CITY
UNIVERSITY OF LONDON
EST 1894

Introduction to AI

Lecture 1: Introduction to AI

Daniel Chicharro

Attendance Register

(INM701 register code:)

School of Science and Technology

Department of Computer Science

People

- Lecturers:
  - Dr Daniel Chicharro (Daniel.Chicharro@city.ac.uk) [Theory Lectures]
  - Dr Atif Riaz (Atif.Riaz.3@city.ac.uk) [Lecturer Lab leader]
- Classroom assistants: Halil Lacevic, Felipe Saldarriaga-Echeverri

Alan Turing (1912-1954)

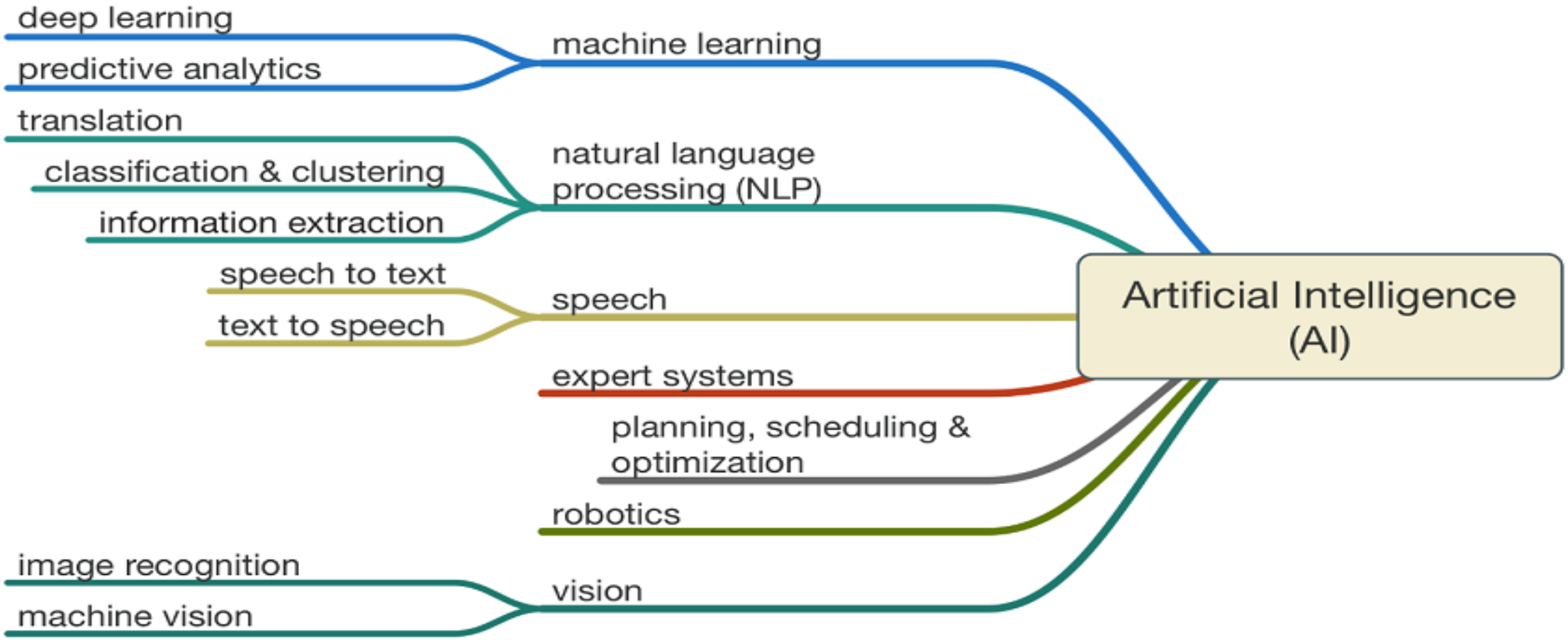
Artificial Intelligence begins with Turing.

The Turing test: first described in his article "Computing machinery and intelligence". Mind, LIX (236): 433–460, 1950.

The Turing test

- An interrogator is connected to one person and one machine through a terminal, and therefore can't see his counterparts.
- The interrogator’s task is to find out which of the two candidates is the machine, and which is human only by asking them questions.
- If the interrogator cannot make a decision within a certain time, the machine is considered to be intelligent.

Artificial Intelligence



AI paradox

✓**Hard problems for people are easy for AI**

e. g., playing chess, folding proteins,

proving theorems

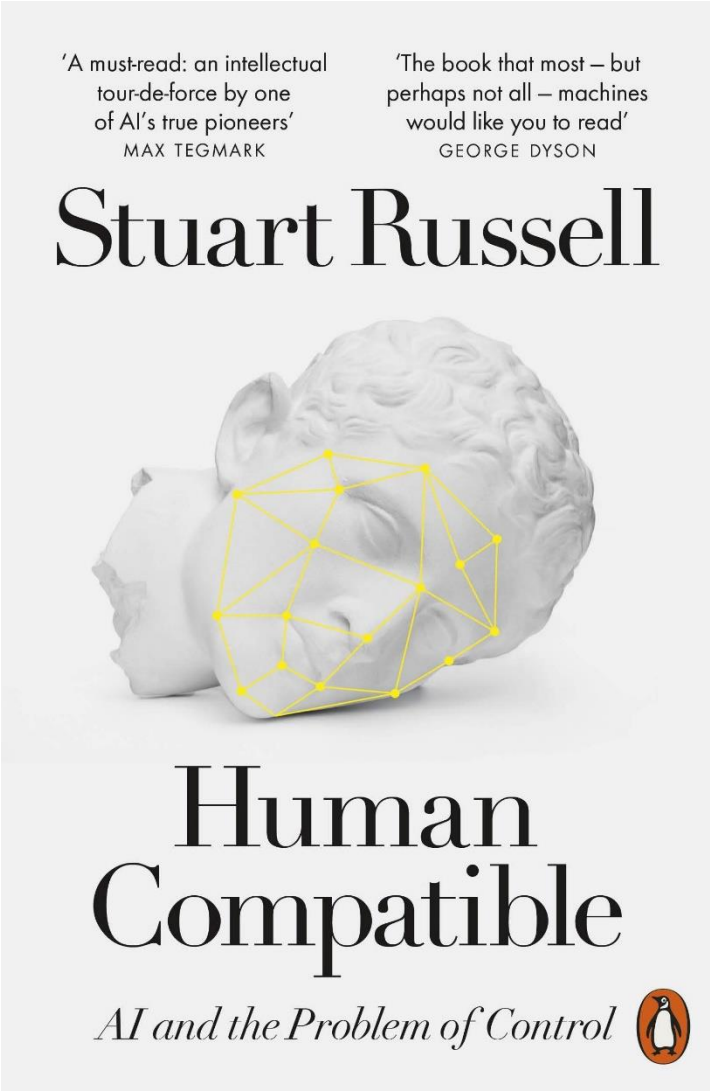
✓**Easy problems are hard for AI**

e. g., interpreting complex sensory information (visual, aural), understanding text, persistence of objects, predicting actions of people, working as a team

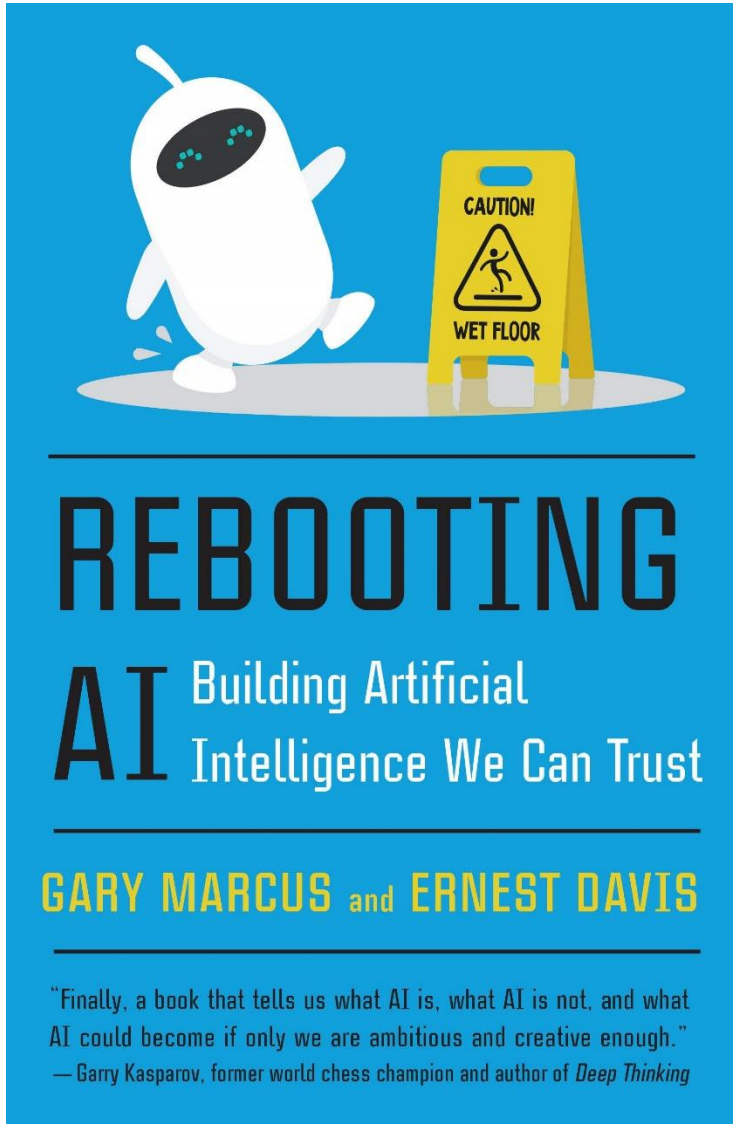
CITY
UNIVERSITY OF LONDON
EST 1894

How Close is General AI?

near?

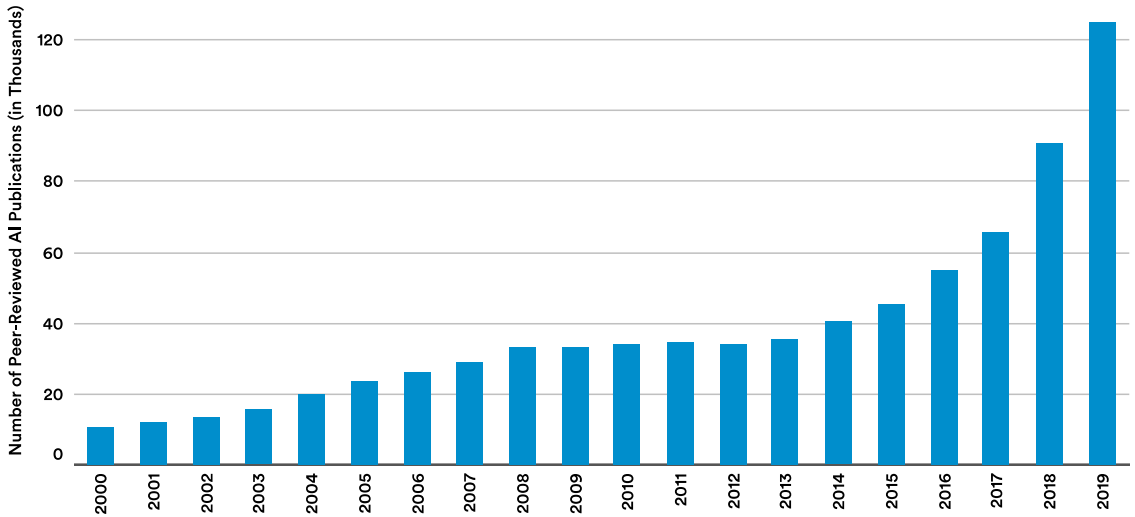


far?

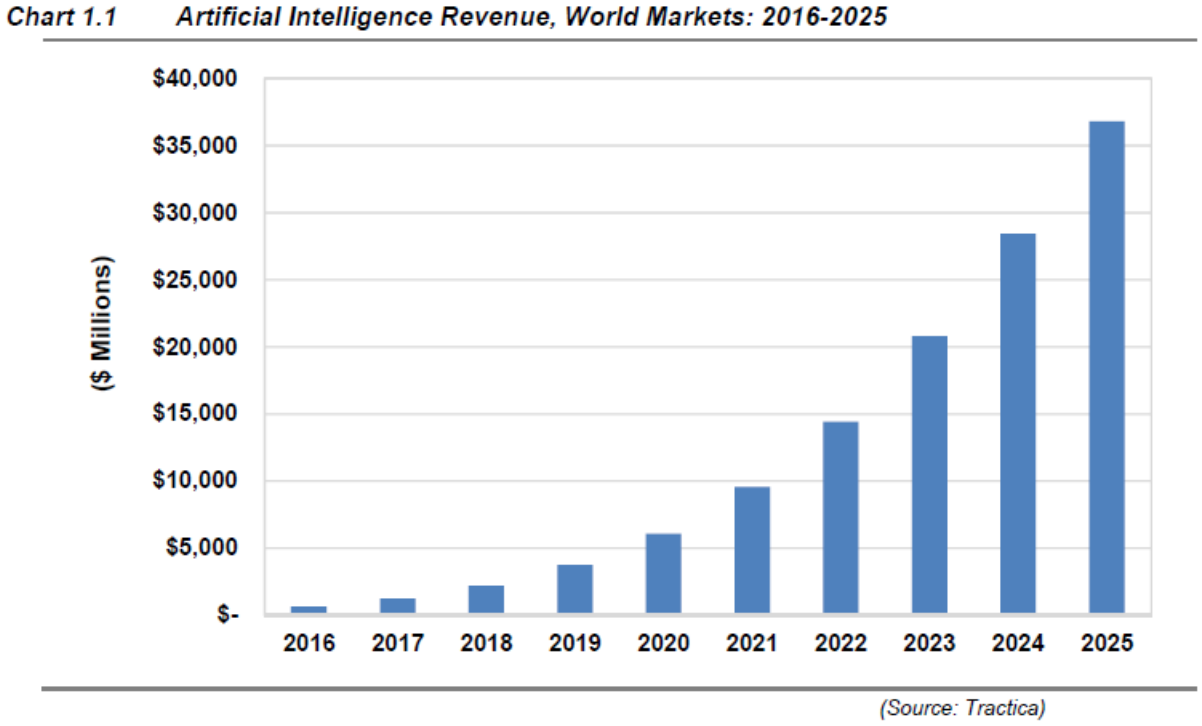


AI is a growth industry

Number of published AI papers



AI Market

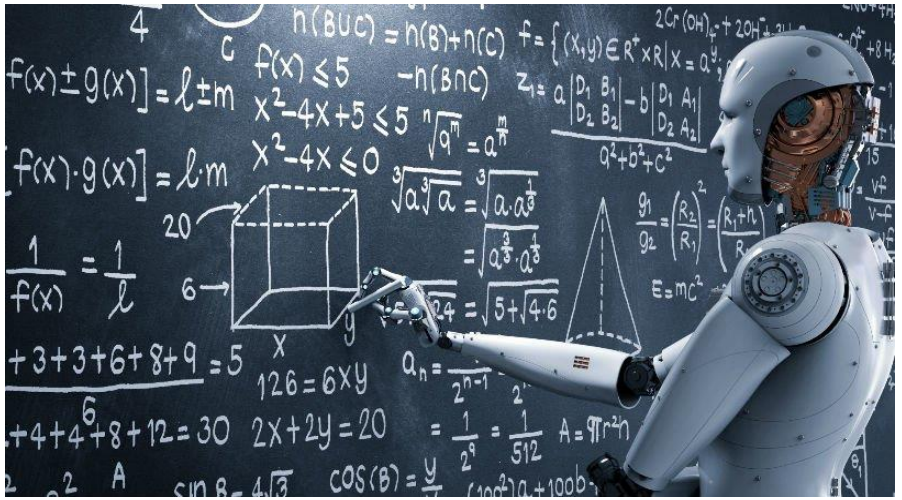


AlphaFold predicts protein structure

- “Highly accurate protein structure prediction with AlphaFold”. [34 authors omitted…] *Nature* **596**, 583–589 (2021). https://doi.org/10.1038/s41586-021-03819-2

More AI applications

- Signal and image processing, e.g. character recognition
- Data analysis
- Medicine
- Fraud detection
- Robotics, computer games
- Bioinformatics
- Translation
- Financial modelling (stock market prediction)



AI in our everyday life

1. **Self-driving and parking vehicles**

Self-driving and parking cars use deep learning,

a subset of AI, to recognize the space around

a vehicle

2. **Digital assistants**

Apple's Siri, Amazon's Alexa, etc. are digital assistants that help users to perform various tasks

(from checking their schedules and searching for something on the web to sending commands to another application).

AI in our everyday life

3. **Recommendations, Social Media and Advertising**

Machine learning targets ads at you...

4. **Automated captioning**

Deep learning models are getting better and better at voice to text, as you might have seen in earlier teaching

5. **Robots**

The Roomba 980 model vacuum (the one that cleans your floor by itself) uses AI to scan the size of a living area to best way to clean the carpet.

6. **Scheduling**

Airlines, etc… use AI techniques for optimising schedules

Module objective

- The ”**Introduction to AI"** module provides the foundational principles of Artificial Intelligence, along with a survey of techniques for manipulating data, classification, regression, and pattern recognition. There is a focus on neural networks.
- The module takes a practical approach, with emphasis on building and analysing AI, whilst theory will also be covered.
- The module focuses on relatively simple models to develop skills training, testing, and critically interpreting the models: exploratory analysis, feature selection, hyperparameters optimization, feature importance
- Good practices for data analysis and machine learning

Lecture Outline

- What is AI?
- Module Information, including development tools: Python and Libraries
- Data
- Learning from data, pattern recognition
- Neural networks, artificial neuron and the perceptron model
- Training perceptrons
- Overview of coursework

Organisation

- Lecture: Thursdays, 14:00-15:50, ELG03 (with a break in the middle)
- Classes (programming labs and exercises):
  - INM701: Thursday 16:00-16:50 ELG06/07
  - IN3062: Thursday 17:00-17:50 ELG06/07
- Drop in: Thursday 9:00-9:50 A218
- Look out for office hours for Daniel and Atif:

https://webapps.city.ac.uk/sst/vle/drop-in/

Organisation

- Lecture: Themes, models and theory

  …linking to…

- Labs: Coding and data,
  - with walk through tutorials
  - exercises
- Use the discussion board

Materials & Assessment

- Lectures (slides and all other material on Moodle)
- Labs (all material on Moodle)
- Coursework counts 100%, no exam
  - Working in groups

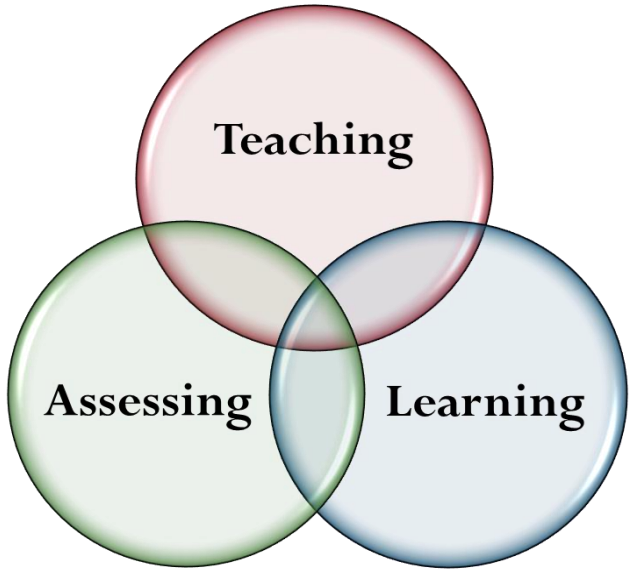
- Task: find sources of data and apply models taught in the module illustrate (and perhaps expand on) things we learn in the module.
- Submission: report, code (also in a github)

Study Hours and Assessment

- 15 Credit Module represents 150 hours of work
- Lectures and labs represent 30 hours
- Self-directed study 70-80 hours
- Coursework 40-50 hours (each)

- Tentative deadline: 5pm Sunday December 22
- Tentative presentations for INM701 in January 19

- You need to distribute your time over 3 months



Coding

Python is the main programming language for this module **(prefer Python 3.10)**

Be sure to install the essential **packages and libraries for numerical computation in Python:**

- Scikit-learn https://scikit-learn.org/stable/index.html
- Numpy https://numpy.org
- TensorFlow/Keras https://keras.io
- Matplotlib https://matplotlib.org
- Pandas https://pandas.pydata.org

CITY
UNIVERSITY OF LONDON
EST 1894

Why Python?

- **Extensive support libraries**: Python has a large number of AI libraries built in
- **Support**: Python is a fully open source
- **Productivity:** Python is an excellent option for developing scalable multi-protocol network applications
- **Flexibility**: Python is good for connecting various data structures
- **Popularity**: Python is one of the most popular languages for today

Expectations

- All students taking this module should have one of the following:
  - Minimum of two year's experience programming (in languages other than Python)
  - Some exposure to Python programming
- We'll be teaching you to work with AI libraries, building models and results in a scripted programming style
- Probably the most difficult bit is understanding your data (exploratory analysis needed, iterations of analysis)
- The emphasis is in learning the methodology of data analysis and machine learning, the use of code is only instrumental for this purpose

Coding

To install Python you can use Anaconda Distribution:

- please, go to this https://www.anaconda.com/download/
- download Python 3.10 version
- Anaconda comes with IDEs you need

Coding IDEs

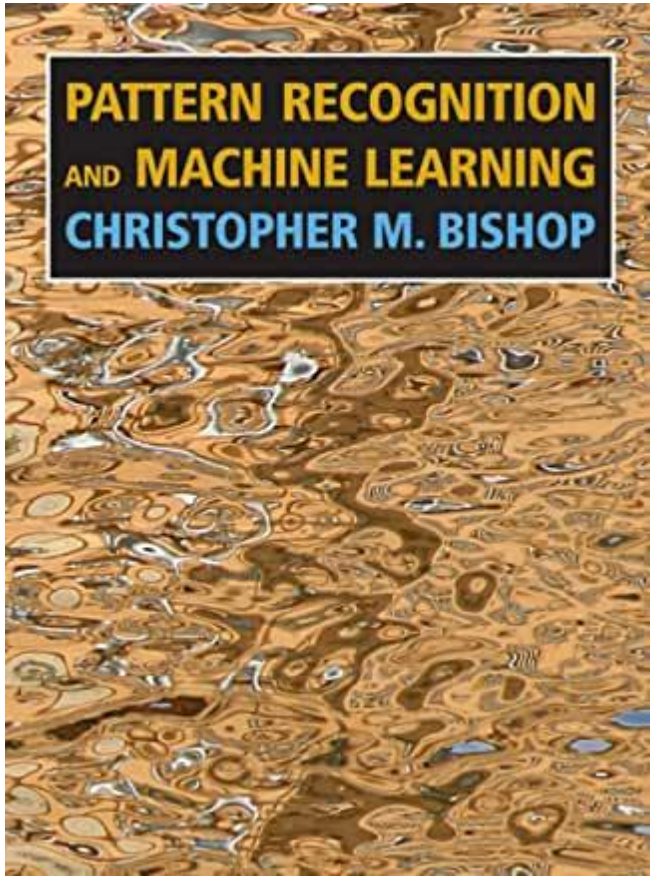
Jupyter Notebook and **Spyder/PyCharm** are used for programming in Python in this module.

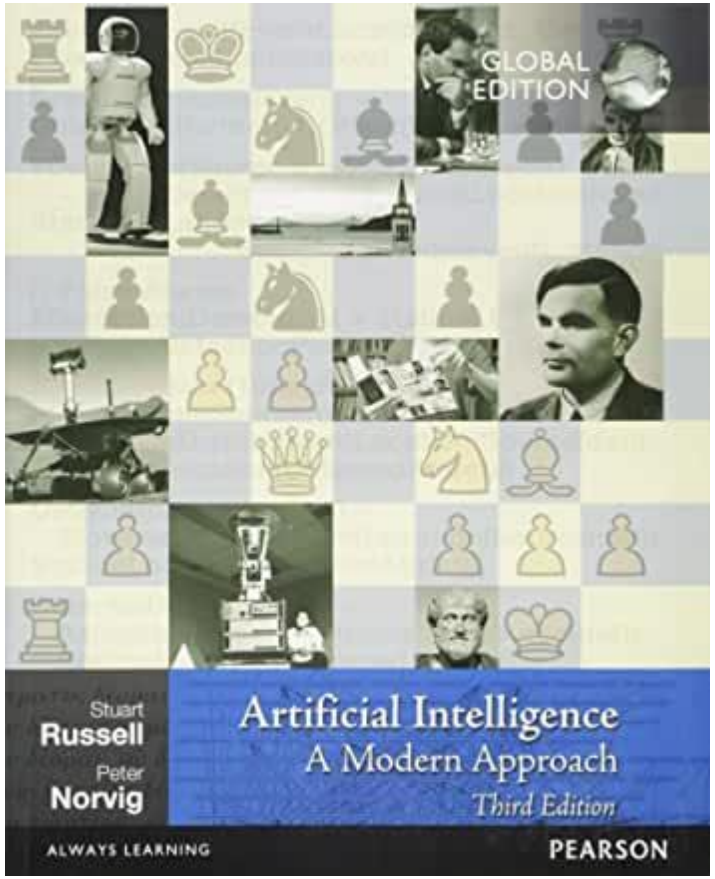
- **Jupyter Notebook** works well as a presentation or education tool, and labs will be presented using this.
- **Spyder/PyCharm** are intuitive IDEs for scientific computing. You should use this for your development work. Solutions to exercises will be python files which you can run in them.

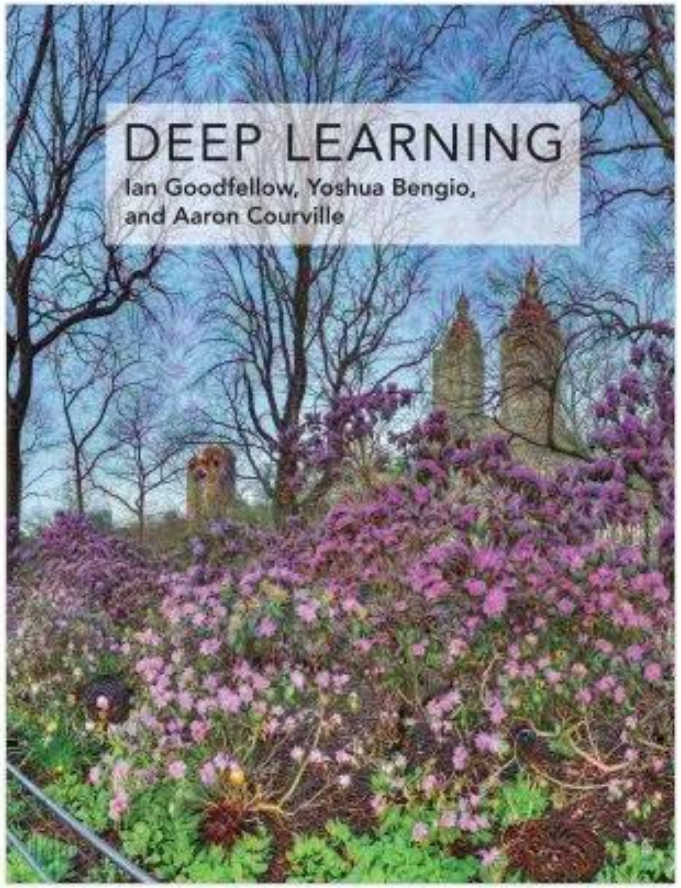
*Sample code from the internet can be used but **must be referenced** if you include it in your work.*

CITY
UNIVERSITY OF LONDON
EST 1894

Books







Books

- You can get the online version of Bishop for free here:
- https://www.microsoft.com/en-us/research/uploads/prod/2006/01/Bishop-Pattern-Recognition-and-Machine-Learning-2006.pdf
- A 4th edition of Russell and Norvig has recently come out, and the chapter structure is not the same
- Also, ”Deep Learning,” Goodfellow, Bengio and Courville. MIT Press, 2016
- https://www.deeplearningbook.org

Module Outline

- Introduction and Perceptron (L1)
- Data, Classification, Decision Trees, and Performance Measures (L2)
- Correlation, Regression and Support Vector Machines (L3)
- Unsupervised learning: PCA and clustering (L4)
- Classification: kNN, Random Forest, Naïve Bayes (L5)
- Neural Networks: Multi-layer Perceptron (L6,L7)
- Convolutional Neural Networks (L8)
- Optimisation and Genetic Algorithms (L9)
- Correlation vs causality, interpreting your models (L10)

AI paradox

✓**Hard problems for people are easy for AI**

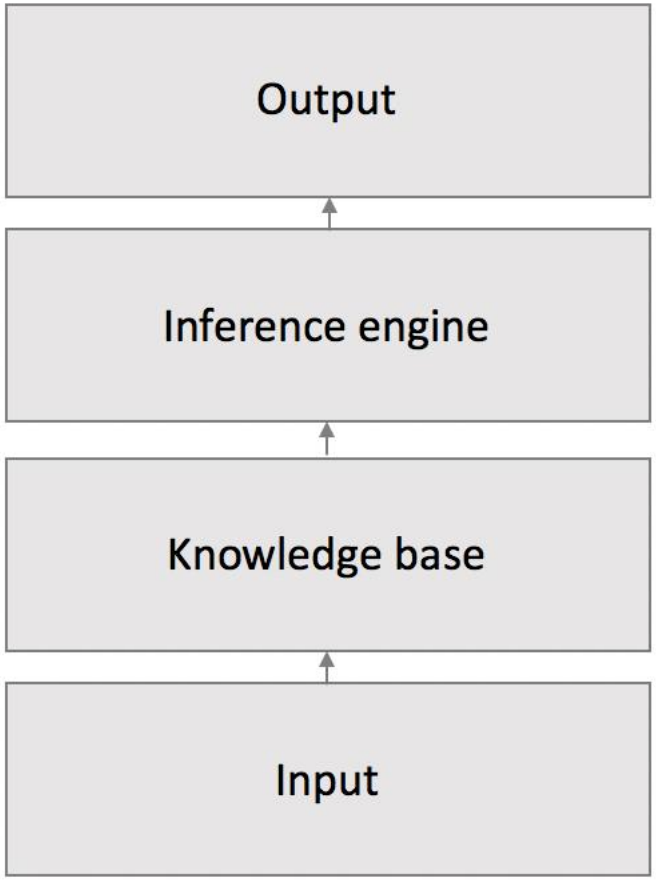
e. g., playing chess, folding proteins,

proving theorems

✓**Easy problems are hard for AI**

e. g., interpreting complex sensory information (visual, aural), understanding text, persistence of objects, predicting actions of people, working as a team

Knowledge-based AI



AI heavily depends on **representation of the given data.**

People try to formalize the rules with enough complexity to describe everything.

AI paradigm shift

- Knowledge-based systems struggled where it is hard to know what the rules are. For example, simple recognition tasks
- Motivates **machine learning**
- Use a given technique to find patterns

Machine learning approach

Collection of data: samples

Selection of the model: which pattern recognition algorithm?

Estimation of parameters: values/distributions

Inference: find answers to requests

Important part of learning is that if all done properly, the trained model will produce good predictions beyond the set of training examples. This is called **generalisation**.

Machine Learning

- Works with a **dataset**
- Learning from data
- Preprocessing the data so that it forms a suitable input to a model
- Application of a **model**, an approach to learning from data
- The model fitting is the process that learns parameter values from data
- Once these are learnt, the output is a function that maps data to a value which (hopefully) works for future input data

Problem types

- **Based on type of data:**
- **supervised learning** (learning with an external teacher)
- **unsupervised learning** (learning with no help)
- [**reinforcement learning** (learning with limited feedback, e.g. reward)]

- **Based on type of output:** regression, classification

- (**Based on type of model:** generative models, discriminative models)

Supervised learning

- The **input** is a labelled **dataset**
- The labels correspond to
  - the **class** the data point belongs to (for classification)
  - the **output value** for given inputs (for regression)
- For a given model of AI, the model is **trained** to **fit** the dataset (as best it can), it learns to approximate the function from the inputs to the outputs.
- The trained model can then be applied to new data points

Unsupervised learning

- Only **input** is given;
- The model learns to form internal representations or codes for the input data that can then be used e.g. clustering.
- Can be a pre-step for supervised learning, identifying a lower dimension in your data relevant for a task (dimensionality reduction).

CITY
UNIVERSITY OF LONDON
EST 1894

Data

1	sepal_l	sepal_w	petal_l	petal_w	species
2	5.1	3.5	1.4	0.2	Iris-setosa
3	4.9	3	1.4	0.2	Iris-setosa
4	4.7	3.2	1.3	0.2	Iris-setosa
5	4.6	3.1	1.5	0.2	Iris-setosa
6	5	3.6	1.4	0.2	Iris-setosa
7	5.4	3.9	1.7	0.4	Iris-setosa
8	4.6	3.4	1.4	0.3	Iris-setosa
9	5	3.4	1.5	0.2	Iris-setosa
10	4.4	2.9	1.4	0.2	Iris-setosa
11	4.9	3.1	1.5	0.1	Iris-setosa
12	5.4	3.7	1.5	0.2	Iris-setosa
13	4.8	3.4	1.6	0.2	Iris-setosa
14	4.8	3	1.4	0.1	Iris-setosa
15	4.3	3	1.1	0.1	Iris-setosa
16	5.8	4	1.2	0.2	Iris-setosa
17	5.7	4.4	1.5	0.4	Iris-setosa
18	5.4	3.9	1.3	0.4	Iris-setosa
19	5.1	3.5	1.4	0.3	Iris-setosa

- **Data** is the starting point
- Numbers
- Images
- Sounds
- Text

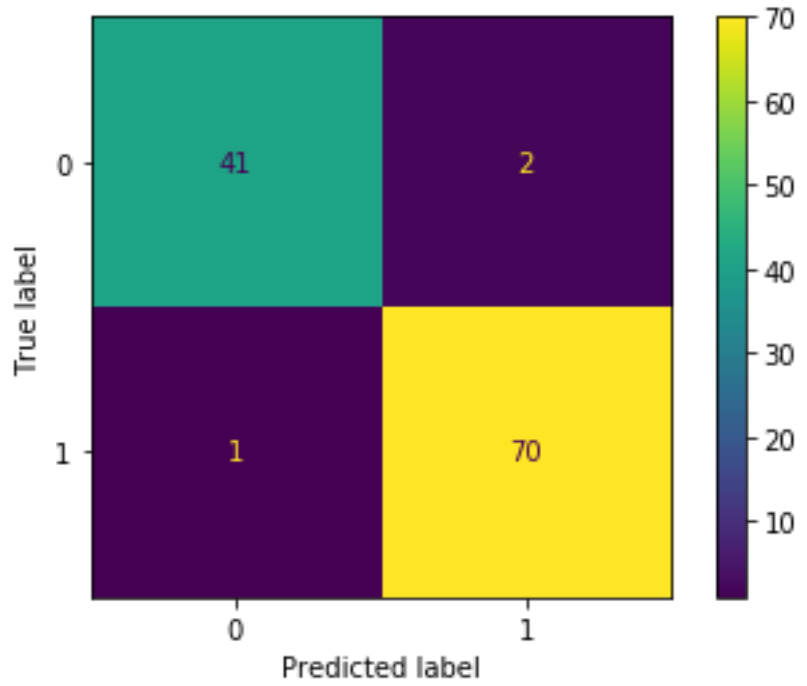
- Data is often a mess
- You need to understand it (**Exploratory Data Analysis**)
- You need to clean it (data wrangling)

Model Training and testing

- As described previously, training is the act of using data to learn a model describing patterns with data (**model fitting**)
- The goal is for the trained model to **generalise**, to apply to new unseen data successfully
- Hence the trained model needs to be tested in order to evaluate how well it really works beyond the data it has been trained with

Measure performance

- Quantify how well the trained model is performing.
- The trained model is rarely perfect because real data is messy and the patterns described might only be approximations to the real world.
- **Model comparison**: you usually do not fit a single model:
  - Same type of models with different hyperparameters
  - Different types of models



End-to-end AI project

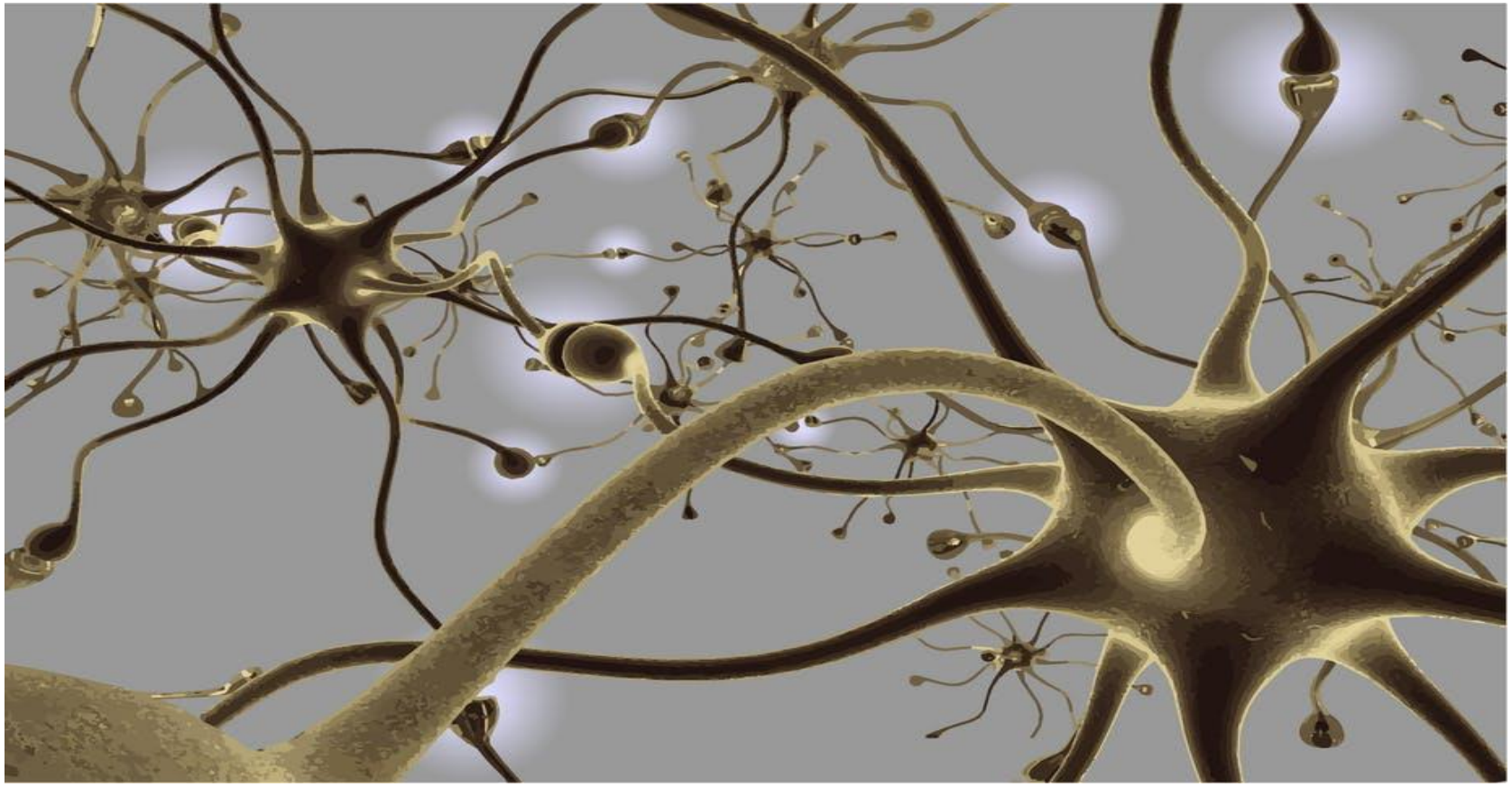
- ✓Working with real data
- ✓Getting the data
- ✓Exploring and visualizing the data to gain insights
- ✓Preparing the data for AI algorithms
- ✓Selecting and training a model
- ✓Testing your model
- ✓Comparing your models
- ✓Interpreting your results

Artificial Neural Networks

- **Artificial Neural Networks (ANNs)** are the heart of the frontier of current developments in machine learning
- (Deep) neural networks allow complicated patterns to be found from very large datasets
- The are many, many architectures. In this module we will look at a couple of the best established (simple) ones.

CITY
UNIVERSITY OF LONDON
EST 1894

Real Neural Networks

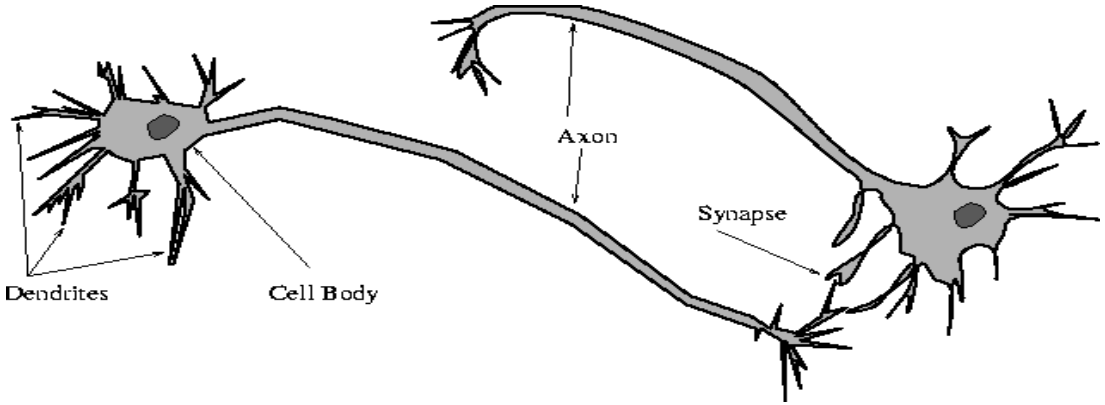


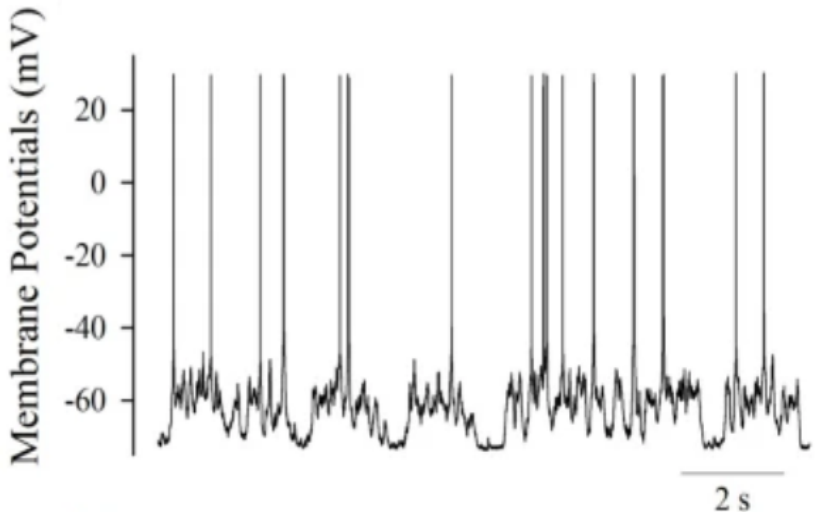
Real neural network

The computational unit in the **nervous system** is **a nerve cell** or **the neuron.**

A neuron has: **dendrites (inputs), cell body, axon (output).**

The human brain contains about 10 billion neurons



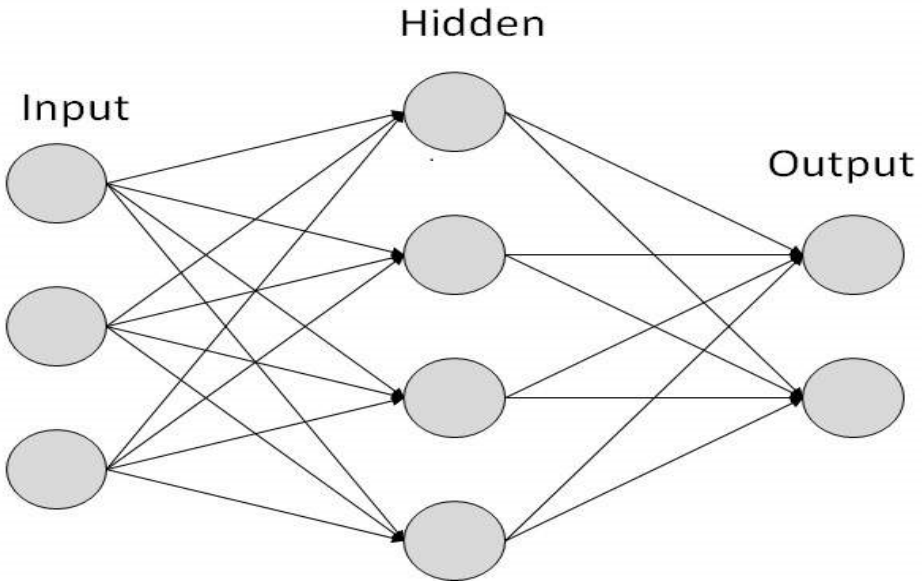


On average, **each neuron is connected to other neurons** through about 10000 synapses.

Artificial neural networks

Artificial neurons are crude approximations of the neurons found in real brains. They may be mathematical constructs or physical devices.

Artificial neural networks (ANNs) are networks of artificial neurons.



Artificial neural network

Artificial neural network characterized by:

- activation function
- Number of neurons and pattern of connections between the neurons (called its architecture)
- method of determining the weights on the connections (called its training, learning or algorithms)

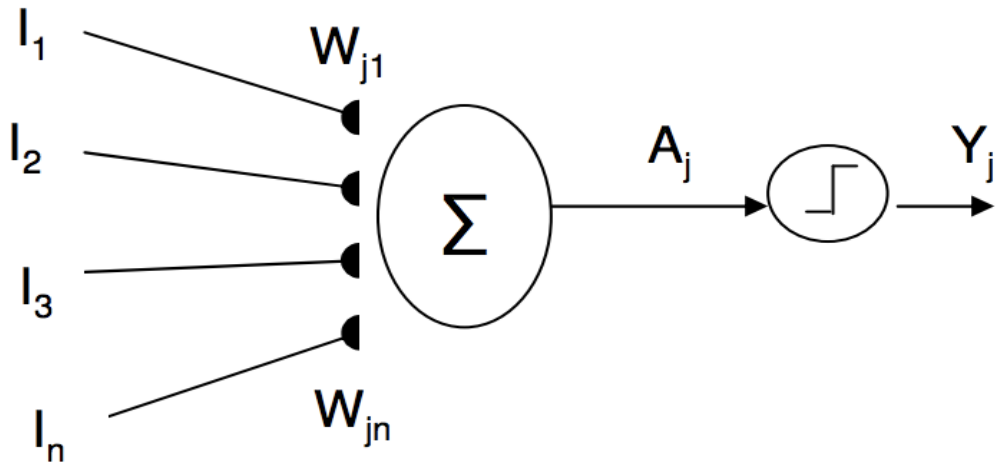
CITY
UNIVERSITY OF LONDON
EST 1894

History of neural networks

- **McCulloch and Pitts** (1943) introduced the idea of neural networks as computing machines.
- **Rosenblatt** (1958) proposed the perceptron as the first model for learning with a teacher (i.e., supervised learning).
- **Back-propagation learning algorithm** for multi-layer perceptrons was rediscovered (Werbos, 1982; Rumelhart, Hinton, 1986).
- **Convolutional Neural Networks trained with backpropagation** (LeCun, et al 1989)
- **Modern Convolutional Neural Networks** (Krizhevsky et al. 2012)
- **Gated Recurrent Units (GRU)** (Cho et al 2014)

McCulloch-Pitts neuron

Also known as a **linear threshold gate**:



1. **A synapses set** (connections) inputs activations from other neurons.
2. A processing unit performing a **weighted sums** of the inputs, producing an activation level, and then a **non-linear activation function** is applied.
3. **An output line** passes the result to other neurons.

How the neuron model works?

- For **neuron j** :
  - each **input I_i** with value x_i is multiplied by a **weight w_{ji}** (synaptic strength);
  - these weighted inputs are **summed** to give the **activation level A_j**;
  - the **activation level** is then **transformed** by an **activation function** to produce the **neuron's output Y_i.**

I_i may be **external input** or the **output of some other neuron.**

w_{ji} is known as the **weight** from unit i to unit j: if $w_{ji} > 0$ then synapse is excitatory; if $w_{ji} < 0$ then synapse is inhibitory.

The McCulloch-Pitts neuron equation

There is an equation for the **output Y_j** of a McCulloch-Pitts neuron as a function of its **inputs I** with values x_i

$$Y_j = sgn(\sum_{i=1}^{n} w_{ji} x_i - \theta),$$

where θ is the **neuron's activation threshold**. Where this step function is defined:

$$Y_j = 1, \ if \ \sum_{i=1}^{n} w_{ji} x_i \geq \theta \qquad Y_j = -1, \ if \ \sum_{i=1}^{n} w_{ji} x_i < \theta$$

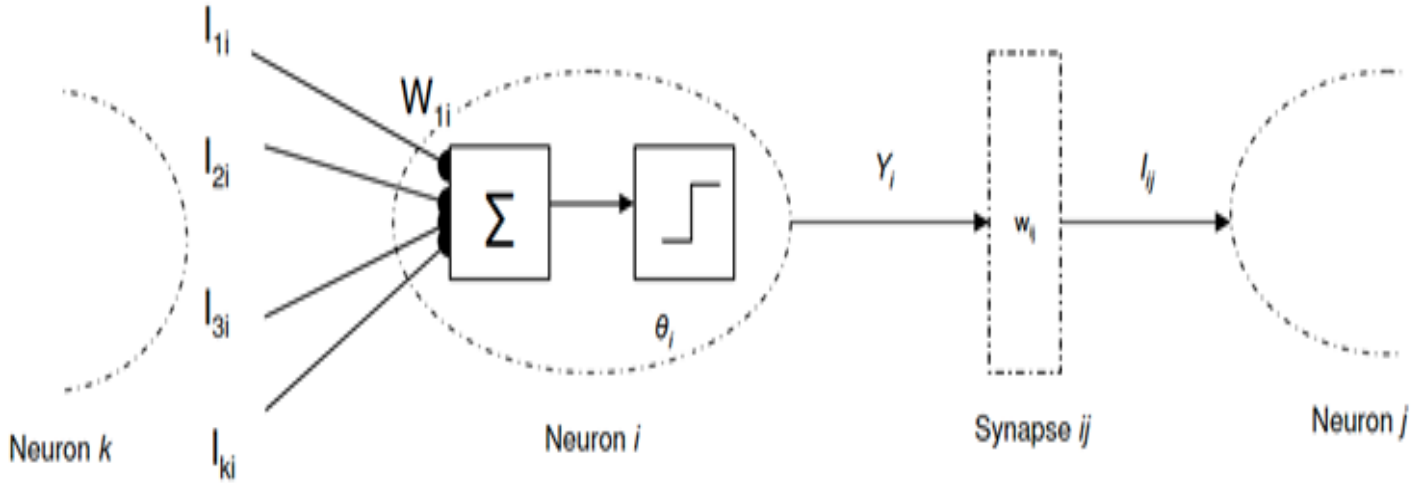
Networks of McCulloch-Pitts neurons

There are a number of neurons labelled by indices k, i, j and activation flows between synapses with strengths w_{ki} , w_{ij}:

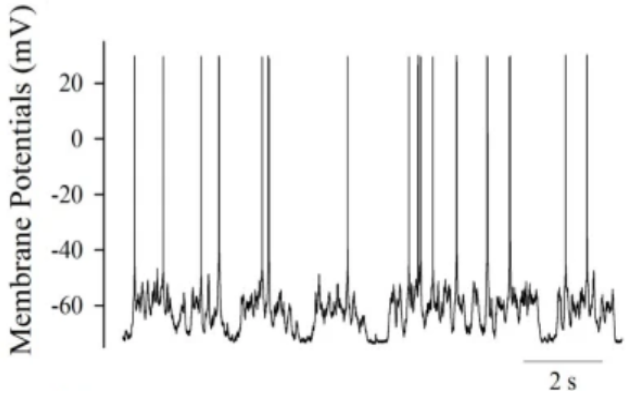
$$x_{ki} = I_{ki} = Y_k w_{ki}$$

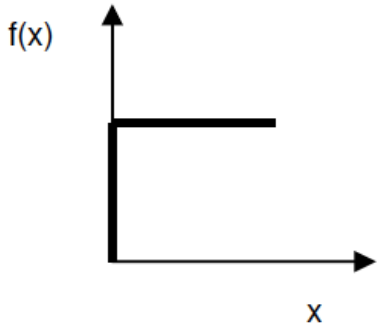
$$Y_i = sgn(\sum_{k=1}^{n} x_{ki} - \theta_i)$$

$$I_{ij} = Y_i w_{ij}$$



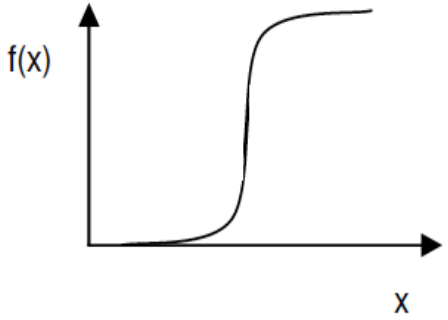
Types of the activation functions





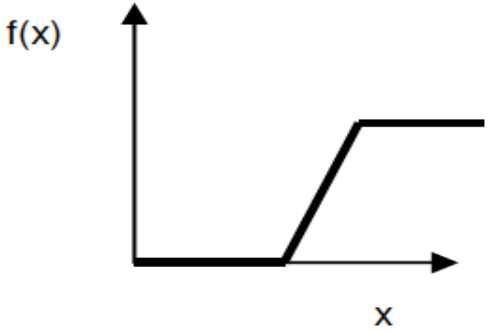
$$f(x) = \begin{cases} 1 & if \ \ x \geq 0 \\ 0 & if \ \ x < 0 \end{cases}$$

1) threshold



$$f(x) = \frac{1}{1+e^{-x}}$$

2) sigmoid



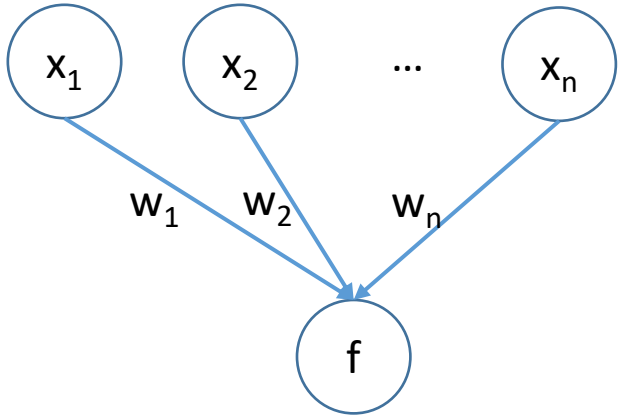
$$f(x) = \begin{cases} 1 & if \ \ x \geq 0.5 \\ x + 0.5 & if \ \ -0.5 \leq x \leq 0.5 \\ 0 & if \ \ x \leq 0.5 \end{cases}$$

3) piecewise-linear

Perceptron

The perceptron is the simplest form of a neural network in which patterns are classified as *linearly separable* (i.e. patterns on opposite sides of a hyperplane). It is closely related to the McCulloch-Pitts neuron, with the added feature that it is trainable.

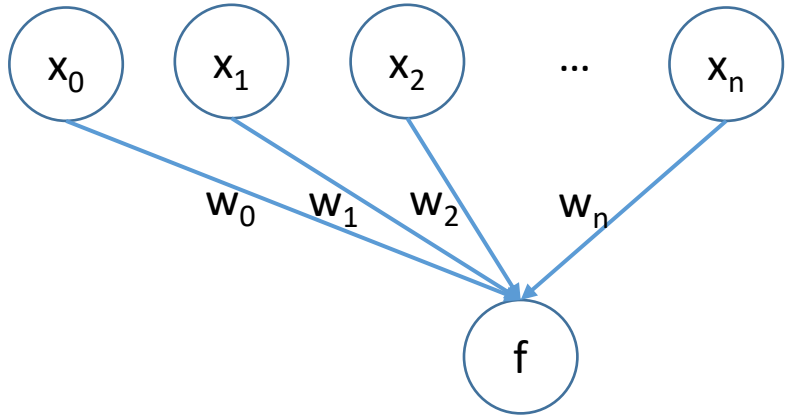
It consists of a **single neuron** with adjustable synaptic **weights** and **bias.**



Perceptron

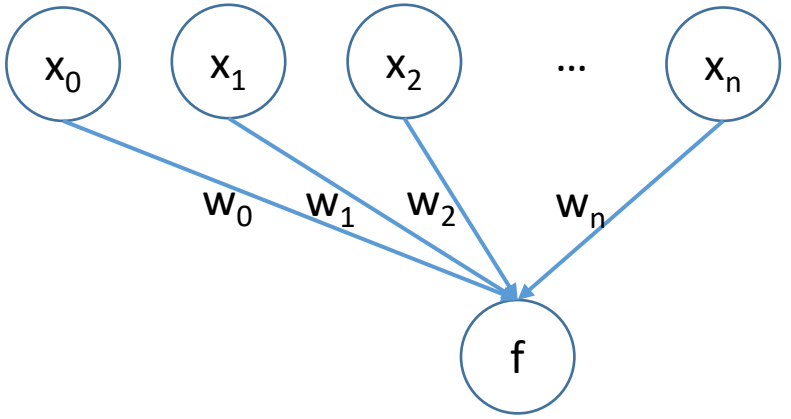
The **bias** is usually incorporated as an extra input, x_0, where the input value is always 1 (corresponding to the activation threshold). Function **f** represents the **activation function**.

The inputs themselves might be transformed by some other function ϕ



Perceptron

- Contains **n input neurons (plus bias)**, **no hidden neuron** and **1 output neuron**
- The **activation function** is a **step function**
- Distinguishes two classes of data (in n-dimensional space, by applying a n-dimensional hyperplane)



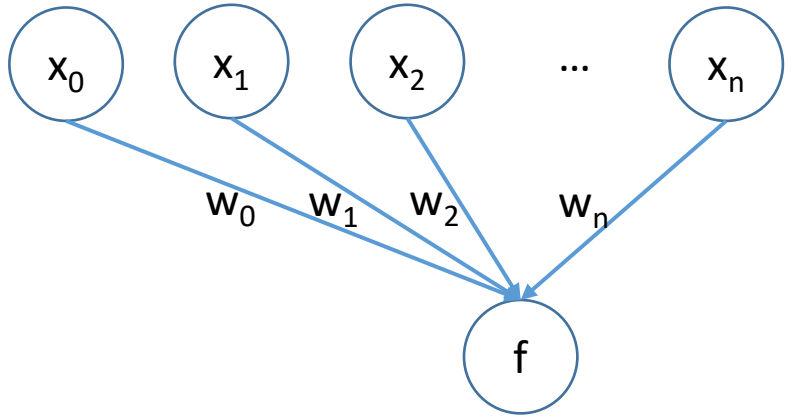
Perceptron

- Input is a vector **x**, the weights also form a vector **w**, the input might be transformed by a fixed non-linear transformation $\boldsymbol{\phi}$, and the activation function is $\boldsymbol{f}$. The output of the perceptron is then given by:

$$\boldsymbol{y(\mathbf{x}) = f(\mathbf{w}^{\mathrm{T}}\boldsymbol{\phi}(\mathbf{x}))}$$

$$w^T \phi(x) = \sum_{i=0}^{n} w_i \phi_i(x)$$

$$f(a)\begin{cases} +1 & if\ a \geq 0 \\ -1 & if\ a < 0 \end{cases}$$

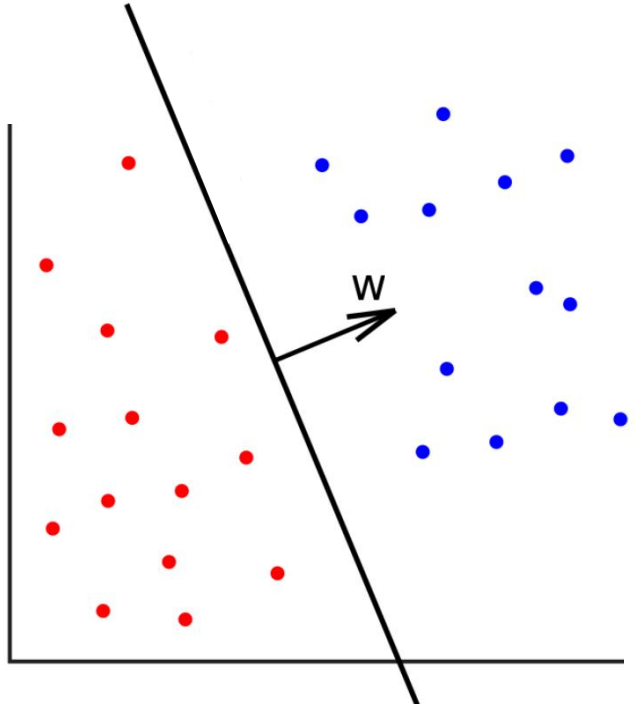


Perceptron

- Input is a vector **x**, the weights also form a vector **w**, the input might be transformed by a fixed non-linear transformation $\boldsymbol{\phi}$, and the activation function is $\boldsymbol{f}$. The output of the perceptron is then given by:

$$y(\mathbf{x}) = f(\mathbf{w}^{\mathrm{T}}\boldsymbol{\phi}(\mathbf{x}))$$

$$f(a)\begin{cases} +1 & if\ a \geq 0 \\ -1 & if\ a < 0 \end{cases}$$



Perceptron Training

- Given a dataset, how do you find a set of weights **w** that give a classifier?
- That is, how do you **train** it?

The algorithm used to adjust the free parameters (**w**) of the perceptron first appeared in a learning procedure developed by **Rosenblatt (1958, 1962)** for his perceptron brain model.

Here we will assume that $\boldsymbol{\phi}$ is the identity:

$$y(\mathbf{x}) = f(\mathbf{w}^{\mathrm{T}}\boldsymbol{\phi}(\mathbf{x}))$$

$$\mathbf{w}^{\mathrm{T}}\boldsymbol{\phi}(\mathbf{x}) = \mathbf{w}^{\mathrm{T}}\mathbf{x} = \mathbf{w}_0\mathbf{x}_0 + \mathbf{w}_1\mathbf{x}_1 + \ldots + \mathbf{w}_n\mathbf{x}_n$$

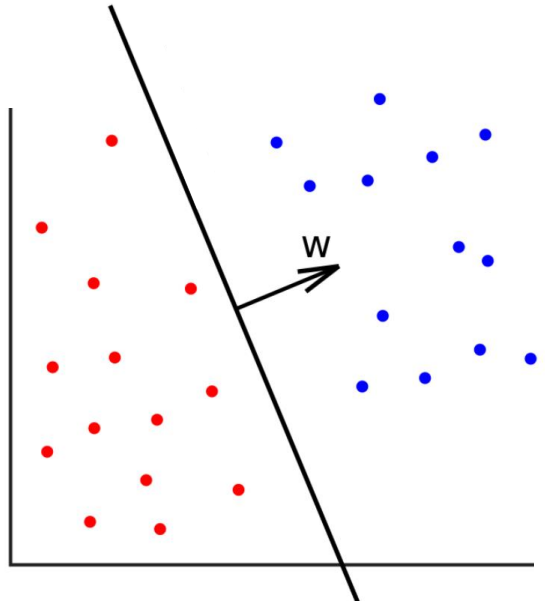
Perceptron's decision surface

As $\mathbf{w}$ and $\mathbf{x}$ have the same dimensionality, n, we can visualize the action of the perceptron in geometric terms.

The surface in the input space, that divides the input space into two classes, according to their label:

Scalar product: *project* in the direction of $\mathbf{w}$

$$\mathbf{w}^T\boldsymbol{\phi}(\mathbf{x}) = \mathbf{w}^T\mathbf{x} = \mathbf{w}_0\mathbf{x}_0 + \mathbf{w}_1\mathbf{x}_1 + \ldots + \mathbf{w}_n\mathbf{x}_n$$



This is an example of a machine's decision boundary that outputs dichotomies.

CITY
UNIVERSITY OF LONDON
EST 1894

Perceptron Training

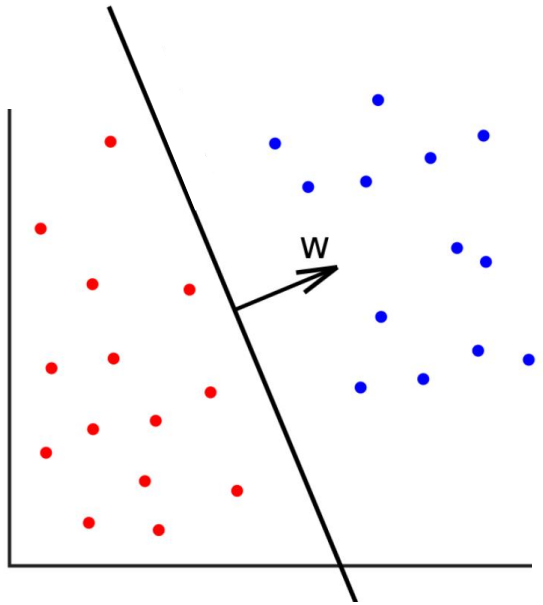
- Given a dataset, the goal is to find a perceptron that will correctly **classify** these points (and other unseen points)
- Initially, a random set of weights is chosen, and the perceptron training algorithm updates these until the perceptron is trained
- Iteratively:
  - To make an update, an element of the dataset is chosen at random
  - If this is correctly classified by the perceptron, no change is made
  - If it is not, then the weights are updated, reducing the **error** in the model

Perceptron Training

$$y(\mathbf{x}) = f(\mathbf{w}^{\mathrm{T}}\boldsymbol{\phi}(\mathbf{x}))$$
$$\mathbf{a} = \mathbf{w}^{\mathrm{T}}\boldsymbol{\phi}(\mathbf{x})$$

$$f(a)\begin{cases} +1 & if\ a \geq 0 \\ -1 & if\ a < 0 \end{cases}$$

- Perceptron error: $E(\boldsymbol{w}) = -\sum_{j=1}^{N_{errors}} \boldsymbol{w^T x_j} t_j$ where t_j is the real label of point j.
- The error is equivalent to $E(\boldsymbol{w}) = -\sum_{j=1}^{N_{errors}} a_j t_j$



Perceptron Training

- Perceptron error: $E(\boldsymbol{w}) = -\sum_{j=1}^{N_{errors}} \boldsymbol{w}^{\boldsymbol{T}} \boldsymbol{x_j} t_j$ where t_j is the real label of point j. $E(\boldsymbol{w}) = -\sum_{j=1}^{N_{errors}} (\sum_{i=0}^{n} w_i x_{ji})\, t_j$

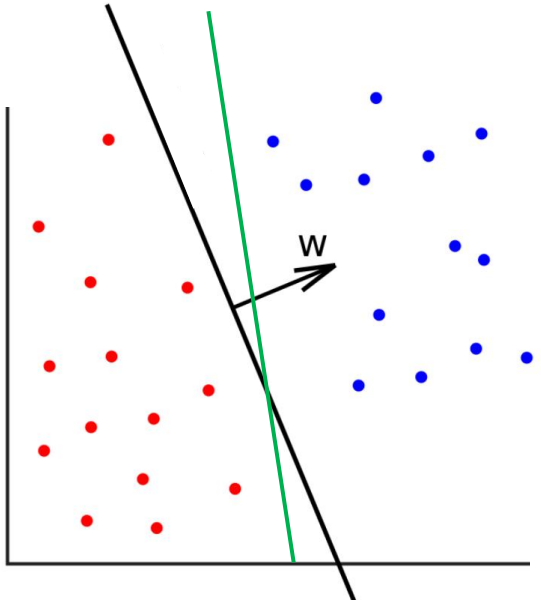
- *Stochastic gradient descent*: Weights updates: at iteration *k* with data point *m*:

$$\boldsymbol{w}^{(k+1)} = \boldsymbol{w}^{(k)} - \eta \nabla E(\boldsymbol{w}) = \boldsymbol{w}^{(k)} - \eta \boldsymbol{x}_m t_m$$

- Convergence to a solution is guaranteed if data linearly separable
- If learning rate η is too high, iterations may oscillate around solution.

Perceptron Training

- Repeated application of this rule is guaranteed to terminate, assuming that the points are **linearly separable**
- The algorithm has been trained to **fit** the data
- There is usually more than one possible answer (separating hyperplane)
- Different runs of the training algorithm might lead to



Perceptron Code

```
from sklearn.linear_model import Perceptron

# Create a Perceptron, with its training parameters
ppn = Perceptron(max_iter=40,tol=0.001,eta0=1)

# Train the model
ppn.fit(X_train,y_train)

# Make predication
y_pred = ppn.predict(X_test)
```

References

- Alan M. Turing. Computing Machinery and Intelligence, Mind, LIX (236): 433–460, https://www.csee.umbc.edu/courses/471/papers/turing.pdf
- Christopher M. Bishop. Neural Networks for Pattern Recognition. Cambridge. UK. 1995. Section 4.1.7 for perceptrons
- Stuart Russell and Peter Norvig. Artificial Intelligence, a Modern Approach (3rd edition). Pearson, 2009. Chapter 1 (history of AI), Section 18.7.1, 18.7.2
- Scikit-learn's Perceptron model: https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.Perceptron.html