Brief introduction to Neural Networks (NN)
Artificial NNs inspired from Biological NNs

Biological Neural Network (BNN)

Artificial Neural Network (ANN)
ANN as simulation of Biological NN

- NN are developed by simulating networks of human neurons

The human neuron has a cell body, a number of input wires (dendrites) and an output wire (axon)

- we can think of it as a computational unit that gets inputs, computes, and spits output to other neurons (which we will call nodes or units)

- we can think of it simply as a logistic unit that computes some $h$ as a sigmoid

Communication in BNNs is done via pulses of electricity

- we can think of I/O passing in/out numerical values

- a wide network..

Vastly oversimplified model.. → see next
All boils down to a set of optimal “weights”
Why NNs as ML algorithms?

Important to model non-linear relationships among attributes.

NNs are at the very core of Deep Learning

- versatile, powerful, scalable

Ideal to tackle large and highly complex ML tasks, such as:

- classifying billions of images (e.g., Google Images)
- powering speech recognition services (e.g. Siri, Alexa, ..)
- recommending the best videos to watch to $O(100M)$ users every day (e.g. YouTube)
- learning to beat humans in very specific tasks (e.g. from medical applications - e.g. radiology - to gaming - e.g. Go and DeepMind’s AlphaZero)
Brief history of NNs

1943 (!): first introduction by McCulloch and Pitts

• a neurophysiologist and a mathematician
• “A Logical Calculus of Ideas Immanent in Nervous Activity”,
• the first ANN architecture in human history
  ❖ a simplified computational model of how biological neurons might work together in animal brains to perform complex computations using propositional logic

1960s: the successes of ANNs stopped - the “first AI winter”

Early 1980s: revival of interest in connectionism, new architectures, better training techniques. Slow progress, though.

1990s: other (not NN) powerful ML techniques were invented, e.g. SVMs. Seemed to offer better results and stronger theoretical foundations than ANNs - the “second AI winter”

2000s-2010s: a new “AI spring”: why now? will it last?
Logical computation with neurons

The McCulloch/Pitts model for a biological neuron was later called “artificial neuron”

- it has 1 (or more) **binary (on/off) inputs** and 1 **binary** output (*NOTE: all binaries!*)
- it activates its output when more than a certain number of its inputs are active (e.g. 2 in the examples below)

They showed that even with such a simplified model it is possible to build a network of artificial neurons that computes **any basic logical proposition** you want

Imagine yourself how to combine them to form more complex logical functions
The TLU and the Perceptron

1957, Rosenblatt: “Perceptron” as one of the simplest possible ANN architectures, based on a slightly modified artificial neuron, called threshold logic unit (TLU) or linear threshold unit (LTU)

- the inputs/output are now numbers (not binary on/off) values
- each input connection is associated with a weight
- the TLU computes a weighted sum of its inputs, applies a step function to that sum, and outputs the result

TLU

Output: $h_w(x) = \text{step}(x^Tw)$

Step function: $\text{step}(z)$

Weighted sum: $z = x^Tw$

1 single TLU can act as a simple linear binary classifier

- linear combination of the inputs, and depending of the thresholds $\rightarrow$ positive/negative class (just like Logistic Regression classifier or a linear SVM !)
The TLU and the Perceptron

A Perceptron is then composed of a single layer of TLUs with each TLU connected to all the inputs

- in the input layer, an extra bias feature is generally added \(x_0 = 1\), via a "bias neuron" that just outputs 1 all the time.

- When all the neurons in a layer are connected to all the neuron in the previous layer (i.e. here the input neurons), it is called a fully-connected or dense layer

This Perceptron can classify instances simultaneously into 3 different binary classes, which makes it a linear multi-class classifier
Training a Perceptron

Input from biology and neurosciences

- 1949, Hebb; “Organization of Behavior”

- **Hebb’s rule** (Hebbian learning): when a biological neuron triggers another neuron, the connection between these two neurons **grows stronger**
  
  - Lowel: “Cells that fire together, wire together”

Largely inspired by Hebb’s rule, the Perceptron training algorithm proposed by Rosenblatt was a **weight update**

- the Perceptron is fed one training instance at a time. For each instance it makes its prediction(s). For every output neuron that produced a wrong prediction, it reinforces the connection weights from the inputs that would have contributed to the correct prediction
Multi-Layer Perceptron (MLP)

1969, Minsky/Papert, “Perceptrons”, the monograph

A picture very similar to the ones we still draw today!
Minsky/Papert:

- serious weaknesses of Perceptrons highlighted, in particular its incapability of solving relatively trivial problems, e.g. the exclusive OR (XOR) classification problem

- Well, true of any other linear classification model (e.g. Logistic Regression classifiers). But researchers had expected much more from Perceptrons! Great disappointment. People dropping off the field.

But.. it soon turned out most limitations of Perceptrons could be eliminated by stacking multiple Perceptrons!

- the resulting ANN is called a Multi-Layer Perceptron (MLP)

Note that you added an entire brand new layer to the original Perceptron architecture

(see next)
Signal flows only in one direction: Feed Forward NN (FFNN)

All layers are FC

A bias unit for each layer apart from the output one

Activation functions important to have some non-linearity between layers

Deep stack of hidden layers?

→ Deep NN (DNN)

(Deep Learning sometimes used also for shallow, non deep, NN)
For many years, researchers struggled to find a way to train MLPs.

1986, Rumelhart/Hinton/Williams introduced the back-propagation (BP) training algorithm - still used today

BP algo in one sentence:

- For each training instance, the BP algo first makes a prediction (forward pass), measures the error, then goes through each layer in reverse to measure the error contribution from each connection (backwards/reverse pass), and finally slightly tweaks the connection weights to reduce the error (in a GD step).

- the whole process is iteratively repeated - in epochs - until the network converges
https://www.asimovinstitute.org/neural-network-zoo/
https://www.asimovinstitute.org/neural-network-zoo/
https://www.asimovinstitute.org/neural-network-zoo/
That’s it, for our Intro to NNs