1 Objective functions, neural networks and linear algebra

We share a philosophy about linear algebra: we think basis-free, we write basis-free, but when the chips are down we close the office door and compute with matrices like fury.

Irving Kaplansky

The basic problem is formulated in the context of interpolation of functions. Let f^{obj} be an objective function which models some problem of interest

$$f^{\text{obj}} : \mathbb{R}^m \longrightarrow \mathbb{R}^n. \tag{1.1}$$

Let x_i be an interpolation point in $\mathbb{R}^m$. Let $\varepsilon_i \in \mathbb{R}^n$ which denotes a noise (small enough in a sense which needs to be specified). Noise is an important notion for real data, because real measurements or real data are never perfect. Set

$$y_i = f^{\text{obj}}(x_i) + \varepsilon_i \in \mathbb{R}^n. \tag{1.2}$$

The pair $(x_i, y_i) \in \mathbb{R}^m \times \mathbb{R}^n$ is called an interpolated data with noise $\varepsilon_i \in \mathbb{R}^n$. If $\varepsilon_i = 0$, the interpolation data is noiseless.
The finite collection of interpolation data is the dataset

$$\mathcal{D} = \{(x_i, y_i),\ i = 1, \dots, N\} \subset \mathbb{R}^m \times \mathbb{R}^n. \tag{1.3}$$

Starting from a given dataset $\mathcal{D}$, a general question is to implement in a computer an approximation denoted as f of the objective function f^{obj}. Some evident difficulties for the design of f are that the level of the noise, the curse of dimension (that is m and/or n take high values), the construction of good datasets (good datasets sample the objective function in meaningful regions), the fact that the implementation must be efficient for large $N = \#(\mathcal{D})$, the quality of the reconstructed function. Other aspects will be developed in the notes.

At the end of the chapter, the structure of basic NNs will be defined.

1.1 Notations

Standard linear algebra notations are used. For an arbitrary vector $z \in \mathbb{R}^p$ in a space of arbitrary dimension p, we note its square $|z|^2 = \sum_{i=1}^{p} |z_i|^2 \geq 0$. One has

$|z|^2 = \langle z, z\rangle$ where the scalar product $\langle \cdot, \cdot\rangle$ is defined by

$$\langle a, b\rangle = \sum_{i=1}^{p} a_i b_i, \qquad a, b \in \mathbb{R}^p.$$

Let us consider two matrices $M, N \in \mathcal{M}_{mn}(\mathbb{R})$, two vectors $a, b \in \mathbb{R}^m$ and one vector $c \in \mathbb{R}^n$ such that $b = Mc$. The contraction of matrices is noted with equivalent notations

$$\langle M, N\rangle = M : N = \sum_{i=1}^{m}\sum_{j=1}^{n} m_{ij} n_{ij} = M^t : N^t = \left\langle M^t, N^t \right\rangle. \tag{1.4}$$

The tensorial product of two vectors is noted $a \otimes c = ac^t \in \mathcal{M}_{mn}(\mathbb{R})$. One has

$$\langle a, b\rangle = \langle a, Mc\rangle = \langle a \otimes c, M\rangle = a \otimes c : M \tag{1.5}$$

A norm for matrices writes

$$\|W\| = \sup_{\|y\|=1, \|x\|=1} \langle y, Mx\rangle = \sup_{\|y\|=1, \|x\|=1} \langle y \otimes x : M\rangle, \tag{1.6}$$

which has the property that $\|AB\| \leq \|A\|\ \|B\|$.

Lemma 1.1.1. *Let $A \in \mathcal{M}_{pq}(\mathbb{R})$, $B \in \mathcal{M}_{qr}(\mathbb{R})$ and $C \in \mathcal{M}_{pr}(\mathbb{R})$. One has the formula*

$$AB : C = \langle AB, C\rangle = \left\langle A, CB^t \right\rangle = A : CB^t.$$

Proof. One can rely on the following property. Let $M, N \in \mathcal{M}_{mn}(\mathbb{R})$ by two matrices. With the notation of the trace of a square matrix which is the sum of its diagonal elements, one has $M : N = \mathrm{tr}(M^t N)$. One obtains

$$AB : C = \mathrm{tr}((AB)^t C) = \mathrm{tr}(B^t A^t C).$$

A known result in linear algebra is that matrices under the trace operator commute. So

$$AB : C = \mathrm{tr}(A^t C B^t) = \mathrm{tr}(A^t (CB^t)) = A : CB^t.$$

□

Matrices with more than two axis of coefficients are called **tensors**. They will be introduced in Section 1.3.3 about Convolutive Neural Networks.

1.2 The least squares method

A first ingredient is the least squares method . Consider a linear function f

$$\begin{aligned} f: \quad \mathbb{R}^m &\longrightarrow \mathbb{R}^n, \\ x &\longmapsto f(x) = Wx + b \end{aligned} \tag{1.7}$$

where the parameters are $W \in \mathcal{M}_{nm}(\mathbb{R})$ which is called the **matrix of weights** and $b \in \mathbb{R}^n$ which is called the **bias** or the offset. Consider now the function

$$\begin{aligned} J: \quad \mathcal{M}_{nm}(\mathbb{R}) \times \mathbb{R}^n &\longrightarrow \mathbb{R}, \\ (W, b) &\longmapsto \sum_{(x,y)\in\mathcal{D}} |Wx + b - y|^2 . \end{aligned} \tag{1.8}$$

The space of parameters is $\mathcal{Q} = \mathcal{M}_{nm}(\mathbb{R}) \times \mathbb{R}^n = \mathcal{M}_{n,m+1}(\mathbb{R})$. A way to minimize the difference between f^{obj} which is encoded in the dataset $\mathcal{D}$ and the function f is to determine the parameters which minimize J. That is one considers an optimal value $(W_*, b_*) \in Q$ **in the sense of least squares**

$$J(W_*, b_*) \le J(W, b) \qquad \forall (W, b) \in Q. \tag{1.9}$$

Lemma 1.2.1. *There exists an optimal value $(W_*, b_*) \in Q$*

$$(W_*, b_*) = \underset{(W,b)\in Q}{\operatorname{argmin}} \, J(W, b).$$

If the vectors $\left\{ \begin{pmatrix} x_i \\ 1 \end{pmatrix} \right\}_{i=1}^{N}$ span the whole space $\mathbb{R}^{m+1}$, then the optimal value is unique.

Remark 1.2.2. *For many problems in data science, one has that $\#(\mathcal{D}) \gg 1$, so the last assumption is reasonable.*

Remark 1.2.3. *The minimization problem (1.9) can be decoupled into n simpler scalar least square problems (one scalar problem per entry in the inputs). On the contrary in the proof below, the approach is more global which has the advantage to introduce some basic tools that will be reused later.*

Proof. The optimal matrix of weights and bias are constructed explicitly. The proof is split in smaller steps.

- In the first step, one freezes $b = 0$ and look for the optimal value of W.
- In the second step, both b and W can take any value.

– In the third comment, this hypothesis of linear independence of the vectors x_i is relaxed .

• **First step**. Consider the function $J(W) = \frac{1}{2}\sum_{i=1}^N |Wx_i - y_i|^2$ where the factor $\frac{1}{2}$ is just for convenience. Set $h(\varepsilon) = J(W + \varepsilon Z)$ where $Z \neq 0$ is an arbitrary non zero matrix $Z \in \mathcal{M}_{mn}(\mathbb{R})$. At a local minimum W_*, the derivative of h vanishes whatever Z is. One has therefore

$$\lim_{\varepsilon} \frac{h(\varepsilon) - h(0)}{\varepsilon} = \sum_{i=1}^N \langle W_* x_i - y_i, Zx_i \rangle = 0, \tag{1.10}$$

The scalar product can be rewritten as a matrix-matrix or vector-vector multiplication as

$$\sum_{i=1}^N \langle Wx_i - y_i, Zx_i \rangle = \left\langle \sum_{i=1}^N x_i \otimes (Wx_i - y_i), Z \right\rangle = 0.$$

Since Z is arbitrary one gets $\sum_{i=1}^N x_i \otimes (W_* x_i - y_i) = 0$. More transformations yield

$$\sum_{i=1}^N x_i (W_* x_i - y_i)^t = \sum_{i=1}^N x_i x_i^t W^t - x_i y_i^t = \left(\sum_{i=1}^N x_i \otimes x_i \right) W^t - \sum_{i=1}^N x_i \otimes y_i = 0.$$

This is rewritten after transposition as

$$WM = B, \quad M = \sum_{i=1}^N x_i \otimes x_i, \quad B = \sum_{i=1}^N y_i \otimes x_i. \tag{1.11}$$

As a consequence of the last assumption of the Theorem, let us assume the linear independence of the vectors x_i. It yields that the matrix $\left(\sum_{i=1}^N x_i \otimes x_i\right)$ is invertible. One gets

$$W = BM^{-1} \in \mathcal{M}_{nm}(\mathbb{R}). \tag{1.12}$$

The function $h(\varepsilon) = J(W + \varepsilon Z)$ is a polynomial with respect to ε, with degree $\deg(h) \leq 2$. Therefore

$$h(\varepsilon) = h(0) + h'(0)\varepsilon + \frac{1}{2}h''(0)\varepsilon^2 = h(0) + \frac{1}{2}\left(\sum_{i=1}^N |Zx_i|^2 \right) \varepsilon^2 \geq h(0) \tag{1.13}$$

because $h'(0) = 0$ by definition of the matrix W. That is $J(W + \varepsilon Z) \geq J(W)$ for all possible ε and Z. So W is indeed a minimum argument of the function J.

• **Second step.** Now b can be non zero. Let us introduce the extended notation

$$\widehat{W} = (W \mid b) \in \mathcal{M}_{n,m+1}(\mathbb{R}) \tag{1.14}$$

and the extended input

$$\widehat{x}_i = \begin{pmatrix} x_i \\ 1 \end{pmatrix}. \tag{1.15}$$

By construction $\widehat{W}\widehat{x}_i = Wx_i + b$. Therefore one can define the extended function

$$\widehat{J}(\widehat{W}) = \frac{1}{2}\sum_{i=1}^{N} \left|\widehat{W}\widehat{x}_i - y_i\right|^2 = \frac{1}{2}\sum_{i=1}^{N} |Wx_i + b - y_i|^2. \tag{1.16}$$

Under the independence condition of the Theorem, the first step yields the formula for the optimal solution

$$\widehat{W} = \widehat{B}\widehat{M}^{-1} = \left(\sum_{i=1}^{N} y_i \otimes \widehat{x}_i\right)\left(\sum_{i=1}^{N} \widehat{x}_i \otimes \widehat{x}_i\right)^{-1} \in \mathcal{M}_{n,m+1}(\mathbb{R}). \tag{1.17}$$

• **Third step.** So far, the proof assumed the linear independence assumption, which is useful to obtain the invertibility of the matrix M in (1.11)-(1.12) and of the matrix $\widehat{M}$ in (1.17). Let us show that the equation (1.11) has at least one solution, even if M is not invertible.
– One considers the linear operator

$$\begin{array}{cccc} \mathcal{L}: & \mathcal{M}_{mn}(\mathbb{R}) & \to & \mathcal{M}_{mn}(\mathbb{R}) \\ & W & \mapsto & W = WM. \end{array}$$

One desires to show that $B \in \mathrm{ran}(\mathcal{L})$, where $\mathrm{ran}(\mathcal{L})$ is the range of the operator $\mathcal{L}$.
– An fundamental linear algebra property is $\mathrm{ran}(\mathcal{L}) = \ker(\mathcal{L}^t)^\perp$, where the orthogonality is defined with respect to the contraction of matrices (the operator $\mathcal{L}^t$ is the symmetric or adjoint operator of $\mathcal{L}$). With the help of Lemma 1.2.1, one obtains

$$\langle \mathcal{L}W : Z\rangle = \langle WM : Z\rangle = \left\langle W : ZM^t\right\rangle = \langle W : ZM\rangle = \langle W : \mathcal{L}Z\rangle \quad \forall\, W, Z.$$

It shows that $\mathcal{L} = \mathcal{L}^t$ is a symmetric operator, so $\ker(\mathcal{L}^t) = \ker(\mathcal{L})$.
– Next one studies

$$\ker(\mathcal{L}) = \left\{ W \in \mathcal{M}_{mn}(\mathbb{R}) \text{ where } W\sum_{i=1}^{N} \mathbf{x}_i \otimes \mathbf{x}_i = 0 \right\}.$$

For $W \in \ker(\mathcal{L})$, one has $0 = \left\langle W\sum_{i=1}^N \mathbf{x}_i \otimes \mathbf{x}_i : W\right\rangle = \sum_{i=1}^N \langle W\mathbf{x}_i \otimes \mathbf{x}_i : W\rangle$ $= \sum_{i=1}^N \left\langle W\mathbf{x}_i\mathbf{x}_i^t : W\right\rangle = \sum_{i=1}^N \langle W\mathbf{x}_i : Ww_i\rangle = \sum_{i=1}^N |W\mathbf{x}_i|^2$. So $Wx_i = 0$ for all $1 \le i \le N$. Reciprocally, if $Wx_i = 0$ for all $1 \le i \le N$, then $W\sum_{i=1}^N \mathbf{x}_i \otimes \mathbf{x}_i = \sum_{i=1}^N Wx_ix_i^t = 0$. Therefore one has also

$$\ker(\mathcal{L}) = \{W \in \mathcal{M}_{mn}(\mathbb{R}) \text{ where } Wx_i = 0 \text{ for all } 1 \le i \le N\}. \tag{1.18}$$

– Finally one takes $B = \sum_{i=1}^{N} \mathbf{y}_i \otimes \mathbf{x}_i$. One has

$$\langle W, B\rangle = W : \sum_{i=1}^{N} \mathbf{y}_i \mathbf{x}_i^t = \sum_{i=1}^{N} W : \mathbf{y}_i \mathbf{x}_i^t = \sum_{i=1}^{N} W x_i : \mathbf{y}_i = 0.$$

So $B \in \ker(\mathcal{L})^{\perp}$. It shows the existence of a matrix solution to $WM = B$. □

The matrix of weights W and the bias b are important tools in Machine Learning in combination with two more ingredients. The first one is **recursivity**, the second one is **non linearity**. Historically, non linearity was introduced in 1958 by Rosenblatt [87] for the construction of the **perceptron**. Recursivity was introduced later with the progresses of computational sciences. However the mathematical structure is perhaps better highlighted with the presentation in the reverse order, that recursivity before non linearity.

1.2.1 Recursivity

Let us first describe recursivity . One takes a sequence of integers

$$(a_0, a_1, a_2, \dots, a_p, a_{p+1}) \in \mathbb{N}^{p+2} \tag{1.19}$$

such that the first one is $a_0 = m$ and the last one is $a_{p+1} = n$, and a sequence of matrices of weights and biases

$$W_r \in \mathcal{M}_{a_{r+1}, a_r}(\mathbb{R}) \quad \text{and } b_r \in \mathbb{R}^{a_{r+1}}, \qquad 0 \le r \le p. \tag{1.20}$$

One defines a sequence of linear functions

$$\begin{aligned} f_r : \quad \mathbb{R}^{a_r} &\longrightarrow \mathbb{R}^{a_{r+1}}, \\ x_r &\longmapsto f_r(x_r) = W_r x_r + b_r. \end{aligned} \tag{1.21}$$

To introduce recursivity, a first but naive possibility is to consider the function

$$\begin{aligned} f : \quad \mathbb{R}^{m} &\longrightarrow \mathbb{R}^{n}, \\ x &\longmapsto f(x) = f_p(f_{p-1}(\dots(f_2(f_1(f_0(x)))\dots)). \end{aligned} \tag{1.22}$$

This is also noted as a composition

$$f = f_p \circ f_{p-1} \dots f_2 \circ f_1 \circ f_0. \tag{1.23}$$

One could think that, just because there is more degrees of freedom to optimize, then this formula is potentially more accurate. This is just the opposite, and actually the result is as accurate as with just one linear function.

Lemma 1.2.4. *The function (1.23) is equal to the simple affine function (1.7) with*

$$W = W_p W_{p-1} \dots W_2 W_1 W_0 \in \mathcal{M}_{nm}(\mathbb{R}) \quad \textit{and} \quad b = b_p + \sum_{r=0}^{p-1} W_p \dots W_{r+1} b_r \in \mathbb{R}^n.$$

Proof. The claim, evident for $p = 0$, is shown for $p \geq 1$ by recurrence reasoning. □

Nevertheless, with non linearity, recursivity opens wonderful new possibilities.

1.2.2 Non linearity

In order to enrich the construction and to break the linear degeneracy explained in Lemma 1.2.4, one introduces a non linear ingredient which is called an **activation function** . There is whole zoo of different activation functions already implemented in softwares. Following the standard notations, we consider either a sigmoid function or the rectified linear unit function and its variants. These activation functions were considered since the beginning of the discipline [20].

Definition 1.2.5. *A continuously differentiable function $S \in C^1(\mathbb{R})$ is called a sigmoid if $S' > 0$, if it has the limit value 0 at $-\infty$ and if it has the value 1 at $+\infty$.*

Another widely use notation for a sigmoid function is σ, but in this text σ will be reserved for permutions in finite sets. By definition $0 = S(-\infty) < S(x) < S(+\infty) = 1$ for all $x \in \mathbb{R}$.

Definition 1.2.6. *The rectified linear unit function (ReLU) $R \in C^0(\mathbb{R})$ is defined by*

$$R(x) = \frac{1}{2}(x + |x|) = x_+ = \max(x, 0) = \begin{cases} 0 & \textit{for} \quad x \leq 0, \\ x & \textit{for} \quad 0 \leq x. \end{cases}$$

Definition 1.2.7. *The thresholded rectified linear unit (TReLU) function $T \in C^0(\mathbb{R})$ is*

$$T(x) = \min(R(x), 1) \in [0, 1]. \tag{1.24}$$

Definition 1.2.8. *Let $\alpha \in \mathbb{R}$ be a given parameter (a standard choice in softwares is $\alpha = \frac{1}{2}$). The leaky rectified linear unit (LReLU) function $L \in C^0(\mathbb{R})$ is*

$$L(x) = \alpha x \textit{ for } x \leq 0 \textit{ and } L(x) = x \textit{ for } x \geq 0.$$

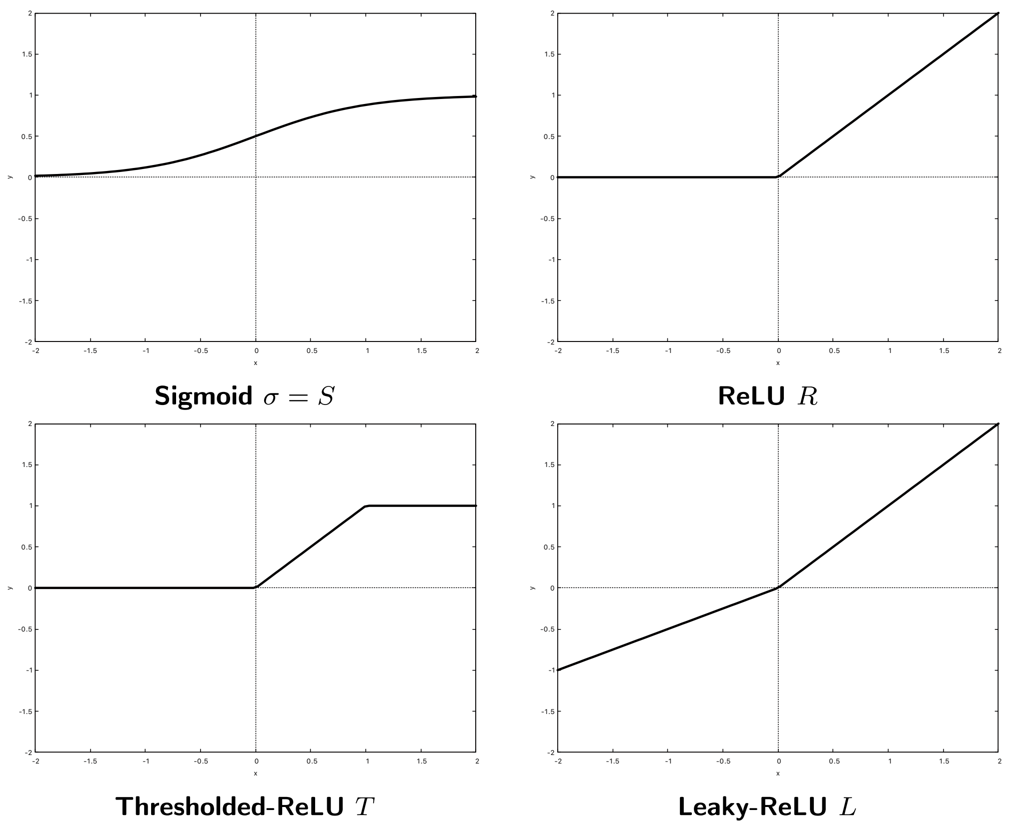

Fig. 1.1: Activation functions.

Any activation functions can be applied component wise to all entries of a vector. For $X = (x_1, \ldots, x_m) \in \mathbb{R}^m$, we will write

$$h(X) = (h(x_1), \ldots, h(x_m)) \in \mathbb{R}^m \quad \text{where } h = R, S, T \text{ or } L.$$

Provided $|\alpha| \neq 1$, the ReLU function is equivalent to the leaky-ReLU.

Lemma 1.2.9. *One has the equalities $L(x) = R(x) - \alpha R(-x)$ and*

$$R(x) = \frac{1}{1-\alpha^2} L(x) + \frac{\alpha}{1-\alpha^2} L(-x).$$

Proof. Evident by expansion. $\square$

If $\alpha = 1$, then the leaky-ReLU function is the linear function $x \mapsto x$ which is in $C^1(\mathbb{R})$, so it is evident that R cannot be obtained from L in this case. If $\alpha = -1$, then the leaky-ReLU function is the absolute-value function $x \mapsto |x|$.

1.2.3 Functions and Neural Networks

The standard way to introduce **non linearity** in the design of function is to intertwinn activation functions with linear functions. Instead of (1.23), one considers

$$f = f_p \circ g_{p-1} \cdots \circ g_2 \circ g_1 \circ g_0, \qquad \text{with } g_r = h_r \circ f_r \text{ where } h_r = S,\ R,\ T \text{ or } L. \tag{1.25}$$

The activation function can change from one layer to the other, this the reason of the index in h_r.

Definition 1.2.10. *We say a function is represented as a Neural Network if it can be written within the structure (1.25). More precisely, such a structure is call a Feedforward Deep Network in the literature [34][Chapter 6].*
The vector $X \in \mathbb{R}^m$ is **the variable** *or* **the input** *. The vector $Y \in \mathbb{R}^n$ is* **the output***. All weights and biaises in W are* **the parameters***. All other layers of size $a_1, \ldots, a_{p-1}$ are called the* **hidden layers** *(which means that p is the number of hidden layers with our notations). The number of hidden layers, the type of activation functions and all other parameters used to control the architecture of the Neural Network are called* **the hyper-parameters***.*

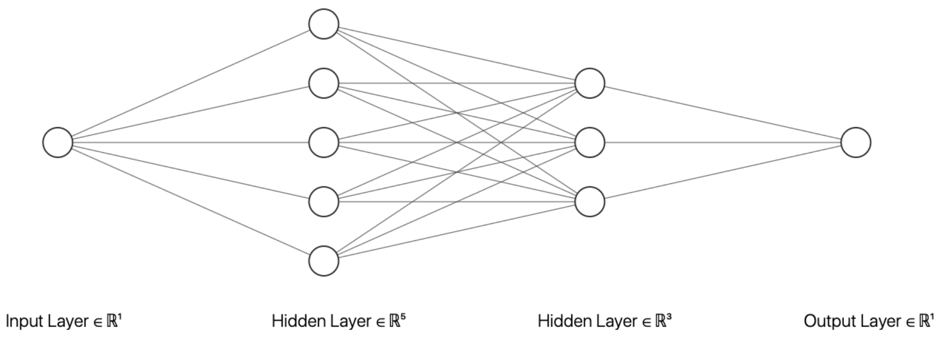

Fig. 1.2: Graph structure of a neural network with $p = 2$ dense hidden layers, one input layer and one output layer which have the same dimension $m = 1$.

It is valuable to present the notation with graphs, as in http://alexlenail.me/NN-SVG/index.html. The input layer is $\mathbb{R}^m$. The output layer is $\mathbb{R}^n$. The graph, to be read from the left to the right, explains the computational flowchart. The hidden layers in graph of Figure 1.2 are called **dense layers** because all entries of the matrices of weights are degrees of freedom.

Definition 1.2.11. *Let $f_0 : \mathbb{R}^m \longrightarrow \mathbb{R}$ be affine. Then $g = S \circ f_0$ is called a Rosenblatt perceptron (1958) with m inputs, see [87].*

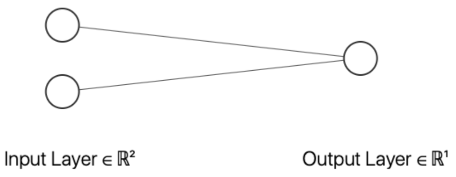

Fig. 1.3: Graph structure of the perceptron. The input layer has dimension $m = 2$. The output layer has dimension $n = 1$. There is no hidden layer.

From now on all matrices of weights and vectors of biases will be denoted with one letter W. It is a super-matrix of weights and biaises

$$W = \big(W_p, b_p, W_{p-1}, b_{p-1}, \dots, W_0, b_0\big) \in \mathcal{M} \tag{1.26}$$

where the dimension is $\dim(\mathcal{M}) = \sum_{r=0}^{p}(a_r + 1)a_{r+1}$. One obtains the function

$$f : \quad \mathbb{R}^m \times \mathcal{M} \quad \longrightarrow \mathbb{R}^n. \tag{1.27}$$

If we do not need to indicate the dependence with respect to the parameter W, the same function (1.27) will denoted as

$$f : \quad \mathbb{R}^m \quad \longrightarrow \mathbb{R}^n. \tag{1.28}$$

As an example, let us run Algorithm 1[1] which is the declaration in Python-Keras-Tensoflow of a simple NN with 4 hidden layers. The width of the hidden layers is 3-3-3-4. The parameters of the NN are visible in the last column in Table 1.1. The number of parameters between layer r and layer $r+1$ is equal to $(a_r + 1)a_{r+1}$ where a_r is the width of layer r and a_{r+1} is the width of layer $r+1$. The exception is between layer 2 and layer 3, because the biais is removed with the command `use_bias=False`: in this case the number of parameters is equal to $a_2 a_3 = 9$. The `None` in the second column corresponds to the number of elements in the dataset, which is unspecified at this stage. Line 10 creates the summary reported in Table 1.1. The

1 https://www.ljll.math.upmc.fr/despres/BD_fichiers/mnist_summary.py

```
Python-Keras-Tensorflow Initialization
input_shape=1
model1=Sequential()
model1.add(Dense(3,input_dim=input_shape,name="hidden1",
                                   use_bias=True, activation='relu'))
model1.add(Dense(3,name="hidden2",use_bias=True, activation='relu'))
model1.add(Dense(3,name="hidden3",use_bias=False, activation='relu'))
model1.add(Dense(4,name="hidden4",use_bias=True, activation='relu'))
model1.add(Dense(1,name="out",    use_bias=True, activation='linear'))
model1.summary()
```

Algorithm 1: Dense NN with 4 hidden layers and width 1-3-3-3-4-1.

```
_________________________________________________________________
 Layer (type)                 Output Shape              Param #
=================================================================
couche_entree (Dense)        (None, 3)                 6
_________________________________________________________________
couche_hidden_1 (Dense)      (None, 3)                 12
_________________________________________________________________
couche_hidden_2 (Dense)      (None, 3)                 9
_________________________________________________________________
couche_hidden_3 (Dense)      (None, 4)                 16
_________________________________________________________________
couche_sortie (Dense)        (None, 1)                 5
=================================================================
Total params: 51
```

Tab. 1.1: Parameters of the dense NN specified in Algorithm 1.

It remains to fit the parameters, that is to construct the best matrix $W \in \mathcal{M}$. A generalization of the least squares procedure considers the minimization problem

$$W_* = \underset{W}{\operatorname{argmin}} \sum_{(x,y)\in\mathcal{D}} |f(x,W) - y|^2$$

where the dataset $\mathcal{D}$ must be specified. All questions considered in this text are variations around this theme. We immediately notice the fact that the weights cannot be unique.

Lemma 1.2.12. *The function f is unchanged after permutations of the neurons of the hidden dense layers. The number of such permutations is*

$$\#(\textit{permutations}) = a_1! \times \cdots \times a_p!.$$

Proof. Permutations of neurons keeping the connections the same (that is permuting the order with which one stores the coefficients in W_r) and keeping the biases the same (that is permuting as well the order with which one stores the coefficients

in b_r) do not change the result.
The formulas can be made explicite. Let $\Pi_r \in \mathcal{M}_{a_r}(\mathbb{R})$ a permutation matrix for the layer of dimension a_r. The coefficients of a permutation matrix can be 0 or 1 only, and a permutation matrix is unitary $\Pi_r^t \Pi_r = I_r$. The number of permutation matrices is equal to the number of permutations of a_r elements, that is $a_r!$. One has the commutation formula

$$\Pi_r \circ h = h \circ \Pi_r \qquad \text{where } h = S, R, T \text{ or } L,$$

because the permutation of modified values is equal to the modified permuted values (h is any activation function). Then the function f (1.25) with the parameter W in (1.26) is the same as the function $\widetilde{f}$ with the parameter

$$\widetilde{W} = \Big(W_p \Pi_p^t, b_p, \ldots, \Pi_{r+1} W_r \Pi_r^t, \Pi_{r+1} b_r, \ldots, \Pi_1 W_0, \Pi_1 b_0 \Big) \in \mathcal{M} \tag{1.29}$$

For example for $p = 2$, one checks that

$$\begin{array}{rcl} \widetilde{f}(x) & = & W_2 \Pi_2^t \circ h_2 \big(\Pi_2 W_1 \Pi_1^t \circ h_1 \left(\Pi_1 W_0 x + \Pi_1 b_0 \right) + \Pi_2 b_1 \big) + b_2 \\ & = & W_2 \Pi_2^t \circ h_2 \circ \Pi_2 \big(W_1 \Pi_1^t \circ h_1 \circ \Pi_1 \left(W_0 x + b_0 \right) + b_1 \big) + b_2 \\ & = & W_2 \Pi_2^t \circ \Pi_2 \circ h_2 \big(W_1 \Pi_1^t \circ \Pi_1 \circ h_1 \left(W_0 x + b_0 \right) + b_1 \big) + b_2 \\ & = & W_2 \circ h_2 \left(W_1 \circ h_1 \left(W_0 x + b_0 \right) + b_1 \right) + b_2 = f(x). \end{array}$$

The proof is the same for all $p \geq 1$.
The number of different $\widetilde{W}$ (1.29) is the product of the number of permutations. □

Remark 1.2.13. *The weights and biases can be changed without changing the function. It reveals a profound lack of uniqueness with respect to the weight W. This is of course detrimental to ML (the algorithmic side of NNs).*

Consider a a cost function $J(W)$ which is convex with only one global minimum, as in Lemma 1.2.1. Assume also that the function comes from a NN with the property that $J(W) = J(\widetilde{W})$ for all $\widetilde{W}$ defined by (1.29). Since the minimum is unique, then $W = \widetilde{W}$ for all possible $\widetilde{W}$. It shows that many coefficients in the matrix W are the same, and probably the structure of the NN can be simplified a lot.

Remark 1.2.14. *In summary one has the alternative which is the consequence of Lemma 1.2.12 with one hidden layer (or more): either the cost function (1.2.12) is not strictly convex, or the function $J(W) = \sum_{(x,y)\in\mathcal{D}} |f(x,W) - y|^2$ could have been encoded with a more economical structure with less neurons.*

1.3 Logistic function, cross-entropy, classification, CNNs

A recent great success of Neural Networks is the **classification** of images [55, 34]. Typically one has many images, each of them is a vector of numerical values (the pixels), and one desires to establish a classification of these images. Classification means that a certain number of images are of the first type, another number of images are of the second type, and so one and so forth. Since one desires to recognize the images knowing a priori their type, this is called **supervised learning**.

Very naturally, the fact that a certain image $x \in \mathbb{R}^m$ is of a certain type is expressed by saying that the probability of the considered type knowing x is 1 and that the probability of other types knowing x is 0. So in classification, the output y is a discrete probability, that is

$$y \in \mathcal{P}_n = \left\{ y \in [0,1]^n, y_1 + \dots y_n = 1 \right\} \underset{\neq}{\subset} [0,1]^n \subset \mathbb{R}^n.$$

The set $\mathcal{P}_n$ will be a discrete probability space, where $n > 1$ is the number of different classes. It is possible to implement a transformation $f : x \mapsto y$ in the least square sense where the output $y \in \mathcal{R}^n$ is seen as a real vector without any particular structure. However, not only it might be quite naive because least square methods do not recognize discrete probabilities, but the numerical results in the literature for classification [34, 55] are indubitably in favor of using the information that the output can be formulated as a discrete probability $y \in \mathcal{P}_n$. The statistical foundations of this methodology are described in the works of Vapnik [99, 100].

We review hereafter the Logistic function and the cross-entropy function which are appropriate to manipulate discrete probabilities, then we establish that classification with no hidden layer is mathematically well posed, and finally we introduce Convolutive Neural Networks (CNN) which are heuristically the best for efficient classification of images. Convolutive Neural Networks reintroduce the evident fact that images are more matrices than vectors.

1.3.1 Logistic function, cross-entropy

Consider a function $f : \mathbb{R}^m \to \mathbb{R}^n$, modeled through a representation like (1.25). The question is to postprocess $z = f(x) \in \mathbb{R}^n$ such that the result is a discrete probability.

Definition 1.3.1. *For any $n > 1$, the logistic function is*

$$\begin{array}{llll} \text{Logis}: & \mathbb{R}^n & \to & \mathcal{P}_n \\ & z & \mapsto & p = (p_1, \dots, p_n), \qquad \textit{with } p_r = \frac{e^{z_r}}{\sum_{s=1}^n e^{z_s}}. \end{array} \tag{1.30}$$

The logistic function transforms any vector into a discrete probability. Note that the values on the boundary of $\mathcal{P}_n$ are not reached by the logistic function

$$\text{Logis}(\mathbb{R}^n) = \{y \in \mathcal{P}_n,\ 0 < y_i < 1 \text{ for all } i\} \subsetneq \mathcal{P}_n.$$

The objective of classification is to reconstruct true or a priori discrete probabilities from the data, so the fact that the Logistic function is not able generate perfect discrete probabilities might appear as a fundamental obstruction. It is absolutely not the case. To illustrate let us take the example where $y_1^{\text{true}} = 1$ and $y_i^{\text{true}} = 0$ for $i \geq 2$. It is sufficient that y_1^{predict} is sufficiently close to one (for example $0.7 \leq y_1 < 1$) to decide that the input belongs to the first class. In other words, it is sufficient for classification applications to post-process the result of the Logistic function.

Let us denote the vector

$$e = (1, \dots, 1) \in \mathbb{R}^n.$$

Lemma 1.3.2. *One has* $\text{Logis}(z + \lambda e) = \text{Logis}(z)$ *for all* $z \in \mathbb{R}^n$ *and* $\lambda \in \mathbb{R}$.

Proof. Indeed $\frac{e^{z_r+\lambda}}{\sum_{s=1}^n e^{z_s+\lambda}} = \frac{e^{z_r}}{\sum_{s=1}^n e^{z_s}}$. □

Definition 1.3.3. *For any* $n > 1$, *the cross-entropy function is*

$$\begin{array}{llll} \text{Cross}: & \mathbb{R}^n \times \mathcal{P}_n & \to & \mathbb{R}^+ \\ & (z, y) & \mapsto & -\sum_{s=1}^n \log(p_s) y_s, \qquad \textit{with } p = \text{Logistic}(z). \end{array}$$

By definition it is non negative

$$\text{Cross}(z, y) \geq 0 \textit{ for all } (z, y) \in \mathbb{R}^n \times \mathcal{P}_n$$

and $\text{Cross}(0, y) = \sum_{s=1}^n (\log n\ y_s) = \log n > 0$ *for all* $y \in \mathcal{P}_n$.

The cross-entropy is related the Kullback-Leibler divergence, also denoted as the relative entropy

$$\text{KL}(y, p) = \sum_{s=1}^n y_s \log \frac{y_s}{p_s} = \sum_{s=1}^n y_s \log y_s - \text{Cross}(z, y).$$

One has the formula

$$\text{Cross}(z, y) = -\langle z, y \rangle + \log \left(\sum_{s=1}^n e^{z_s} \right). \tag{1.31}$$

The cross-entropy function can be analyzed in the context of convex analysis theory in finite dimension [1, 44].

Lemma 1.3.4. *The cross-entropy is convex with respect to its first variable $z \in \mathbb{R}^n$, but is not strictly convex. For $y \in \mathcal{P}_n \bigcap (0,1)^n$, there exists a minimum argument $z_* \in \mathbb{R}^n$*

$$\mathrm{Cross}(z_*, y) \le \mathrm{Cross}(z, y) \text{ for all } z \in \mathbb{R}^n$$

which is characterized by $\mathrm{Logis}(z_*) = y$. *The minimum argument z_* is given by an explicit formula and is unique up to the addition of λe for any $\lambda \in \mathbb{R}$.*

Proof. With (1.31), one has for $z, y \in \mathbb{R}^n$

$$\nabla_z \mathrm{Cross}(z, y) = \mathrm{Logis}(z) - y. \tag{1.32}$$

We will use the notation $p = \mathrm{Logis}(z) \in \mathcal{P}_n$ and $d_r = \frac{e^{z_r}}{\sum_{s=1}^n e^{z_s}}$ for $1 \le r \le n$. The rest of the proof is as follows.

• The Hessian matrix of the cross-entropy is the symmetric matrix $\nabla_z^2 \mathrm{Cross}(z, y) = \nabla_z \mathrm{logis}(z)$, that is after calculations

$$\nabla_z^2 \mathrm{Cross}(z, y) = \mathrm{diag}(p_r) - (p_r p_s)_{1 \le r,s \le n} \in \mathcal{M}_n(\mathbb{R}).$$

Using that $\sum_r p_r = 1$, one obtains $\nabla_z^2 \mathrm{Cross}(z, y) = \mathrm{diag}\left(p_r \sum_{s \ne r} p_s\right) - (p_r p_s)_{1 \le r \ne s \le n}$. Let $d = (d_r)_{1 \le r \le n} \in \mathbb{R}^n$ be an arbitrary test direction or test vector. Then

$$\left\langle \nabla_z^2 \mathrm{Cross}(z, y) d, d \right\rangle = \sum_{1 \le r \ne s \le n} p_r p_r (d_r - d_s)^2 \ge 0. \tag{1.33}$$

Since it holds for all $d \in \mathbb{R}^n$, the Hessian is non negative, and so the cross-entropy is convex with respect to $z \in \mathbb{R}^n$.

• At a minimum, the gradient vanishes. It writes $\frac{e^{z_r}}{\mu} = y_r$ with $\mu = \sum_{s=1}^n e^{z_s} > 0$. So $z_r = \log(y_r \mu)$. It is correctly defined only for $y_r > 0$: this is the reason of the restriction $y \in (0,1)^n$. One can write $z_r = \log y_r + \lambda$ with $\lambda = \log \mu$, which is of course in accordance with Lemma 1.3.2. Taking $\mu = 1$, it yields

$$z_* = (\log y_r)_{1 \le r \le n} \,. \tag{1.34}$$

Since the function is convex and the gradient vanishes at z_*, it means that z_* is a minimum given by an explicit formula.

• Finally $\mathrm{Cross}(z + \lambda e, y) = \mathrm{Cross}(z, y)$ for all $\lambda \in \mathbb{R}$ which shows the last item of the claim. □

1.3.2 Classification

An implementation the logistic regression with affine reconstruction over the dataset $\mathcal{D}$ writes

$$J(W) = \sum_{i=1}^{N} \mathrm{Cross}(Wx_i + b, y_i) \geq 0. \tag{1.35}$$

There is no hidden layer and no activation function in this representation. With the convention $\widehat{W}\widehat{x}_i = Wx_i + b$ already introduced in (1.16), it is also noted $\widehat{J}(\widehat{W}) = \sum_{i=1}^{N} \mathrm{Cross}(\widehat{W}\widehat{x}_i, y_i)$. An expansion yields

$$\widehat{J}(\widehat{W}) = -\sum_{i=1}^{N}\sum_{r=1}^{m} \log\left(\frac{e^{\left\langle \widehat{W}\widehat{x}_i, e_r\right\rangle}}{\sum_{s=1}^{m} e^{\left\langle \widehat{W}\widehat{x}_i, e_s\right\rangle}}\right) y_i^r, \qquad y_i^r = \langle y_i, e_r\rangle .$$

Let us assume that there exists $\widehat{W}_*$ such that $\widehat{J}(\widehat{W}_*) \leq \widehat{J}(\widehat{W})$ for all $\widehat{W} \in \mathcal{M}_{n,m+1}(\mathbb{R})$, then the function f is defined by $f(X, \widehat{W}_*) = \mathrm{Logis}(\widehat{W}_*\widehat{X}) = \mathrm{Logis}(W_*X + b_*)$. Since the cross-entropy function is convex, it is clear that $\widehat{J}$ is convex with respect to the pair (W, b). Nevertheless, it is well known in convex analysis[2] that convexity is not enough to guarantee the existence of a minimum argument $\widehat{W}_*$ such that $\widehat{J}(\widehat{W}_*) \leq \widehat{J}(\widehat{W})$ for all possible $\widehat{W}$. A technical difficulty is that, contrary to Lemma 1.3.4, no explicit solution of the Euler-Lagrange equation $\nabla\widehat{J}(\widehat{W}_*) = 0$ is known because the gradient (1.37) has a complicated structure due to the sum over all pairs in the dataset. However we will prove in this Section that a (non unique) minimum exists, using standard methods in convex analysis theory [1, 44].

The introduction of a more general Neural Network structure in the classification formulation (1.35) is performed by considering the function (1.27) which has the structure (1.25) with the parameters W (1.26). One obtains the generalization of (1.35)

$$J(W) = \sum_{i=1}^{N} \mathrm{Cross}(f(x_i, W), y_i). \tag{1.36}$$

In the general case, that is when there is one (or more) hidden layer with (possibly various) non linear activation functions, the function J has no reason to be convex and it is extremely delicate to analyze its structure. Some numerical methods used in practice to calculate a feasible solution to the minimization problem $W_* = \mathrm{argmin}\ J(W)$ will be presented in Chapter 4.

2 The typical example in dimension one is the exponential function $x \mapsto e^x$ which is strictly convex but has no minimum. Instead it has an infimum at $-\infty$.

To explain why the minimization problem associated to the simple formulation (1.35) with affine reconstruction is well posed, we begin with elementary formulas.

Lemma 1.3.5. *The gradient $\nabla_{\widehat{W}} \widehat{J}(\widehat{W})$ is*

$$\nabla_{\widehat{W}} \widehat{J}(\widehat{W}) = \sum_{i=1}^{N} \left(\text{Logistic}\left(\widehat{W}\widehat{x}_i\right) - y_i\right) \otimes \widehat{x}_i \in \mathcal{M}_{n,m+1}(\mathbb{R}). \tag{1.37}$$

The Hessian matrix $\nabla^2_{\widehat{W}} \widehat{J}(W)$ is characterized by

$$\left\langle \nabla^2_{\widehat{W}} \widehat{J}(W)\overline{W}, \overline{W} \right\rangle = \sum_{i=1}^{N} \left\langle \nabla_z^2 \text{Cross}(\widehat{W}\widehat{x}_i, y_i)\overline{W}\widehat{x}_i, \overline{W}\widehat{x}_i \right\rangle \geq 0, \quad \forall \overline{W} \in \mathcal{M}_{n,m+1}(\mathbb{R}).$$

The function $\widehat{J}$ is not strictly convex since the second derivative vanishes in the direction $\overline{W} = \lambda(0_{n,m}, e)$.

Remark 1.3.6. *Here the Hessian matrix must be understood as a tensor which acts on a pair of matrices in $\mathcal{M}_{n,m+1}(\mathbb{R})$. This is a tensor with four axis, see Definition 1.3.12. An alternative is to identify matrices in $\mathcal{M}_{n,m+1}(\mathbb{R})$ as vectors in $\mathbb{R}^{n(m+1)}$. Then the Hessian matrix $\nabla^2_{\widehat{W}} \widehat{J}(W)$ can be written as a classical square matrix in $\mathcal{M}_{n(m+1)}(\mathbb{R})$.*

Proof. The first derivative comes from the summation $\sum_i \cdots$ of

$$\nabla_{\widehat{W}} \text{Cross}(\widehat{W}\widehat{x}_i, y_i) = \nabla z \text{Cross}(\widehat{W}\widehat{x}_i, y_i) \otimes \widehat{x}_i = \left(\text{Logistic}\left(\widehat{W}\widehat{x}_i\right) - y_i\right) \otimes \widehat{x}_i.$$

The Hessian matrix comes from the summation $\sum_i \cdots$ of

$$\left\langle \nabla^2_{\widehat{W}} \text{Cross}(\widehat{W}\widehat{x}_i, y_i)\overline{W} : \overline{W} \right\rangle = \left\langle \nabla_z^2 \text{Cross}(\widehat{W}\widehat{x}_i, y_i)\overline{W}\widehat{x}_i, \overline{W}\widehat{x}_i \right\rangle$$

for all test matrices $\overline{W} \in \mathcal{M}_{n,m+1}(\mathbb{R})$. The non negativity is a consequence of (1.33). □

Our objective is now to find a minimizer $\widehat{W}_* \in \mathcal{M}_{n,m+1}(\mathbb{R})$ such that $0 \leq \widehat{J}(\widehat{W}_*) \leq \widehat{J}(\widehat{W})$ for all $\widehat{W} \in \mathcal{M}_{n,m+1}(\mathbb{R})$. Taking $\widehat{W} = \widehat{0}$, one necessarily has that the minimizer, if it exists, is such that

$$0 \leq \widehat{J}(\widehat{W}_*) \leq \widehat{J}(\widehat{0}) = N \log n.$$

So it is sufficient to consider the set

$$\mathcal{K}_0 = \left\{ \widehat{W} \in \mathcal{M}_{n,m+1}(\mathbb{R}), \widehat{J}(\widehat{W}) \leq N \log n \right\}$$

and to look for a minimum $\widehat{W}_* \in \mathcal{K}_0$ in this set

$$0 \leq \widehat{J}(\widehat{W}_*) \leq \widehat{J}(\widehat{W}) \text{ for all } \widehat{W} \in \mathcal{K}_0. \tag{1.38}$$

Definition 1.3.7. *Let us define the vectorial subspace $\mathcal{M} \subset \mathcal{M}_{n,m+1}(\mathbb{R})$*

$$\mathcal{M} = \underset{1\le i\le N,\ 2\le r\le m}{Span} \{(e_r - e_1) \otimes \widehat{x}_i\}$$

and the vectorial subspace $\mathcal{L} \subset \mathcal{M}_{n,m+1}(\mathbb{R})$

$$\mathcal{L} = \left\{\widehat{W} \in \mathcal{M}_{n,m+1}(\mathbb{R}), \left\langle \widehat{W}\widehat{x}_i, e_r - e_1\right\rangle = 0 \text{ for all } 1 \le i \le N \text{ and } 2 \le r \le m\right\}.$$

Let us take $\widehat{W} = (0_{n,m} \mid e) \in \mathcal{M}_{n,m+1}(\mathbb{R})$. Then $\left\langle \widehat{W}\widehat{x}_i, e_r - e_1\right\rangle = \langle e, e_r - e_1\rangle =$ 0 for all i and all r, that is $\widehat{W} \in \mathcal{L}$. It yields that

$$\dim(\mathcal{L}) \ge 1. \tag{1.39}$$

By definition the spaces are orthogonal with respect to the scalar product (1.4)

$$\mathcal{L} = \mathcal{M}^\perp \text{ and } \mathcal{M} = \mathcal{L}^\perp.$$

Lemma 1.3.8. *Consider the orthogonal decomposition of $\widehat{W} \in \mathcal{M}_{n,m+1}(\mathbb{R})$*

$$\widehat{W} = \widehat{W_1} + \widehat{W_2}, \quad \widehat{W_1} \in \mathcal{M},\ \widehat{W_2} \in \mathcal{L}.$$

Then one has $\widehat{J}(\widehat{W}) = \widehat{J}(\widehat{W_2})$.

Proof. By definition, one has

$$\widehat{J}(\widehat{W_1} + \widehat{W_2}) = -\sum_{i=1}^N \sum_{r=1}^m \log\left(\frac{e^{\left\langle \widehat{W_1}\widehat{x}_i, e_r\right\rangle} e^{\left\langle \widehat{W_2}\widehat{x}_i, e_r\right\rangle}}{\sum_{s=1}^m e^{\left\langle \widehat{W_1}\widehat{x}_i, e_s\right\rangle} e^{\left\langle \widehat{W_2}\widehat{x}_i, e_s\right\rangle}}\right) y_i^r.$$

The condition $\widehat{W_2} \in \mathcal{L}$ yields the simplification

$$\widehat{J}(\widehat{W_1} + \widehat{W_2}) = -\sum_{i=1}^N \sum_{r=1}^m \log\left(\frac{e^{\left\langle \widehat{W_1}\widehat{x}_i, e_r\right\rangle}}{\sum_{s=1}^m e^{\left\langle \widehat{W_1}\widehat{x}_i, e_s\right\rangle}}\right) y_i^r = J(\widehat{W_1}).$$

□

Theorem 1.3.9. *Assume $y_i \in \mathcal{P}_n \bigcap (0,1)^n$ for all $1 \le i \le N$. Then there exists a matrix $\widehat{W}_* \in \mathcal{M}_{n,m+1}(\mathbb{R})$ such that of $\widehat{J}(\widehat{W}_*) \le \widehat{J}(\widehat{W})$ for all $\widehat{W} \in \mathcal{M}_{n,m+1}(\mathbb{R})$. This matrix is non unique.*

Proof. Take a sufficiently large $K > 0$ and consider the set

$$S(K) = \left\{\widehat{W} \in \mathcal{M}_{n,m+1}(\mathbb{R}) \mid \sup_{i=1}^N \sup_{r=2}^m \left|\left\langle \widehat{W}\widehat{x}_i, e_1 - e_r\right\rangle\right| \le K\right\}.$$

Using the technical Lemma 1.3.11, the set $S(K) \cap \mathcal{M}$ is compact. So the continuous function $\widehat{J}$ has a minimum over $S(K) \cap \mathcal{M}$, denoted as $\widehat{W}_* \in S(K) \cap \mathcal{M}$. Using Lemma 1.3.8, $\widehat{W}_*$ is also a minimizer of $\widehat{J}$ over $S(K)$

$$\widehat{J}(\widehat{W}_*) \leq \widehat{J}(\widehat{W}), \quad \forall \widehat{W} \in S(K).$$

With the other technical Lemma 1.3.10, it is a minimizer over $\mathcal{M}_{n,m+1}(\mathbb{R})$. □

It remains to prove the two technical Lemmas.

Lemma 1.3.10. *Assume $y_i \in \mathcal{P}_n \bigcap (0,1)^n$ for all $1 \leq i \leq N$. Then there exists a sufficiently large constant $K > 0$ such that $\mathcal{K}_0 \subset S(K)$.*

Proof. Let $\widehat{W} \in \mathcal{K}_0$ and take an integer l $(1 \leq l \leq N)$. One has

$$0 \leq \text{Cross}(\widehat{W}\widehat{x}_l, y_l) \leq \sum_{i=1}^{N} \text{Cross}(\widehat{W}\widehat{x}_i, y_i) = \widehat{J}(\widehat{W}) \leq \widehat{J}(0).$$

Take an integer r $(1 \leq r \leq m)$. One deduces the upper bound

$$-\log\left(\left\langle \text{Logistic}(\widehat{W}\widehat{x}_l), e_r \right\rangle\right) y_l^r \leq \widehat{J}(0).$$

By assumption $y_l^r = (y_l, e_r) > 0$ for all l and r. Therefore there exists a (small but positive) constant $\varepsilon = e^{-\frac{\widehat{J}(0)}{\mu}} \in (0,1)$ with $\mu = \min\limits_{lr} y_l^r$ such that

$$\left\langle \text{Logistic}(\widehat{W}\widehat{x}_l), e_r \right\rangle \geq \varepsilon > 0. \tag{1.40}$$

The constant ε is independent of l and r. The inequality (1.40) can be rewritten as

$$\frac{e^{\left\langle \widehat{W}\widehat{x}_l, e_r \right\rangle}}{\sum_{s=1}^{m} e^{\left\langle \widehat{W}\widehat{x}_l, e_s \right\rangle}} = \frac{1}{\sum_{s=1}^{m} e^{\left\langle \widehat{W}\widehat{x}_l, e_s - e_1 \right\rangle}} \geq \varepsilon \qquad \text{(note that } \varepsilon < 1\text{)}.$$

So one has $e^{\left\langle \widehat{W}\widehat{x}_l, e_s - e_1 \right\rangle} \leq \frac{1}{\varepsilon}$ for all l and s. Without restriction, one can assume the ordering

$$\left\langle \widehat{W}\widehat{x}_l, e_1 \right\rangle \leq \left\langle \widehat{W}\widehat{x}_l, e_2 \right\rangle \leq \cdots \leq \left\langle \widehat{W}\widehat{x}_l, e_m \right\rangle.$$

Therefore one gets the bound

$$0 \leq \left\langle \widehat{W}\widehat{x}_l, e_s - e_1 \right\rangle = \left| \left(\widehat{W}\widehat{x}_l, e_s - e_1 \right) \right| \leq K$$

where the constant is taken as $K = -\log \varepsilon$. Therefore $\widehat{W} \in S(K)$. □

Lemma 1.3.11. *The form*

$$|||\widehat{W}||| = \sup_{i=1}^{N} \sup_{r=2}^{m} \left| \left\langle \widehat{W}\widehat{x}_i, e_1 - e_r \right\rangle \right|, \quad \widehat{W} \in \mathcal{M}_{n,m+1}(\mathbb{R})$$

is a norm on $\mathcal{M}$.

Proof. It is sufficient to show that $\widehat{W} \in \mathcal{M}$ and $|||\widehat{W}||| = 0$ implies that $\widehat{W} = 0$. Assume $|||\widehat{W}||| = 0$, so $\left\langle \widehat{W}\widehat{x}_i, e_1 - e_r \right\rangle = 0$ for all possible i ad r. It means that $\widehat{W} \in \mathcal{L}$. Therefore $\widehat{W} \in \mathcal{M} \cap \mathcal{L} = \{0\}$, and the proof is ended. □

1.3.3 Convolutive neural networks

As mentioned at the beginning of Section 1.3, an image (that is a collection of numbers called pixels because they correspond to a certain position in space) can be identified as a vector

$$x \in \mathbb{R}^a. \tag{1.41}$$

Even if this statement is logically true, it is of course very naive because this representation looses the multi-dimensional structure of an image.

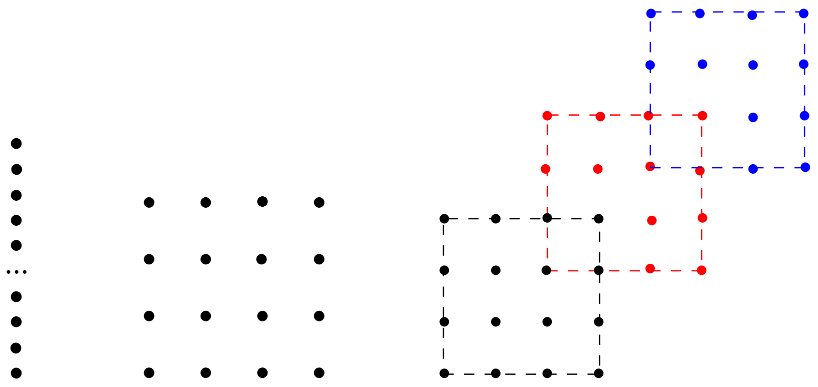

Fig. 1.4: Three different storing of a 2D image. Left as a vector. Center as a 2D structure. Right as a 2D structure plus a color information.

Actually a two-dimensional image is a rectangular map of pixels which may encode an additional information about the color code, which yields a tri-dimensional map of numbers. The color code is ultimately due to the number of different chromatic cones[3] in the human eye, which is equal to 3. Bi-dimensional images in

3 The number of different chromatic cones runs from 1 (marine mammals) to 5 (butterfly). Humans are in between.

grey are identified to tri-dimensional map where the third dimension in 1 (like a monochromatic picture). Tomography applications[4] reconstruct a tri-dimensional map of pixels with an additional color information, so it is corresponds to a four-dimensional map of numbers. The objective hereafter is to present some basic notions which correspond to these facts.

Definition 1.3.12. *A tensor with q axis denoted as $M \in \mathcal{M}_{\alpha_1,\dots,\alpha_q}(\mathbb{R})$ is a collection of real numbers*

$$m_{\beta_1,\dots,\beta_q} \in \mathbb{R}, \text{ where } 1 \le \beta_r \le \alpha_r \text{ for } 1 \le r \le q.$$

For $q = 1$, the tensor M is a vector. For $q = 2$ it is a matrix.

Let us consider the general situation where the entries the vector $x \in \mathbb{R}^a$ correspond to the storage in a **tensor** of the numerical values of the black-and-white pixels of an q-dimensional image

$$X \in \mathcal{M}_{\alpha_1,\dots,\alpha_q}(\mathbb{R}). \tag{1.42}$$

The case of colored pixels is detailed later.
There is a univoque correspondance between the indices x_b of the vector x and the indices $X_{\beta_1,\dots,\beta_q}$ of the tensor $X \in \mathcal{M}_{\alpha_1,\dots,\alpha_q}(\mathbb{R})$. This correspondance is given by[5]

$$\begin{cases} b = \beta_1 + \alpha_1(\beta_2 - 1) + \alpha_1\alpha_2(\beta_3 - 1) + \dots + \alpha_1 \dots \alpha_{q-1}(\beta_q - 1), \\ 1 \le b \le a = \alpha_1 \dots \alpha_q, \\ 1 \le \beta_r \le \alpha_r \text{ for } 1 \le r \le q. \end{cases} \tag{1.43}$$

The index b can be identified as well with the multi-index

$$b \approx \beta = (\beta_1, \dots, \beta_q). \tag{1.44}$$

The formula (1.43) is a bijection between the two different ways of considering/storing an image.

Let us describe linear operations that apply to vectors (1.41) which have a multi-dimensional structures like (1.42-1.44). In simple cases, such linear operations can be written as Toeplitz matrices, band matrices or convolution matrices (so

4 Tomography is a tri-dimensional imaging technique very popular in geophysics, medical imaging, astrophysics, material imaging and many other disciplines.
5 This formula is commonly used in Finite Difference methods for reindexing the indices, from a multi-entry index to a single index.

the name[6]). Two preliminary notions are Toeplitz matrices and band matrices. A square Toeplitz matrix $W \in \mathcal{M}_a(\mathbb{R})$ is such that

$$w_{b,c} = \mathrm{w}_{b-c} \qquad 1 \le b, c \le a \tag{1.46}$$

where $\mathrm{w}_d \in \mathbb{R}$ for $1 - a \le d = b + c \le a - 1$. A square band matrix $W \in \mathcal{M}_a(\mathbb{R})$ of size $1 \le d \le a$ is such that

$$w_{b,c} = 0 \text{ for } |b - c| \ge d.$$

The number of independent entries of a Toeplitz band matrix of size d is $2d < a^2$ which is much less than the total number of entries of a generic matrix. The general definition follows.

```
Python-Keras-Tensorflow Initialization
input_shape = (28, 28,1)
num_classes = 10
model2 = keras.Sequential([
  keras.Input(shape=input_shape),
  layers.Conv2D(32, kernel_size=(3, 3),
  use_bias=False,padding="same",activation="relu"),
  layers.MaxPooling2D(pool_size=(2, 2)),
  layers.Conv2D(64, kernel_size=(3, 3),use_bias=True,
                          padding="valid",activation="relu"),
  layers.MaxPooling2D(pool_size=(2, 2)),
  layers.Flatten(),
  layers.Dense(num_classes, activation="softmax"),   ])
model2.summary()
```

Algorithm 2: CNN adapted to the MNIST database.

Definition 1.3.13. *A convolutive[7] matrix $W \in \mathcal{M}_a(\mathbb{R})$ associated to the correspondance (1.41-1.43) is*

$$w_{b,c} = \mathrm{w}_{\beta_1-\gamma_1,\dots,\beta_q-\gamma_q} \tag{1.48}$$

where $b \approx \beta = (\beta_1, \dots, \beta_q)$ and $c \approx \gamma = (\gamma_1, \dots, \gamma_q)$.

6 Toeplitz band matrices are discrete version of convolutions. One starts with two functions w and y defined on the real line $\mathbb{R}$. The function $x = w * y$ is the convolution of w and y

$$y(t) = \int_{\mathbb{R}} w(t - s)x(s)ds = \int_{\mathbb{R}} w(y)x(t - s)ds. \tag{1.45}$$

If the kernel w has a compact support, it can be discretized as a Toeplitz matrix with a band structure. Since convolution kernels are quite common in nature and science, it explains the importance of this topic.

7 The multidimensional generalization of (1.45) is as follows. Consider $y(\mathbf{t}) = \int_{\mathbb{R}^d} w(\mathbf{t} - \mathbf{s})x(\mathbf{s})ds = \int_{\mathbb{R}} w(\mathbf{s})x(\mathbf{t} - \mathbf{s})ds$, where $\mathbf{t}, \mathbf{s} \in \mathbb{R}^d$ with $d \ge 1$. Assume w has a compact

To model colored images, it is natural to introduce an additional axis which corresponds to different colors of one single pixel of the image. This additional axis is also of great importance for further algorithmic purposes. The channels do not have a geometrical structure like the pixels of an image, they are more the superposition of some properties of the pixels which are called **channels**. Instead of the tensor (1.42), one considers tensors

$$X \in \mathcal{M}_{\alpha_1,\dots,\alpha_q,\mathbf{inp}}(\mathbb{R}) \tag{1.49}$$

where the last axis corresponds to $\mathbf{inp} \geq 1$ different input channels. They are many possibility in terms of manipulations of such structures. Hereafter we describe one of the simplest ones, which is convolution of the underlying image described by the q first axis with a different number of entries for the input x and the output y. The structure of y corresponds to the tensor

$$Y \in \mathcal{M}_{\alpha_1,\dots,\alpha_q,\mathbf{out}}(\mathbb{R}) \tag{1.50}$$

where the last axis corresponds to $\mathbf{out} \geq 1$ different output channels.
Technically, the coefficients of the input follow the structure $x_b = X_{\beta_1,\dots,\beta_q,\beta_{q+1}}$ where the indices are related by

$$\begin{cases} b = \beta_1 + \alpha_1(\beta_2 - 1) + \alpha_1\alpha_2(\beta_3 - 1) + \dots + \alpha_1 \dots \alpha_q(\beta_{q+1} - 1), \\ 1 \leq b \leq a_{\mathbf{inp}} = (\alpha_1 \dots \alpha_q)\mathbf{inp}, \\ 1 \leq \beta_r \leq \alpha_r \text{ for } 1 \leq r \leq q, \\ 1 \leq \beta_{q+1} \leq \mathbf{inp}. \end{cases} \tag{1.51}$$

The coefficients of the output follow the structure $y_c = Y_{\gamma_1,\dots,\gamma_q,\gamma_{q+1}}$ where the indices are related by

$$\begin{cases} c = \gamma_1 + \alpha_1(\gamma_2 - 1) + \alpha_1\alpha_2(\gamma_3 - 1) + \dots + \alpha_1 \dots \alpha_q(\gamma_{q+1} - 1), \\ 1 \leq c \leq a_{\mathbf{out}} = (\alpha_1 \dots \alpha_q)\mathbf{out}, \\ 1 \leq \gamma_r \leq \alpha_r \text{ for } 1 \leq r \leq q, \\ 1 \leq \gamma_{q+1} \leq \mathbf{out}. \end{cases} \tag{1.52}$$

support $|\mathbf{t}| \geq R \Longrightarrow w(\mathbf{t}) = 0$. So one can write

$$y(t_1,\dots,t_d) = \int_{(-R,R)^d} w(t_1 - s_1,\dots,t_d - s_d)x(s_1,\dots,s_d)ds_1 \dots ds_d. \tag{1.47}$$

After sampling along the coordinates, one can discretize this operation with a convolutive matrix.

Definition 1.3.14. *A convolutive matrix $W \in \mathcal{M}_{a_{\mathbf{out}}, a_{\mathbf{inp}}}(\mathbb{R})$ with* **inp** ≥ 1 *input channels and* **out** ≥ 1 *output channels associated*[8] *to the correspondance (1.51-1.52) is*

$$w_{b,c} = \mathrm{w}_{\beta_1-\gamma_1,\dots,\beta_q-\gamma_q:\mathbf{inp},\mathbf{out}} \tag{1.53}$$

where $b \approx \beta = (\beta_1, \dots, \beta_q, \beta_{q+1})$ and $c \approx \gamma = (\gamma_1, \dots, \gamma_q, \gamma_{q+1})$.

Remark 1.3.15. *In other words, a convolutive matrix with channels is a convolutive matrix for the axis which describe the geometrical structure of the datas, while it is a classical matrix for the input channel and output channel parameters.*

Below, we describe techniques such as padding and striding which are used to restrict even further the range of convolution formulas to blocks such that the full convolution kernel applies. Using (1.43), striding in the rth direction means means that only a few numbers of indices of indices β_r are used, not the full set $\beta_r = 1, \dots, \alpha_r$. To implement the idea in the rth direction, a possibility is to consider a stride-parameter $\widehat{\alpha}_r$ $(1 \leq \alpha_r^{\mathrm{str}} \leq \alpha_r)$ and the integers β_r^{red} ("red" is for reduced) such that

$$\beta_r = 1 + \alpha_r^{\mathrm{str}}(\beta_r^{\mathrm{red}} - 1) \tag{1.54}$$

satisfies $1 \leq \beta_r \leq \alpha_r$. It corresponds to the interval

$$1 \leq \beta_r^{\mathrm{red}} \leq \alpha_r^{\mathrm{red}}$$

where the reduced size is $\alpha_r^{\mathrm{red}} = 1 + \left\lfloor \frac{\alpha_r - 1}{\alpha_r^{\mathrm{str}}} \right\rfloor$. Starting from $\beta^{\mathrm{red}} = \left(\beta_1^{\mathrm{red}}, \dots, \beta_q^{\mathrm{red}}\right)$, one defines naturally

$$b^{\mathrm{red}} = \beta_1^{\mathrm{red}} + \alpha_1^{\mathrm{str}}(\beta_2^{\mathrm{red}} - 1) + \alpha_1^{\mathrm{str}}\alpha_2^{\mathrm{str}}(\beta_3^{\mathrm{red}} - 1) + \dots + \alpha_1^{\mathrm{str}} \dots \alpha_{q-1}^{\mathrm{str}}(\beta_q^{\mathrm{red}} - 1)$$

and the combination of (1.43) and (1.54)

$$\begin{cases} b = \beta_1 + \alpha_1(\beta_2 - 1) + \alpha_1\alpha_2(\beta_3 - 1) + \dots + \alpha_1 \dots \alpha_{q-1}(\beta_q - 1), & \\ \beta_r = 1 + \alpha_r^{\mathrm{str}}(\beta_r^{\mathrm{red}} - 1), & 1 \leq r \leq q. \end{cases}$$

One can now write a linear operation which starts from a space of dimension $a = \alpha_1 \dots \alpha_q$ and ends in a space of dimension $a^{\mathrm{red}} = \alpha_1^{\mathrm{red}} \dots \alpha_q^{\mathrm{red}}$.

8 As in formula (1.47), the matrix W can be seen as the discretization of the weight w in the convolution formula

$$y(\mathbf{t}, \alpha) = \int_{\mathbf{t} \in (-R,R)^d} w(\mathbf{t} - \mathbf{s}, \alpha, \beta) x(\mathbf{s}) ds d\beta.$$

Definition 1.3.16. *A rectangular convolutive matrix with stride* $W \in \mathcal{M}_{a^{\mathrm{red}},a}(\mathbb{R})$ *has the following structure.*
For

$$\begin{cases} c = \gamma_1 + \alpha_1(\gamma_2 - 1) + \alpha_1\alpha_2(\gamma_3 - 1) + \cdots + \alpha_1 \dots \alpha_{q-1}(\gamma_q - 1), & \\ \gamma_r = 1 + \alpha_r^{\mathrm{str}}(\gamma_r^{\mathrm{red}} - 1), & 1 \le r \le q, \end{cases}$$

then $w_{b^{\mathrm{red}},c} = w_{\beta_1^{\mathrm{red}}-\gamma_1^{\mathrm{red}},\dots,\beta_q^{\mathrm{red}}-\gamma_q^{\mathrm{red}}}$. *In other cases then* $w_{b^{\mathrm{red}},c} = 0$.
Convolutive matrices with stride and channels are obtained by combination with Definition 1.3.14.

Further compression is performed with the maxpool function [34].

Definition 1.3.17. *The maximal value of two numbers is called the* maxpool *function:* $\mathrm{maxpool}(a,b) = \max(a,b)$. *This function is extended to multidimensional blocks with natural notations.*

So $\mathrm{maxpool}(a,b) = a + R(b-a) = b + R(a-b)$ can be encoded with the ReLU function R. Similarly, one has $\min(a,b) = a + b - \mathrm{maxpool}(a,b) = b - R(b-a) = a - R(a-b)$.

Exercise 1.3.18. *Show that*

$$\min(a,b) = W_1 R\,(W_0) \tag{1.55}$$

where $W_1 = \frac{1}{2}(1,-1,-1,-1)$, $W_0 = \begin{pmatrix} 1 & 1 \\ -1 & -1 \\ 1 & -1 \\ -1 & 1 \end{pmatrix}$ *and* $x = \begin{pmatrix} a \\ b \end{pmatrix}$.

Algorithm 2 is the declaration in Python-Keras-Tensorflow of a CNN[9][10] for the MNIST database [55]. In this database, the images are made of 28×28 black-and-white pixels. That is an image is a 28×28 table of integers between 0 and 255. The number of classes is 10 which is the number of different digits. In Algorithm 2, one sees that one considers the image as a multi-channel image, but with a channel equal to 1, that is why the shape of the input is $(28, 28, 1)$. Lines 5 to 14 are the declaration of the CNN with a syntax which is slightly different from the one of the previous Algorithm. The convolution is performed with `layers.Conv2D` which is the convolution in space dimension 2, that is q with the notation of Definition 1.3.12.

9 https://www.ljll.math.upmc.fr/despres/BD_fichiers/mnist_summary.py
10 https://www.ljll.math.upmc.fr/despres/BD_fichiers/mnist_CNN_TReLU.py

```
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 28, 28, 32)        288
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 14, 14, 32)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 12, 12, 64)        18496
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 6, 6, 64)          0
_________________________________________________________________
flatten (Flatten)            (None, 2304)              0
_________________________________________________________________
dense (Dense)                (None, 10)                23050
=================================================================
Total params: 41,834
```

Tab. 1.2: Parameters of the CNN specified in Algorithm 2.

The number of parameters is reported in Table 1.2. The first convolution layer as $3^2 \times 1 \times 32 = 288$ parameters, in accordance with what is described in (1.48). The output of this layer has the shape $(28, 28, 32)$. Then pooling on 2×2 blocks yields a tensor with the shape $(14, 14, 32)$. Another convolution layer is activated, but with two differences. Firstly the `padding` is declared as valid, it has the effect that the convolution is activated only on blocks such that the full convolution can be applied: since the kernel is 3×3, then the shape becomes $12 \times 12 \times 64$ because we use output 64 channels. Secondly the bias is activated: that is why the number of parameters is $3^2 \times 32 \times 64 = 18432$ which corresponds to the general formula (1.48) plus 64 biais parameters, so the total number is $18432 + 64 = 18496$. Additional pooling yields a tensor of shape $(6, 6, 64)$ which is flattened on a unique vector of size $6 \times 6 \times 64 = 2304$. It remains to activated a dense layer with 10 outputs which are the 10 classes, that is $2304 \times 11 = 23050$ parameters. For classification the `softmax` is used in line 13 of Algorithm 2.

Algorithm 3 describes an adaptation of Algorithm 2 to the CIFAR database which is dedicated to colored 2D images[11].

11 https://www.cs.toronto.edu/~kriz/cifar.html

```
Tensorflow Initialization
no_classes = 10
img_width, img_height, img_num_channels = 32, 32, 3
input_shape = (img_width, img_height, img_num_channels)
model3 =Sequential()
model3.add(Conv2D(32, kernel_size=(3, 3), activation='relu', input_shape=input_shape))
model3.add(MaxPooling2D(pool_size=(2, 2)))
model3.add(Conv2D(64, kernel_size=(3, 3), activation='relu'))
model3.add(MaxPooling2D(pool_size=(2, 2)))
model3.add(Conv2D(128, kernel_size=(3, 3), activation='relu'))
model3.add(MaxPooling2D(pool_size=(2, 2)))
model3.add(Flatten())
model3.add(Dense(256, activation='relu'))
model3.add(Dense(128, activation='relu'))
model3.add(Dense(no_classes, activation='softmax'))
model3.summary()
```

Algorithm 3: CNN adapted to the CIFAR database with 3 channels.

1.4 Summary of the chapter

Various Neural Networks were introduced in this chapter. It was shown that these Neural Networks allow to construct functions, denoted as $f : \mathbb{R}^m \to \mathbb{R}^n$, which hopefully have the ability to approximate a given objective function, denoted as $f^{\text{obj}} : \mathbb{R}^m \to \mathbb{R}^n$. The function f can be implemented with any number of (deep) layers. The objective function is characterized with a dataset which contains interpolation data.

The next chapter is dedicated to show that Neural-Network-generated functions have in theory the ability to approximate any objective function with known regularity, provided the structure and the numerical value of the weights satisfy some theoretical requirements.