ML & Optimization Lec 16

Adaptive Gradient Descent: RMSProp (2012), ADAM (2014)

Neural Networks:
- concept of neurons and neural networks
- expressivity
- vector notation for neural networks
- example using Pytorch

RMSProp (2012)

attempted to fix "problem" of Adagrad's inevitable learning rate decay

typically 0.9

$$D_t = \beta D_{t-1} + (1-\beta)\,\mathrm{diag}\left(\nabla f_t(x_t)\,\nabla f_t(x_t)^T\right)$$

vs

$$D_t = D_{t-1} + \mathrm{diag}\left(\nabla f_t(x_t)\,\nabla f_t(x_t)^T\right)$$

The update is the same

$\approx 10^{-8}$

$$x_{t+1} = x_t - \alpha\,(D_t + \epsilon I)^{-1/2}\,\nabla f_t(x_t)$$

ADAM (2014)

very popular in the ML community

- tries to fix the learning rate decay issue of Adagrad
- also tries to estimate a better search direction using momentum

$$\mu_t = \beta_1 \mu_{t-1} + (1-\beta_1) \nabla f_t(x_t)$$

$$D_t = \beta_2 D_{t-1} + (1-\beta_2) \operatorname{diag}\left(\nabla f_t(x_t) \nabla f_t(x_t)^T\right)$$

$$\omega_t = \omega_{t-1} - \alpha \left(D_t + \varepsilon I\right)^{-1/2} \mu_t$$

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 by using well-engineered architectures (will discuss many in later lectures)
led to SOTA performance in many domains

- can take advantage of modern massive datasets

What is a NN?

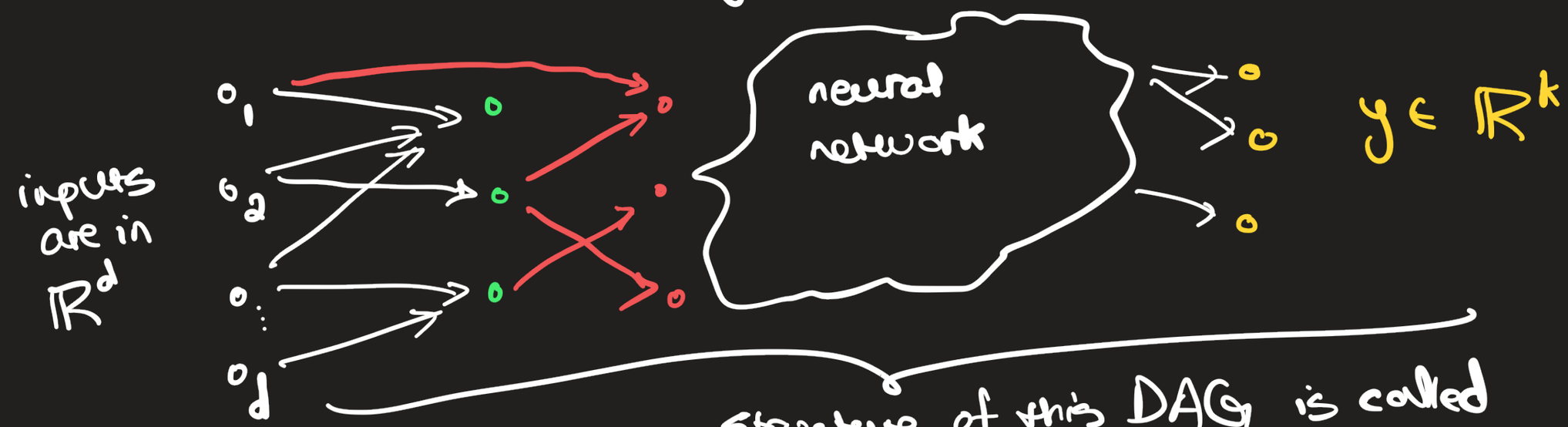
- motivated by biological neural networks (collection of neurons)

output = f(input)

[called the "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"

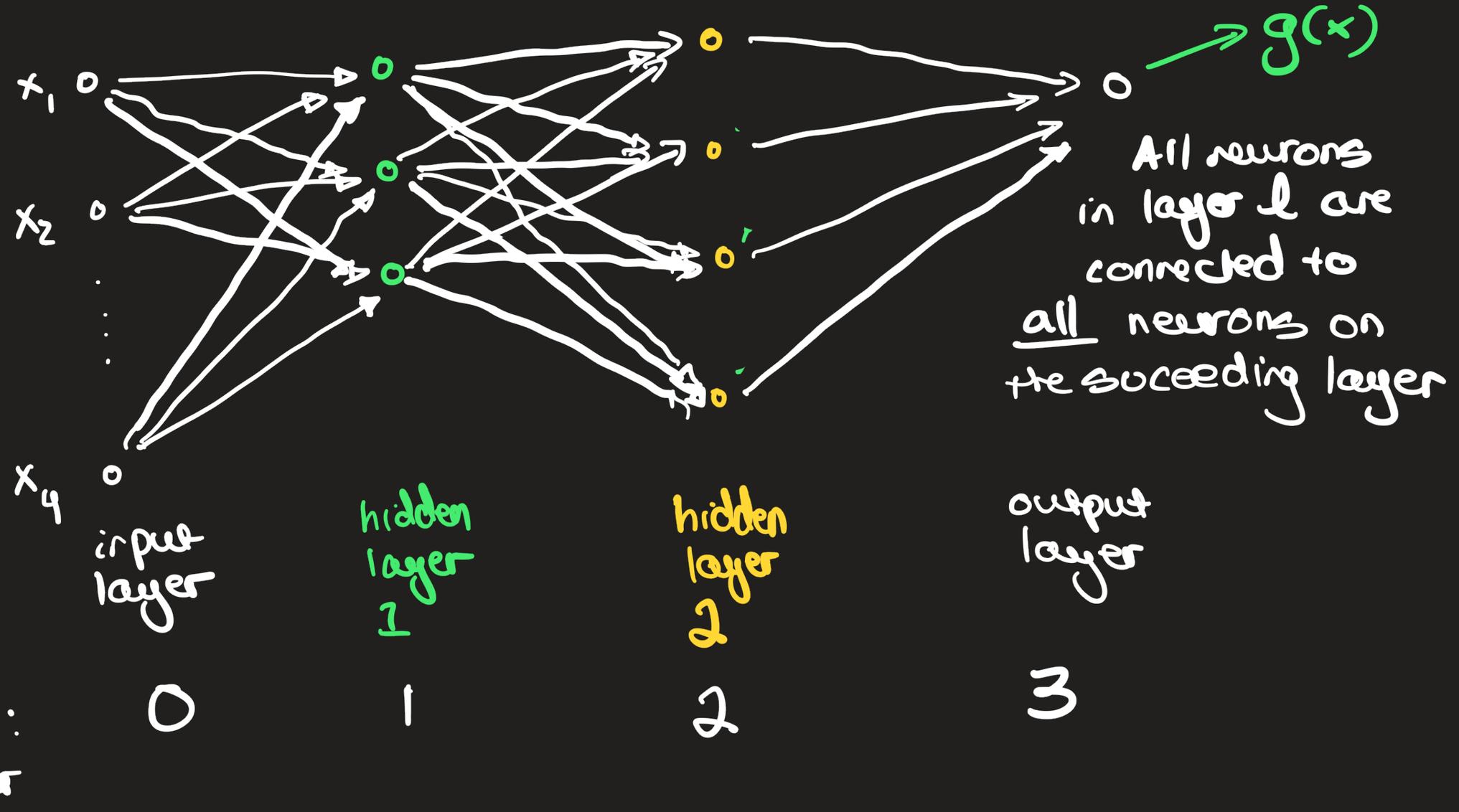
Given a particular architecture of the NN,
learn the parameters associated with each neuron
to minimize the RERM

$$w^* = \underset{w}{\operatorname{argmin}} \; \frac{1}{n}\sum_{i=1}^{n} \ell\big(\underbrace{f(x_i; w)}_{\text{NN evaluated w/ } x_i \text{ as input when } w \text{ are its parameters}}, y_i\big) + \lambda R(w)$$

The particular architecture determines:

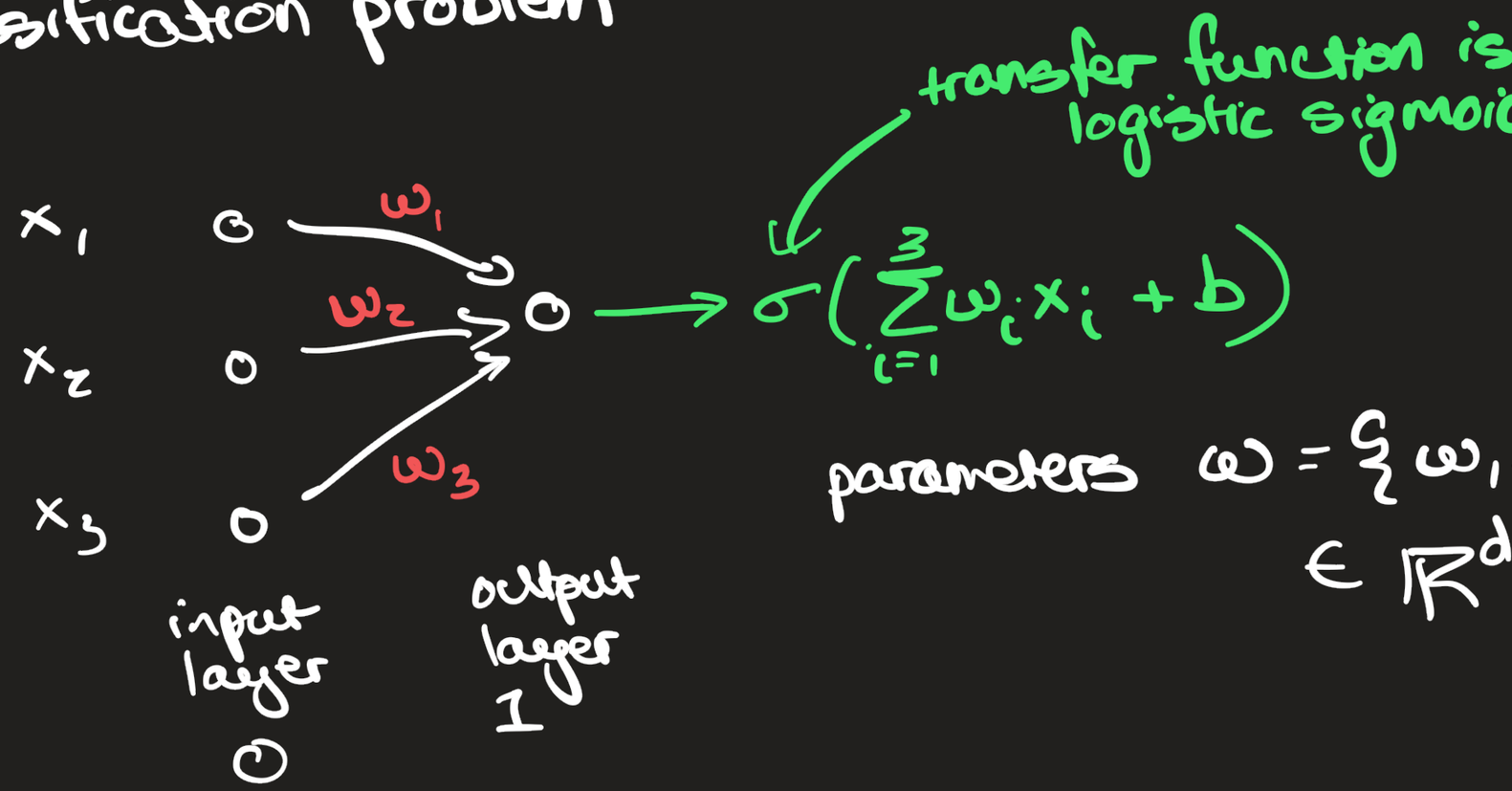
- how many parameters there are
- how f depends on these parameters

Architecture Choice 1: Fully-connected Feedforward NNs



Example of a FCFFN

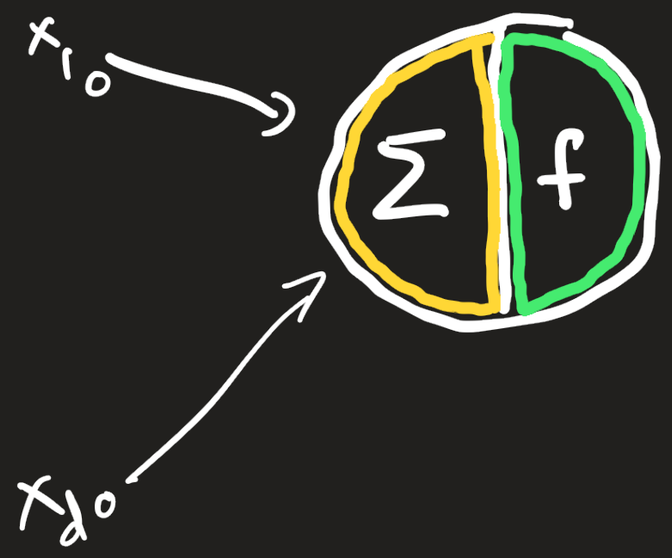
$x \in \mathbb{R}^3$ and $y \in [0,1]$ → prob of class 1 in a two-class classification problem



parameters $w = \{w_1, w_2, w_3, b\}$
$\in \mathbb{R}^{d+1}$

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

Typical setup of neurons



output is $f(\langle \bar{\omega}, x \rangle + b)$

where f is a (nonlinear) activation function

preactivation

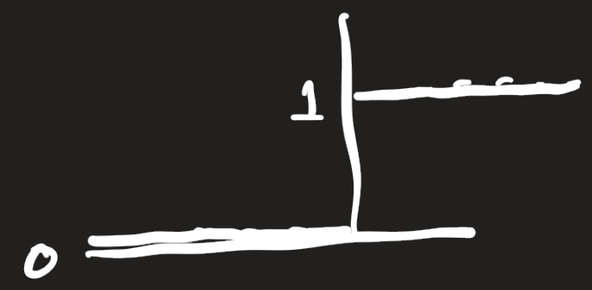
activation

parameters $w = \{\bar{\omega}, b\}$
$\in \mathbb{R}^{d+1}$

perceptron-type neurons

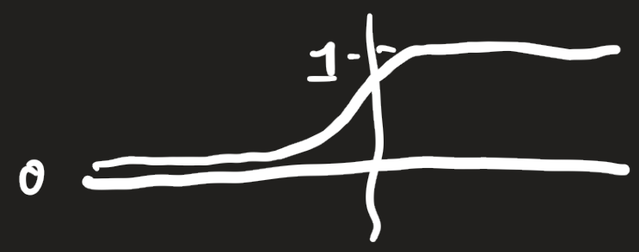
Ex activation functions

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



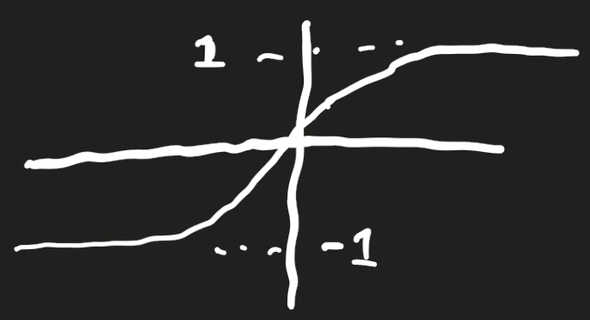
sign/thresholding

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



logistic sigmoid

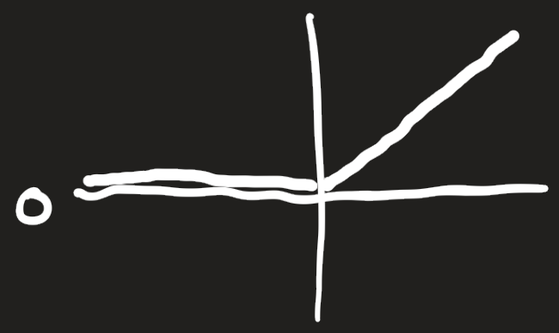
$\sigma(x) = \tanh(x)$



hyperbolic tan

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

$= x_+$



ReLU
rectified linear unit

≈ 2015

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

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

by taking the application domain & validation dataset into consideration

Expressivity of NNs

What class of functions can be expressed/represented accurately using networks of perceptron-type neural networks w/ L layers and an arbitrary number of neurons?

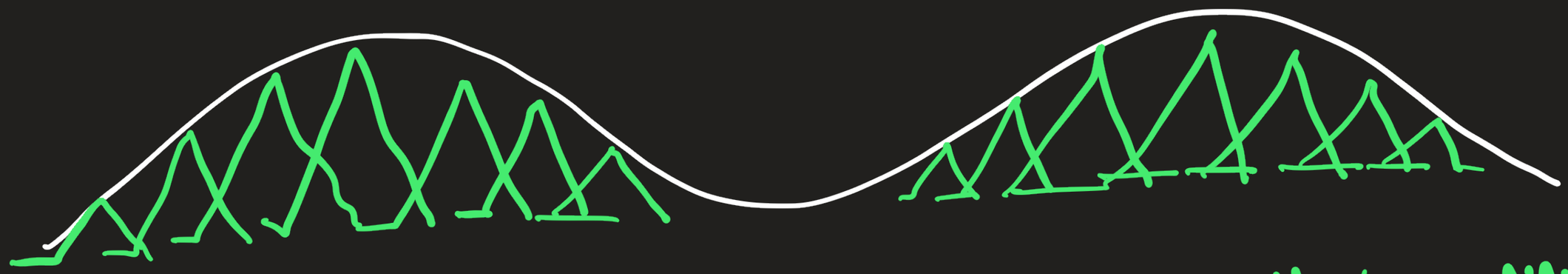
Consider MLPs (multilayer perceptrons): FCFFNs with perceptron-type neurons

Answer: for any activation function that is nonlinear and monotonic, it is the case that for any square-integrable function $f \in L_2$ can be approximated arbitrarily well in L_2-norm by a two-layer NN w/ this activation function

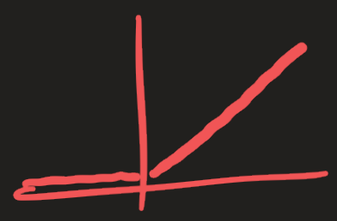
That is, there exists a 2-layer MLP $g(x)$ such that

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

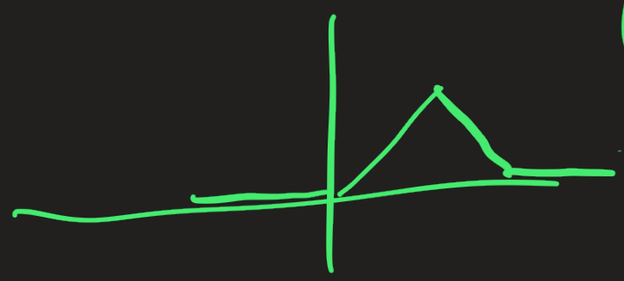
two hidden layers and an output layer



Recall ReLU activation:



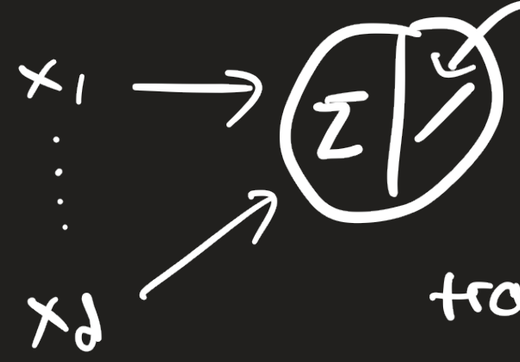
Claim: by using two-hidden layer NN can represent



(exercise!)

All models till now are NNs

Linear regression



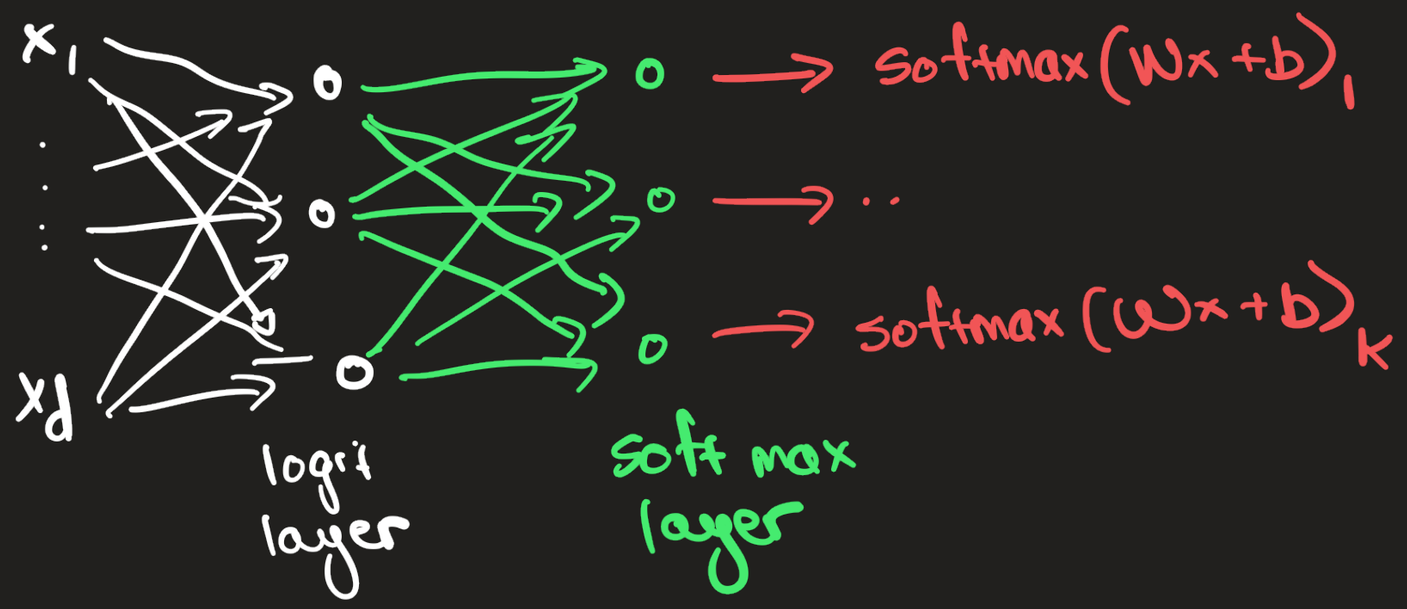
activation function $\sigma(x) = x$

$\rightarrow \langle w, x \rangle + b$

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

then we get the linear regression model

Multinomial Logistic regression



Vector notation for MLPs

Let our MLP have L layers

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

say 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 $n_L = k$

$L = 2$

$n_0 = 4$

$n_1 = 3$

$n_2 = 2$

We write the preactivations and activation values 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}

$$a^l = \begin{matrix} \\ n_l \end{matrix} \left\{ \begin{bmatrix} \langle \omega_1^l, o^{l-1} \rangle + b_1^l \\ \vdots \\ \langle \omega_{n_l}^l, o^{l-1} \rangle + b_{n_l}^l \end{bmatrix} \right.$$

This means layer l has $n_l \times n_{l-1} + n_l$ parameters

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

where weight matrix $w^l \in \mathbb{R}^{n_l \times n_{l-1}}$:

$$w^l = \begin{bmatrix} - & \omega_1^{l\,T} & - \\ & \vdots & \\ - & \omega_{n_l}^{l\,T} & - \end{bmatrix}$$

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

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

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

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 $\omega = \{w^L, \cdots, w^1, b^L, \cdots, b^1\}$

Training a given architecture consists of learning ω by solving the RERM