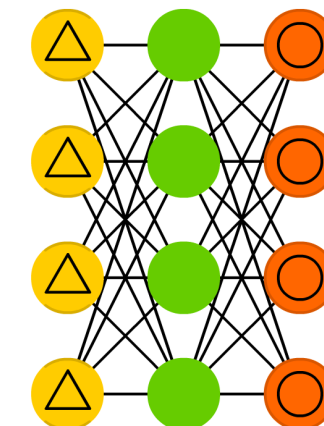
Auto Encoder (AE)



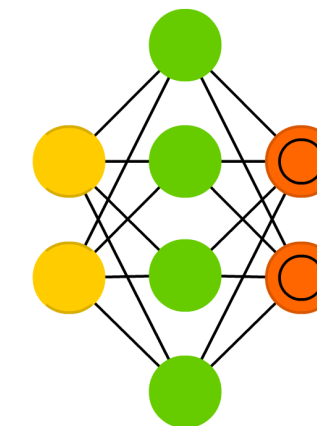
Variational AE (VAE)



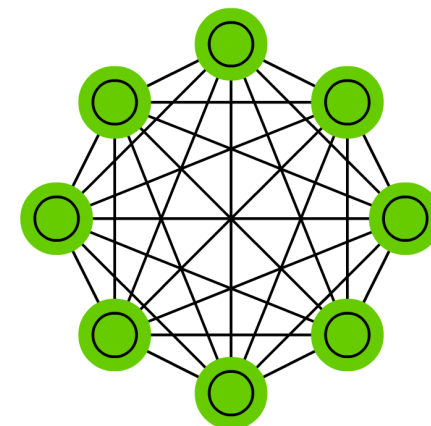
Denoising AE (DAE)



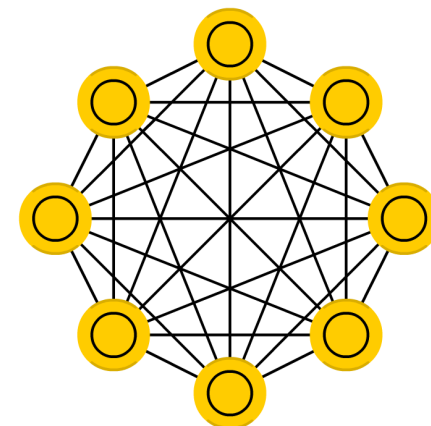
Sparse AE (SAE)



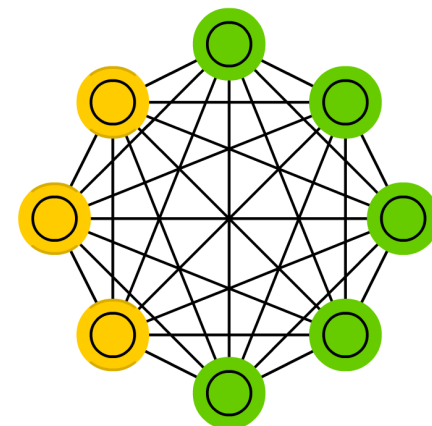
Markov Chain (MC)



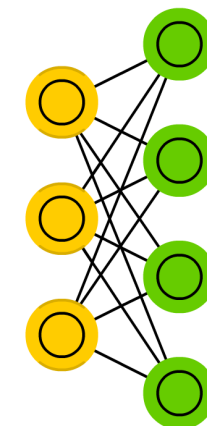
Hopfield Network (HN)



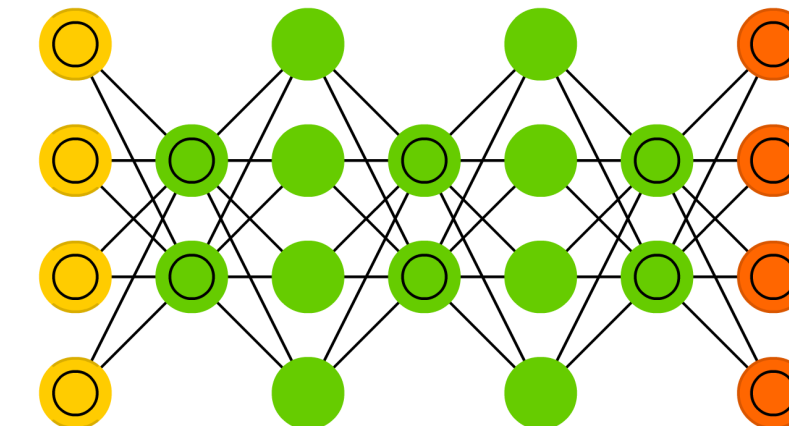
Boltzmann Machine (BM)



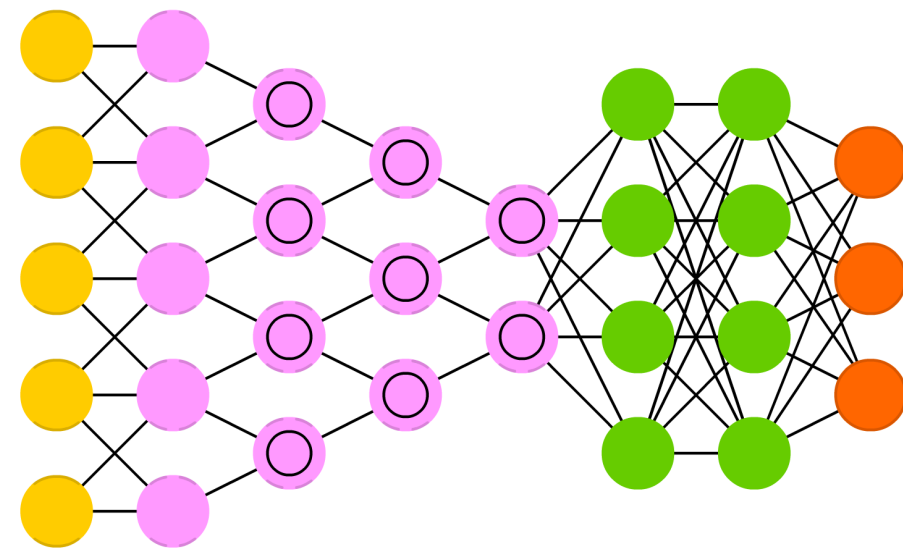
Restricted BM (RBM)



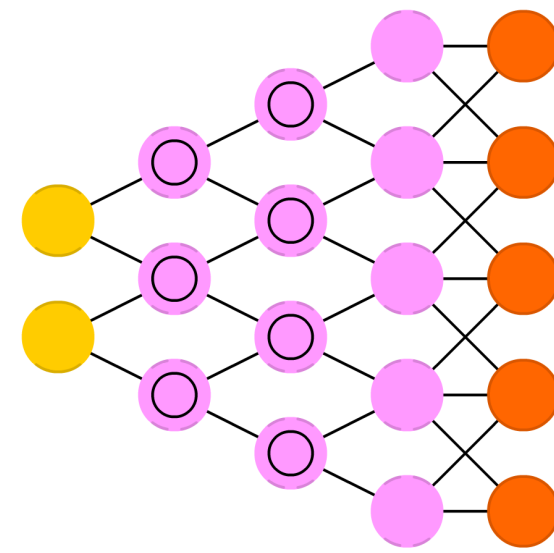
Deep Belief Network (DBN)



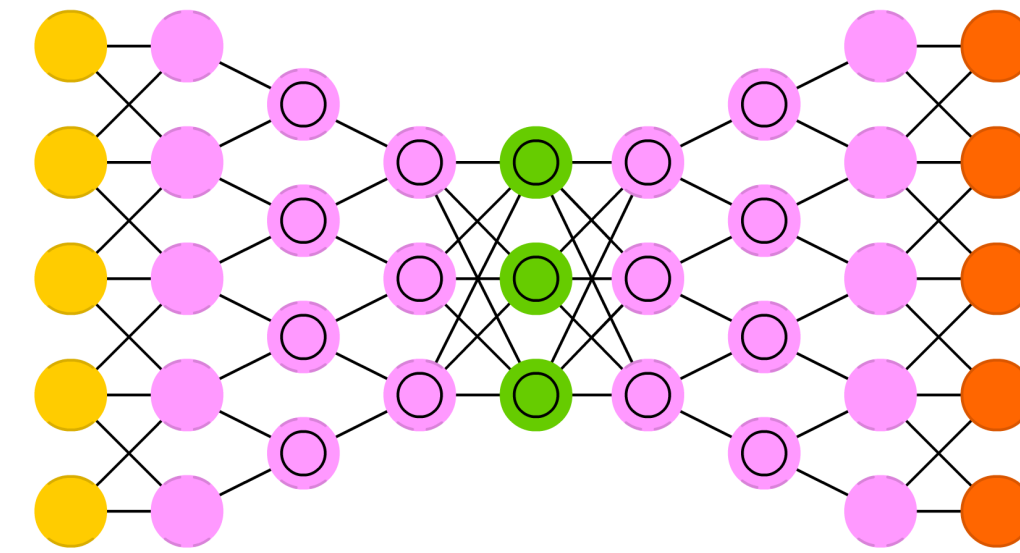
Deep Convolutional Network (DCN)



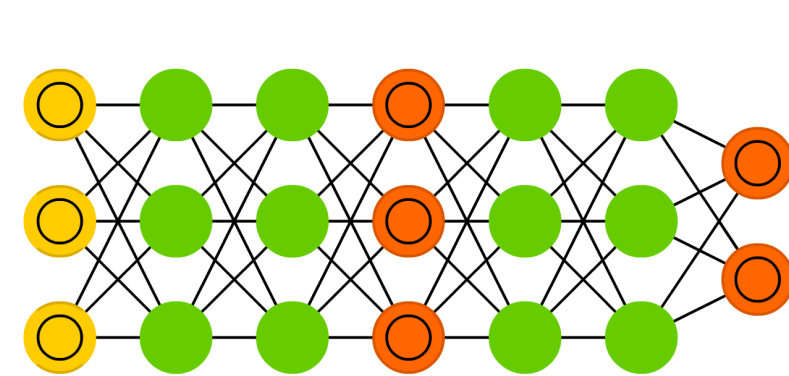
Deconvolutional Network (DN)



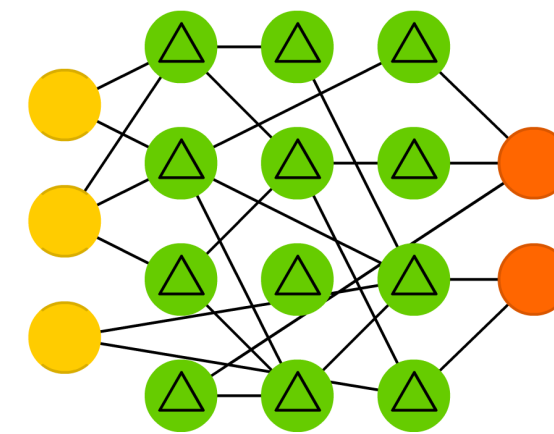
Deep Convolutional Inverse Graphics Network (DCIGN)



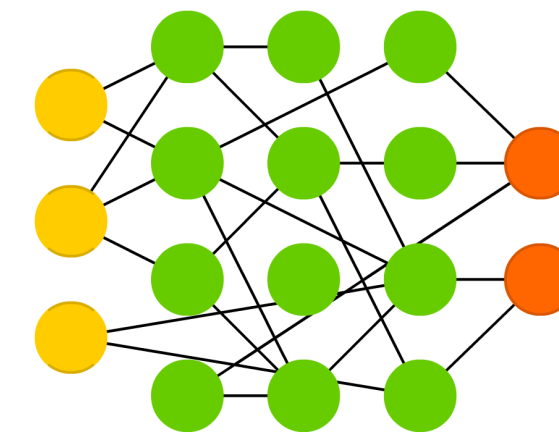
Generative Adversarial Network (GAN)



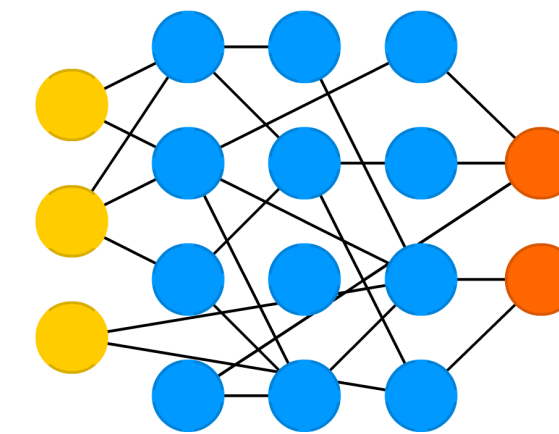
Liquid State Machine (LSM)



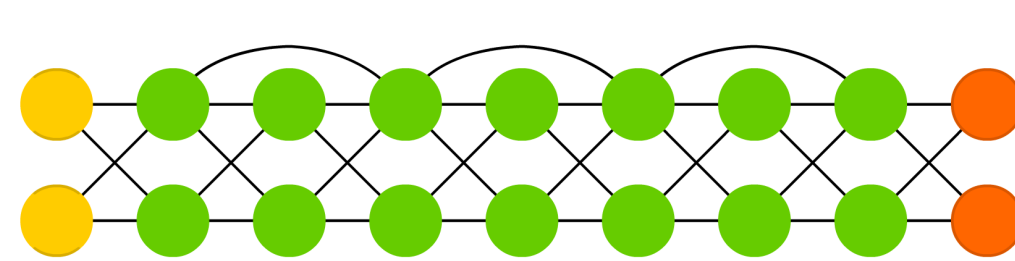
Extreme Learning Machine (ELM)



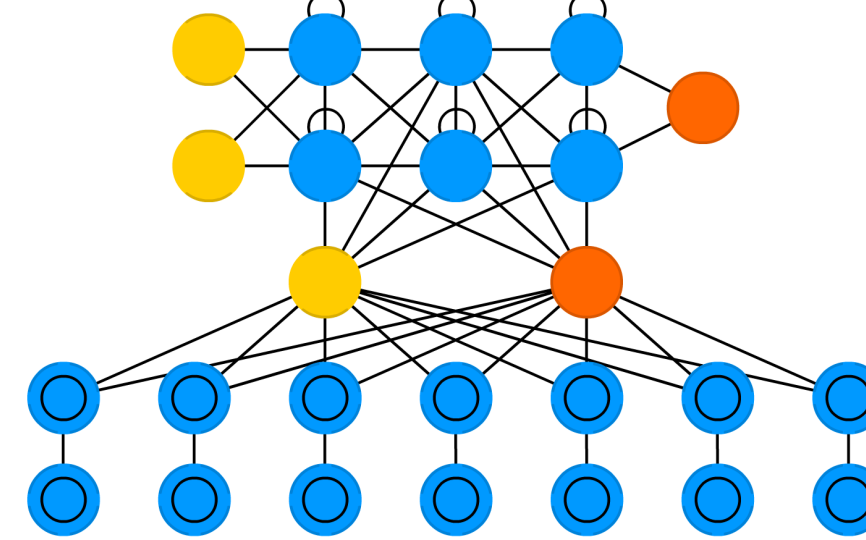
Echo State Network (ESN)



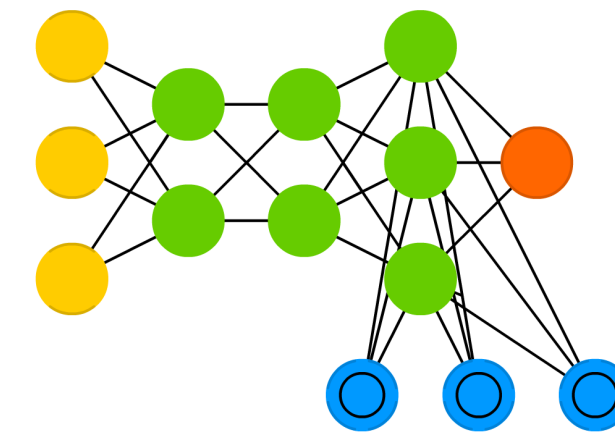
Deep Residual Network (DRN)



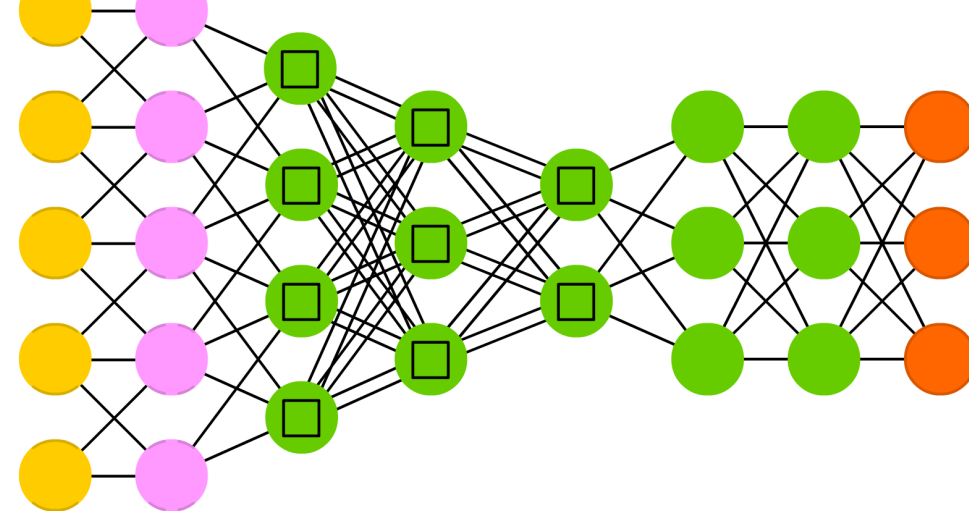
Differentiable Neural Computer (DNC)



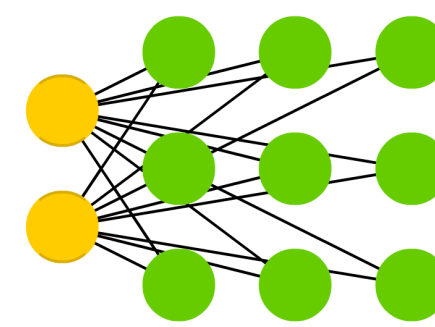
Neural Turing Machine (NTM)



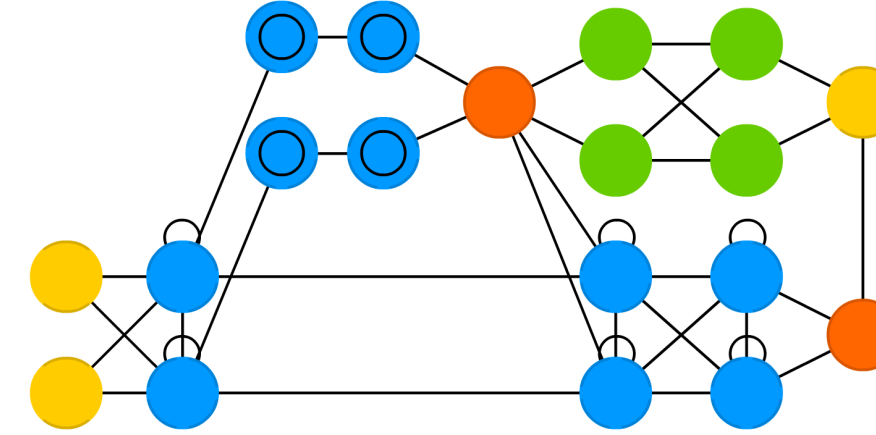
Capsule Network (CN)



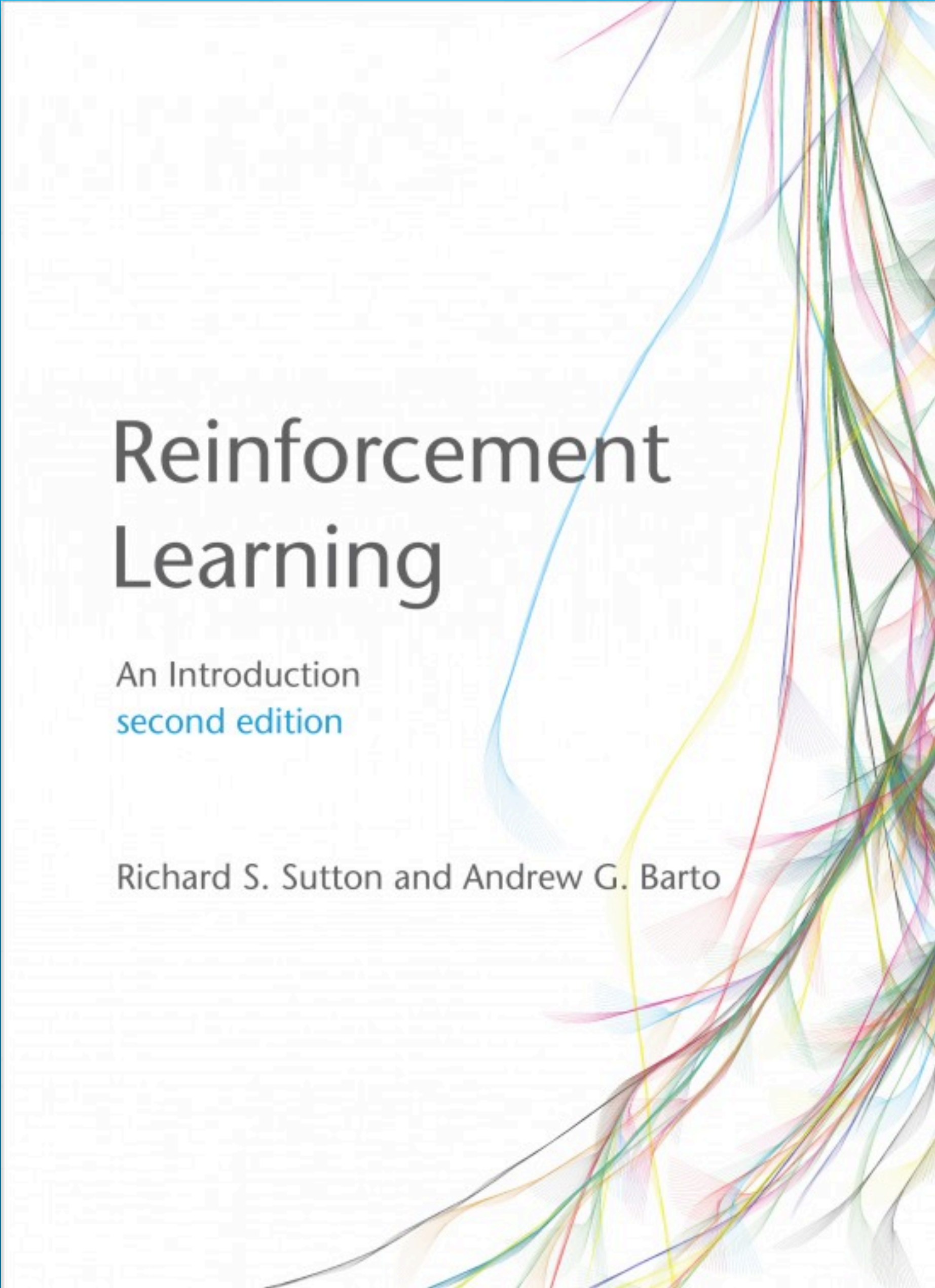
Kohonen Network (KN)



Attention Network (AN)



APPENDIX: Reinforcement Learning



Reinforcement Learning

An Introduction
second edition

Richard S. Sutton and Andrew G. Barto

“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

2024 M. SCHNEIDER

JUDEA PEARL
WINNER OF THE TURING AWARD
AND DANA MACKENZIE

**THE
BOOK OF
WHY**



**THE NEW SCIENCE
OF CAUSE AND EFFECT**

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.

Reinforcement Learning

–Q–Learning

Reward Function

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

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

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.

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

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

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 $\mathbb{R}^n$, 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://cpf.tpq.io> | training@tpq.io
@dyjh | YouTube

