ML & Optim Lecture 17 – Neural Networks

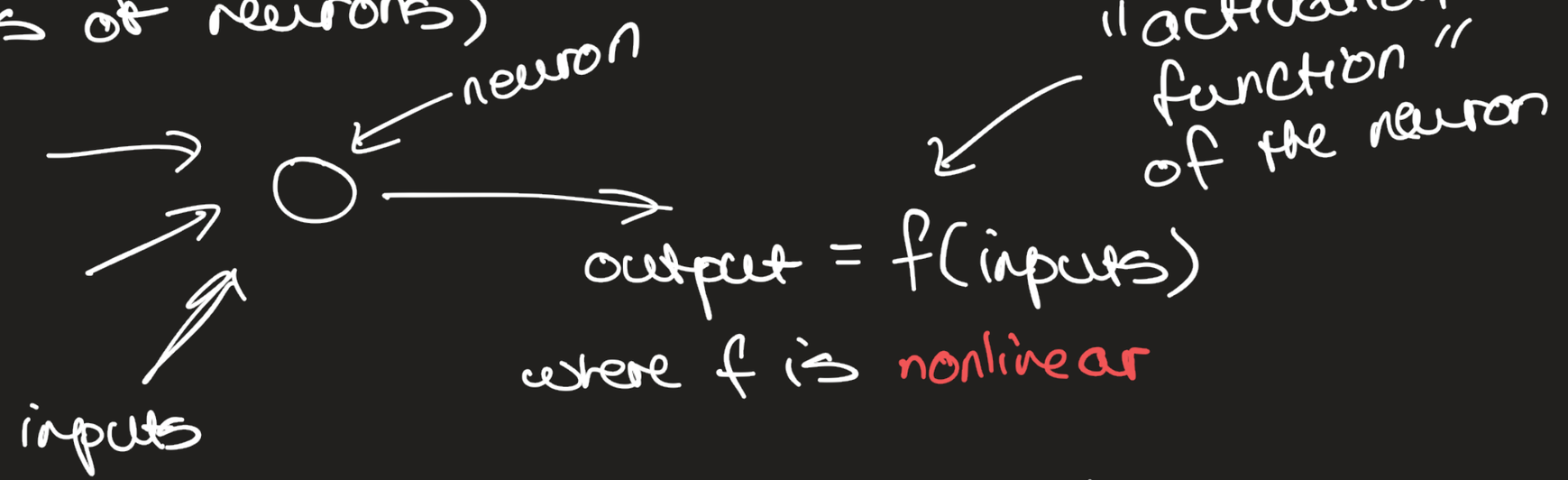
- Neural Networks as "modern" alternatives to kernel machines
- Expressivity of NNs
- Backpropagation alg for training NNs

Motivations for modern NNs

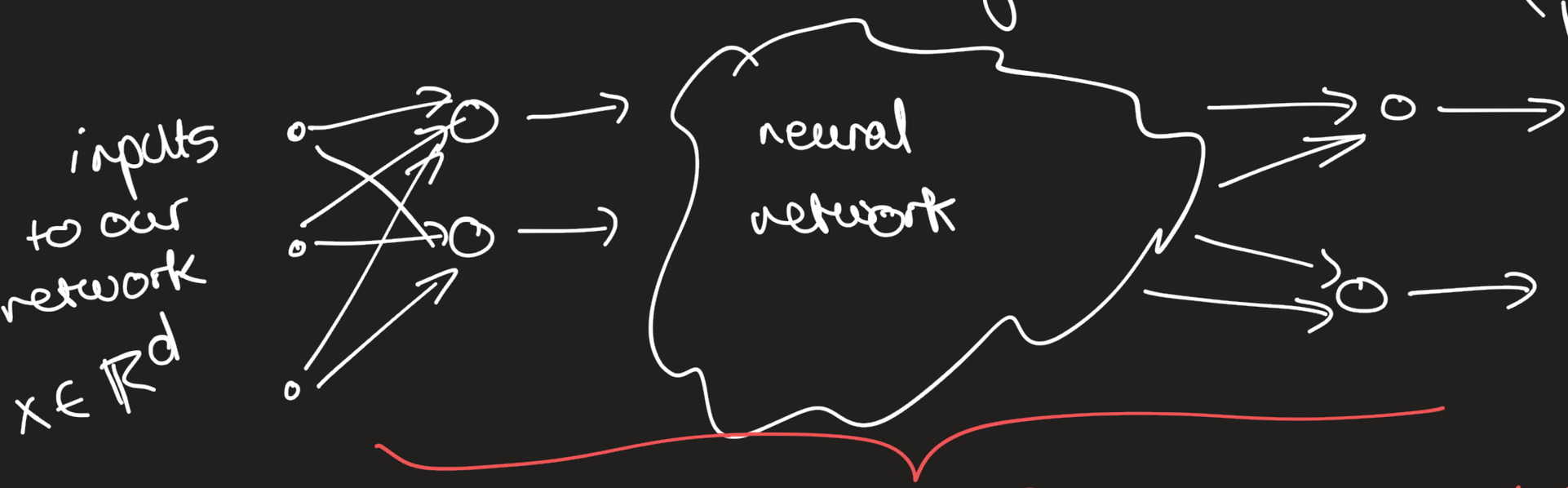
- more easily scalable than kernel approaches, due to GPGPUs and SGD algorithms
- very good inductive biases for many domains by using well-engineered architectures (will discuss some later) → state-of-the-art leads to SOTA results in many domains
- can take advantage of modern massive datasets

What is a neural network?

- motivated by biological neural networks (collections of neurons)



- neurons are connected together as weighted DAG (directed acyclic graph: no feedback loops or self loops)



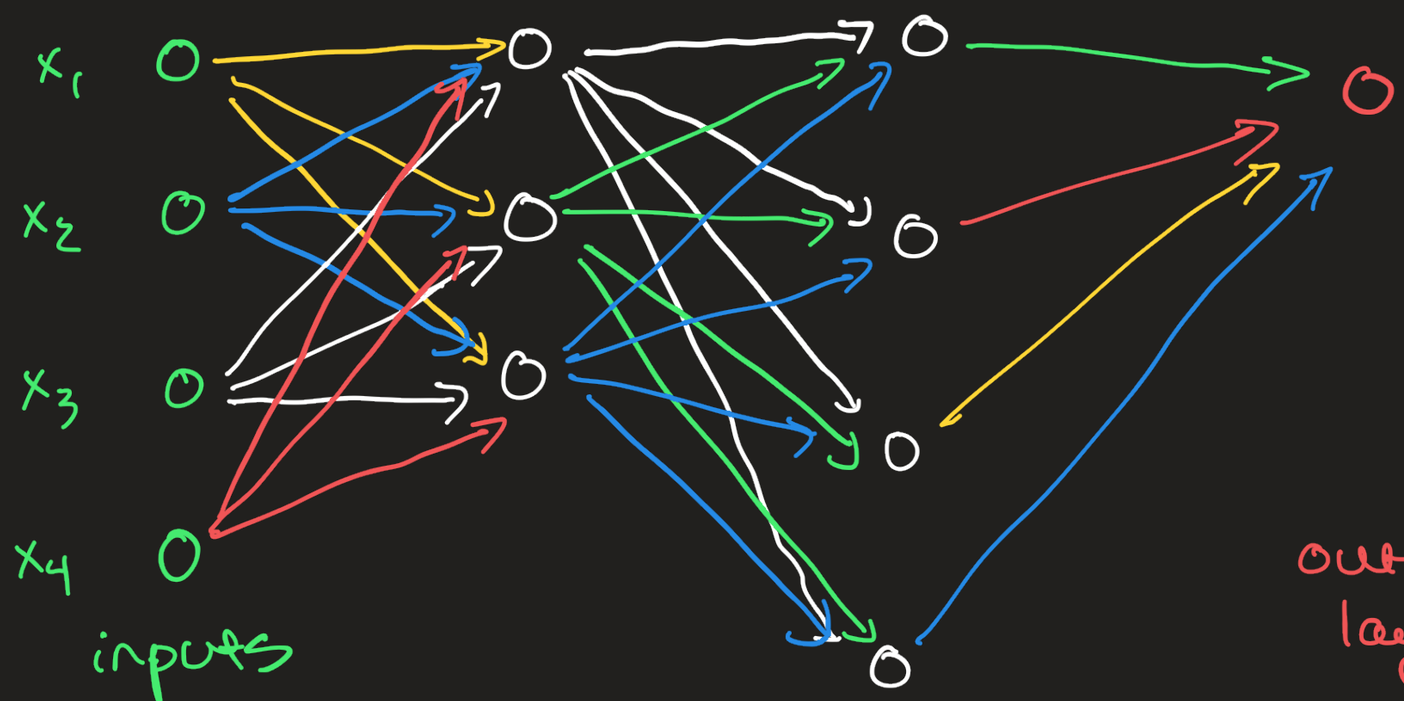
Given a particular DAG (architecture of the NN), we learn the weights of the edges as our parameters to minimize the RERM

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

$f(x_i; w)$: NN evaluated w/ x_i as input and w as the weights

The architecture determines how many parameters we have, and the way in which f depends on those parameters.

Architecture Choice 1: Fully-connected Feedforward NNs

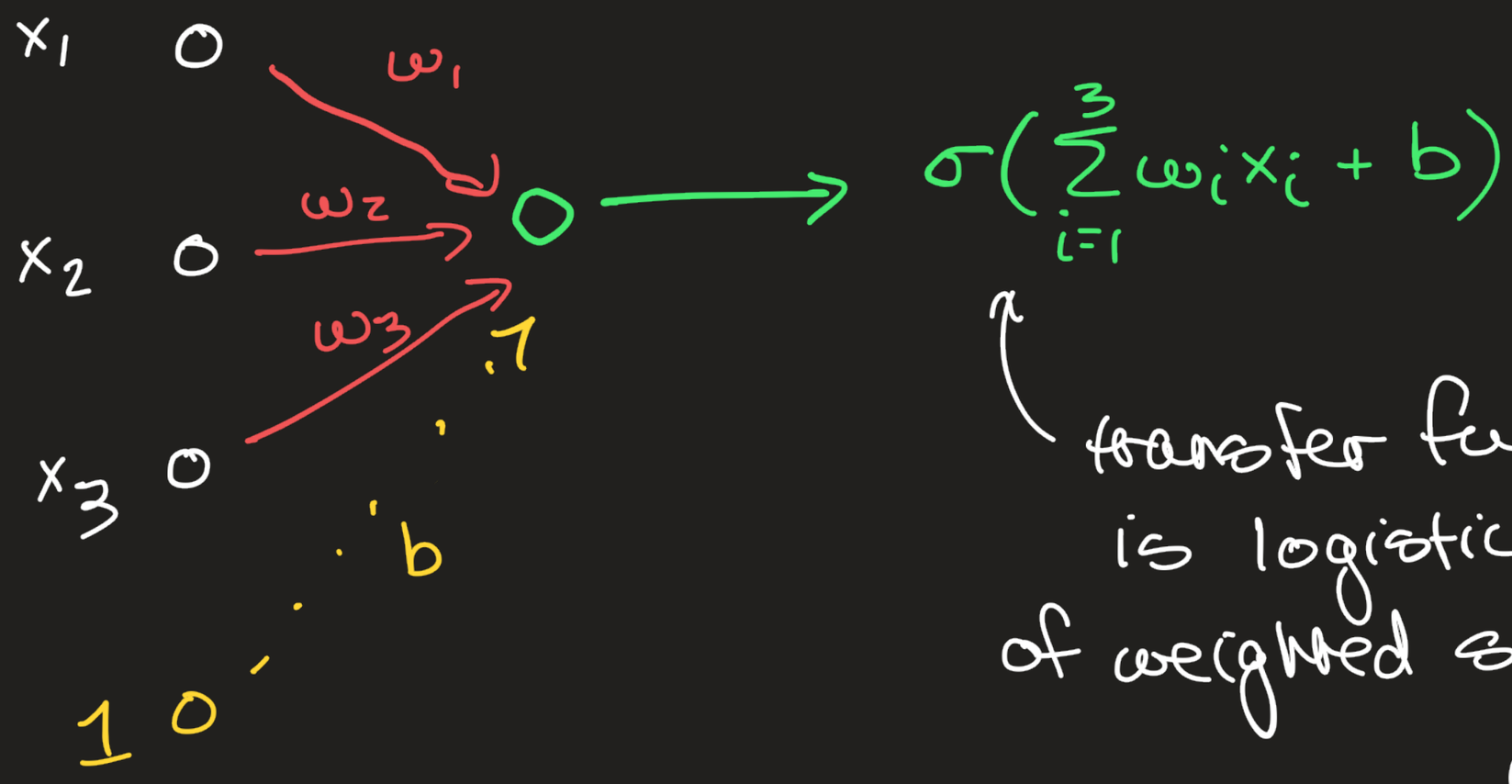


All neurons on a layer are connected to all neurons on the succeeding layer

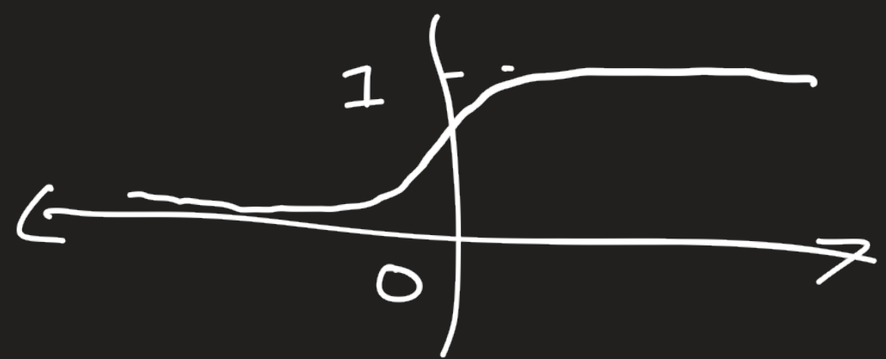
- The neurons are divided into:
  - an input layer (layer 0), corresp to x
  - an output layer, corresp to y
  - zero or more hidden layers

Ex of a FCFFN

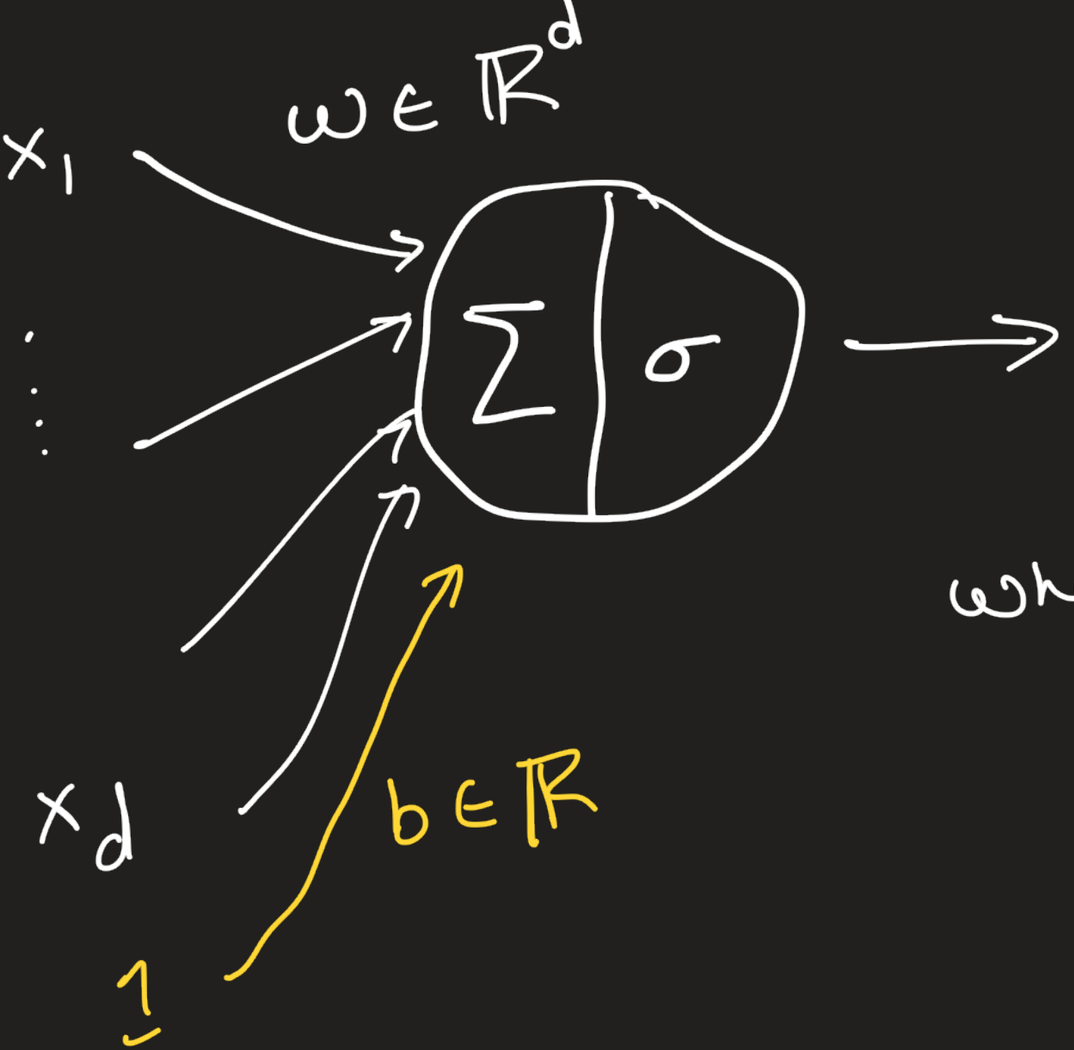
$x \in \mathbb{R}^3$ and $y \in [0,1]$, prob. of class 1 in a binary classification prob



transfer function is logistic sigmoid of weighted sum



Typical setup of neurons



$$\sigma\left(\underbrace{\langle \omega, x \rangle + b}_{\text{"preactivation" of the neuron}}\right)$$

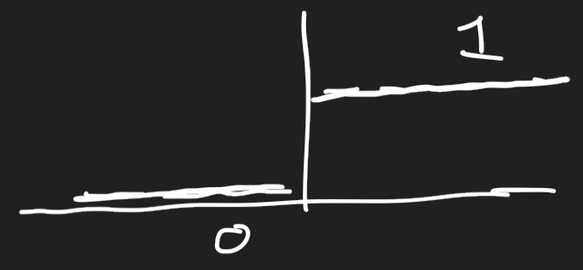
"activation" of the neuron

where σ is a nonlinearity

perceptron-type neurons

Ex activation functions

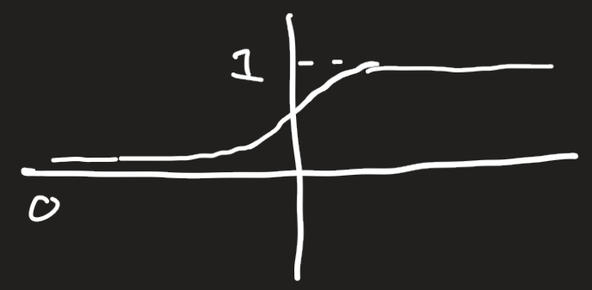
$\sigma(x) = \text{sign}(x)$



σ

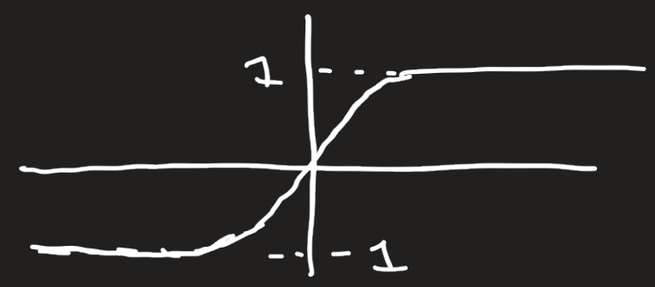
sign / threshold

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



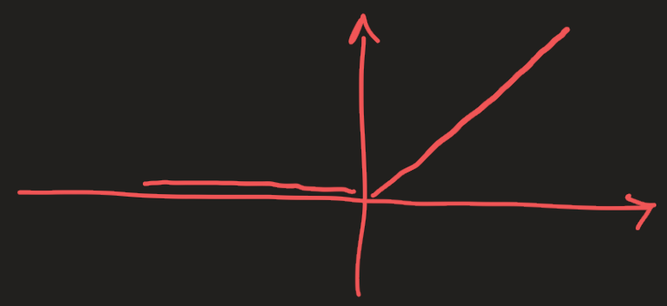
logistic sigmoid

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



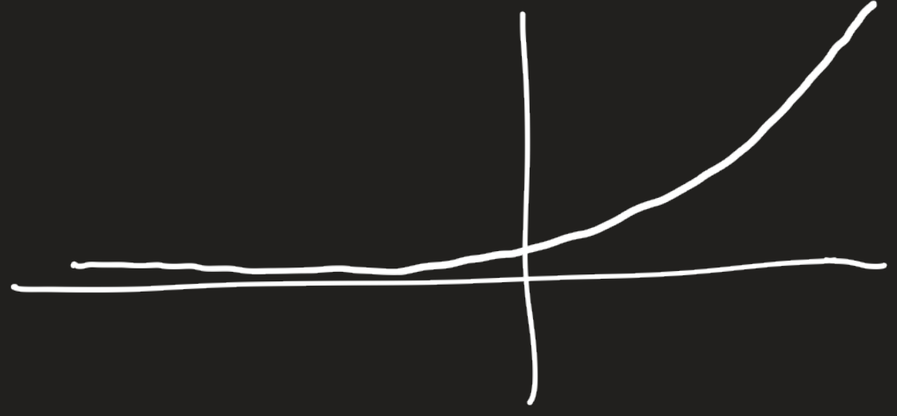
hyperbolic tangent

$\sigma(x) = x_+$

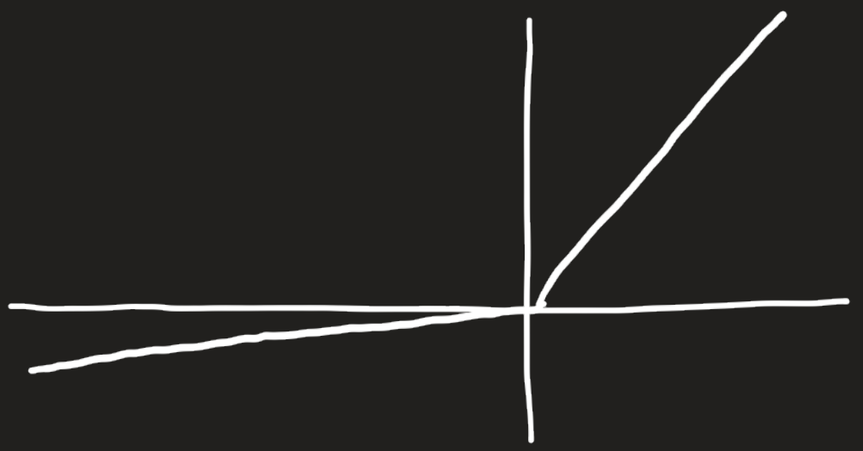


ReLU
rectified linear units

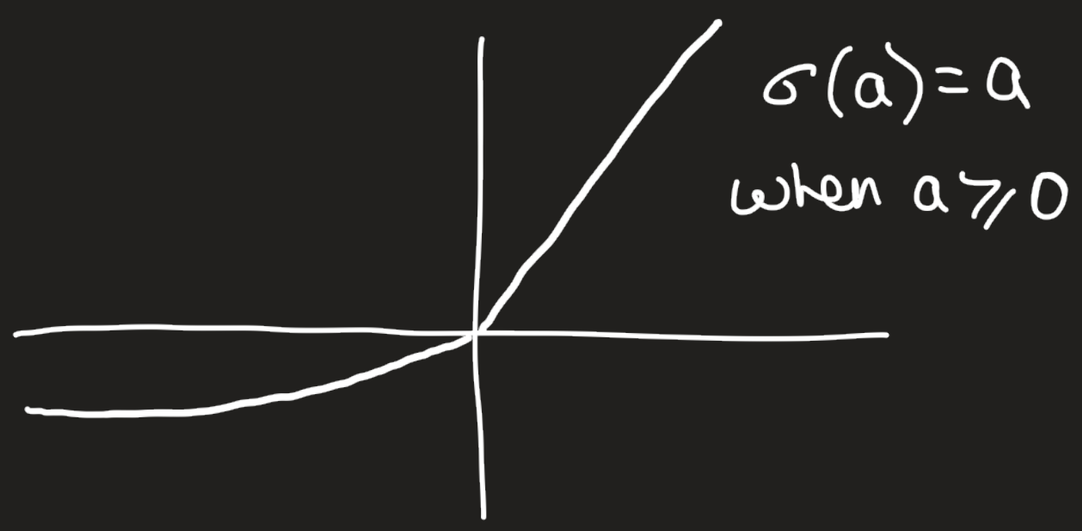
$\simeq$ 2015



$$\sigma(x) = \ln(1 + e^x)$$ softplus

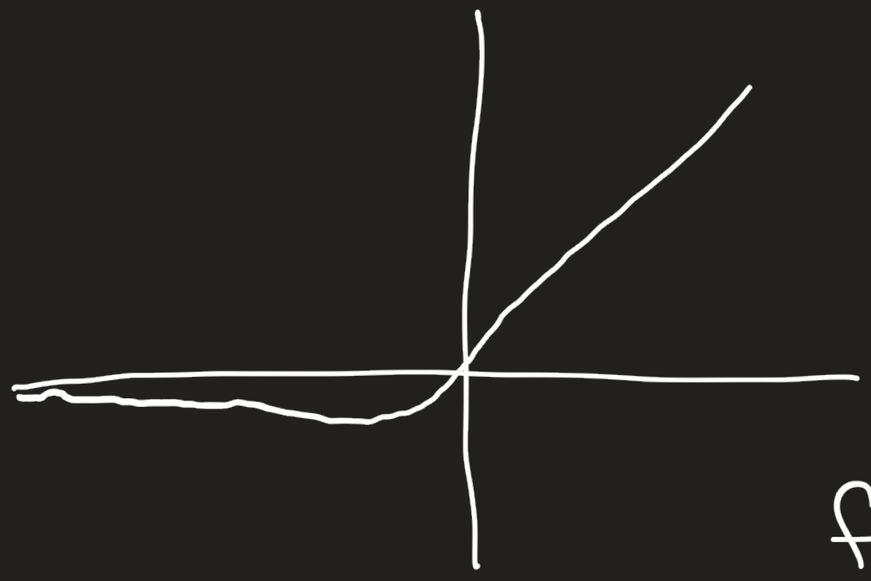


$$\sigma(x) = \begin{cases} x & \text{if } x \geq 0 \\ \beta x & \text{if } x < 0 \end{cases}$$ leaky ReLU



exponential LU (ELU)
or
scaled ELU (SELU)

$\sigma(a) = \gamma(e^a - 1)$ when $a < 0$



$$\sigma(x) = x\,\sigma(\beta x)$$

↑ logistic sigmoid

"swish" function 2017

found by search over a large expression space of potential activation functions

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

- the architecture
- the optimization alg
- the regularizer
- etc.

by taking the application domain & dataset into account

Expressivity of MCPs (multi-layer perceptron NNs)

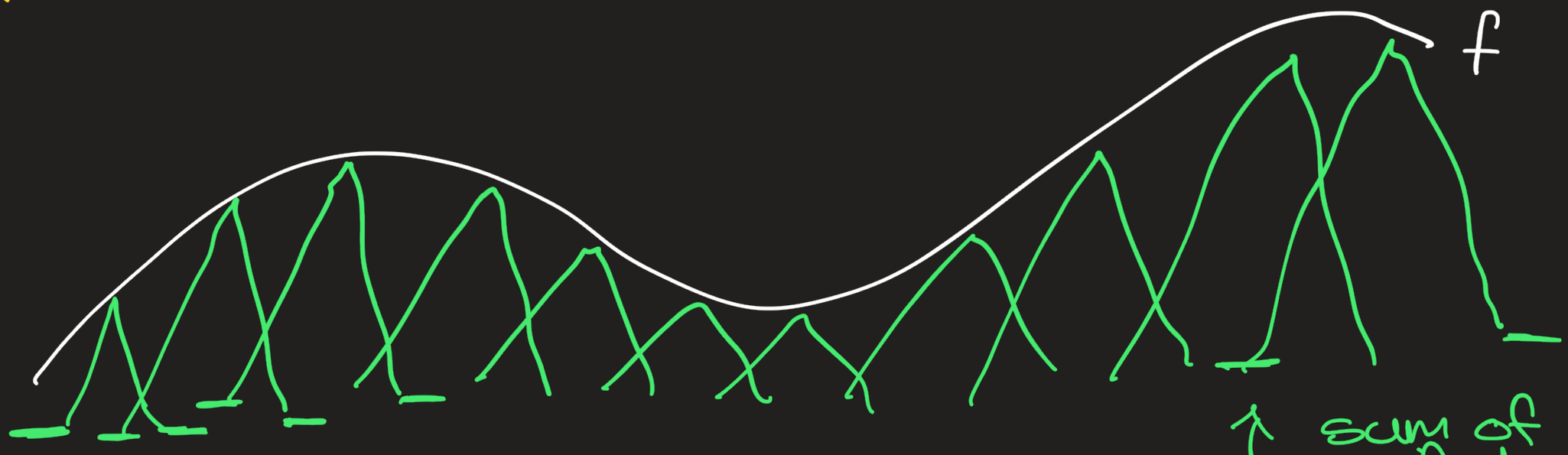
What class of functions can be expressed/ represented using a MLP w/ L layers and an arbitrary number of neurons?

Answer: for any activation function that is nonlinear & monotonic, it is the case that any function $f \in L_2$ can be approximated arbitrarily well in L_2-norm by a two-layer NN w/ this activation function. (output layer takes linear combinations).

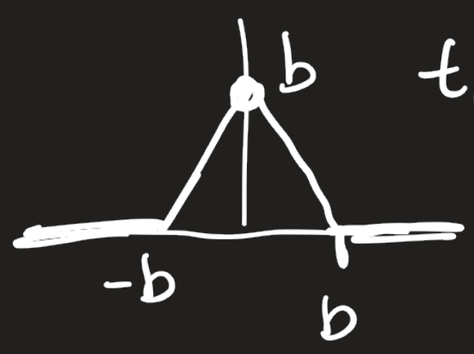
That is, there exists a 2-layer MCP $g(x)$ such that

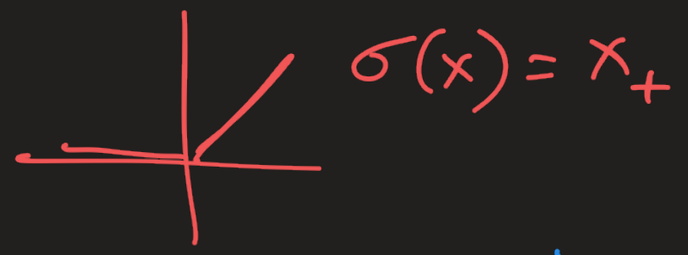
$$\|g - f\|_2^2 = \int |f(x) - g(x)|^2 \, dx < \epsilon.$$

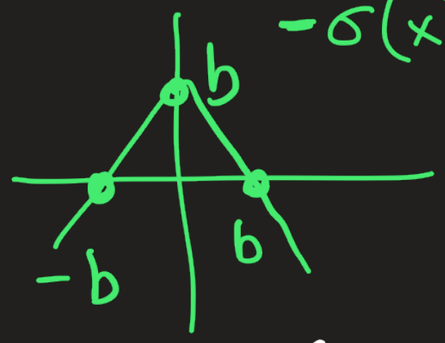
Ex: ReLU

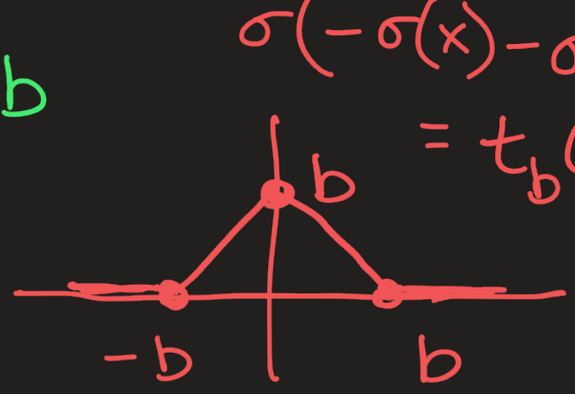


Question: how to form a triangle $t_b(x)$



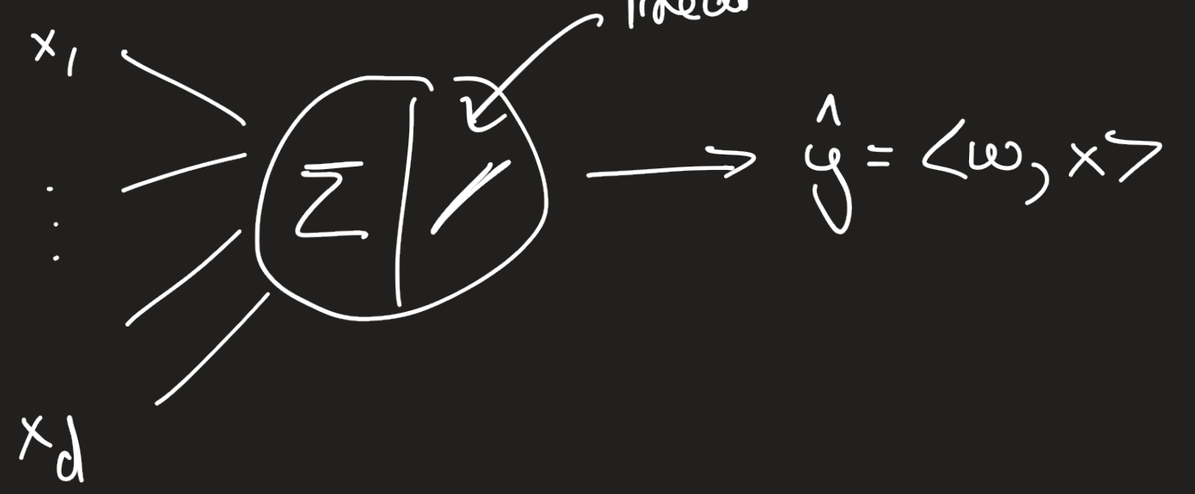






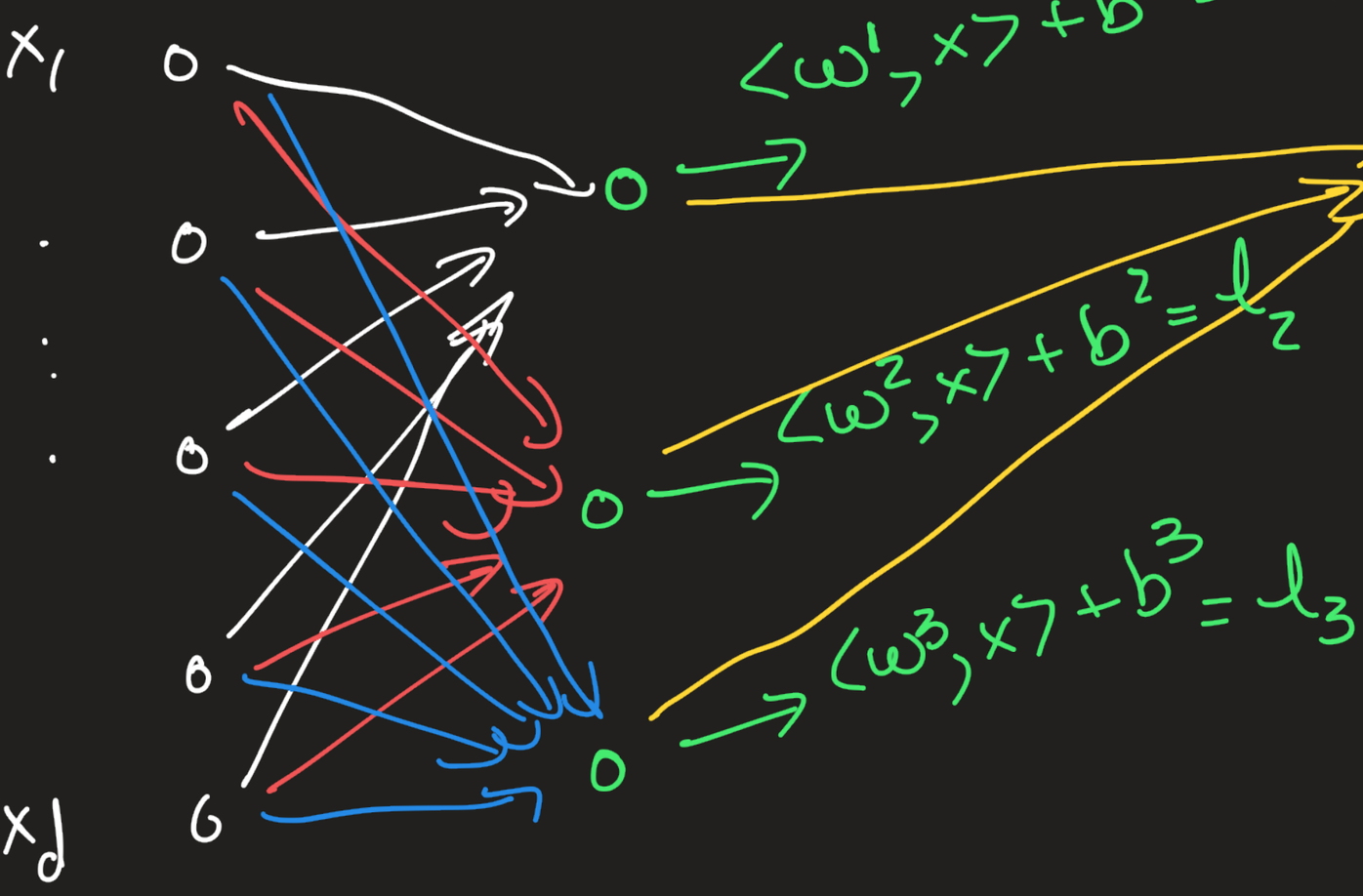
All models till now are NNs

Linear regression



train this NN w/
$l(y, \hat{y}) = (\hat{y} - y)^2$
and we get linear regression

Multinomial Logistic Regression (3-class)



$$o \longrightarrow p_1 = \frac{e^{l_1}}{e^{l_1} + e^{l_2} + e^{l_3}}$$

$$o \longrightarrow p_2 = \frac{e^{l_2}}{e^{l_1} + e^{l_2} + e^{l_3}}$$

$$o \longrightarrow p_3 = \frac{e^{l_3}}{e^{l_1} + e^{l_2} + e^{l_3}}$$

layer 0

hidden layer 1 (logits)

linear activation

output layer (class probs)

softmax activation

softmax layer

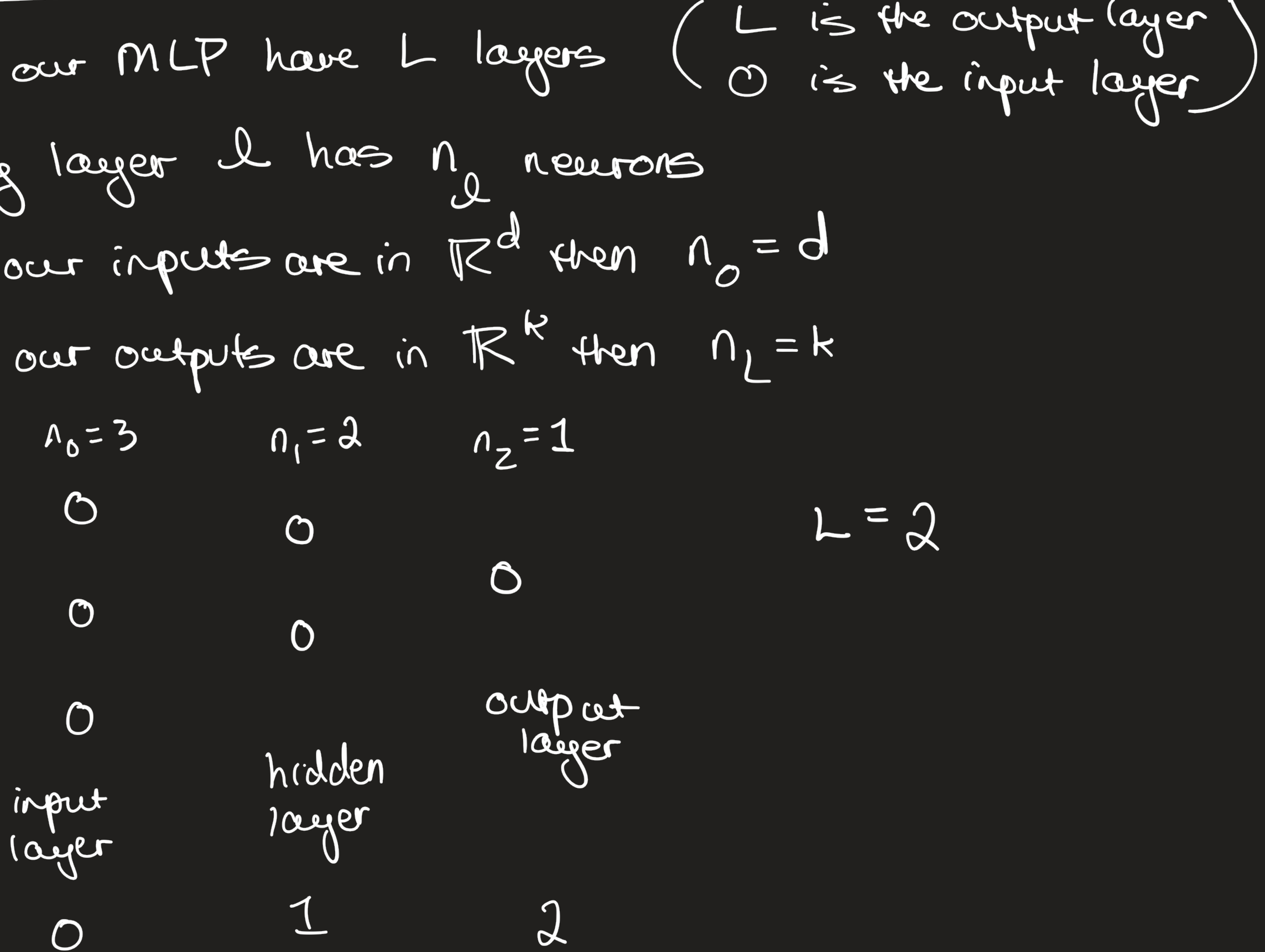
Vector notation for MLPs

Let our MLP have L layers (L is the output layer, 0 is the input layer)

say layer l has n_l neurons

if our inputs are in $\mathbb{R}^d$ then $n_0 = d$

if our outputs are in $\mathbb{R}^k$ then $n_L = k$



We write the preactivation and activation values of the neurons on layer l as vectors of length n_l:

$$a^l \in \mathbb{R}^{n_l} \quad \text{(preactivation vector of layer } l\text{)}$$

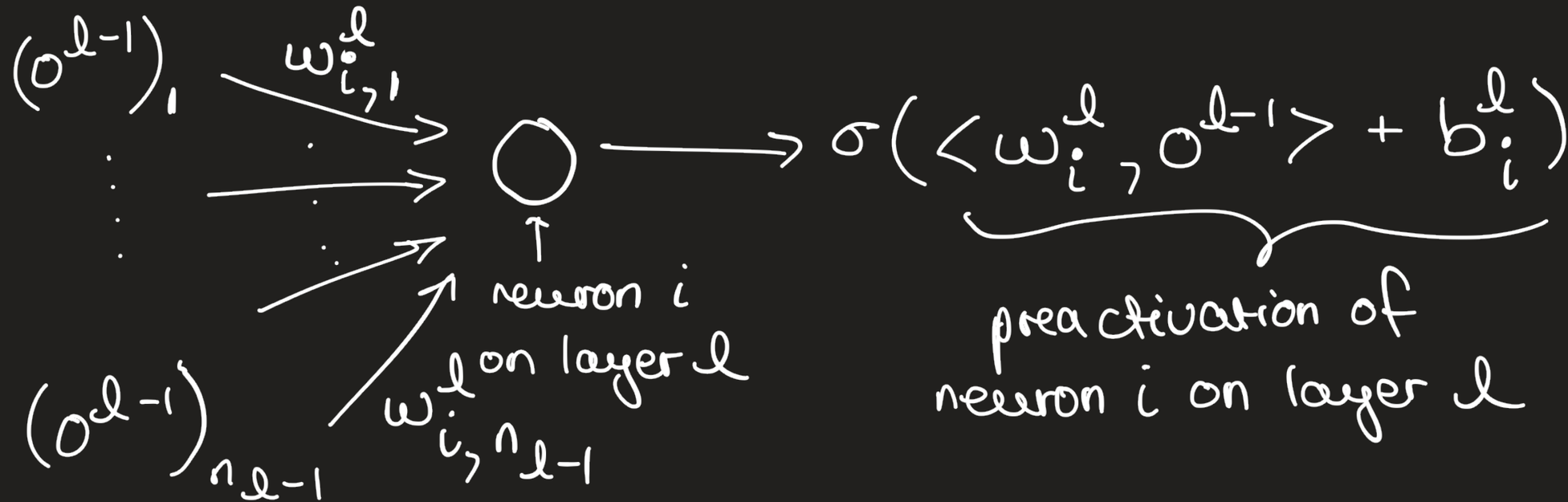
$$o^l \in \mathbb{R}^{n_l} \quad \text{(activation vector of layer } l\text{)}$$

and write

$$o^l = \sigma(a^l)$$

meaning entrywise application of the activation function.

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



so

$$a_i^l = \langle w_i^l, o^{l-1} \rangle + b_i^l \quad \text{for each } i \in [n_l]$$

$$\Rightarrow \quad a^l = \begin{bmatrix} (w_1^l)^T \\ \vdots \\ (w_{n_l}^l)^T \end{bmatrix} o^{l-1} + \begin{bmatrix} b_1^l \\ \vdots \\ b_{n_l}^l \end{bmatrix}$$

or

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

where $$W^l = \begin{bmatrix} (w_1^l)^T \\ \vdots \\ (w_{n_l}^l)^T \end{bmatrix} \in \mathbb{R}^{n_l \times (n_{l-1})}$$

$$b^l = \begin{bmatrix} b_1^l \\ \vdots \\ b_{n_l}^l \end{bmatrix} \in \mathbb{R}^{n_l}$$

So

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

With this notation, an L-layer MLP is a function of the form

$$\hat{y}(x) = o^L = \sigma(a^L) = \sigma(w^L o^{L-1} + b^L)$$

$$= \sigma(w^L \sigma(w^{L-1} o^{L-2} + b^{L-1}) + b^L)$$

$$= \dots$$

$$= \sigma(w^L \sigma(w^{L-1} \dots (\sigma(w^1 x + b^1) + b^2) \dots + b^L)$$

Note that the activation function may be different for different layers (or within the same layer)

Training a given architecture consists of learning $\{w_i, b_i\}_{i=1}^{L}$ by solving the RERM