Ridge regression

$f(x_i) = \vartheta_0 + \vartheta_1 x_{i1} + \dots + \vartheta_m x_{im}$, $i = 1, \dots, n$

$$\begin{bmatrix} x_{11} & \dots & x_{1m} \\ \vdots & & \\ x_{n1} & \dots & x_{nm} \end{bmatrix} \qquad \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}$$

$\rightsquigarrow f(z) = \vartheta_0 + \vartheta_1 z_1 + \dots + \vartheta_m z_m$, $z \in \mathbb{R}^m$

minimise the loss function

$$L_\vartheta = \sum_{i=1}^{n} \left(y_i - \left(\vartheta_0 + \vartheta_1 x_{i1} + \dots + \vartheta_m x_{im} \right) \right)^2 + \lambda \left(\vartheta_1^2 + \dots + \vartheta_m^2 \right)$$

a penalty for large coefficients

$\lambda \geq 0$ - hyperparameter

Lasso:

$$L_\vartheta = \sum_{i=1}^{n} \left(- \text{''} - \right)^2 + \lambda \left(|\vartheta_1| + \dots + |\vartheta_m| \right)$$

Lasso has an advantage that it often gives $\vartheta_j = 0$ for some j's

interpretability of the model is better

For the usual regression the relative size of the features is irrelevant (e.g., x_1 could be $\in [1,2)$, $x_2 \in [100, 200]$, ...), while for Ridge and Lasso it is important and usually it is desirable that the features are of the same order of magnitude

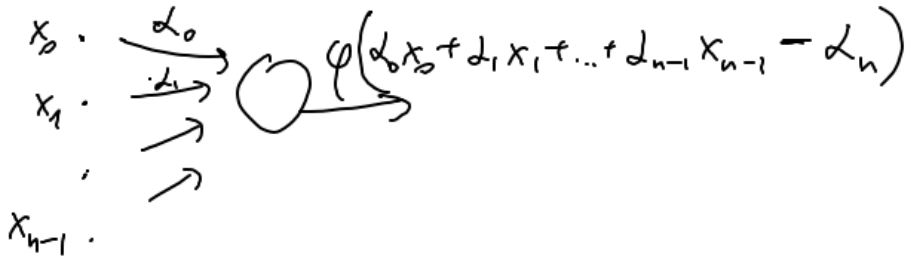
Fix: preprocess the data, for example so that

e.g. each column of X has mean zero and variance 1

or each column of X attains values between 0 and 1

Neural nets

neuron: a function of the form

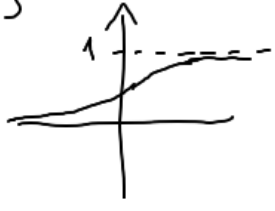


$$\mathbb{R}^n \ni (x_0, x_1, \ldots, x_{n-1}) \xmapsto{\text{neuron}} \varphi(\alpha_0 x_0 + \alpha_1 x_1 + \ldots + \alpha_{n-1} x_{n-1} - \alpha_n) \in \mathbb{R}$$

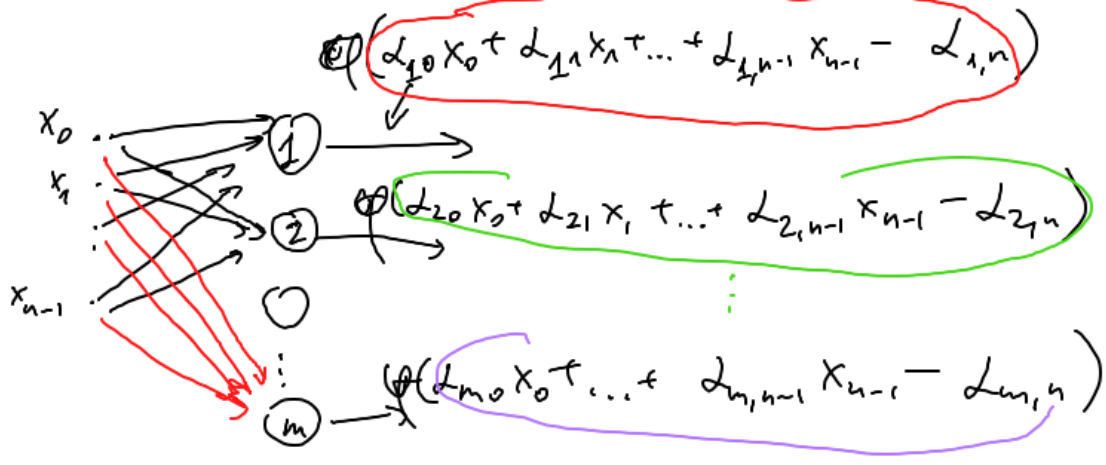
$\alpha_0, \ldots, \alpha_n$ – weights $\in \mathbb{R}$,

φ – activation function, $\varphi: \mathbb{R} \to \mathbb{R}$ increasing

e.g. $\varphi(x) = \dfrac{1}{1+e^{-x}} = \sigma(x)$ sigmoid



$\varphi(x) = \max(x, 0)$ ReLU



$\mathbb{R}^n \ni (x_0, x_1, \ldots, x_{n-1}) \longmapsto \varphi\left(\begin{bmatrix} \alpha_{10} & \alpha_{11} & \ldots & \alpha_{1,n-1} & \alpha_{1n} \\ \vdots & & & & \vdots \\ \alpha_{m0} & \ldots & & \alpha_{m,n-1} & \alpha_{mn} \end{bmatrix} \cdot \begin{bmatrix} x_0 \\ \vdots \\ x_{n-1} \\ -1 \end{bmatrix}\right) = \varphi\left(\begin{bmatrix} \quad \\ \quad \\ \vdots \\ \quad \end{bmatrix}\right) = \begin{bmatrix} \varphi(\quad) \\ \varphi(\quad) \\ \\ \varphi(\quad) \end{bmatrix} \in \mathbb{R}^m$

convention

(dense)

The layer of neurons is a function:

$\mathbb{R}^n \longrightarrow \mathbb{R}^m$ given by this formula

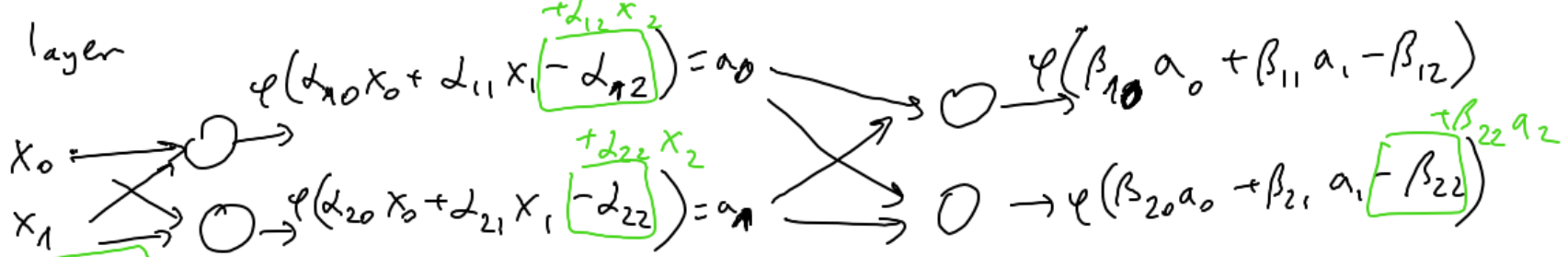
The neural net is a function which is a composition of layers

$$\mathbb{R}^n \ni (x_0, \dots, x_{n-1}) \xrightarrow{\text{layer}} \begin{bmatrix} \varphi(a_{1,0}x_0 + \dots + a_{1,n-1}x_{n-1} - a_{1,n}) \\ \vdots \\ \varphi(a_{m,0}x_0 + \dots + a_{m,n-1}x_{n-1} - a_{m,n}) \end{bmatrix} \xrightarrow{\text{layer}} \in \mathbb{R}^{10}$$

$$\uparrow \mathbb{R}^m \qquad \downarrow \text{layer}$$

$$\vdots$$

For example, let us consider a neural net consisting of 2 layers with 2 neurons in each layer



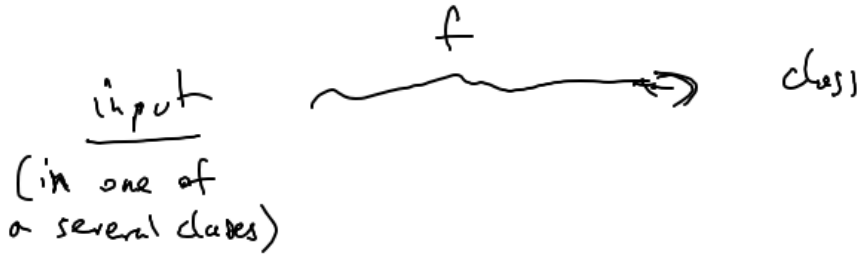
$$\mathbb{R}^2 \ni (x_0,x_1) \xrightarrow{x_2=-1} \Big(\varphi\Big(\beta_{10}\varphi(\alpha_{10}x_0+\alpha_{11}x_1-\alpha_{12})+\beta_{11}\varphi(\alpha_{20}x_0+\alpha_{21}x_1-\alpha_{22})-\beta_{12}\Big), \varphi\Big(\beta_{20}\varphi(\text{---})+\beta_{21}\varphi(\text{---})-\beta_{22}\Big)\Big) \in \mathbb{R}^2$$

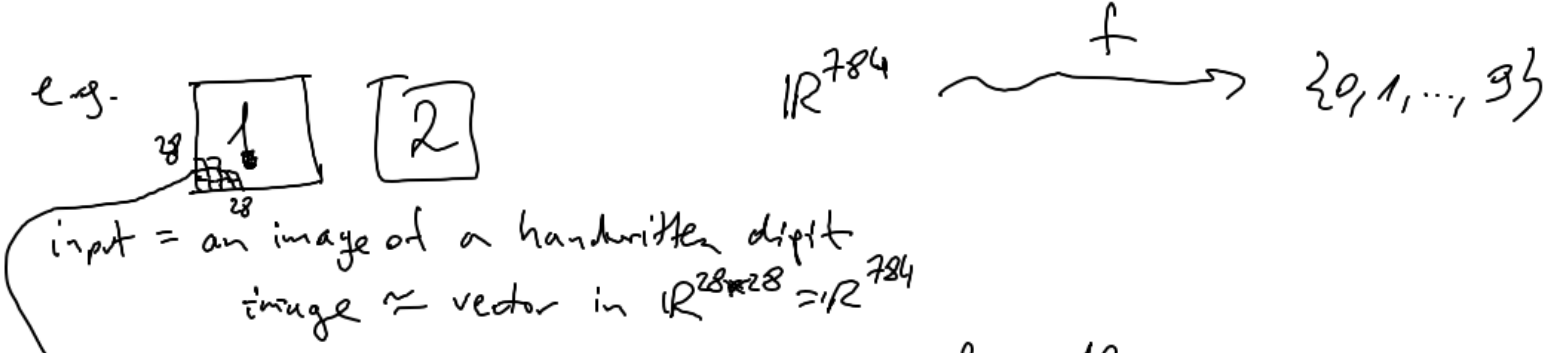
($a_2=-1$)

If φ is ~~linear~~ affine, then the compositions above ↑ are just ~~linear~~ affine functions of (x_0,x_1)

Therefore we do not want to take an affine function as φ.

A common application of neural nets: classification problem





The training procedure:

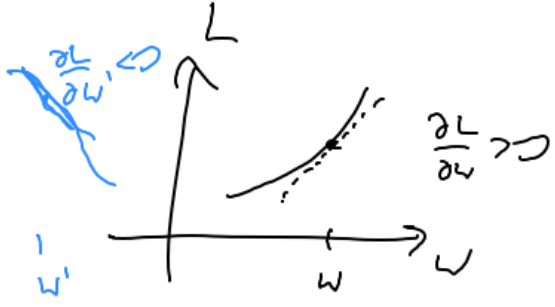
- take some input $x \in \mathbb{R}^{784}$ and feed it to the network:

  we obtain some $y = f(x) \in \mathbb{R}^{10}$

for that input, we know the label, and so we know the target vector:

$$t = (0, 1, 0, \dots, 0) \in \mathbb{R}^{10}$$

↑ this one corresponds to a digit '1'



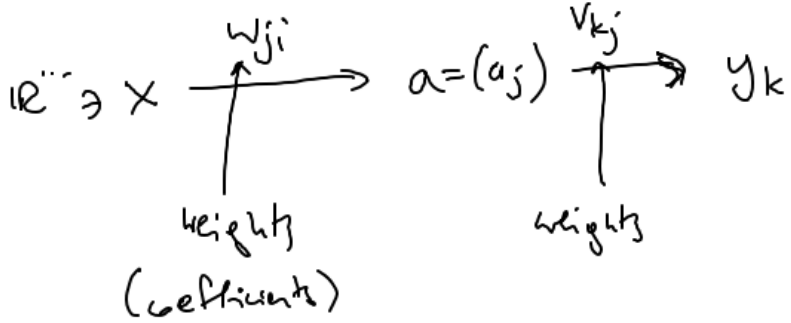
we introduce some loss function, e.g.

$$L(y,t) = \frac{1}{2} \sum_{i=1}^{10} (y_i - t_i)^2$$

Compute $\frac{\partial L}{\partial w}$ for each coefficient w and adjust the coefficients:

$$w := w - c \frac{\partial L}{\partial w}$$

↑ learning const., e.g. 0.01



2 layers,
(w_{ji}) - weights in the 1st layer
(v_{kj}) - 11 — — 2nd — 11 —

$a_j = \varphi\left(\sum_i w_{ji} x_i\right)$ — (with the convention that to x we append a "1" as last coordinate)

$y_k = \varphi\left(\sum_j v_{kj} a_j\right)$ (with a similar convention for a)

(e.g. $k=1,2$, $j=1,2$, $i=0,1,2$ for 2 neurons in each layer)

$x_0 \cdot \quad \circ a_0 \quad 0 \to y_1$
$x_1 \cdot \quad \circ a_1 \quad \circ \to y_2$
$-1 = x_2 \cdot \quad -1 = a_2$

target: (t_k), $L(t,y) = \frac{1}{2}\sum_k (t_k - y_k)^2 = \frac{1}{2}\sum_k \left(t_k - \varphi\left(\sum_j v_{kj} a_j\right)\right)^2$

$v = v_{k_0 j_0}$: $\frac{\partial L}{\partial v} = \sum_k \left(t_k - \varphi\left(\sum_j v_{kj} a_j\right)\right) \cdot \frac{\partial}{\partial v}\left(t_k - \varphi\left(\sum_j v_{kj} a_j\right)\right) = \left(t_{k_0} - \varphi\left(\sum_j v_{k_0 j} a_j\right)\right)\cdot(-1)\, \varphi'\left(\sum_j v_{kj} a_j\right) \cdot a_{j_0}$

$= \frac{\partial}{\partial v}\left(\sum_j v_{kj} a_j\right)$

$\frac{\partial L}{\partial y_{k_0}}$

$$a_j = \varphi\left(\sum_i w_{ji}\, x_i\right)$$

$$y_k = \varphi\left(\sum_j v_{kj}\, a_j\right)$$

$$L(t,y) = \frac{1}{2} \sum_k (t_k - y_k)^2$$

$w = w_{j_0 i_0}$

$$\frac{\partial L}{\partial w} = \sum_k (t_k - y_k)(-1) \cdot \varphi'\left(\sum_j v_{kj}\, a_j\right) \cdot \frac{\partial}{\partial w}\left(\sum_j v_{kj}\, a_j\right) =$$

(the derivative vanishes if $j \neq j_0$)

$$= \sum_k (t_k - y_k)(-1)\ \varphi'\left(\sum_j v_{kj}\, a_j\right) \cdot \sum_j v_{kj}\ \varphi'\left(\sum_i w_{ji}\, x_i\right) \cdot \frac{\partial}{\partial w}\left(\sum_i w_{ji}\, x_i\right) =$$

$$= \sum_k \boxed{(t_k - y_k)(-1)\ \varphi'\left(\sum_j v_{kj}\, a_j\right)} v_{kj_0}\ \varphi'\left(\sum_i w_{j_0 i}\, x_i\right) \cdot x_{i_0}$$