ML and Optimization Lecture 14

Neural networks:

- concepts of neurons and neural networks
- expressivity
- vector notation for neural networks

Motivations for modern (2nd wave / "deep") NNs

- more easily scalable than kernel approaches (w.r.t. size of the training data), because we can use SGD algorithms and GPGPUs.

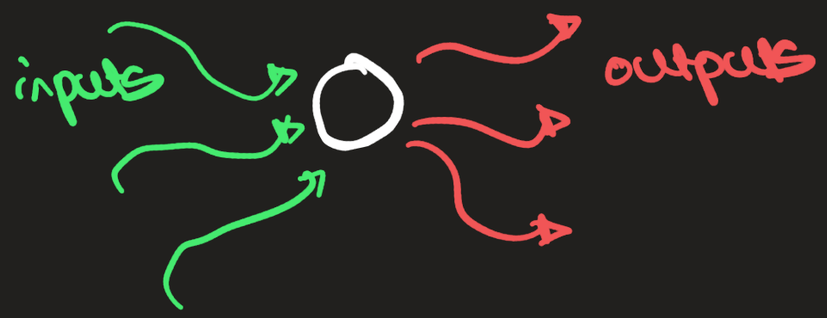
- very good inductive biases for many domains bc using well-engineered architectures (will discuss many in later lectures) led to SOTA performance on a lot of tasks in many domains.

- can take advantage of modern massive datasets

up to now our inductive bias has been towards linear separability

What is a NN?

- motivated by biological neural networks (collections of neurons)

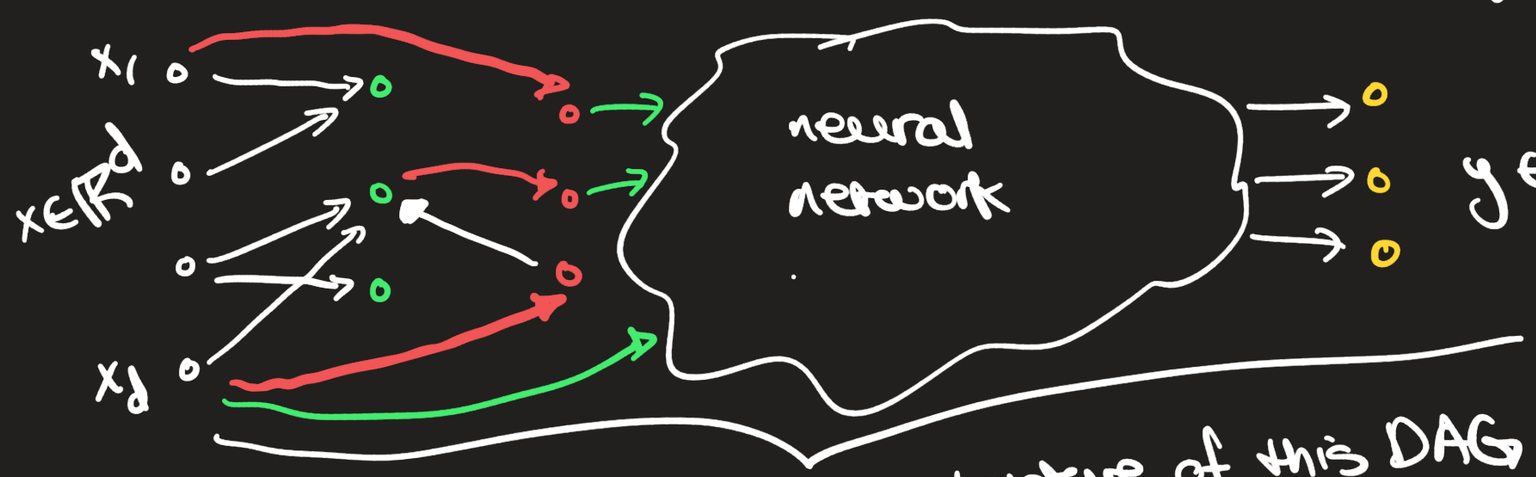


output = f(input)

↑ "activation function"

where f is nonlinear

- neurons are connected together as a DAG (directed acyclic graph)



structure of this DAG is called the "architecture of the NN"

Given a particular architecture of the NN,
learn the parameters associated w/ each neuron
to minimize a RERM problem

$$\omega^* = \underset{\omega}{\operatorname{argmin}} \frac{1}{n} \sum_{i=1}^{n} \ell(\underbrace{f(x_i;\omega)}, y_i) + \lambda R(\omega)$$

output of NN evaluated evaluated w/ x_i
as input, when its parameters are given by ω

The particular architecture determines:
- how many parameters there are
- how f depends on the parameters

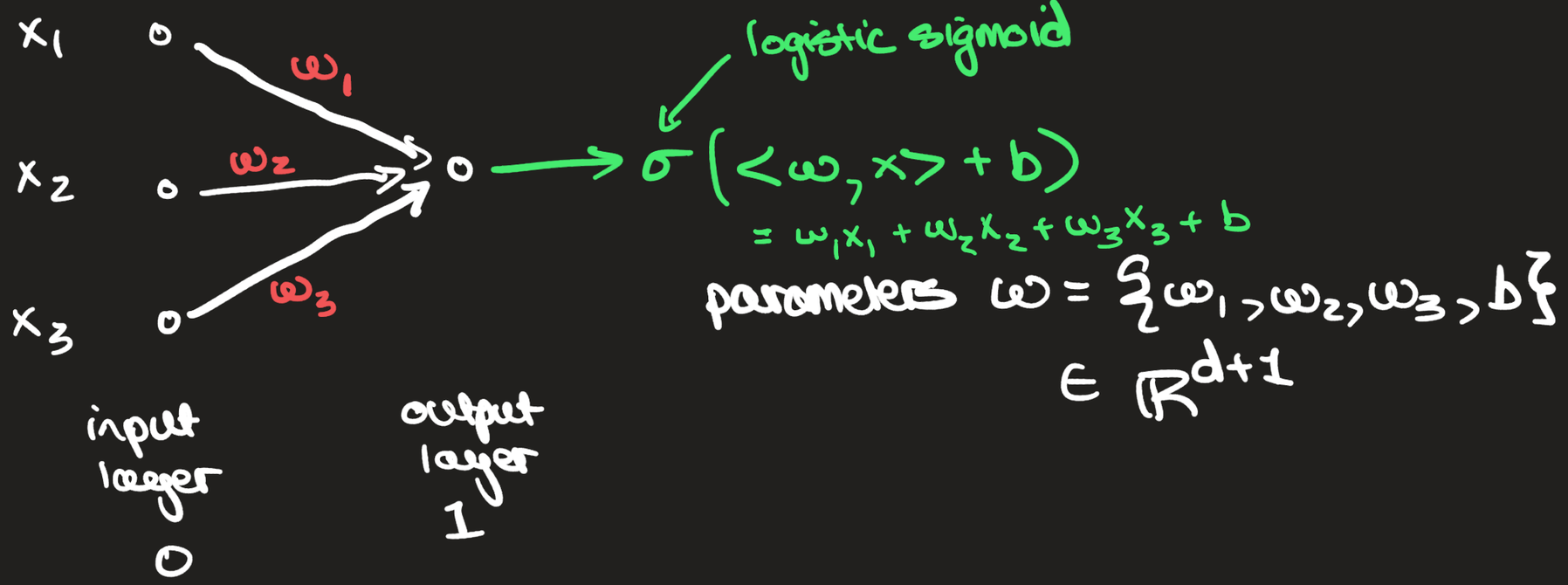
Architecture Choice 1: Fully-connected Feedforward NNs (FCFFNs or FFNs)

All neurons on layer l are connected to all neurons on layer $l-1$, and to no other neurons.

	input layer	hidden layer 1	hidden layer 2	output layer
layer number:	0	1	2	3

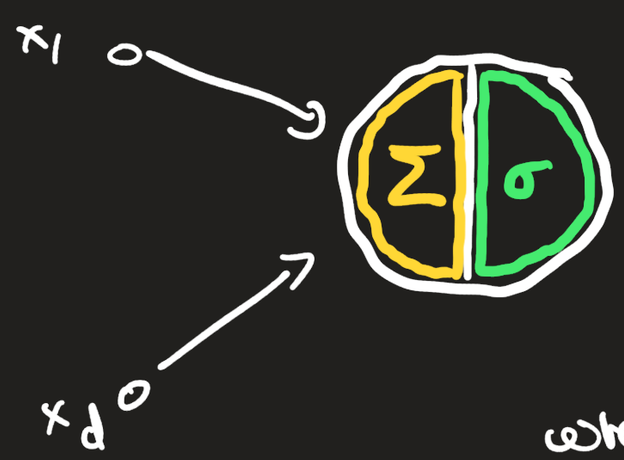
Example of a FCFFN

$x \in \mathbb{R}^3$ and $y \in [0,1]$, probability of class 1 in a two-class classification problem



logistic regression is an example of a 1-layer, 1 neuron FCFFN

Typical setup of neurons



$\longrightarrow$ output is $\sigma(\langle w, x \rangle + b)$

activation of the neuron

called the preactivation of the neuron

where σ is a nonlinear function

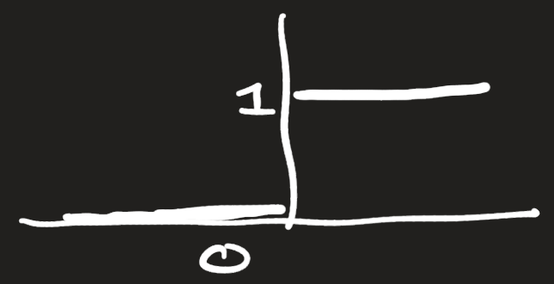
parameters of this neuron are

$$w = \{w_1, \dots, w_d, b\} \in \mathbb{R}^{d+1}$$

perceptron-type neurons

Example activation functions

$\sigma(x) = \text{step}(x)$



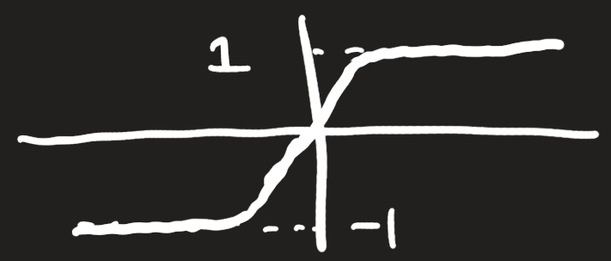
sign/thresholding

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

logistic sigmoid

$\sigma(x) = \tanh(x)$

$= \frac{e^{2x}-1}{e^{2x}+1}$



tanh

$\sigma(x) = \text{ReLU}(x)$

$= \max(0, x)$

ReLU
rectified linear unit
$\simeq$ 2015

The activation function is a hyperparameter that should be chosen in the same way you choose:

- the architecture
- the optimization algorithm
- the regularizer
- etc.

by taking the application domain & validation dataset into consideration

Need for NNs

Why do we need NNs?

The models we have seen so for take the form

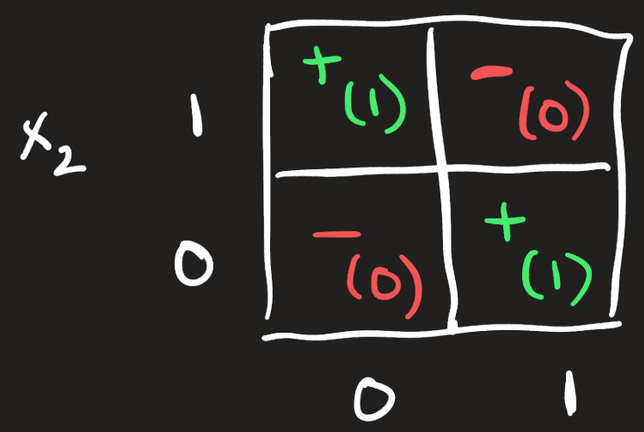
$$f(\langle w, x \rangle + b)$$

These are useful for problems w/ linear separability

Example of a problem that is <u>not</u> linearly separable

Learn the XOR function

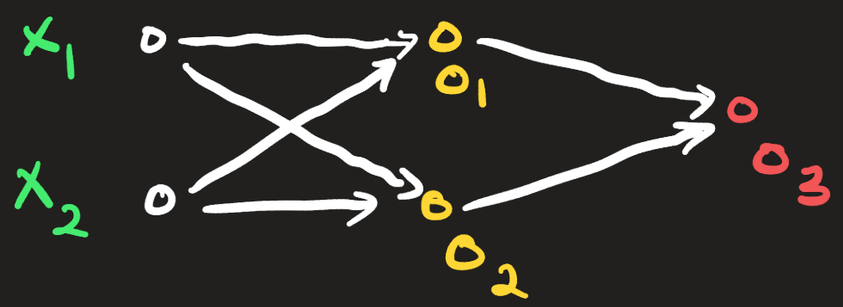
$$f(x_1, x_2) = x_1 \oplus x_2 = \begin{cases} 1 & x_1 \neq x_2 \\ 0 & x_1 = x_2 \end{cases}$$



← These classes are <u>not</u> linearly separable, so svm, logistic regression, etc will not work

NNs can learn XOR

Claim: A FCFFN with 2 hidden neurons can express XOR, using σ = threshold as activation



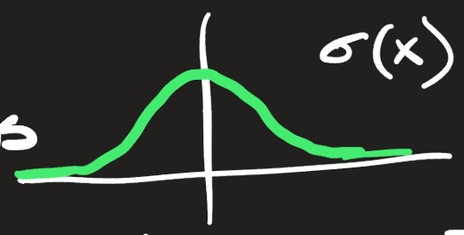
Idea: $XOR(x_1, x_2) = (x_1 \vee x_2) \wedge \overline{(x_1 \wedge x_2)}$

Idea: use o_1 to compute $x_1 \wedge x_2 = \sigma(x_1 + x_2 - 1.5)$
use o_2 to compute $x_1 \vee x_2 = \sigma(x_1 + x_2 - 0.5)$
use o_3 to compute $o_2 \wedge \overline{o_1} = \sigma(o_2 - o_1)$

Expressivity of NNs

What functions can NNs approximate?

Intuition: Let σ be a "bump"



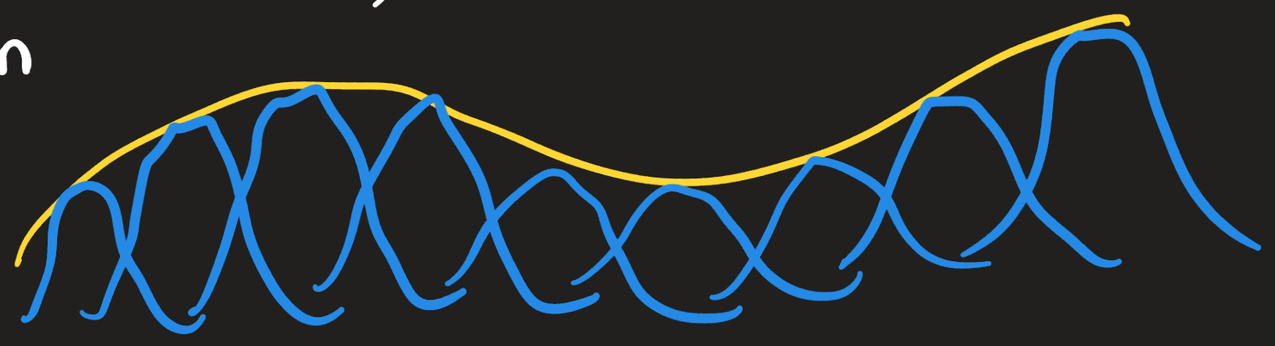
Consider the set of 1-hidden layer NNs

$$\Sigma_d = \text{span}\{\sigma(\langle \omega, x\rangle + b) \text{ for } \omega \in \mathbb{R}^d, b \in \mathbb{R}\}$$

$$= \left\{ f \mid f(x) = \sum_{i=1}^{k} a_i \sigma(\langle \omega_i, x\rangle + b_i) \text{ for some } k \in \mathbb{N}, a_i \in \mathbb{R}, \omega_i \in \mathbb{R}^d, b_i \in \mathbb{R} \right\}$$

= the set of superpositions of scaled (a_i), skewed (ω_i), translated (b_i) bumps

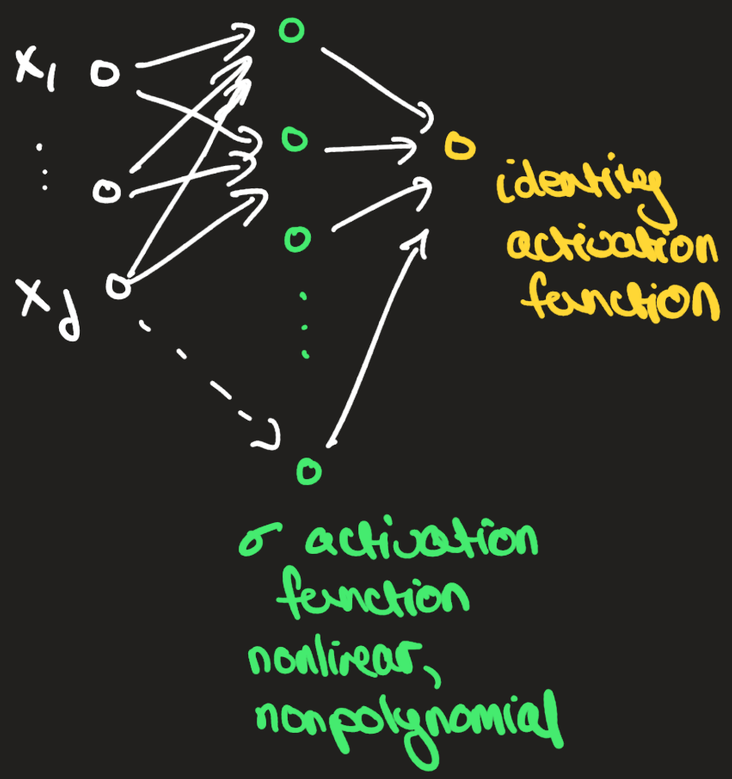
Given any function f, we can approximate it with such a superposition



Expressivity of NNs

Universal approximation property

Thm 1 of (Leshno et al 1993)



Let σ be bounded on every compact set of $\mathbb{R}^d$.

Set

$$\Sigma_d = \text{span}\{\sigma(\langle \omega, x\rangle + b) : \omega \in \mathbb{R}^d, b \in \mathbb{R}\}$$

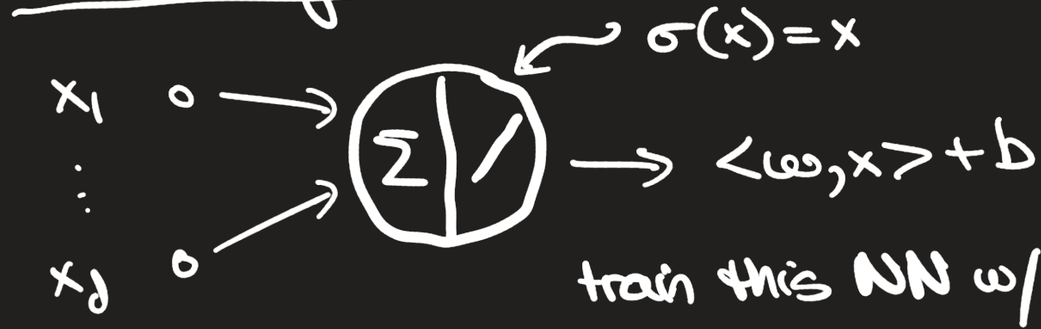
then Σ_d is dense in $C(\mathbb{R}^d)$ if and only if σ in not a polynomial.

That is, if f is a continuous function on $\mathbb{R}^d$, then for each $\varepsilon > 0$, there exists on $f_\varepsilon \in \Sigma_d$ s.t.

$$\|f - f_\varepsilon\|_\infty = \max_{x \in R^d} |f(x) - f_\varepsilon(x)| < \varepsilon$$

All models see until now are NNs

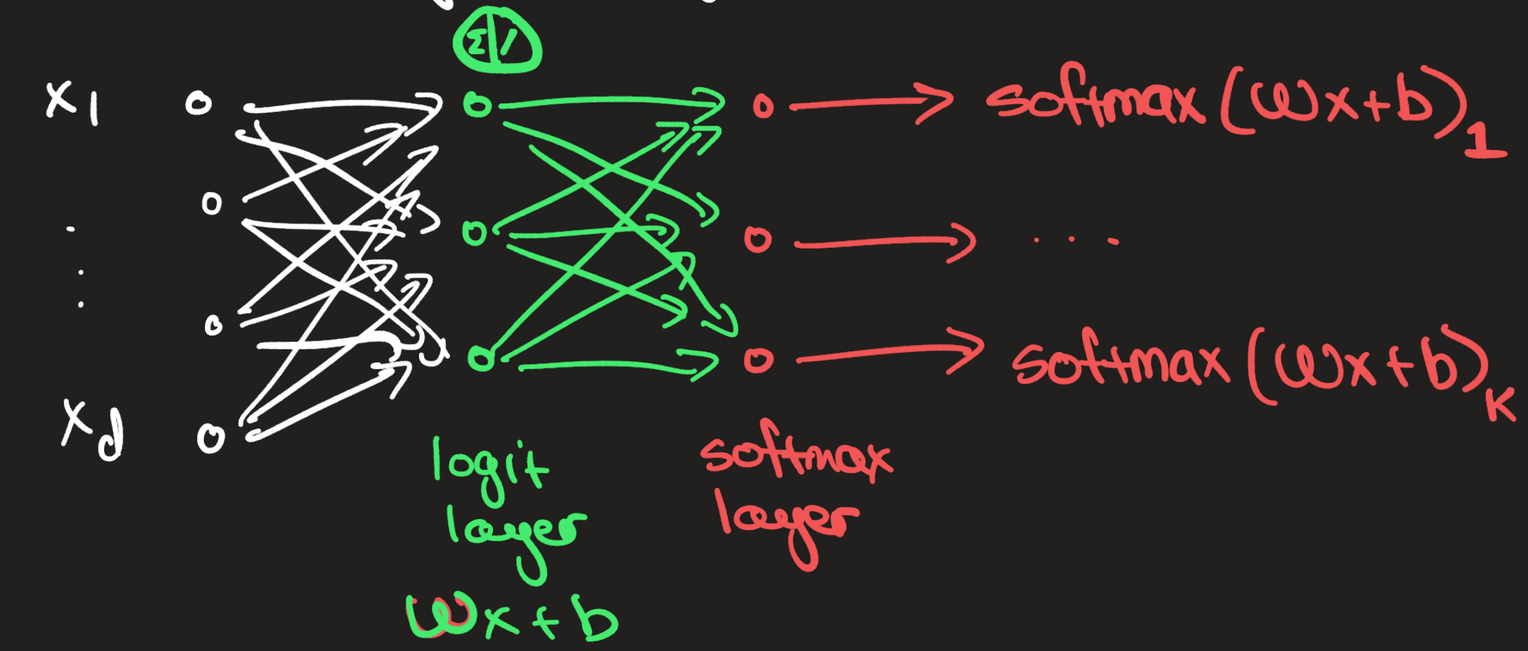
Linear regression



train this NN w/ $\ell(y, \hat{y}) = (y - \hat{y})^2$

then we get a linear regression model.

Multiclass Logistic Regression

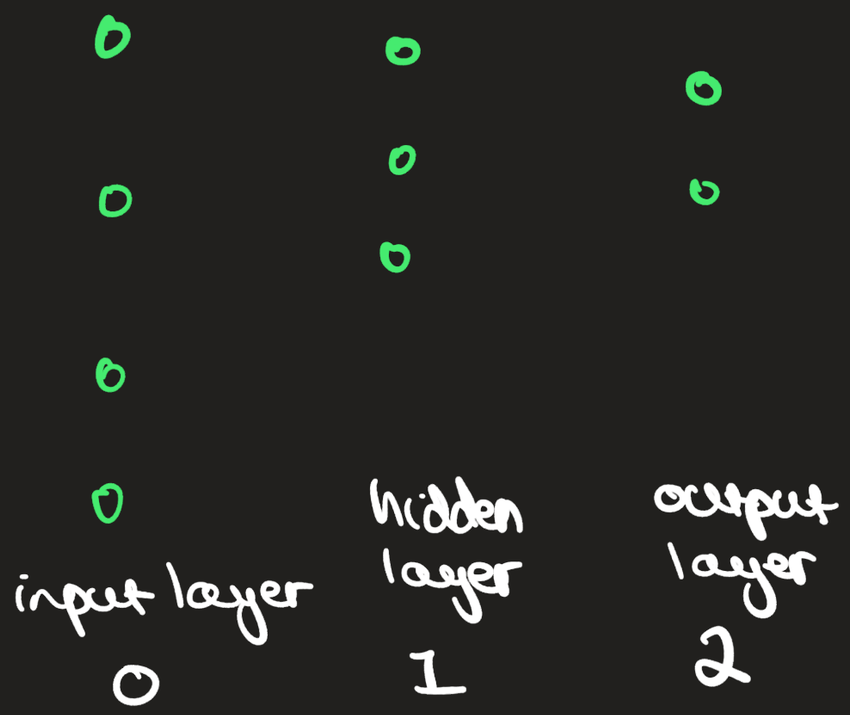


Vector notation for MLPs

→ Multilayer perceptron NNs: FCFFNs where each neuron is of perceptron-type

Let our MLP have L layers
each layer l has n_l neurons
if inputs are in $\mathbb{R}^d$ then $n_0 = d$
if outputs are in $\mathbb{R}^k$ then $n_L = k$

(0 is the input layer
L is the output layer)



$L = 2$
$n_0 = 4$
$n_1 = 3$
$n_L = 2$

We write the preactivations and activations of the neurons on layer l as vectors of length n_l:

$a^l \in \mathbb{R}^{n_l}$ – preactivation vector

$o^l \in \mathbb{R}^{n_l}$ – activation vector

Expression for a^l in terms of o^{l-1}

$$a^l = \eta_l \left\{ \begin{bmatrix} \langle w_1^l, o^{l-1} \rangle + b_1^l \\ \vdots \\ \langle w_{\eta_l}^l, o^{l-1} \rangle + b_{\eta_l}^l \end{bmatrix} \right.$$

This means layer l has $\eta_l \times \eta_{l-1} + \eta_l$ parameters

$$= W^l o^{l-1} + b^l$$

where weight matrix $W^l \in \mathbb{R}^{\eta_l \times \eta_{l-1}}$:

$$W^l = \begin{bmatrix} (w_1^l)^T \\ \vdots \\ (w_{\eta_l}^l)^T \end{bmatrix}$$

$(W^l)_{ij}$ = weight connecting neuron j on layer $l-1$ to neuron i on layer l

and bias vector $b^l \in \mathbb{R}^{\eta_l}$:

$$b^l = \begin{bmatrix} b_1^l \\ \vdots \\ b_{\eta_l}^l \end{bmatrix}$$

Expression for $f(x;\omega)$ an L-layer MLP

$o^l = \sigma(a^l)$ apply activation function element-wise

$= \sigma(W^l o^{l-1} + b^l)$

With this notation, an L-layer MLP computes

$$f(x;\omega) = o^L = \sigma(a^L) = \sigma(W^L o^{L-1} + b^L)$$
$$= \sigma(W^L \sigma(W^{L-1} o^{L-2} + b^{L-1}) + b^L)$$
$$= \cdots$$
$$= \sigma(W^L \sigma(W^{L-1} \cdots (\sigma(W^1 x + b^1) + b^2) \cdots + b^L)$$

parameters of NN are $\omega = \{W^1, \cdots, W^L, b^1, \cdots, b^L\}$

Training a given architecture consists of learning ω by solving the RERM (e.g. w/ SGD)