Coding session will be exciting. We'll see how to get our machines to recognize <u>handwritten digits</u>!

$\mathcal{X}$ = array of 'pixels' $\approx [0,1]^{m\times n} \approx \left\{ \frac{k}{N} : 0 \le k \le N \right\}^{n\times m}$

$\mathcal{Y} = \{0, \ldots, 9\}$

D = classified images of handwritten digits

<u>Recall</u>: We have data points $(x_i, y_i)_{i=1}^{n} = D$. We <u>assume</u> $y_i = f^*(x_i)$ (+ noise) for some $f: \mathcal{X} \to \mathcal{Y}$.

Our goal is to determine f^*.

① Consider a pool of candidate functions $\mathcal{F}$

<u>Ex.</u>: $\mathcal{X} = \mathcal{Y} = \mathbb{R}$, $\{x \mapsto \beta_1 x + \beta_2 x^2 + \beta_{100} x^{100} : \beta_1, \beta_2, \beta_{100} \in \mathbb{R}\}$

② Use D and a loss function $\ell(\cdot,\cdot)$ to define the <u>empirical risk</u> $\hat{R}_D : \mathcal{F} \to [0, \infty]$

$\hat{R}_D(f) = \frac{1}{n} \sum_{i=1}^{n} \ell(y_i, f(x_i))$. <u>Ex.</u>: $\ell(y, y') = \frac{1}{2}\|y - y'\|^2$ (quadratic loss)

③ Use optimization methods to find (an approximate) optimizer $\hat{f}$ of $\hat{R}_D$

<u>Ex.</u>: use GD, SGD, SGD with minibatching, ...

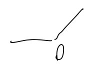
(Should be clear to everyone what ①, ②, ③ are in the case of linear regression)

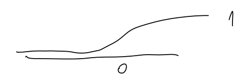
A neural network (NN) is just <u>a specific choice of $\mathcal{F}$.</u>

- NN: originally introduced to model biological neurons (→ 'perceptron') and their interaction, hence the name
- Allow for efficient implementation of (a version of) GD (via 'backpropagation')
- Perform well in practice

We discuss the most basic case, fully-connected feedforward NN.

Let $\sigma : \mathbb{R} \to \mathbb{R}$ be a continuous function. Ex.: $\sigma(x) = x^+$ (ReLU), $\sigma(x) = \frac{e^x}{1+e^x}$, ...





Notation: $x \in \mathbb{R}^n$, $\sigma(x) := (\sigma(x_1), \dots, \sigma(x_n)) \in \mathbb{R}^n$.

$k \in \mathbb{N}$, $n_0, \dots, n_k \in \mathbb{N}$, $W_i \in \mathbb{R}^{n_i \times n_{i-1}}$, $b_i \in \mathbb{R}^{n_i}$, $1 \leq i \leq k$,

$$f(x) = W_k\Big(\dots \sigma\Big(W_2\, \sigma\big(\underbrace{W_1 x + b_1}_{\in \mathbb{R}^{n_1}}\big) + b_2\Big)\dots\Big) + b_k, \quad x \in \mathbb{R}^{n_0}.$$

$W_1 \dots W_k$ 'weights'

$b_1 \dots b_k$ 'biases'

σ 'activation fun.'

$\max_{0 \leq i \leq k} n_i$ = 'width', k = 'depth', $k - 2$ = # hidden layers

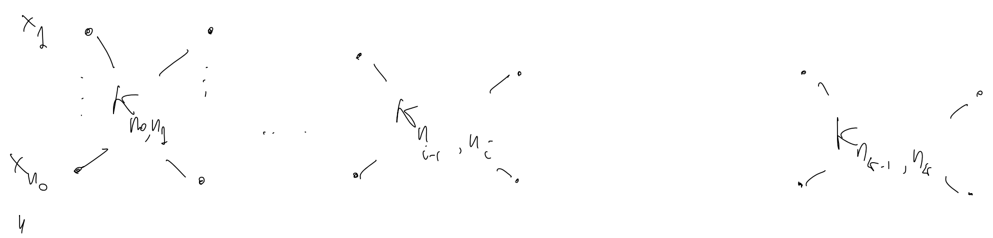
$$f : \mathbb{R}^{n_0} \xrightarrow{(W_1, b_1)} \mathbb{R}^{n_1} \xrightarrow{\sigma} \mathbb{R}^{n_1} \xrightarrow{(W_2, b_2)} \mathbb{R}^{n_2} \xrightarrow{\sigma} \mathbb{R}^{n_2} \dots \xrightarrow{\sigma} \mathbb{R}^{n_{k-1}} \xrightarrow{(W_k, b_k)} \mathbb{R}^{n_k}$$

Sometimes, $b_k = 0$ is imposed.

<u>Ex.</u>: $x \in \mathbb{R}^{n_0}$, $x =$ handwritten digit, $f(x) = (f_0(x), \dots, f_9(x))$,

$f_i(x) = \sim$ 'probability that x represents the digit i' (after normalisation, eg with softmax)

'Pictoral' representation of f: ($K_{s,t}$ = complete (s,t)-bipartite graph, $s,t \in \mathbb{N}$)



$x \in \mathbb{R}^{n_0}$ $\quad F_1 := \sigma(W_1 x + b_1)$ $\quad F_i := \sigma(W_i F_{i-1}(\cdot) + b_i)$ $\quad F_{n_k} = f$ = output

(There are other ways to connect neurons, see eg. convolutional NN.)

<u>In practice</u>, one fixes $k \in \mathbb{N}$, $n_0 \; n_k \in \mathbb{N}$ (and σ) ('hyperparameters') at the beginning, and then optimises over W_i, b_i ('training').

$\mathcal{X} \subseteq \mathbb{R}^n$, $\mathcal{Y} \subseteq \mathbb{R}^m$.

We take $n_0 = n$ and $n_L = m$.

$$\mathcal{F}(n_1 \dots n_{L-1}; \sigma) := \text{functions as above}$$

$$\mathcal{F}(\sigma) := \bigcup_{L \in \mathbb{N}_{\geq 2}} \; \bigcup_{n_1 \dots n_{L-1} \in \mathbb{N}} \mathcal{F}(n_1 \dots n_{L-1}; \sigma).$$

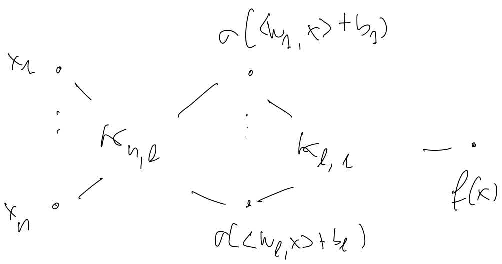
Recall steps ①, ②, ③. Each one introduces an error:

① $\rightsquigarrow$ approximation error, ② $\rightsquigarrow$ estimation error, ③ $\rightsquigarrow$ optimisation error.

We will ~~not~~ discuss ③ and only focus on $L = 2$, $m = n_2 = 1$, $b_2 = 0$, and arbitrary $n_1 \in \mathbb{N}$, i.e. functions of the form

$$f(x) = \sum_{j=1}^{l} \eta_j \, \sigma(\underbrace{\langle w_j, x \rangle}_{\text{scalar product}} + b_j), \qquad x \in \mathbb{R}^n,$$

where $l \in \mathbb{N}$ and $\eta_j \in \mathbb{R}$, $(w_j, b_j) \in \mathbb{R}^n \times \mathbb{R}$ for $1 \leq j \leq l$.



Gather all such functions in the set $\mathcal{G}(n;\sigma)$, $\mathcal{G}(n,\sigma)$ = single hidden layer of arbitrary width.

(1) Which functions can we approximate with elements of $\mathcal{G}(n,\sigma)$?

$\sigma \in C(\mathbb{R}) \Rightarrow \mathcal{G}(n,\sigma) \subseteq C(\mathbb{R}^n)$ $\rightsquigarrow$ Is $\mathcal{G}(n,\sigma)$ <u>dense</u>?

$C(\mathbb{R}^n)$ endowed with the (metrisable) top. of unif. conv. on cpt sets, i.e.

$f_n, f \in C(\mathbb{R}^n)$, $f_n \to f \iff \forall K \subseteq \mathbb{R}^n$ cpt, $\max_{x \in K} |f_n(x) - f(x)| \xrightarrow[n\to\infty]{} 0$.

<u>Thm</u> (Pinkus, '99) Let $\sigma \in C(\mathbb{R})$. TFAE

- $\mathcal{G}(1,\sigma)$ is dense in $C(\mathbb{R})$
- $\mathcal{G}(n,\sigma)$ is dense in $C(\mathbb{R}^n)$ $\forall n$
- σ is not a polynomial

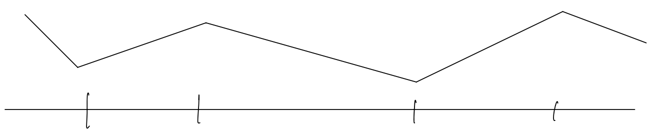
In part., $\sigma = \text{ReLU}$ or $\sigma = \text{sigmoid}$ are okay.

<u>Problem</u>: No information on $\ell \in \mathbb{N}$ nor the size of $(w_j, b_j)_{j=1}^{\ell}$.

Could be useless in practice: We may be unable to construct a NN of the required size or to find the correct weights & biases

It is possible to obtain more 'quantitative' approx. results, e.g. with some smoothness assumption on the fcn. we want to approx. More technical. See e.g. Weinan E.

Ex.1 $\sigma = \text{ReLU}$, $n=1$, $f: \mathbb{R} \to \mathbb{R}$ piecewise affine.



② Estimation error.

Prop (9.1): Let $\sigma = \text{ReLU}$, $D, R > 0$ and

$$\tilde{\mathcal{G}} = \left\{ f \in \mathcal{G}(n,\sigma) : \|a\|_1 \le D, \|w_j\|_2^2 + \left(\frac{b_j}{R}\right)^2 = 1 \ \forall j \right\}.$$

If the loss fun. $\ell(\cdot,\cdot)$ is G-LIP in the second variable, then

$$\mathbb{E}\left[\hat{R}_N(\hat{f}) - \inf_{f \in \tilde{\mathcal{G}}} R(f) \right] \le \frac{16\, G D R}{\sqrt{N}} \quad (1),$$

N = sample size.

In particular, the RHS of (1) does <u>not</u> depend on the number of parameters, but only on their <u>size</u>.