Chapter 1

Starter on neural networks

Contents

Historically, neural networks (NN) have been introduced to model the human brain behavior [MP43, Ros58] in order to bring some theory foundations for artificial intelligence. Now that deep NN can be trained with big data, these tools know an impressive come back with excellent performances in various application fields, as speech recognition, automatic translation (e.g., Deepl), self-driving cars, automatic diagnoses, quantum chemistry, molecular dynamics, and so on. The suspicious reader can type "neural network" on google scholar to measure the "worldwide triumphant march of deep neural network" to quote [GKNV21]. Despite this success, the theoretical properties of NN are far less understood. The aim of this lecture is to bring some theoretical explanations to the excellent practical behavior of NN and in particular to put forward the role of the depth.

In this chapter, we introduce elementary definitions for neural networks and we provide a first view on the approximation ability of these objects and on the role of the depth. It mostly relies on the lecture notes of [Pet22].

1.1 Definition

Neural network Let us first provide the definition of a neural network (NN in short).

Definition 1.1. *Let $d, k \geq 1$, $L \geq 2$ be three integers. A neural network (NN) with input dimension d, ouput dimension k and depth L is a pair $\Phi = (W, \rho)$ where $\rho : \mathbb{R} \to \mathbb{R}$ is some function, called the activation function, and W is a sequence of matrix-vector pairs*

$$W = ((A_1, b_1), \dots, (A_L, b_L)),$$

whose elements are called the weights (or coefficients), where for $1 \leq \ell \leq L$, A_ℓ is a $N_\ell \times N_{\ell-1}$ real matrix, and b_ℓ is a N_ℓ-dimensional real vector, for some integers $N_1, \dots, N_{L-1} \geq 1$ (convention $N_0 := d$, $N_L := k$).

For a NN Φ, we can define the following quantities, all-together defining the *architecture* or *topology* of the network:

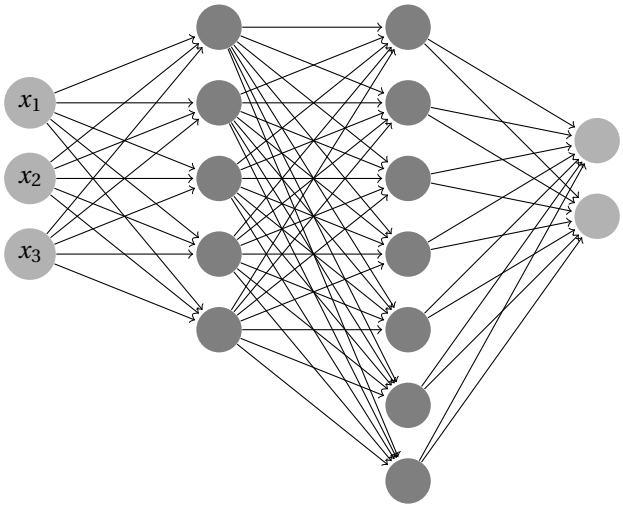

Figure 1.1: Instance of neural network with depth $L = 3$, $N = 17$, $N_{\max} = 7$ and fully connected.

- $L(\Phi) = L$ is the *depth* of Φ.
- The layers corresponding to $\ell \in \{1, \dots, L-1\}$ are called the hidden layers. Hence, L is also the number of layers, while $L-1$ is the number of hidden layers.
- For each $\ell \in \{1, \dots, L\}$, $N_\ell(\Phi) = N_\ell$ denotes the number of neurons of the ℓ-th layer, also called the *ℓ-th width* of Φ. The *maximum width* of the NN is $N_{\max}(\Phi) = \max_{1 \le \ell \le L-1} N_\ell$. The *number of neurons of* Φ is $N(\Phi) = d + \sum_{\ell=1}^{L} N_\ell$.
- The *sparsity* of Φ is the number of non-zero weights of Φ, that is, $\|\Phi\|_0 = \sum_{\ell=1}^{L} (\|A_\ell\|_0 + \|b\|_0)$.
- The *support* of Φ is given by the sequence of supports

$$\{(i, j) \in \{1, \dots, N_{\ell-1}\} \times \{1, \dots, N_\ell\} : (A_\ell)_{i,j} \neq 0\},\ 1 \le \ell \le L.$$

A representation for a NN is often done via its architecture, by drawing an edge between neuron i of layer $\ell - 1$ and neuron j in layer ℓ, if $(A_\ell)_{i,j} \neq 0$, see Figures 1.1 and 1.2.

A NN is associated to a particularly useful function, called the realization of the NN.

Definition 1.2. *For some NN $\Phi = (W, \rho)$, with $W = ((A_1, b_1), \dots, (A_L, b_L))$ having input dimension d and output dimension k, the realization of Φ is defined by the function*

$$R(\Phi) : \mathbb{R}^d \to \mathbb{R}^{N_L} : x \mapsto R(\Phi)(x) = x_L\,,$$

where the output x_L is given by the relations

$$\begin{aligned} x^{(0)} &= x\,; \\ x^{(\ell)} &= \rho(A_\ell x^{(\ell-1)} + b_\ell),\ 1 \le \ell \le L-1\,; \\ x^{(L)} &= A_L x^{(L-1)} + b_L, \end{aligned}$$

for which the function ρ acts component-wise, that is, for some vector $y = (y_i)_{1 \le i \le I}$, $\rho(y)$ denotes the vector $(\rho(y_i))_{1 \le i \le I}$.

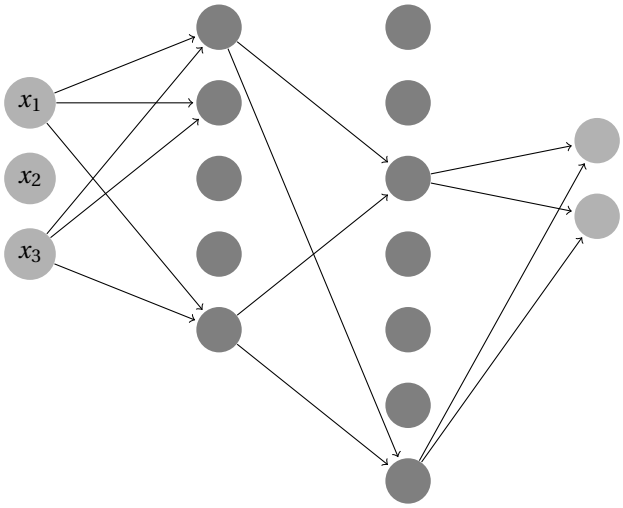

Figure 1.2: Same neuron structure as in Figure 1.1, but with sparsity 14 (if $b_\ell = 0$ for $1 \le \ell \le L$).

As it is apparent from Figure 1.2, two different NN can have the same realization (just remove some useless neurons), hence distinguishing a NN from its realization is necessary to define properly the complexity of a NN. The realization of a NN is also called a *multilayer perceptron* (MLP) with d-dimensional input, k-dimension ouput, L layers (that is, $L-1$ hidden layers) and activation function ρ.

Put differently, by denoting $T_\ell : x \in \mathbb{R}^{N_{\ell-1}} \mapsto A_\ell x + b_\ell \in \mathbb{R}^{N_\ell}$, $R(\Phi)$ can be written as the composition function

$$x \in \mathbb{R}^d \mapsto R(\Phi) = T_L(\rho(T_{L-1}(\dots\rho(T_1(x))\dots)))\,,$$

or

$$R(\Phi) = T_L \circ \rho \circ \cdots \circ T_2 \circ \rho \circ T_1\,.$$

Example 1.3 (Shallow network)**.** *Any NN with depth $L = 2$ is called a shallow network. Hence, the realization of a shallow NN (with input/output dimension 1), takes the following particular form*

$$x \in \mathbb{R} \mapsto \sum_{i=1}^{N} (A_{2,1,i}\rho(A_{1,i}x + b_{1,i}) + b_{2,i}) = b' + \sum_{i=1}^{N} a'_i \rho(a_i x + b_i),$$

where $N \ge 1$ is the number of neurons of the hidden layer, and a_i, b, a'_i, b' are real numbers.

Finally, with some abuse, when the choice of ρ is clear from the context, we will sometimes simply say that the NN is given by $\Phi = ((A_1, b_1), \dots, (A_L, b_L))$.

Network expressiveness The class of functions generated from a NN is more or less complex depending on the complexity of the NN and of the choice of the activation function. The set of realized NN of depth L, input dimension d, output dimension k and activation function ρ is denoted by $\mathbf{NN}^{k,d}(L, \rho)$. Informally, we say that the size of $\mathbf{NN}^{k,d}(L, \rho)$ measures the expressiveness of the network. The larger this class, the more expressive the NN.

For instance, any NN using an affine activation function realizes an affine function, which means that it is not expressive. Hence, we are going to consider an activation function which is non-affine, which will correspond to more expressive NN. Some popular examples are:

- The rectified linear unit (ReLU) $\rho(x) = x_+ = \max(0, x)$;
- The rectified quadratic unit (ReQU) $\rho(x) = x_+^2$.
- Any sigmoidal function, that is, a continuous function $\rho : \mathbb{R} \to \mathbb{R}$ such that $\lim_\infty \rho = 1$, $\lim_{-\infty} \rho = 0$. For instance, $\rho = 1 + (2/\pi)\arctan$.

For instance, it is clear that ReLU NN generate always continuous and piece-wise affine function, that is, $\mathbf{NN}^{d,k}(L, (\cdot)_+)$ is contained in the set of affine functions. This can be seen as a limit of expressiveness of the ReLU network and could perhaps suggest to avoid using such functions. However, we will see that the interest of ReLU NN lies in its ability to *approximate* regular functions. In great generality, the expressiveness also refers to this approximation ability. This property will be further formalize later on.

Some examples

Example 1.4 (Induced smoothness)**.** *The smoothness of ρ induces some minimum smoothness for the realization:*

- *a Lipshitz ρ gives an (at least) Lipschitz realization;*
- *a k-times differentiable ρ gives an (at least) k-times differentiable realization;*
- *any NN with depth $L \geq 2$ and using an activation function being polynomial of degree $r \geq 1$ realizes a polynomial of degree at most $r(L-1)$.*

Example 1.5 (Realizing identity)**.** *The identity function $f : x \in \mathbb{R} \mapsto x = x_+ - (-x)_+$ can be realized as a ReLU NN $\Phi = ((A_1, b_1), (A_2, b_2))$ of depth $L = 2$ and $N_1 = 2$, $A_1 = (1\ -1)^T$, $b_1 = (0\,0)^T$, $A_2 = (1\ -1)$, $b_2 = (00)$. In particular, any affine function can be realized as a ReLU NN. Also, this shows that a realized NN can be more regular than the initial activation function.*

Example 1.6 (Realizing identity in higher dimension and arbitrary depth)**.** *The previous example can be generalized to arbitrary input/output dimension $d = k$, and depth $L \geq 2$, by considering the network*

$$\Phi_{d,L}^{Id} = \left(\left(\begin{pmatrix} I_d \\ -I_d \end{pmatrix}, 0 \right), (I_{2d}, 0), \ldots, (I_{2d}, 0), ((I_d\ -I_d), 0) \right), \tag{1.1}$$

in which $(I_{2d}, 0)$ is repeated $L-2$ times. With the ReLU activation function, we easily check that $R(\Phi_{d,L}^{Id})$ is the identity function on $\mathbb{R}^d$.

The above examples show that the identity function can be realized (exactly) as a ReLU NN. It is also interesting to note that the identity function can be realized with other more regular activation functions, up to some approximation terms. For instance, if we use any (non-constant) differentiable activation function, the identity function be approximated within a compact set. This roughly comes from the Taylor approximation of $\rho(x)$ by an affine function.

Proposition 1.7 (Approximating identity)**.** *Let K be a segment of $\mathbb{R}$ with non-empty interior and $\rho : \mathbb{R} \to \mathbb{R}$ be differentiable at some x^* belonging to the interior of K with $\rho'(x^*) \neq 0$. Then for every $\epsilon > 0$, there exists a NN $\Phi = ((A_1, b_1), (A_2, b_2))$ with depth $L = 2$, $N_1 = 1$ and $\|\Phi\|_0 \leq 4$ such that for all $x \in K$, $|R(\Phi)(x) - x| < \epsilon$.*

Proof. Assume $K \subset [-M, M]$ for $M > 0$. First observe that by definition of the derivative of ρ at point x^* (and using the fact that K is compact and $\rho'(x^*) \neq 0$), we have that for all $\epsilon > 0$ there exists a $\Lambda = \Lambda(\epsilon) > 0$ such that for all $\lambda > \Lambda$ for all $x \in K$ we have

$$\left| \lambda \frac{\rho(x/\lambda + x^*) - \rho(x^*)}{x} - \rho'(x^*) \right| < \epsilon |\rho'(x^*)| / M.$$

This leads to

$$\left| \lambda \frac{\rho(x/\lambda + x^*) - \rho(x^*)}{\rho'(x^*)} - x \right| < \epsilon |x| / M \leq \epsilon.$$

Now, for $\Phi = ((A_1, b_1), (A_2, b_2))$ with $A_1 = 1/\lambda$, $b_1 = x^*$, $A_2 = \lambda/\rho'(x^*)$, $b_2 = -\rho(x^*)\lambda/\rho'(x^*)$, we have $R(\Phi)(x) = \lambda \frac{\rho(x/\lambda + x^*) - \rho(x^*)}{\rho'(x^*)}$ for all $x \in \mathbb{R}$. Noting that the weights of Φ only depend on $\lambda, \epsilon, x^*, \rho$, the proof is finished. □

1.2 Non-constructive approximations

The following result shows that a shallow network is enough to approximate arbitrary well any continuous function on a compact set.

Theorem 1.8 (Universal approximation theorem, [Cyb89, HSW89])**.** *Let ρ be any sigmoidal activation function and $K \subset \mathbb{R}^d$ be a compact set. Then $\mathbf{NN}^{d,1}(2, \rho)$ is dense in the set of real-valued continuous functions in K, that is, for any function $f : K \to \mathbb{R}$ continuous on K and any $\epsilon > 0$, there exists a NN $\Phi = \Phi^{f,\epsilon,d}$ with $L(\Phi) = 2$ such that $\sup_{x \in K} |R(\Phi)(x) - f(x)| \leq \epsilon$.*

A proof of this result can be found in [Pet22] (Theorem 2.4). It relies on the Hahn-Banach theorem and thus is not constructive. Therefore, it cannot be used to determine Φ. Moreover, while the choice of the activation function is relatively free and the depth is fixed at $L = 2$, the other complexity measures of Φ are not guaranteed to be bounded.

Another branch of results prescribes the choice of the activation function to offer a better complexity control, as in the following theorem.

Theorem 1.9 (Fast approximation)**.** *There exists a continuous piece-wise polynomial activation function ρ such that for any continuous real-valued function f on $[0,1]$ and any $\epsilon > 0$, there exists a NN $\Phi = \Phi^{f,\epsilon}$ with $L(\Phi) = 2$, $N(\Phi) \leq 3$, $\|\Phi\|_0 \leq 3$ such that $\sup_{x \in [0,1]} |R(\Phi)(x) - f(x)| \leq \epsilon$.*

The idea of the proof is quite simple (see Proposition 2.21 in [Pet22] for a formal proof): since polynomials with rational coefficients are dense in $C([0,1], \mathbb{R})$, we can record all possible (countable) approximation functions on segments $[2i, 2i+1]$, $i \in \mathbb{Z}$, and concatenate the result to define a piecewise polynomial activation function ρ. Now, any given f can be well approximated on $[0,1]$ by translating appropriately ρ. To complete the proof, it is sufficient to note that the latter can be realized by a NN with only 1 non-zero coefficient in the hidden layer.

Compared to Theorem 1.8, Theorem 1.9 further bounds the sparsity of the network by ensuring $\|\Phi\|_0 \leq 3$. Also, it can be generalized in higher dimension by using the Kolmogorov superposition theorem, see Theorem 2.24 in [Pet22].

However, a drawback of this kind of result is that we have put too much information in the activation function, which makes the corresponding network not usable in practice (even computing $R(\Phi)$ is not possible!).

In this lecture, we will rather focus on network approximations with bounded $L(\Phi)$, $N(\Phi)$ and $\|\Phi\|_0$ for a feasible activation function, like the ReLU or ReQU.

1.3 Basic operations

Combining networks will be particularly useful when building approximating networks.

Concatenation The perhaps most natural way to combine networks is the concatenation, which corresponding to compose the corresponding realizations. Let $\Phi_1 = ((A_1^1, b_1^1), \dots, (A_{L_1}^1, b_{L_1}^1))$ and $\Phi_2 = ((A_1^2, b_1^2), \dots, (A_{L_2}^2, b_{L_2}^2))$ be two NN such that the input dimension d_1 of Φ_1 is equal to the output dimension k_2 of Φ_2. The concatenation of Φ_1 and Φ_2 is defined by

$$\Phi^1 \bullet \Phi^2 = ((A_1^2, b_1^2), \dots, (A_{L_2-1}^2, b_{L_2-1}^2), (A_1^1 A_{L_2}^2, A_1^1 b_{L_2}^2 + b_1^1), (A_2^1, b_2^1), \dots, (A_{L_1}^1, b_{L_1}^1)). \tag{1.2}$$

(Be careful of the ordering! $R(\Phi^2)$ is applied before $R(\Phi^1)$ here.) Since $A_1^1(A_{L_2}^2 x + b_{L_2}^2) + b_1^1 = A_1^1 A_{L_2}^2 x + A_1^1 b_{L_2}^2 + b_1^1$, we easily check that

$$R(\Phi^1 \bullet \Phi^2) = R(\Phi_1) \circ R(\Phi_2).$$

Note that $L(\Phi^1 \bullet \Phi^2) = L(\Phi^1) + L(\Phi^2) - 1$, $N(\Phi^1 \bullet \Phi^2) = N(\Phi^1) + N(\Phi^2) - 2d_1$. Also, we have

$$\|\Phi^1 \bullet \Phi^2\|_0 \leq \|\Phi^1\|_0 + \|\Phi^2\|_0 + N_2^1(N_{L_2-1}^2 + 1) - \|A_{L_2}^2\|_0 - \|b_{L_2}^2\|_0 - \|A_1^1\|_0 - \|b_1^1\|_0. \tag{1.3}$$

Sparse concatenation The previous concatenation operator can be wasteful with respect to the complexity $\|\cdot\|_0$, because $A_1^1 A_{L_2}^2$ can have much more non-zeros than A_1^1 and $A_{L_2}^2$ (think about $A_1^1 = (1 \dots 1)^T$ and $A_{L_2}^2 = (1 \dots 1)$). Obviously, a concatenation operator which is more economic to that respect would be to simply branch the output of Φ_2 to the input of Φ_1. However, this does not directly provide the correct realization, because of the activation function. In case of the ReLU activation function, we can nevertheless propose a concatenation in this vein because the identity function can be realized by a ReLU network, see Example 1.5. This is referred to the sparse concatenation.

Consider a ReLU activation function and let $\Phi_1 = ((A_1^1, b_1^1), \dots, (A_{L_1}^1, b_{L_1}^1))$ and $\Phi_2 = ((A_1^2, b_1^2), \dots, (A_{L_2}^2, b_{L_2}^2))$ be two NN such that the input dimension d_1 of Φ_1 is equal to the output dimension k_2 of Φ_2. The sparse concatenation of Φ_1 and Φ_2 is defined by

$$\begin{aligned}&\Phi^1 \odot \Phi^2 \\ &= \left((A_1^2, b_1^2), \dots, (A_{L_2-1}^2, b_{L_2-1}^2), \left(\begin{pmatrix} A_{L_2}^2 \\ -A_{L_2}^2 \end{pmatrix}, \begin{pmatrix} b_{L_2}^2 \\ -b_{L_2}^2 \end{pmatrix}\right), ([A_1^1 \ -A_1^1], b_1^1), (A_2^1, b_2^1), \dots, (A_{L_1}^1, b_{L_1}^1)\right).\end{aligned} \tag{1.4}$$

Since for all $x \in \mathbb{R}^{L_2-1}$, we have

$$\begin{aligned}[A_1^1 \ -A_1^1]\rho\left(\begin{pmatrix} A_{L_2}^2 \\ -A_{L_2}^2 \end{pmatrix} x + \begin{pmatrix} b_{L_2}^2 \\ -b_{L_2}^2 \end{pmatrix}\right) + b_1^1 &= A_1^1((A_{L_2}^2 x + b_{L_2}^2)_+ - (-A_{L_2}^2 x - b_{L_2}^2)_+) + b_1^1 \\ &= A_1^1(A_{L_2}^2 x + b_{L_2}^2) + b_1^1,\end{aligned}$$

we have

$$R(\Phi^1 \odot \Phi^2) = R(\Phi^1) \circ R(\Phi^2),$$

as for the previous concatenation $\Phi^1 \bullet \Phi^2$. However, the complexity of $\Phi^1 \odot \Phi^2$ is somewhat different: we have $L(\Phi^1 \odot \Phi^2) = L(\Phi^1) + L(\Phi^2)$ (slightly deeper), $N(\Phi^1 \odot \Phi^2) \leq N(\Phi^1) + 2N(\Phi^2)$ (slightly larger) and $\|\Phi^1 \odot \Phi^2\|_0 \leq 2\|\Phi^1\|_0 + 2\|\Phi^2\|_0$ (potentially much smaller sparsity, compare to (1.3)).

Parallelization Another natural combination is just to parallelize network. Let $\Phi_1 = ((A_1^1, b_1^1), \dots, (A_L^1, b_L^1))$ and $\Phi_2 = ((A_1^2, b_1^2), \dots, (A_L^2, b_L^2))$ (with input dimensions d_1 and d_2) that have the same depth L. Then the parallelization of Φ_1 and Φ_2 is defined by

$$P(\Phi^1, \Phi^2) = ((\tilde{A}_1, \tilde{b}_1), \dots, (\tilde{A}_L, \tilde{b}_L)), \tag{1.5}$$

where $\tilde{A}_\ell = \begin{pmatrix} A_\ell^1 & 0 \\ 0 & A_\ell^2 \end{pmatrix}$ and $\tilde{b}_\ell = \begin{pmatrix} b_\ell^1 \\ b_\ell^2 \end{pmatrix}$ for $1 \le \ell \le L$.

In addition, when $d_1 = d_2 = d$ (same input dimension), it is possible to reduce a bit the complexity when parallelizing. In that case, the parallelization of Φ^1 and Φ^2 with share input is defined by

$$SP(\Phi^1, \Phi^2) = ((\hat{A}_1, \tilde{b}_1), (\tilde{A}_2, \tilde{b}_2), \dots, (\tilde{A}_L, \tilde{b}_L)), \tag{1.6}$$

where $\hat{A}_1 = \begin{pmatrix} A_1^1 \\ A_1^2 \end{pmatrix}$.

We easily check that we have

$$R(P(\Phi^1, \Phi^2))(x) = (R(\Phi_1)(x_1), R(\Phi_2)(x_2)), \;\; x = (x_1, x_2) \in \mathbb{R}^{d_1} \times \mathbb{R}^{d_2}$$
$$R(SP(\Phi^1, \Phi^2))(x) = (R(\Phi_1)(x), R(\Phi_2)(x)), \;\; x \in \mathbb{R}^{d_1} = \mathbb{R}^{d_2}$$

which means that the parallelization just corresponds to a coupling of the realizations. This is useful to build networks with values in a higher dimension. Note that this operation preserve the complexity cost in the following sense $\|SP(\Phi^1, \Phi^2)\|_0 = \|P(\Phi^1, \Phi^2)\|_0 = \|\Phi^1\|_0 + \|\Phi^2\|_0$, $N(SP(\Phi^1, \Phi^2)) = N(\Phi_1) + N(\Phi_2) - d$, $N(P(\Phi^1, \Phi^2)) = N(\Phi_1) + N(\Phi_2)$ and $N_{\max}(P(\Phi^1, \Phi^2)) = N_{\max}(SP(\Phi^1, \Phi^2)) \le N_{\max}(\Phi_1) + N_{\max}(\Phi_2)$.

Other combinations More generally, given Φ_1 and Φ_2 two NN, we can consider the task of building Φ such that $R(\Phi) = R(\Phi_1) * R(\Phi_2)$, for a given operation $*$, for instance addition or multiplication.

For the addition operator, this can be done by combining a (shared input) parallelization with a summation step. More precisely, for $\Phi_1 = ((A_1^1, b_1^1), \dots, (A_L^1, b_L^1))$ and $\Phi_2 = ((A_1^2, b_1^2), \dots, (A_L^2, b_L^2))$ two NN with the same depth L and same input/output dimensions d and k, consider the network

$$\Phi = ((\hat{A}_1, \tilde{b}_1), (\tilde{A}_2, \tilde{b}_2), \dots, (\tilde{A}_{L-1}, \tilde{b}_{L-1}), (U\tilde{A}_L, U\tilde{b}_L)),$$

where $U = (1 \dots 1)$ is a $1 \times k$ vector and with the notation used in (1.6). Then we easily check that $R(\Phi) = R(\Phi_1) + R(\Phi_2)$. If the depths of the two networks are different and we are working with the ReLU activation function, then one can make artificially the two depths equal by concatenating the identity network consider in Example 1.6.

For the multiplication operator, things are much more complicated: the class of MLP is not stable with respect to this operator. For instance, as we will see below, constructing $x \mapsto x^2$ from a ReLU network is only possible approximately.

1.4 Constructive approximations and role of depth

Here, we provide some elementary results already showing that deep NN are better approximations than shallow NN, for the same sparsity constraint.

As we have seen with the concatenation operator, networks are structures that behave well with respect to the composition. For instance, the composition of n realizations of NN of depth $L = 2$ can be realized with a network of depth $L = n + 1$. The point is that the "complexity" of a function composed n times can rapidly increase with n, while the depth of the network only increase linearly in n. Hence, the depth is a key parameter to build sophisticated functions with NN. We illustrate this fact below with the saw-tooth function.

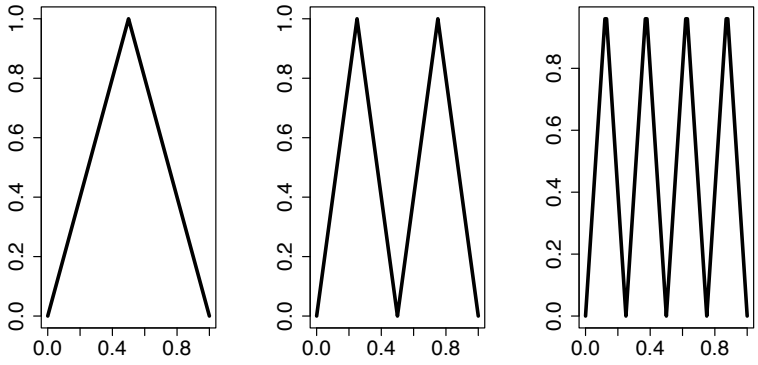

Figure 1.3: Functions T_1 (left), T_2 (center) and T_3 (right).

The saw-tooth function First, the "hat" function is defined by

$$T_1(x) = \begin{cases} 2x & x \in [0,1/2]; \\ 2-2x & x \in (1/2,1]. \end{cases}$$

Then, the saw-tooth function of order n, denoted by T_n, is defined by composing T_1 n-times, that is, $T_n = T_1 \circ \cdots \circ T_1$. It is left as an exercise that T_n has 2^{n-1} "hats", each of size 2^{-n+1}, or more formally, for all $x \in (0,1)$, for all integer $i \in \{0,\dots,2^n-1\}$,

$$T_n(x) = \begin{cases} 2^n(x - i2^{-n}) & x \in [i2^{-n},(i+1)2^{-n}] \quad i \text{ even}; \\ 2^n((i+1)2^{-n} - x) & x \in [i2^{-n},(i+1)2^{-n}] \quad i \text{ odd}. \end{cases} \tag{1.7}$$

Functions T_1, T_2 and T_3 are displayed in Figure 1.3.

Now, note that since $T_1(x) = (2x)_+ - 2(2x-1)_+ + (2x-2)_+$, the "hat" function T_1, can be realized with the NN

$$\Phi^{(1)} = ((A_1,b_1),(A_2,b_2)),\ A_1 = (2\,2\,2)^T,\ b_1 = (0\,-1\,-2)^T,\ A_2 = (1\,-2\,1),\ b_2 = 0 \tag{1.8}$$

(and the ReLU activation function). By the concatenation operation, we have $T_n = R(\Phi^{(1)}) \circ \cdots \circ R(\Phi^{(1)}) = R(\Phi^{(1)} \bullet \cdots \bullet \Phi^{(1)})$ (in which $\Phi^{(1)}$ appears n times). This implies the following result.

Proposition 1.10 (Efficient representation of the saw-tooth function)**.** *For all $n \geq 1$, the saw-tooth function T_n can be realized with a NN $\Phi^{(n)}$ of depth $L(\Phi^{(n)}) = n+1$, $N_{\max}(\Phi) = 3$ and $\|\Phi^{(n)}\|_0 \leq 12n-2$.*

Proof. Let $\Phi^{(n)} = \Phi^{(1)} \bullet \cdots \bullet \Phi^{(1)}$ (in which $\Phi^{(1)}$ appears n times). The result can be proved by a recursion over n. This is true for $n = 1$ because $L(\Phi^{(1)}) = 2$ and $\|\Phi^{(1)}\|_0 = 8$. Now, for some $n \geq 2$, assume that $\Phi^{(n-1)}$ has been built so that $L(\Phi^{(n-1)}) = n$, $N_{\max}(\Phi^{(n-1)}) = 3$, $\|\Phi^{(n-1)}\|_0 \leq 12(n-1)-2$ with the last affine operator being $((1\,-2\,1),0)$. Then $\Phi^{(n)} = \Phi^{(1)} \bullet \Phi^{(n-1)}$ and thus by the depth and $\|\cdot\|_0$ relations shown with the concatenation operator (see in particular (1.3)), $L(\Phi^{(n)}) = L(\Phi^{(1)}) + L(\Phi^{(n-1)}) - 1 = n+1$ and $\|\Phi^{(n)}\|_0 \leq \|\Phi^{(1)}\|_0 + \|\Phi^{(n-1)}\|_0 + 3(3+1) - 3 - 0 - 3 - 2 = 8 + 12(n-1) + 12 - 2 - 8 = 12n - 2$. □

Proposition 1.10 can already underline the role of the depth. Consider a shallow NN Φ, as in (1.3), with $s = \|\Phi\|_0$. It is clear that $R(\Phi)$ has at most s affine pieces. But the saw-tooth NN $\Phi^{(n)}$ of order $n = \lfloor (s+2)/12 \rfloor$ has a complexity $\|\Phi^{(n)}\|_0 \leq s$, and realizes the function T_n with $2^n = 2^{\lfloor (s+2)/12 \rfloor} \gg s$

affine pieces. Hence, considering a deep NN allows to generate much more affine pieces. In fact, the saw-tooth function is nearly unbeatable at this game, because a ReLU NN can only generate a limited number of affine pieces, as the next result shows.

Lemme 1.11. *[Limit of expressiveness of RELU NN] For an integer $L \geq 2$, if ρ is piecewise affine with p pieces. Then, for every NN Φ with $d = 1$, $k = 1$, $L(\Phi) = L$ and $N_1, \dots, N_{L-1} \leq N$, we have that $R(\Phi)$ has at most $(pN)^{L-1}$ affine pieces. In particular, a ReLU MLP of input dimension $d = 1$, depth L and with N maximum neurons on each layer has a most $(2N)^{L-1}$ affine pieces.*

Proof. Quite elementary by a recursion on L, see the proof of Theorem 4.3 in [Pet22] for more details. □

By Proposition 1.10, the saw-tooth function can be realized with $\Phi^{(n)}$, a NN of depth $L = n+1$ and maximum width $N_{\max} = 3$, and has $2^n = 2^{L-1}$ affine pieces (on $[0,1]$). Hence it qualitatively achieves the upper bound of Lemma 1.11 (2^{L-1} instead of 6^{L-1}). Hence, the saw-tooth function somewhat exploits all the flexibility offered by the depth of a ReLU NN to generate complex functions, at least when the compexity is measured with the number of affine pieces.

Approximation of non-affine functions The interest of generating complex functions with deep ReLU NN is that we can correctly approximate functions outside the class of piecewise affine functions by keeping the NN somewhat sparse. Here, we exemplify this general principle with the square function $x \in \mathbb{R} \mapsto x^2$. More examples will be provided in the next chapters. The saw-tooth function is (again !) useful with the following result.

Proposition 1.12. *Consider the functions $(T_n, n \geq 1)$ defined by (1.7). Then for all integer $N \geq 0$, we have*

$$\sup_{x\in[0,1]} \left| x^2 - x + \sum_{n=1}^{N} 2^{-2n} T_n(x) \right| \leq 2^{-2N-2}.$$

The proof is based on the fact that $H_N : x \in [0,1] \mapsto x - \sum_{n=1}^{N} 2^{-2n} T_n(x)$ is a piecewise affine function with breakpoints $k2^{-N}$, $K = 0, \dots, 2^N$ and that coincides with the square function on each of these breakpoints. The result then follows by maximizing $|x^2 - H_N(x)| = H_N(x) - x^2$ between two successive breakpoints, which is easy because it reduces to a polynomial function of degree 2. The complete proof can be found in [Pet22], see Proposition 3.8 therein.

Proposition 1.12 then entails the following approximation of the square function.

Proposition 1.13. *Let $\epsilon \in (0, 1/2)$. There exists a ReLU NN Φ^ϵ such that*

$$\sup_{x\in[0,1]} |R(\Phi^\epsilon)(x) - x^2| \leq \epsilon,$$

while $L(\Phi^\epsilon) \leq 3\log_2(1/\epsilon)$, $N_{\max}(\Phi^\epsilon) \leq 6\log_2(1/\epsilon)$, $\|\Phi^\epsilon\|_0 \leq 26\log_2^2(1/\epsilon)$.

Proof. Choose $N = \lceil \log_2(1/\epsilon)/2 \rceil$, which is an integer $1 \leq N \leq \log_2(1/\epsilon)$ such that $2^{-2N-2} \leq \epsilon$. Since by Proposition 1.12 we have $\sup_{x\in[0,1]} |x^2 - x + \sum_{n=1}^{N} 2^{-2n} T_n(x)| \leq \epsilon$, we only have to realize $x \in [0,1] \mapsto x - \sum_{n=1}^{N} 2^{-2n} T_n(x)$ with a network Φ^ϵ of the appropriate complexity. First note that this can be easily done if the last hidden layer of the NN outputs the vector $(x, T_1(x), \dots, T_N(x)) \in \mathbb{R}^{N+1}$, since all the elements of this vector are non-negative and since we can add $((1 - 1/4 \dots, 2^{-2N}), 0)$ as the last layer of Φ^ϵ to realize the desired function.

Let us thus consider the task of realizing

$$x \in [0,1] \mapsto (x, T_1(x), \dots, T_N(x)) \in \mathbb{R}^{N+1}.$$

First, by Proposition 1.10, for $1 \le n \le N$, the saw-tooth function T_n can be realized with a NN $\Phi^{(n)}$ of depth $L(\Phi^{(n)}) = n+1$, maximum width 3 and $\|\Phi^{(n)}\|_0 \le 12n-2$. Second, note that $x \in [0,1] \mapsto x_+$ is the identity function here, so by adding $N-n$ layers $(1,0)$ to the output of $\Phi^{(n)}$, we obtain new NN, denoted $\Phi_N^{(n)}$, with $T_n = R(\Phi_N^{(n)})$, $L(\Phi_N^{(n)}) = N+1$, $N_{\max}(\Phi_N^{(n)}) \le 3$, $\|\Phi_N^{(n)}\|_0 \le 12n-2+N-n \le 12N$, $1 \le n \le N$. Also, for $n=0$, we can let $\Phi_N^{(0)} = \Phi_{1,N+1}^{Id}$ (see (1.1)) that satisfies $R(\Phi^{(0)})(x) = x$ and the above points. Since $\Phi_N^{(n)}$, $0 \le n \le N$ have all the same depth, we can parallelize these networks (which a shared input) and consider $\Phi' = SP(\Phi_N^{(n)}, 0 \le n \le N)$, with $L(\Phi') = N+1$, $\|\Phi'\|_0 \le 12N(N+1)$, $N_{\max}(\Phi') \le 3(N+1)$ and $R(\Phi') : x \in [0,1] \mapsto (x, T_1(x), \dots, T_N(x))$. Finally, this leads to the NN Φ^ϵ, which is such that $L(\Phi^\epsilon) = N+2$, $\|\Phi^\epsilon\|_0 \le 12N(N+1)+N+1$, $N_{\max}(\Phi^\epsilon) \le 3(N+1)$. □

Proposition 1.13 ensures that there exists a NN approximating the square function by an error of at most ϵ and which as depth of order $\log_2(1/\epsilon)$. This can be seen as an upper bound for the NN depth for a given accuracy ϵ. Interestingly, the next result shows a corresponding lower bound.

Proposition 1.14. *For any $f \in C^3([0,1],\mathbb{R})$ which is not affine there exists a given constant $c = c(f)$ such that the following holds: for any ReLU NN Φ with depth $L \ge 2$, and $N_{\max}(\Phi) \le N_{\max}$, we have*

$$\sup_{x\in[0,1]} |f(x) - R(\Phi)(x)| \ge c(2N_{\max})^{-2(L-1)}.$$

The proof is based on the following lemma, which can be deduced from Theorem 1 in [FSB10].

Lemme 1.15. *For any $f \in C^3([0,1],\mathbb{R})$ which is not affine there exists a given constant $c = c(f)$ such that for all $p \ge 1$, $\inf_g \|f-g\|_\infty > cp^{-2}$, where the infimum is taken over piecewise affine functions with at most p pieces.*

Proof. First, since f belongs to $C^3([0,1],\mathbb{R})$ and is not affine, Lemma 1.15 entails that there exists a constant $c = c(f)$ such that $\sup_{x\in[0,1]} |f(x)-g(x)| \ge cp^{-2}$, for all function g that are piecewise affine with at most p pieces. Now, by Lemma 1.11, $R(\Phi)$ is piecewise affine with at most $(2N_{\max})^{L-1}$ pieces. The two above facts imply the results. □

Proposition 1.14 underline the role of the depth of a network: for approximating some regular function f with accuracy ϵ, one needs to choose a NN with depth L with $c(2N_{\max})^{-2(L-1)} \le \epsilon$, that is,

$$L \ge 1 + 0.5\frac{\log_2(c/\epsilon)}{\log_2(2N_{\max})}.$$

Hence, only sufficiently deep NN can approximate well a smooth function. In addition, for f being the square function, we can compare the order of the above depth lower bound to the previous upper bound. For $N_{\max}$ of order $\log_2(\epsilon)$ (as in Proposition 1.13), we obtain a lower bound of order $\frac{\log_2(\epsilon)}{\log_2(\log_2(\epsilon))}$, which coincide with the depth order $\log_2(\epsilon)$ found in Proposition 1.13 up to a loglog factor.

At this point of the lecture, the picky reader could ask why should we bother to approximate regular functions with NN, since we can use the regular functions in the first place. In fact, a great interest of NN is to approximate with only few parameters certain interesting class of functions in high dimension, see Chapter 5 [Pet22].

References

[Cyb89] George Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, 2(4):303–314, 1989.

[FSB10] Christopher L Frenzen, Tsutomu Sasao, and Jon T Butler. On the number of segments needed in a piecewise linear approximation. *Journal of Computational and Applied mathematics*, 234(2):437–446, 2010.

[GKNV21] Rémi Gribonval, Gitta Kutyniok, Morten Nielsen, and Felix Voigtlaender. Approximation spaces of deep neural networks. *Constructive Approximation*, pages 1–109, 2021.

[HSW89] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366, 1989.

[MP43] Warren S McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133, 1943.

[Pet22] Philipp Petersen. Neural network theory. http://www.pc-petersen.eu/Neural_Network_Theory.pdf, 2022.

[Ros58] Frank Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.