Machine Learning for Finance

Workshop at Texas State University
November 2022

Dr. Yves J. Hilpisch

Introduction

THE PYTHON QUANTS

SERVICES
for financial institutions globally

EVENTS
for Python quants & algorithmic traders

TRAINING
about Python for finance & algorithmic trading

CERTIFICATION
in cooperation with university

BOOKS
about Python and finance

PLATFORM
for browser-based data analytics

OPEN SOURCE
Python library for financial analytics

http://tpq.io

The AI Machine

http://aimachine.io

Dr. Yves J. Hilpisch is founder and CEO of **The Python Quants** (http://tpq.io), a group focusing on the use of 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):

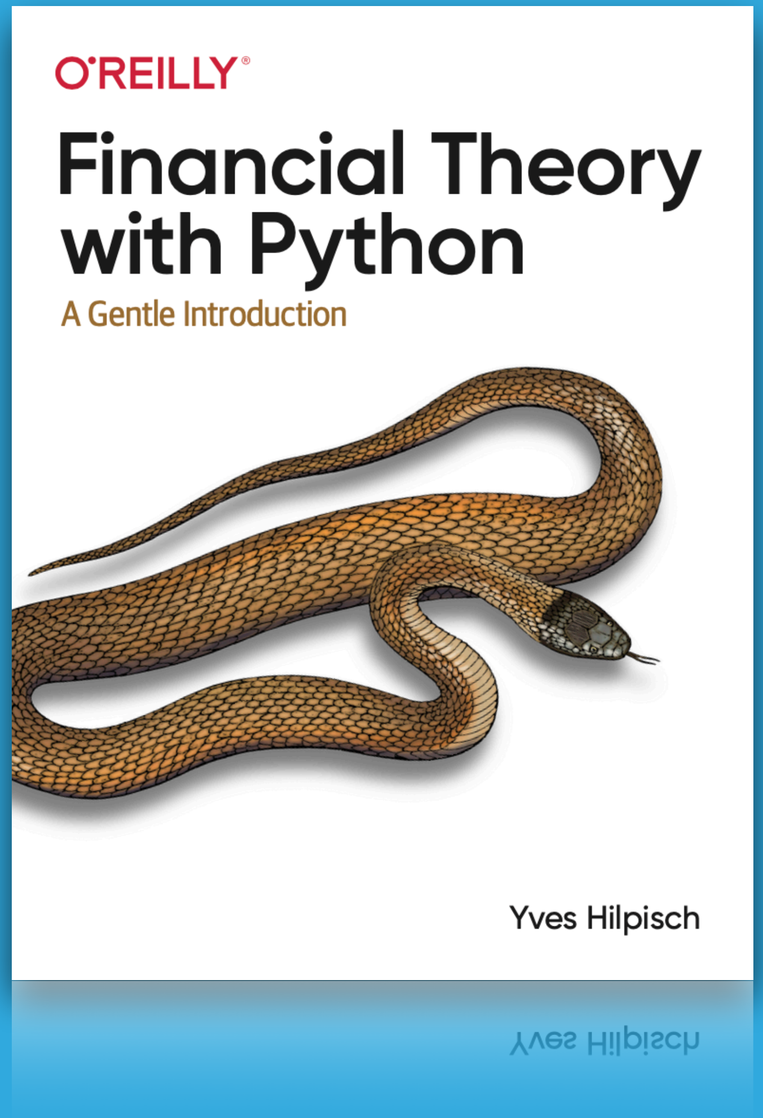
* Financial Theory with Python (2021, O'Reilly)
* Artificial Intelligence in Finance (2020, O'Reilly)
* Python for Algorithmic Trading (2020, O'Reilly)
* Python for Finance (2018, 2nd ed., O'Reilly)
* Listed Volatility and Variance Derivatives (2017, Wiley Finance)
* Derivatives Analytics with Python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

Yves is the director of the first online training program leading to **University Certificates in Python for Algorithmic Trading** (https://home.tpq.io/certificates/pyalgo) and **Computational Finance** (https://home.tpq.io/certificates/compfin). 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), New York (http://aifat.tpq.io), Frankfurt, Berlin, and Paris. He has given **keynote speeches** at technology conferences in the United States, Europe, and Asia.

http://hilpisch.com

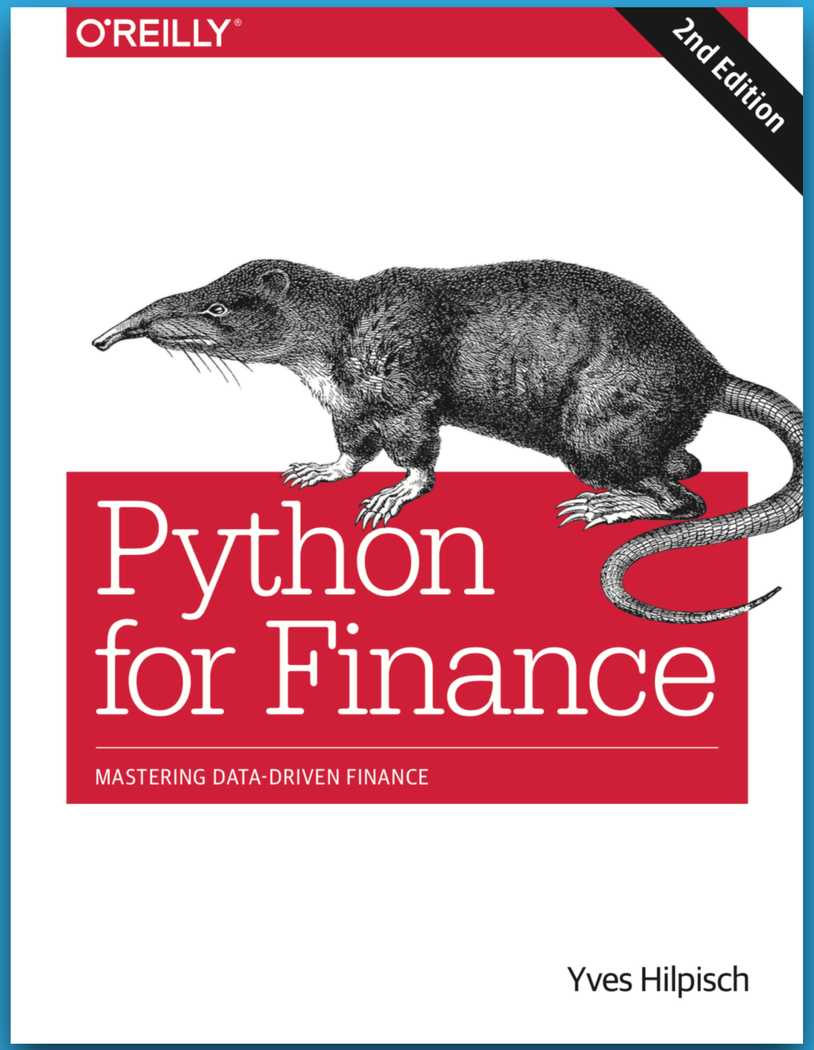
Financial Theory with Python – A Gentle Introduction



Finance with Python and Python environments
Basic Finance Concepts and Models:
- **Risk–Return**
- **Arbitrage Pricing**
- **Expected Utility Theory**
- **Mean–Variance Portfolio Theory**
- **Capital Asset Pricing Model**
- **Portfolio Optimization**

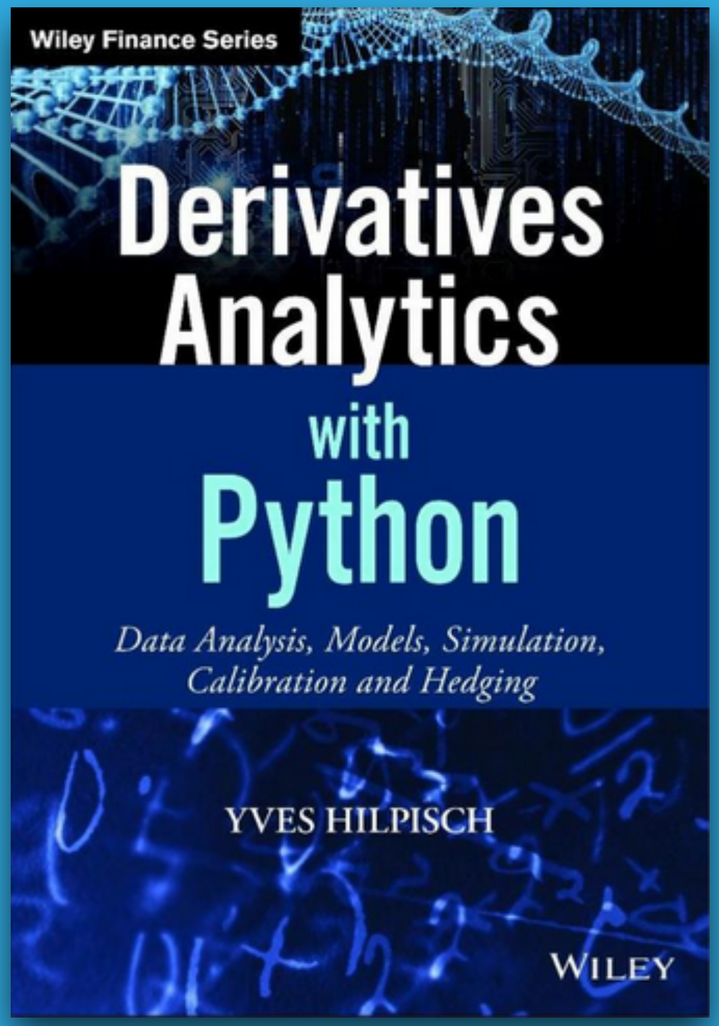
Basic Python Concepts and Packages:
- **Major Python Idioms**
- **NumPy Package**
- **SciPy & SymPy Packages**

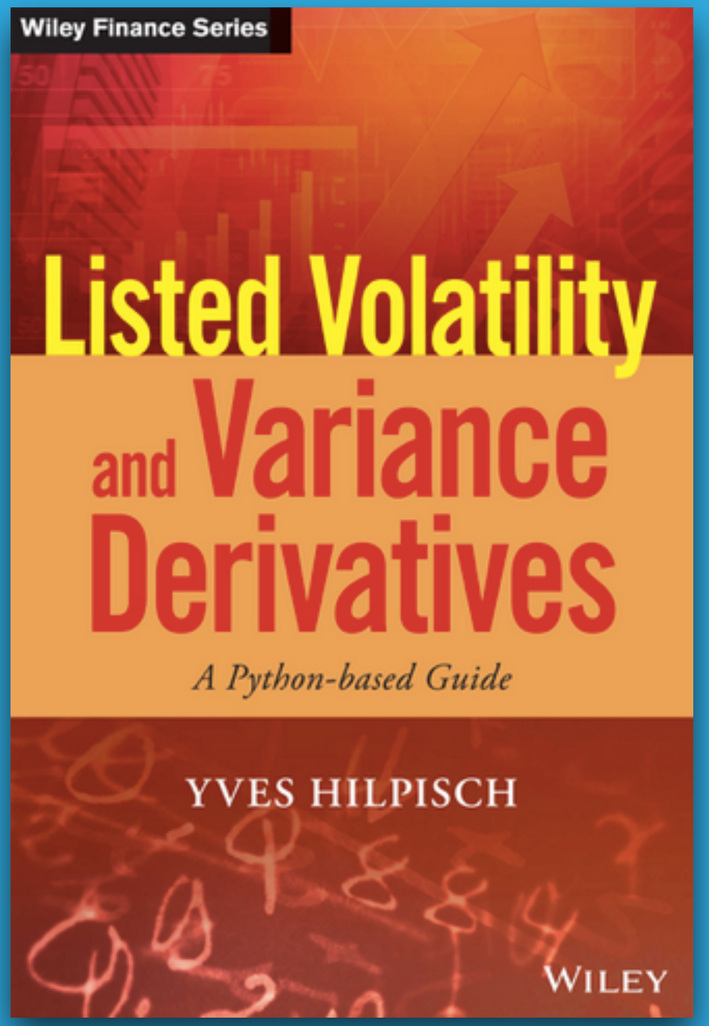
Python for Finance



http://books.tpq.io

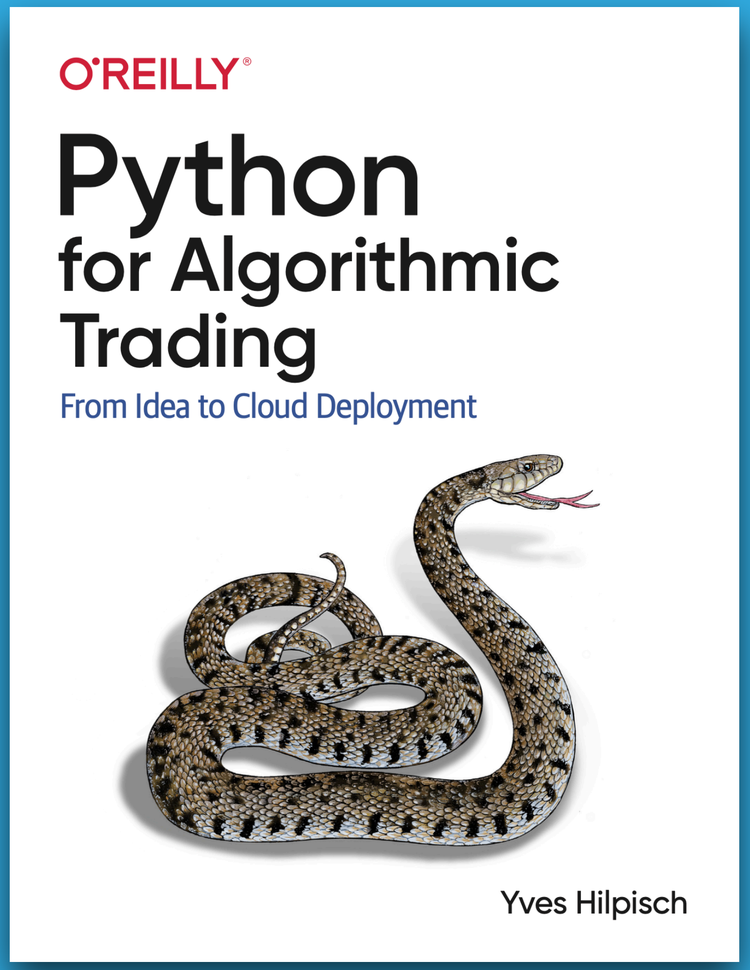
Quant Finance with Python

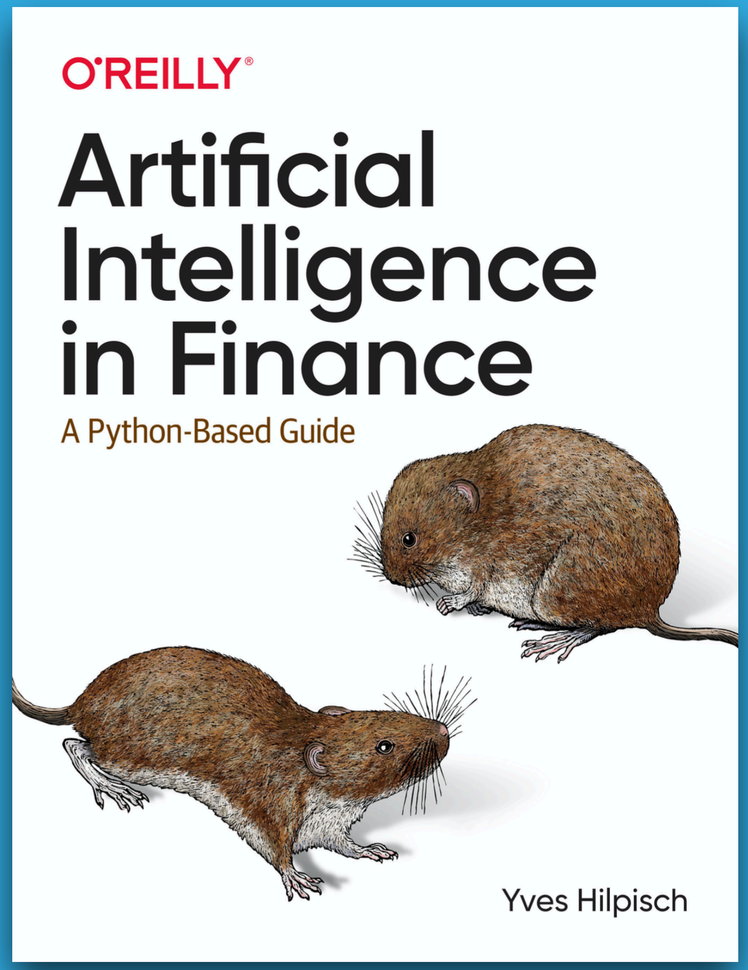




http://books.tpq.io

Python & AI for Finance & Trading





http://books.tpq.io

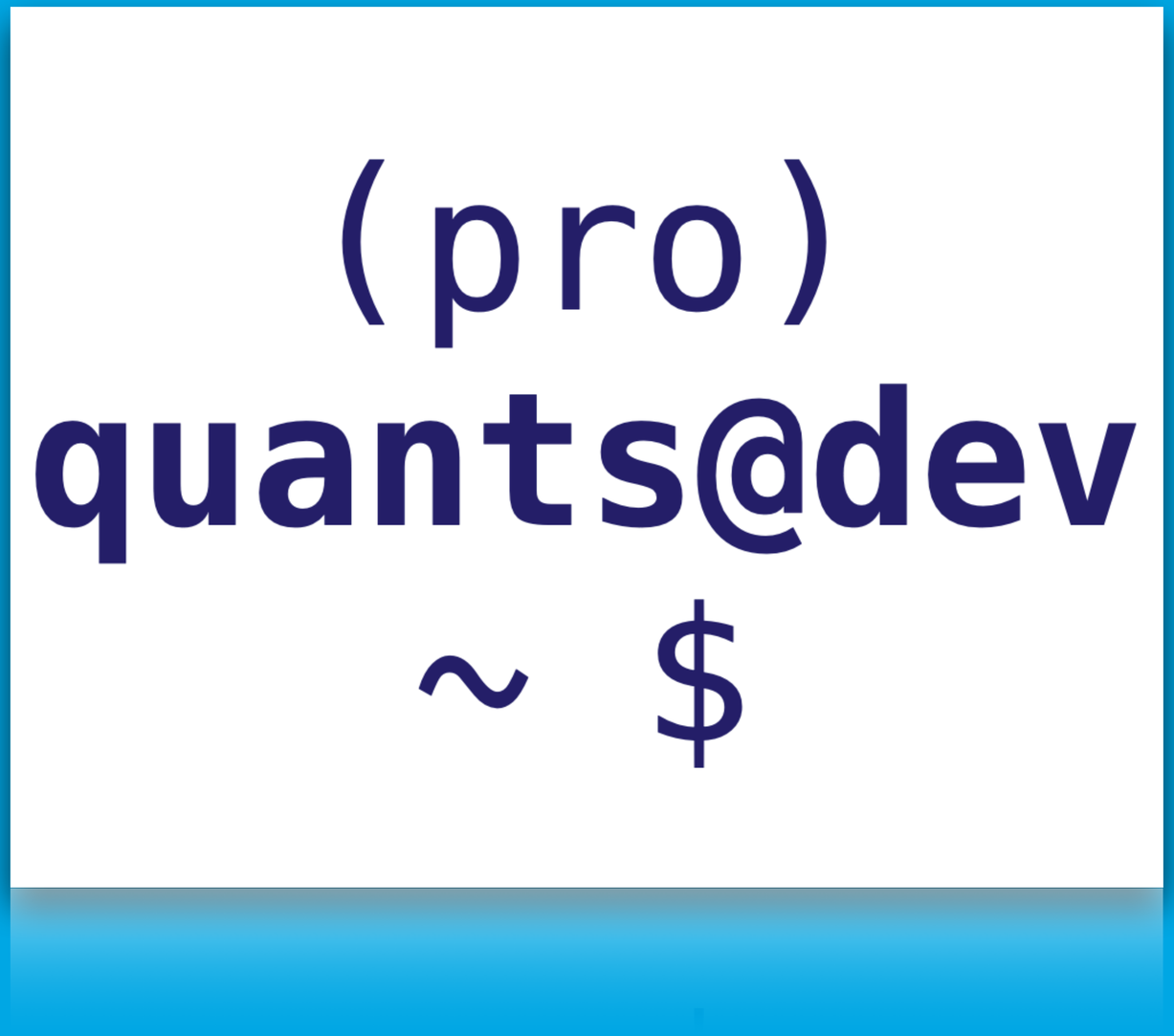
4 to 12 months
live & self-paced

250+ hours
of instruction

25,000+ lines
of code

2,500+ pages
HTML/PDF

https://platinum.tpq.io

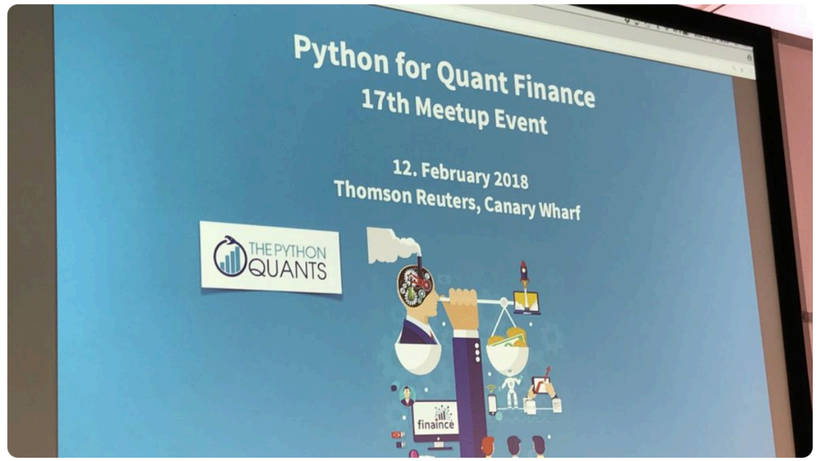


Community of
professional & aspiring
quant developers &
quant researchers.

750 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

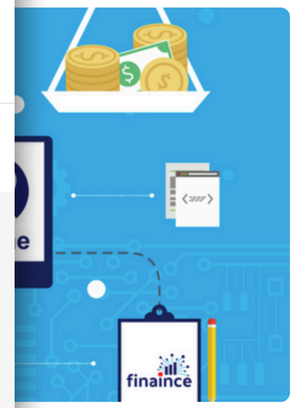
Join this group

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

Agenda

DAY 1

- **Data Science Packages**
  - Jupyter Lab
  - NumPy, pandas
- **Basic Finance**
  - Option Pricing
  - Market Completeness
  - Portfolio Selection
  - Neural Nets from Scratch
- **Basic Statistical Methods**
  - Descriptive Statistics
  - Approximation
  - Pattern Frequency

DAY 2

- Types of Machine Learning
  - Unsupervised Learning
  - Supervised Learning
- **Finance Applications**
  - Regime Detection
  - Black-Scholes-Merton
  - Efficient Markets
- **Machine Learning as a Process**
  - Success, Capacity
  - Bias vs. Variance
  - Cross-Validation
- **Deep Learning**
  - Interactive Neural Networks
  - Neural Network Classes
  - Dense Neural Networks

Statistical & Machine Learning

Mathematics.
Function generates data.

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

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

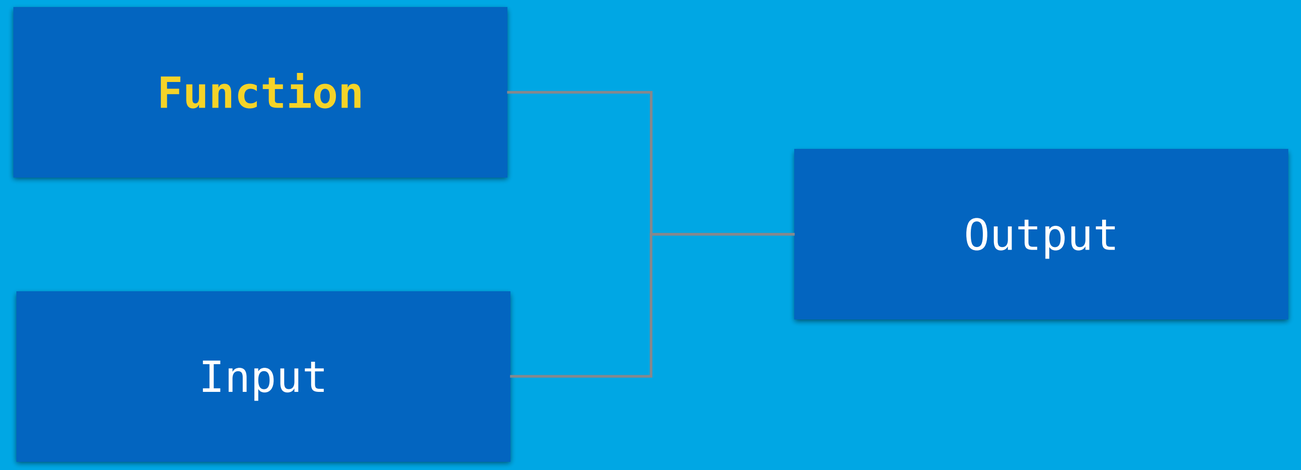
Statistics.
Data generates function.

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

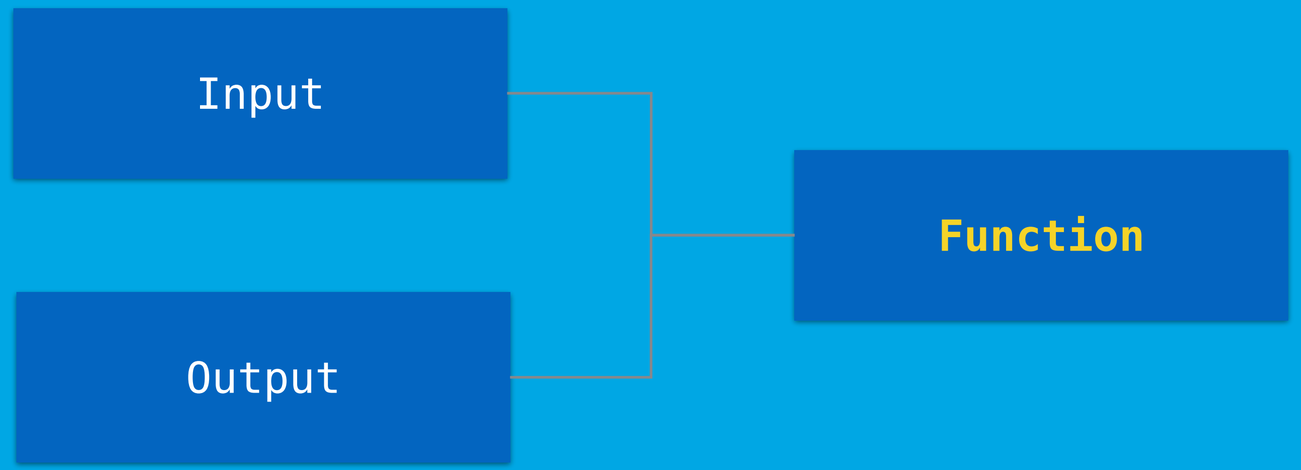
$$\hat{f}(x) = \alpha + \beta x \approx y$$

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

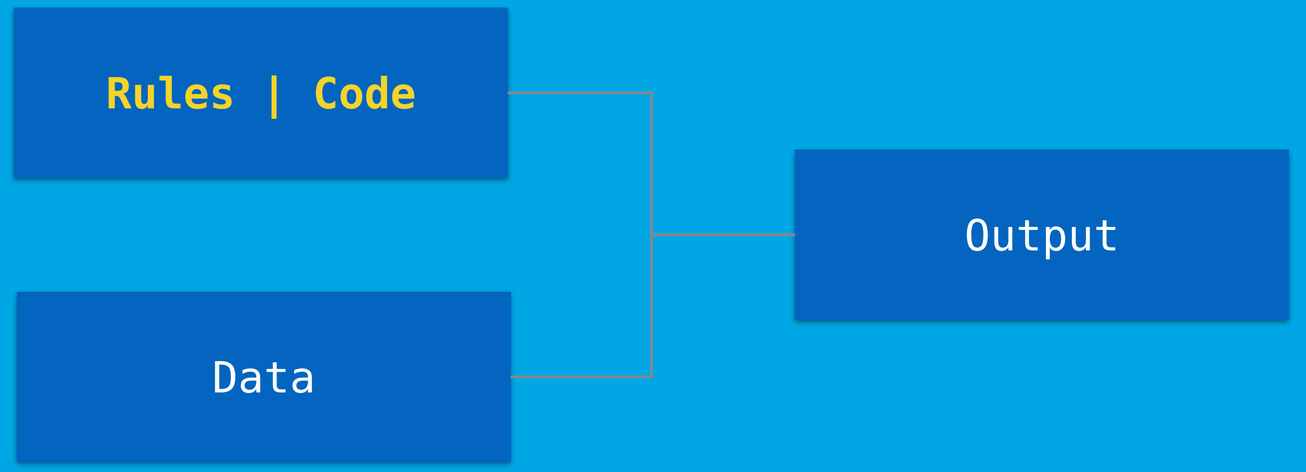
Mathematics.



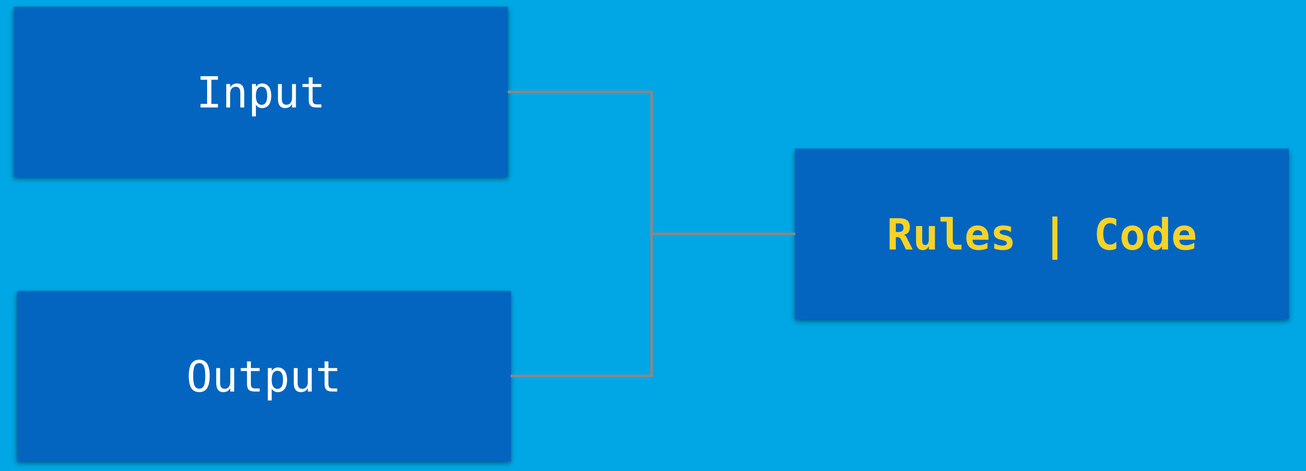
Statistics.



Programming.

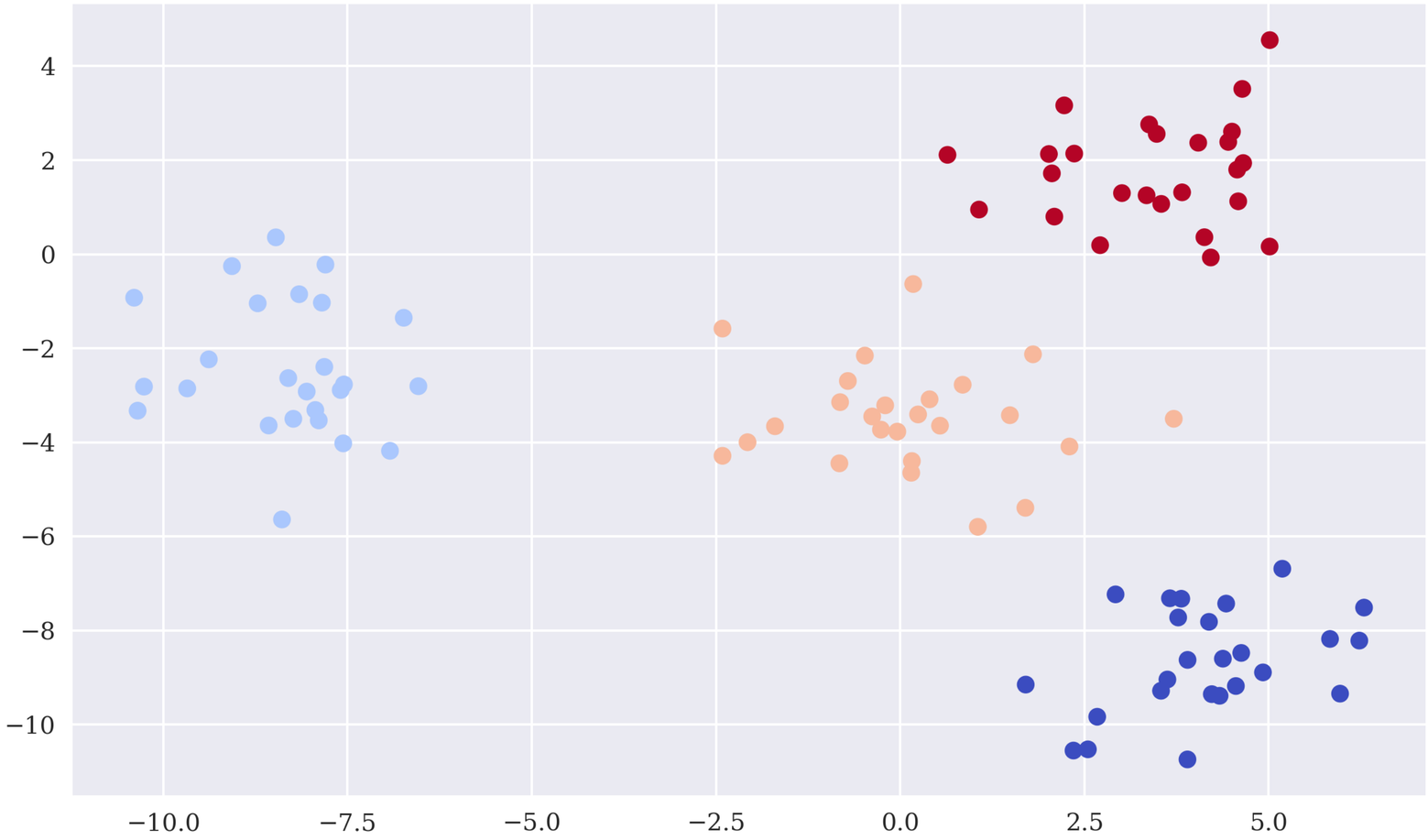


Machine Learning.

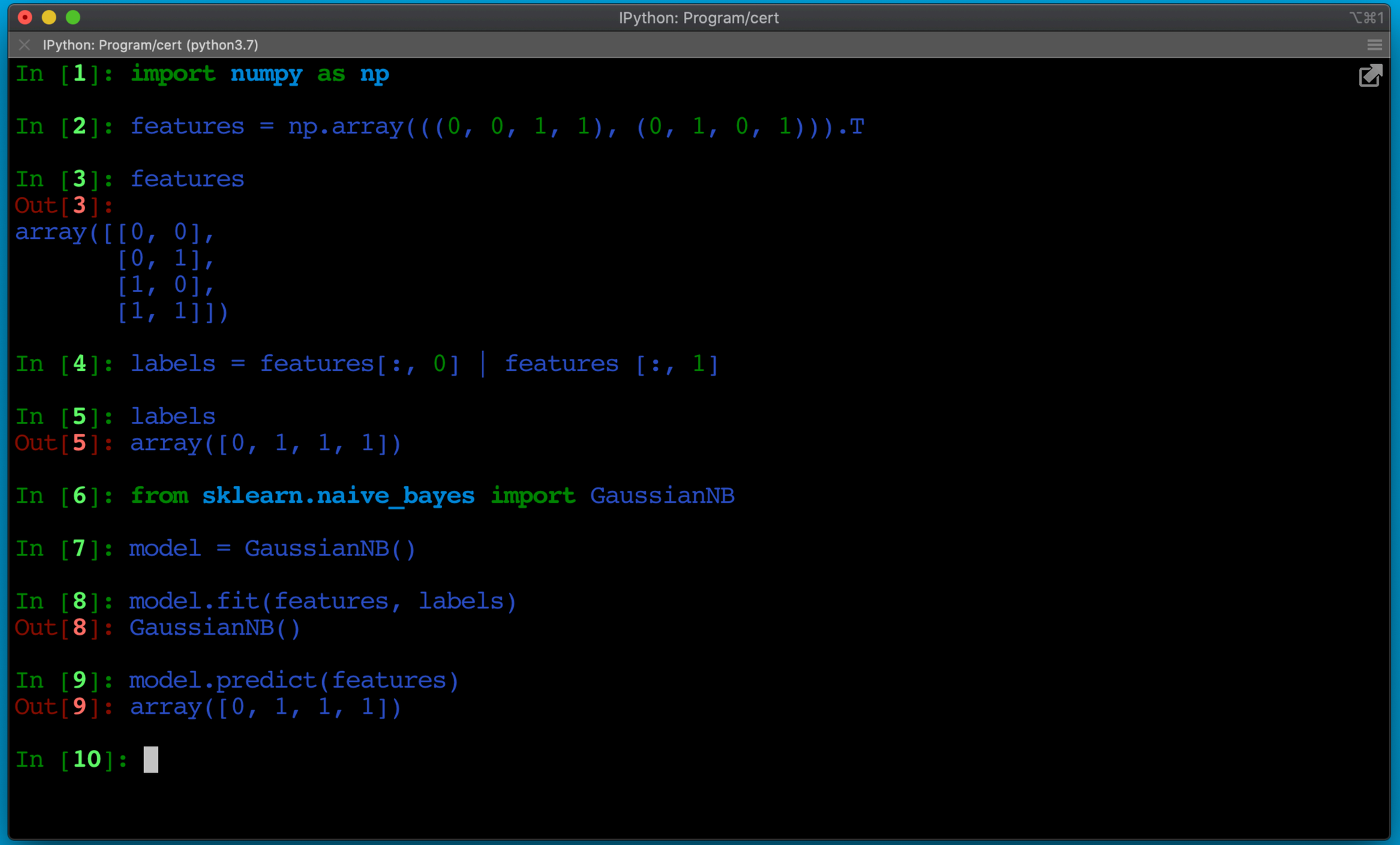


Types of Learning

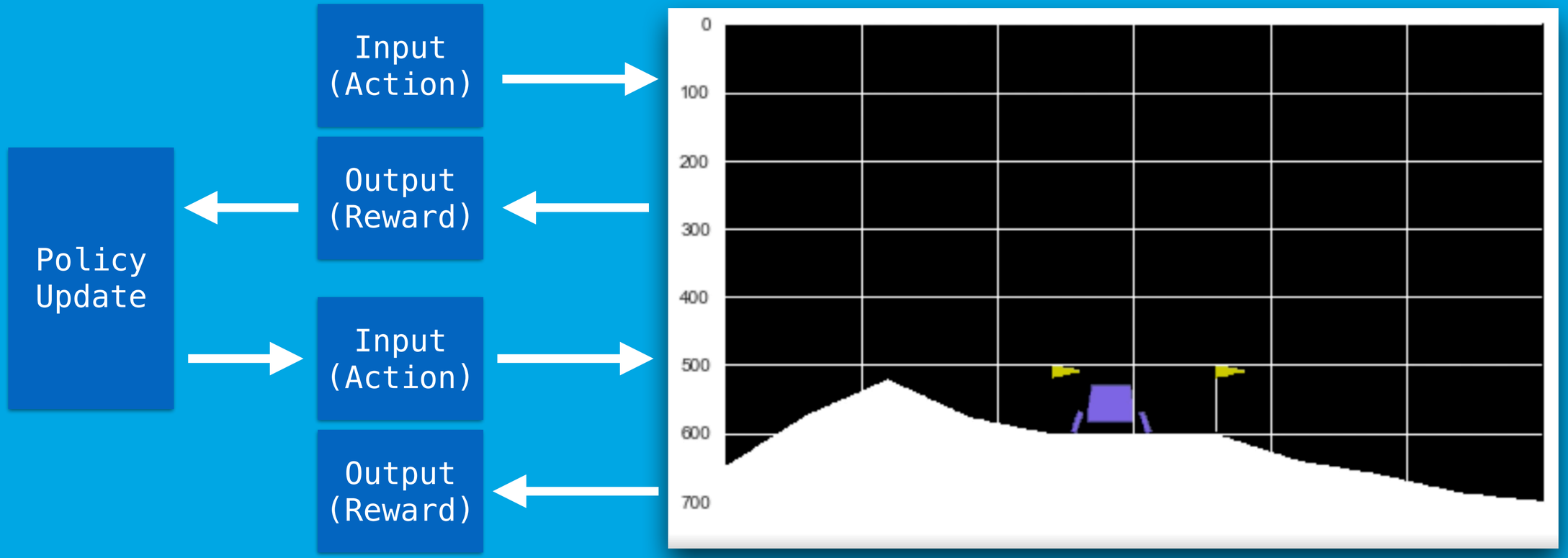
Unsupervised learning (UL)
These are algorithms that learn from a given sample data set of features (input) values only. They are supposed to learn about the input data set, given, for example, some guiding parameters. Clustering algorithms fall into that category. In a financial context, such algorithms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


4
2
0
−2
−4
−6
−8
−10
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0

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.

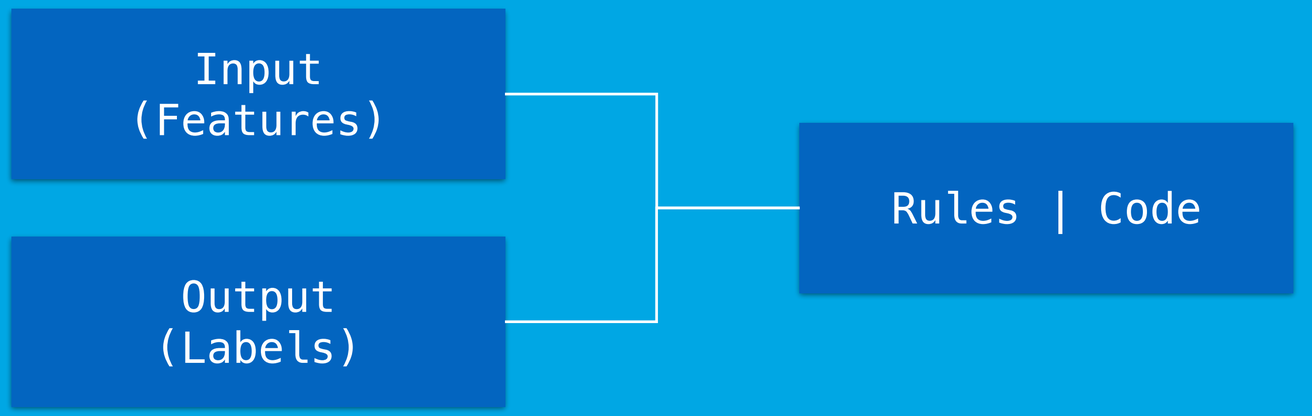

IPython: Program/cert
⌥⌘1
IPython: Program/cert (python3.7)
In [1]: import numpy as np
In [2]: features = np.array(((0, 0, 1, 1), (0, 1, 0, 1))).T
In [3]: features
Out[3]:
array([[0, 0],
[0, 1],
[1, 0],
[1, 1]])
In [4]: labels = features[:, 0] | features [:, 1]
In [5]: labels
Out[5]: array([0, 1, 1, 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.

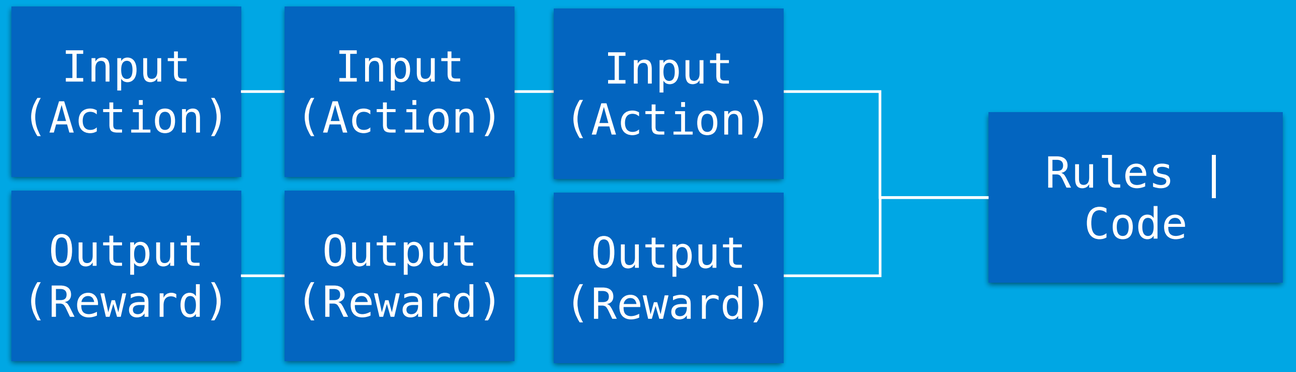

Input
(Action)
Output
(Reward)
Policy
Update
Input
(Action)
Output
(Reward)
0
100
200
300
400
500
600
700

Unsupervised Learning.

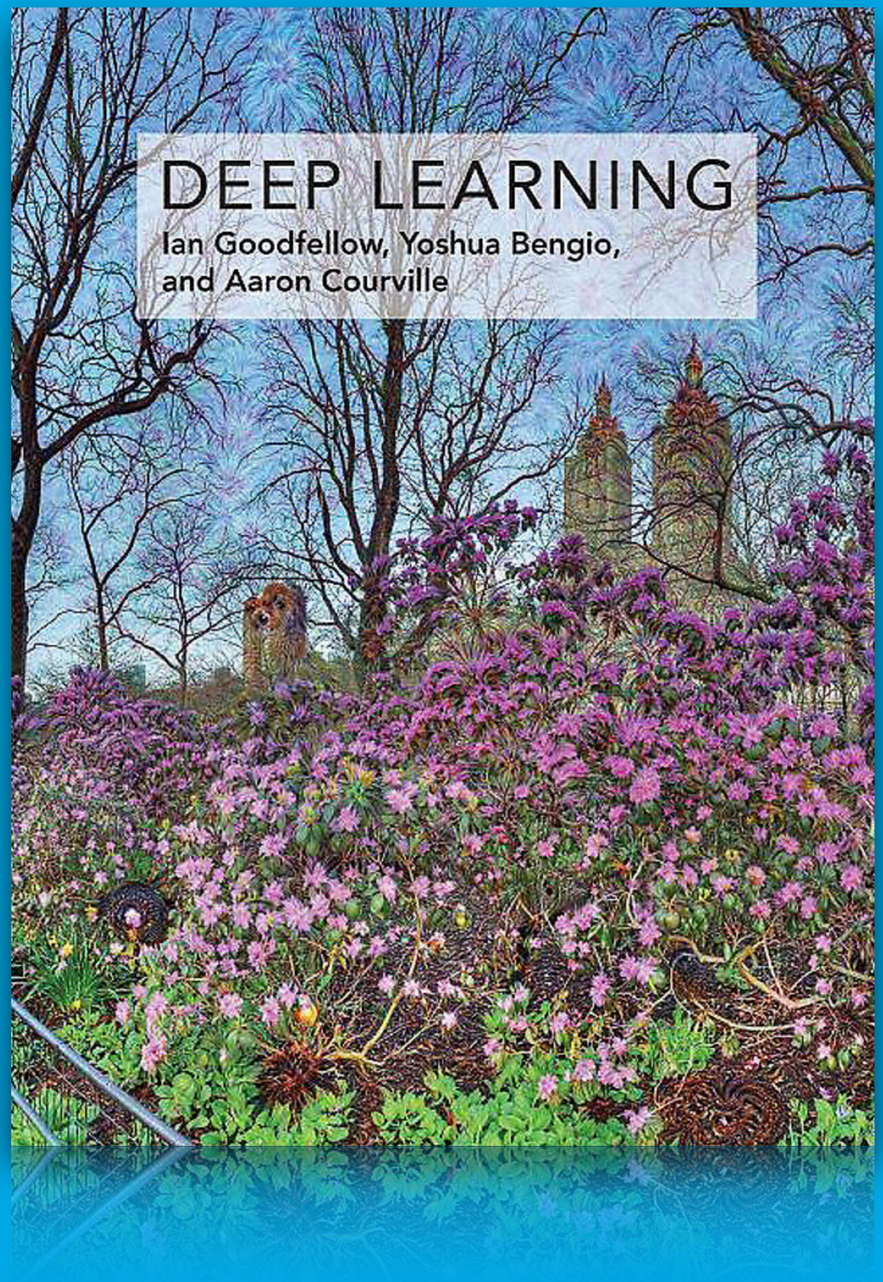
Supervised Learning.



Reinforcement Learning.



Dense Neural Networks

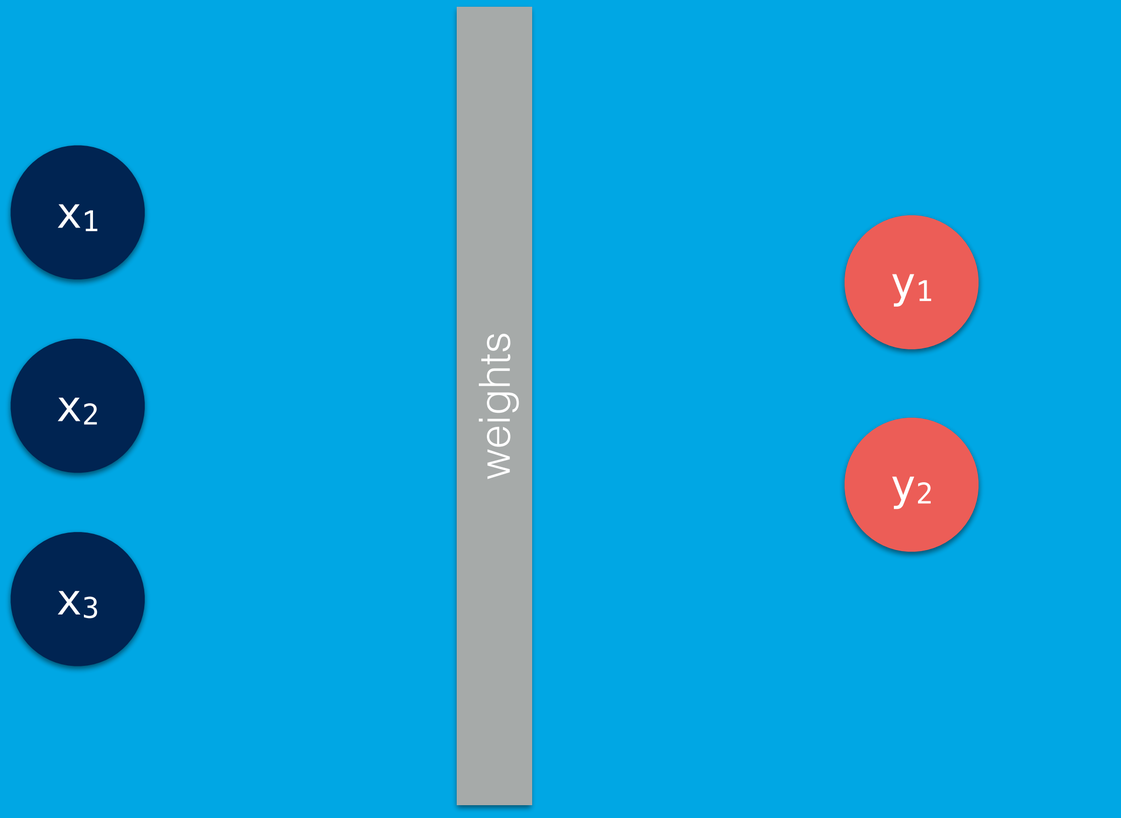


“A major source of difficulty in many real-world artificial intelligence applications is that many of the factors of variation influence every single piece of data we are able to observe.”

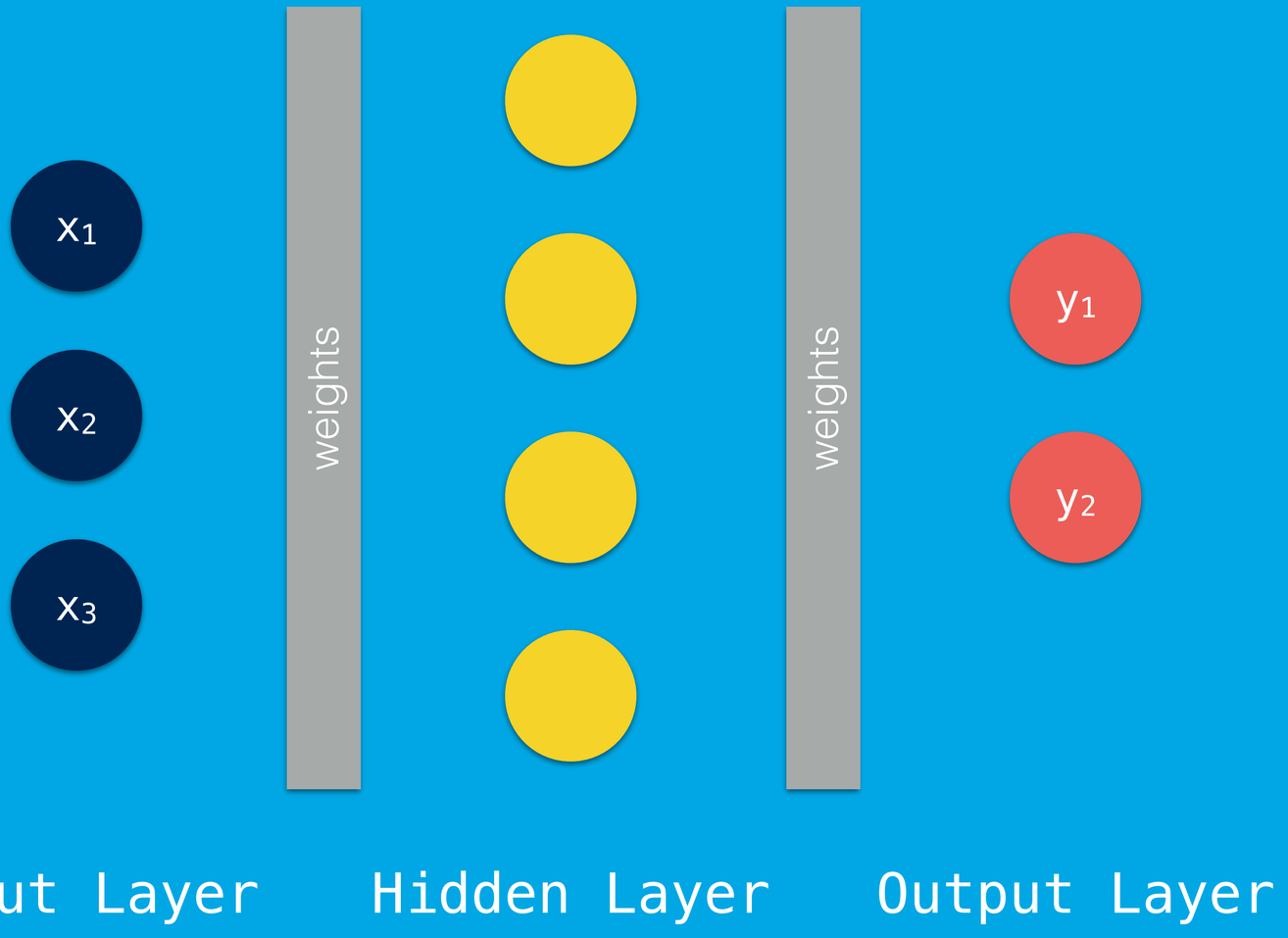
“Deep learning solves this central problem in representation learning by introducing representations that are expressed in terms of other, simpler representations. Deep learning enables the computer to build complex concepts out of simpler concepts.”

“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.”

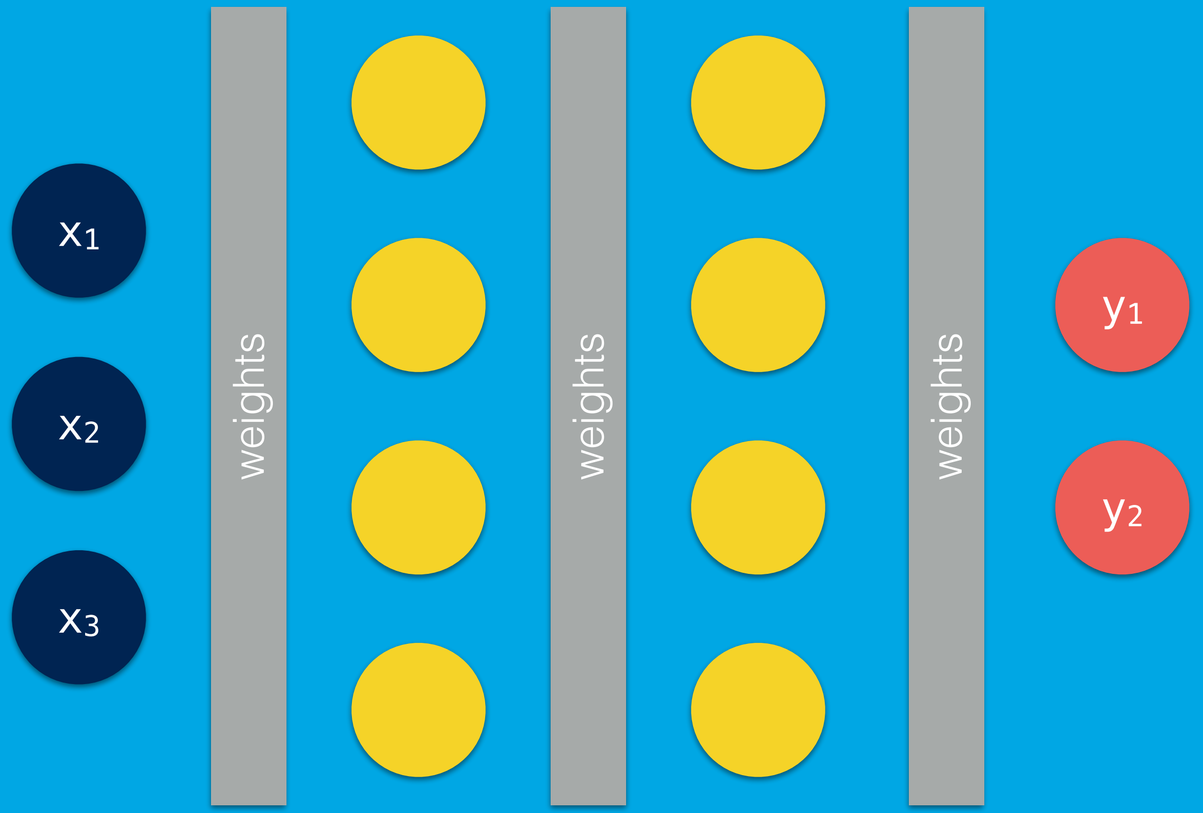
Neural Network (Simple)
0 Hidden Layers



Neural Network (Shallow)
1 Hidden Layer

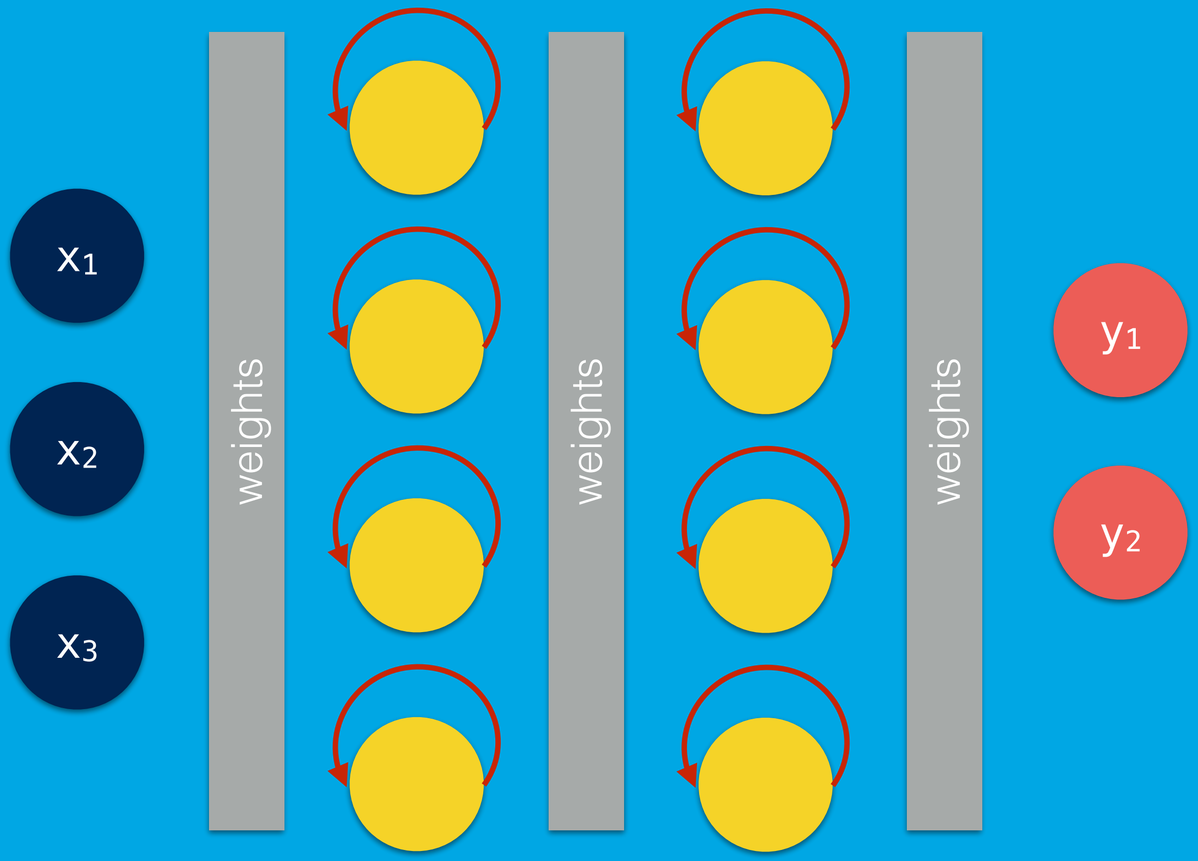


Neural Network (Deep)
2+ Hidden Layers



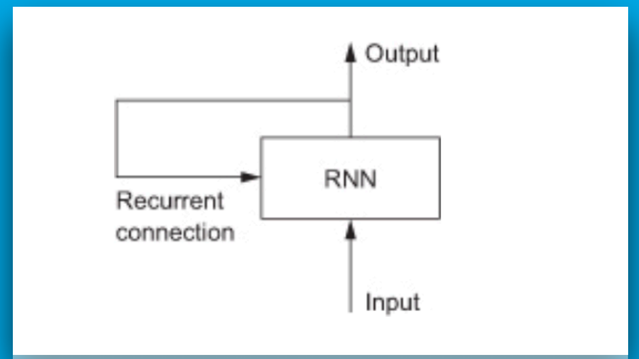
Recurrent Neural Networks

Recurrent Neural Network (Deep)
Remembering the Output/State



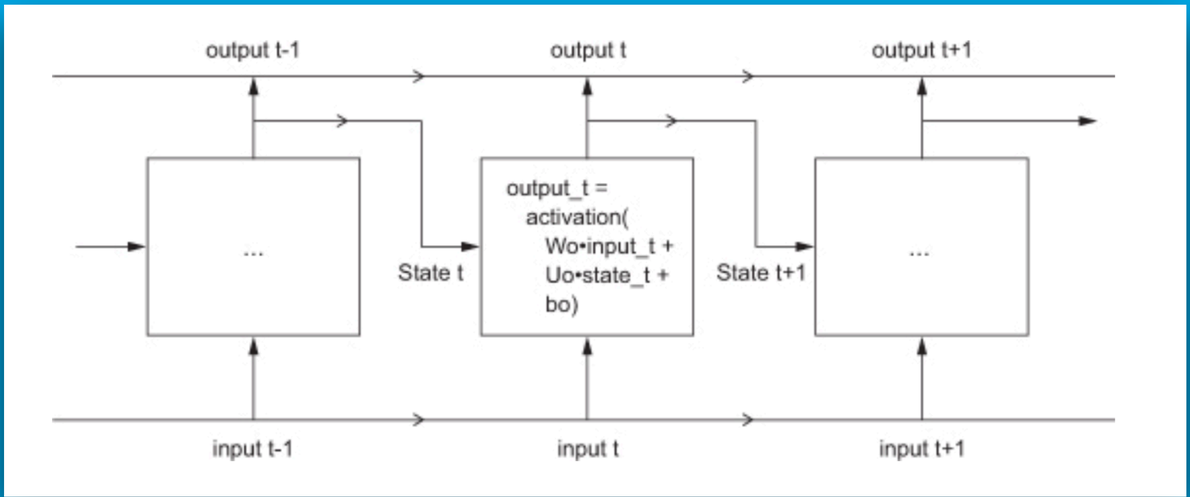
Input Layer Hidden Layers Output Layer

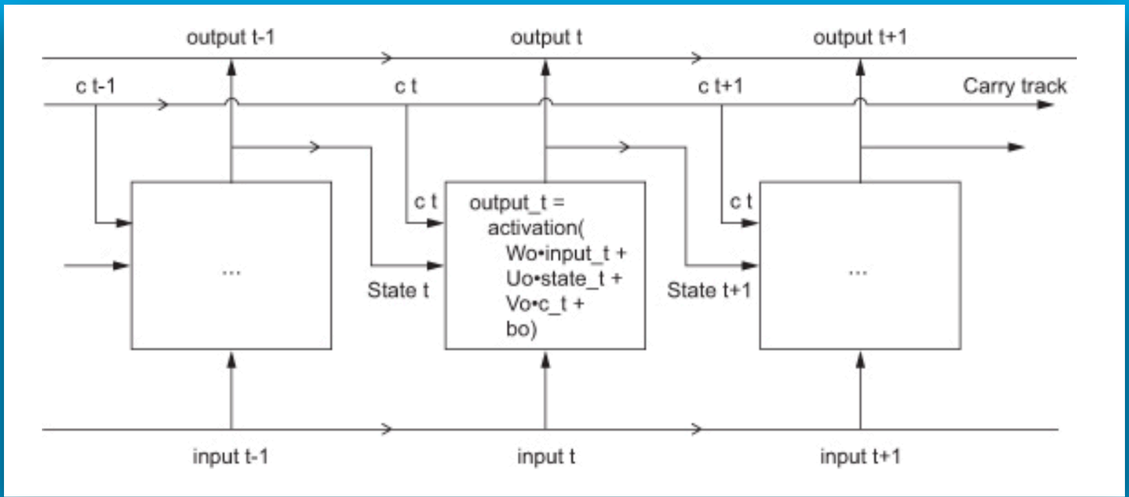
```
state_t = 0
for input_t in input_sequence:
    output_t = f(input_t, state_t)
    state_t = output_t
```

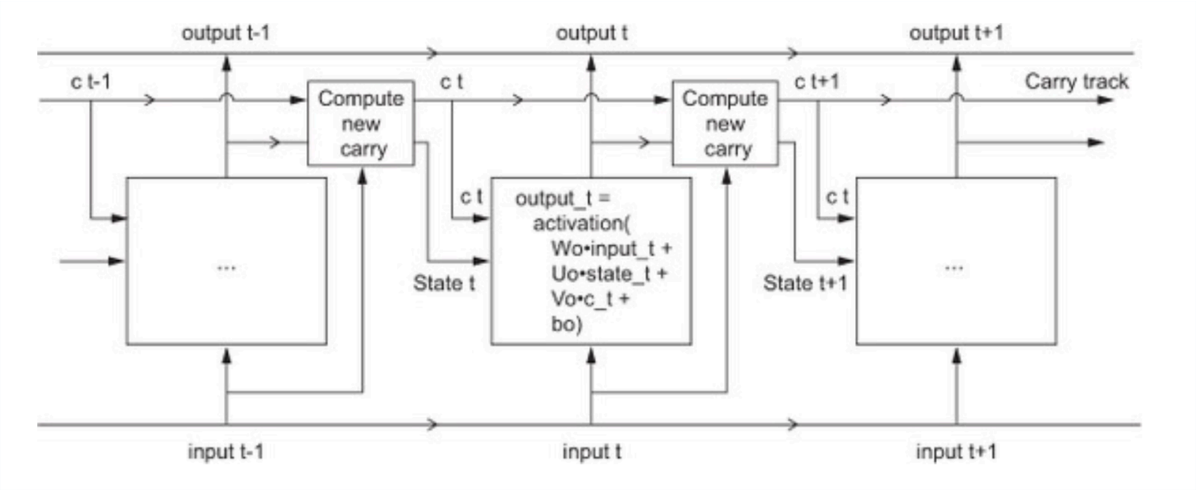


Source: Chollet (2017)

Long Short Term Memory (LSTM)
Adding a Carry Track

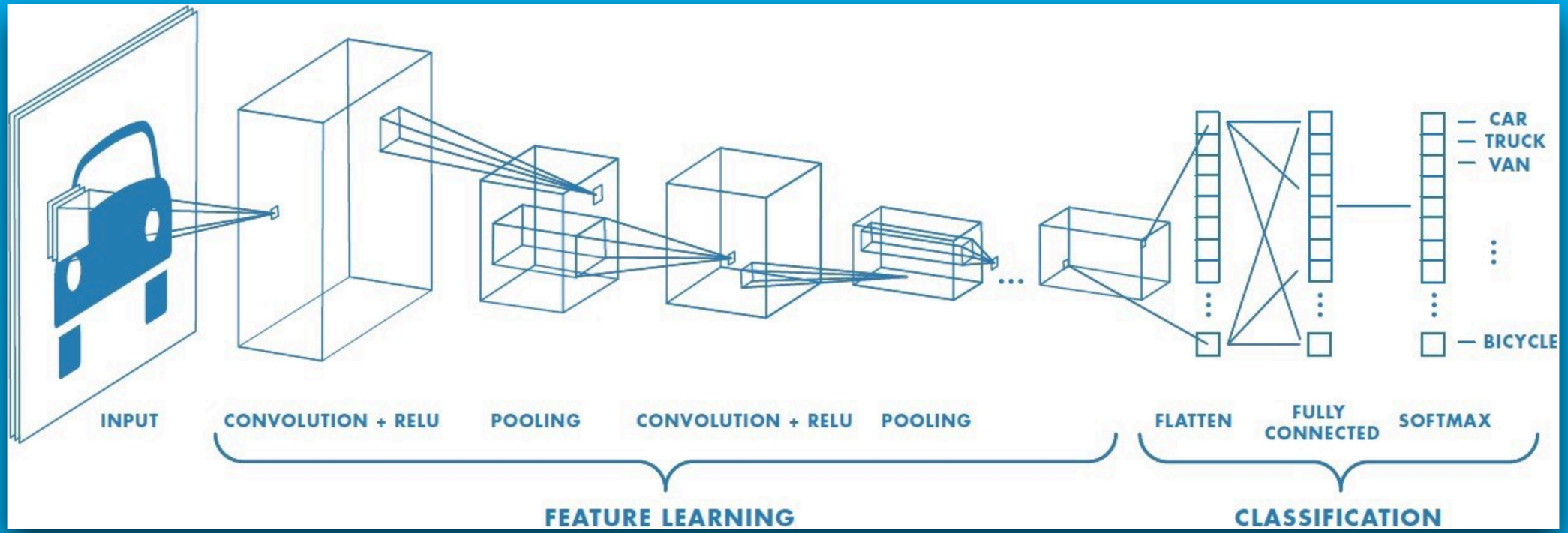






Source: Chollet (2017)

Convolutional Neural Networks


CAR
TRUCK
VAN
BICYCLE
INPUT
CONVOLUTION + RELU
POOLING
CONVOLUTION + RELU
POOLING
FLATTEN
FULLY CONNECTED
SOFTMAX
FEATURE LEARNING
CLASSIFICATION

Overview of Neural Networks

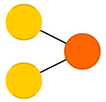
A mostly complete chart of

Neural Networks

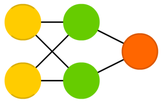
 asimovinstitute.org

- 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

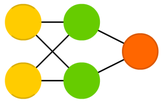
Perceptron (P)



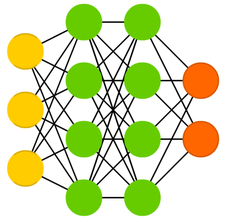
Feed Forward (FF)



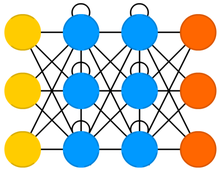
Radial Basis Network (RBF)



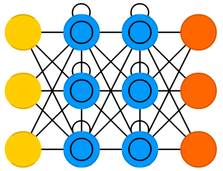
Deep Feed Forward (DFF)



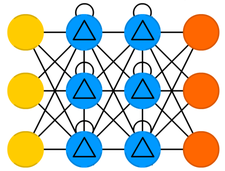
Recurrent Neural Network (RNN)



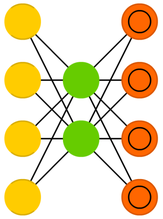
Long / Short Term Memory (LSTM)



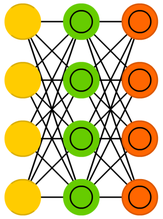
Gated Recurrent Unit (GRU)



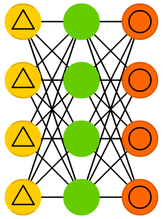
Auto Encoder (AE)



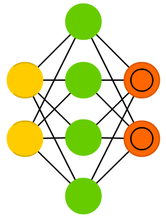
Variational AE (VAE)



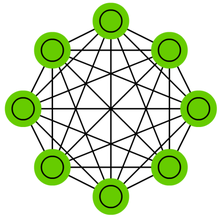
Denoising AE (DAE)



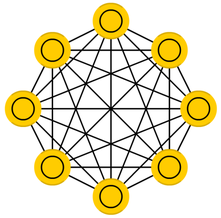
Sparse AE (SAE)



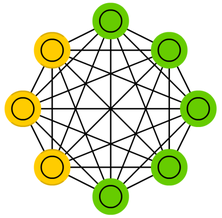
Markov Chain (MC)



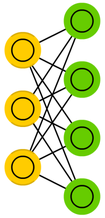
Hopfield Network (HN)



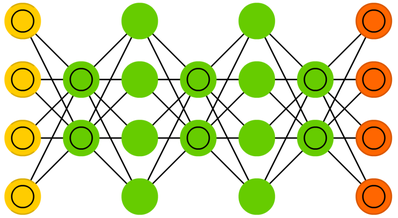
Boltzmann Machine (BM)



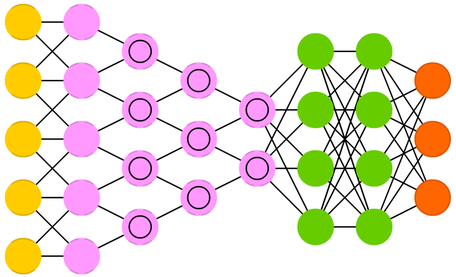
Restricted BM (RBM)



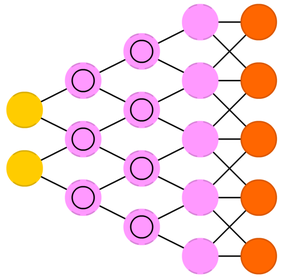
Deep Belief Network (DBN)



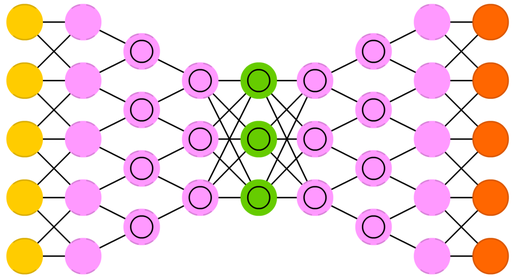
Deep Convolutional Network (DCN)



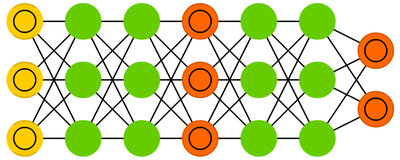
Deconvolutional Network (DN)



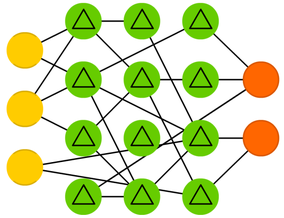
Deep Convolutional Inverse Graphics Network (DCIGN)



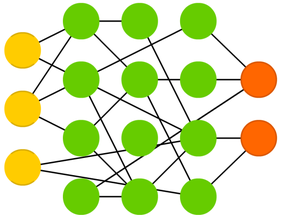
Generative Adversarial Network (GAN)



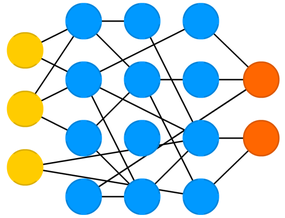
Liquid State Machine (LSM)



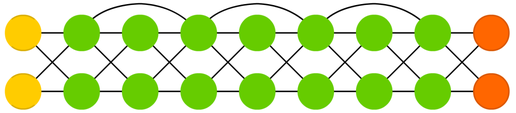
Extreme Learning Machine (ELM)



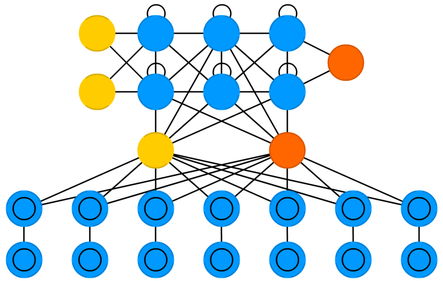
Echo State Network (ESN)



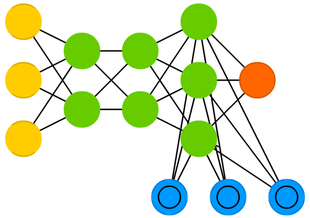
Deep Residual Network (DRN)



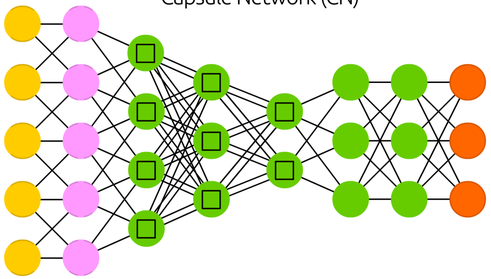
Differentiable Neural Computer (DNC)



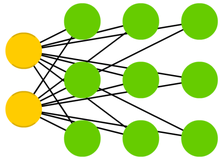
Neural Turing Machine (NTM)



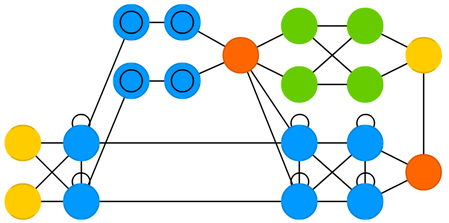
Capsule Network (CN)



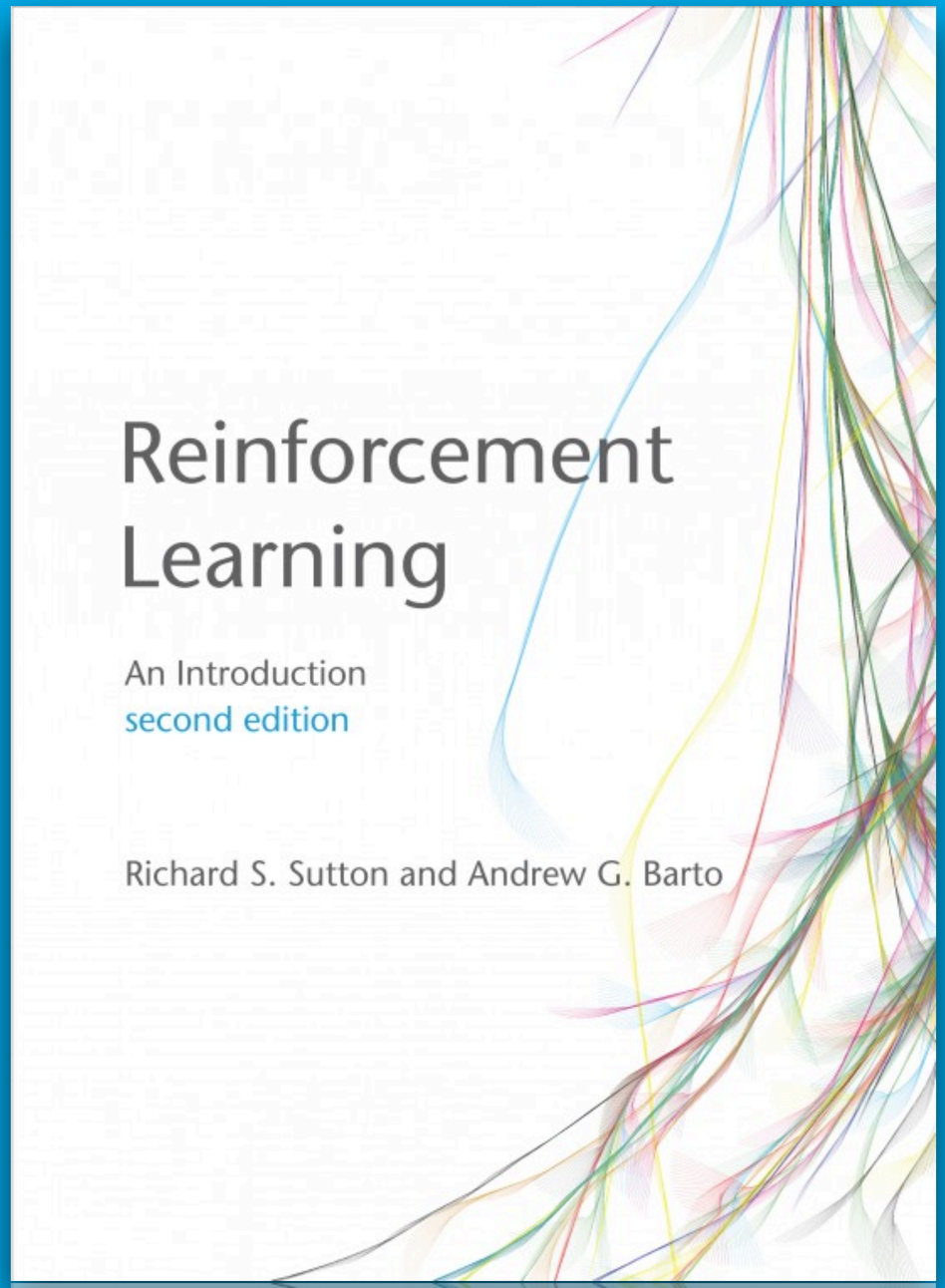
Kohonen Network (KN)



Attention Network (AN)



Reinforcement Learning —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

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.

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.

The Python Quants GmbH

Dr. Yves J. Hilpisch
+49 3212 112 9194
http://tpq.io | training@tpq.io
@dyjh