Machine Learning

Neural Networks

Fabio Vandin

November 27^{th}, 2023

Neural Networks

Informal definition: simplified models of the brain

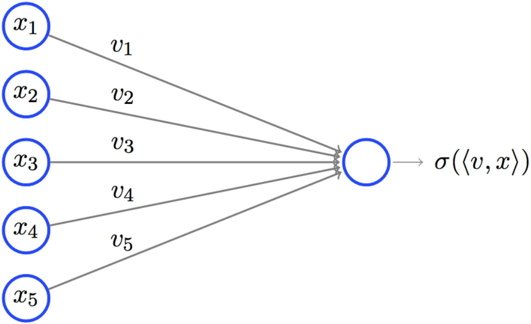
- large number of basic computing units: *neurons*
- connected in a complex network

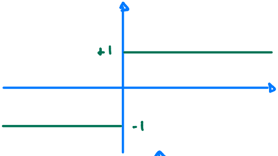
Neuron

Neuron: function $\mathbf{x} \to \sigma(\langle \mathbf{v}, \mathbf{x} \rangle)$, with $\mathbf{x} \in \mathbb{R}^d$

$\sigma : \mathbb{R} \to \mathbb{R}$ is the *activation function*

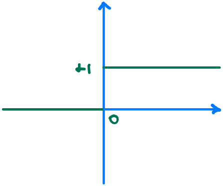
Example: $\mathbb{R}^5$

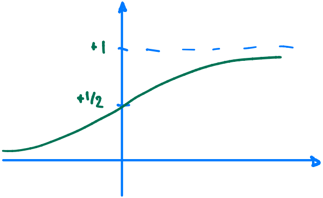




We will consider σ to be one among:

- sign function: $\sigma(a) = \mathrm{sign}(a)$
- threshold function: $\sigma(a) = \mathbb{1}[a > 0]$
- sigmoid function: $\sigma(a) = \frac{1}{1+e^{-a}}$

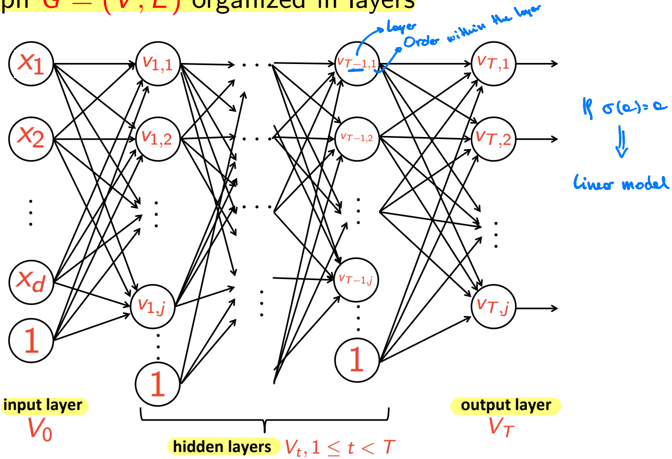




Neural Network (NN)

Obtained by connecting many neurons together.

We focus on **feedforward neural networks**, defined by a directed acyclic graph $G = (V, E)$ organized in layers

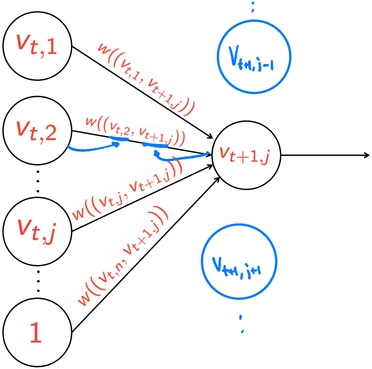


Each edge e has a *weight* $w(e)$ specified by $w : E \to \mathbb{R}$

Point of View of One Node

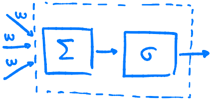
Consider node $v_{t+1,j}$, $0 \leq t < T$. Let

- $a_{t+1,j}(\mathbf{x})$: its *input* when $\mathbf{x}$ is fed to the NN
- $o_{t+1,j}(\mathbf{x})$: its *output* when $\mathbf{x}$ is fed to the NN



Then: $a_{t+1,j}(\mathbf{x}) = \sum_{r:(v_{t,r}, v_{t+1,j}) \in E} w((v_{t,r}, v_{t+1,j}))o_{t,r}(\mathbf{x})$

$$o_{t+1,j}(\mathbf{x}) = \sigma(a_{t+1,j}(\mathbf{x}))$$



Neural Network: Formalism

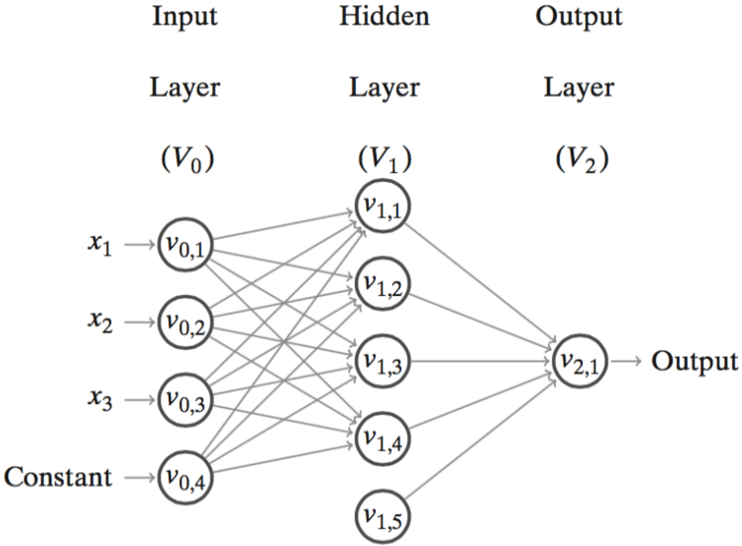
Neural network: described by directed acyclic graph $G = (V, E)$ and weight function $w : E \to \mathbb{R}$

- $V = \cup_{t=0}^{T} V_t, V_i \cap V_j = \emptyset \;\; \forall i \neq j$ → Two layers have different nodes
- $e \in E$ can only go from V_t to V_{t+1} for some t
- V_0 = *input layer*
- V_T = *output layer*
- $V_t, 0 < t < T$ = *hidden layers*
- T = *depth* $(\longleftrightarrow)$
- $|V|$ = *size* of the network
- $\max_t |V_t|$ = *width* of the network $(\updownarrow)$

Notes:

- for binary classification and regression (1 variable): output layer has 1 node
- different layers could have different activation functions (e.g., output layer)

Example



depth $= 2$, size $= 10$, width $= 5$

Exercize

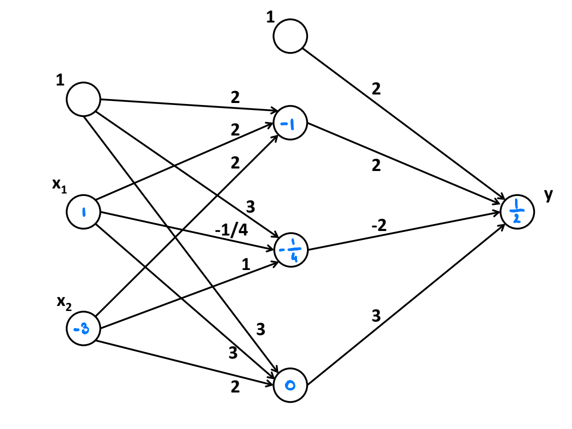
Assume that for each node the activation function $\sigma(z) : \mathbb{R} \to \mathbb{R}$ is defined as

$$\sigma(z) = \begin{cases} 1 & z \geq 1 \\ z & -1 \leq z < 1 \\ -1 & z < -1 \end{cases}$$

and consider the neural network in the next slide, compute the value of the output y when the input $\mathbf{x} \in \mathbb{R}^2$ is

$$\mathbf{x} = [1 \quad -3]^\top$$

Exercise (continue)



Hypothesis Set of a NN

Architecture of a NN: (V, E, σ)

Once we specify the architecture and w, we obtain a function:

$$h_{V,E,\sigma,w} : \mathbb{R}^{|V_0|-1} \to \mathbb{R}^{|V_T|}$$

The *hypothesis class* of a neural network is defined by *fixing* its architecture:

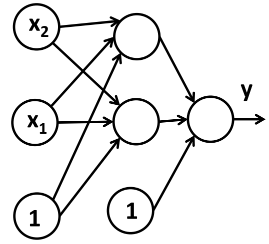
$$\mathcal{H}_{V,E,\sigma} = \{h_{V,E,\sigma,w} : w \text{ is a mapping from } E \text{ to } \mathbb{R}\}$$

Question: what type of functions can be implemented using a neural network?

Exercise (expressiveness of NNs)

Let $\mathbf{x} = [x_1, x_2] \in \{-1, 1\}^2$, and let the training data be represented by the following table:

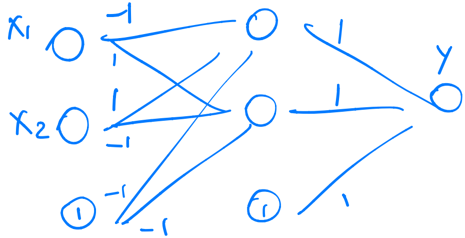
x_1	x_2	y
-1	-1	-1
-1	1	1
1	-1	1
1	1	-1



Consider the NN in the figure above, where the activation function for each hidden node and the output node is the *sign* function. Assume that the network's weights are constrained to be in $\{-1, 1\}$.

1. Find network's weights so that the training error is 0.
2. Use example above to motivate the fact that NNs are *richer* models than linear models.

1)



2) The input data is a XOR, which is not linearly separable ⇒ cannot be perfectly classified by a linear model ⇒ NN can

General construction

Let's take an arbitrary function

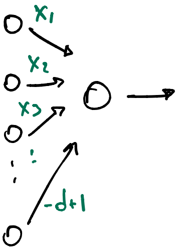
$$f : \{-1;1\}^d \to \{-1,1\}$$

<u>Goal</u>: build a NN that "computes" f: if the input is $\vec{x}$, then the prediction of the NN is $f(\vec{x})$

i) Consider $\vec{x}$ s.t. $f(\vec{x})=1$: $\forall$ such $\vec{x}$, there is a neuron in the (only) hidden layer that corresponds to $\vec{x}$. The neuron represent:

$$g_i(\vec{x}\,') = \text{sign}(\langle \underbrace{\vec{x}}_{\text{weights from the input}}, \underbrace{\vec{x}\,'}_{\text{input to the neuron}} \rangle \overbrace{-d+1}^{\text{weight from the constant}})$$

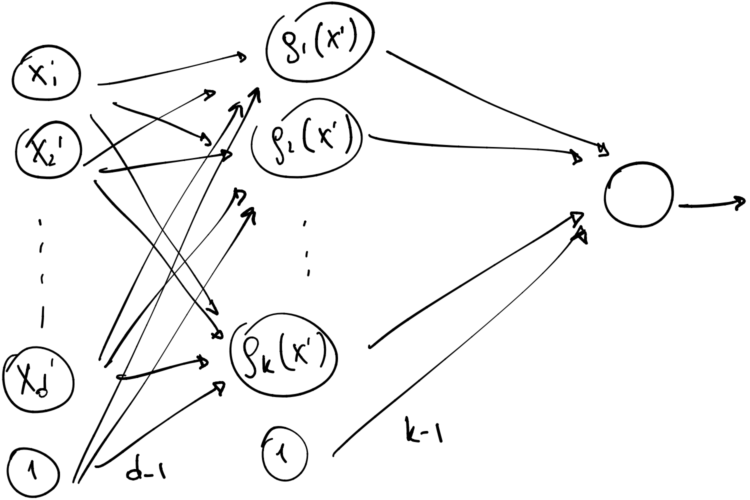
neuron $g_i(\vec{x}')$



the output node implements $f(\vec{x}') = \text{sign}\left(\sum_{i=1}^{k} g_i(\vec{x}') + k - 1\right)$

where $k = \#$ of input edges s.t. $f(\vec{x}) = 1$

The N.N. is in general:



Expressiveness of NN

Proposition

For every d, there exists a graph (V, E) of depth 2 such that $\mathcal{H}_{V,E,\text{sign}}$ contains all functions from $\{-1, 1\}^d$ to $\{-1, 1\}$

NN can implement every boolean function!

Unfortunately the graph (V, E) is very big...

Proposition

For every d, let $s(d)$ be the minimal integer such that there exists a graph (V, E) with $|V| = s(d)$ such that $\mathcal{H}_{V,E,\text{sign}}$ contains all functions from $\{-1, 1\}^d$ to $\{-1, 1\}$. Then $s(d)$ is an exponential function of d.

Note: similar result for $\sigma = \text{sigmoid}$

Proposition

For every fixed $\varepsilon > 0$ and every Lipschitz function $f : [-1,1]^d \to [-1,1]$ it is possible to construct a neural network such that for every input $\mathbf{x} \in [-1,1]^d$ the output of the neural network is in $[f(\mathbf{x}) - \varepsilon, f(\mathbf{x}) + \varepsilon]$.

Note: first result proved by Cybenko (1989) for sigmoid activation function, requires only 1 hidden layer!

NNs are **universal approximators!**

But again...

Proposition

Fix some $\varepsilon \in (0,1)$. For every d, let $s(d)$ be the minimal integer such that there exists a graph (V, E) with $|V| = s(d)$ such that $\mathcal{H}_{V,E,\sigma}$, with $\sigma =$ sigmoid, can approximate, with precision ε, every 1-Lipschitz function $f : [-1,1]^d \to [-1,1]$. Then $s(d)$ is exponential in d.