Chapter 2

Feedforward neural networks

Feedforward neural networks, henceforth simply referred to as neural networks (NNs), constitute the central object of study of this book. In this chapter, we provide a formal definition of neural networks, discuss the *size* of a neural network, and give a brief overview of common activation functions.

2.1 Formal definition

We previously defined a single neuron ν in (1.2.1) and Figure 1.1. A neural network is constructed by connecting multiple neurons. Let us now make precise this connection procedure.

Definition 2.1. Let $L \in \mathbb{N}$, $d_0, \dots, d_{L+1} \in \mathbb{N}$, and let $\sigma \colon \mathbb{R} \to \mathbb{R}$. A function $\Phi \colon \mathbb{R}^{d_0} \to \mathbb{R}^{d_{L+1}}$ is called a **neural network** if there exist matrices $\boldsymbol{W}^{(\ell)} \in \mathbb{R}^{d_{\ell+1} \times d_\ell}$ and vectors $\boldsymbol{b}^{(\ell)} \in \mathbb{R}^{d_{\ell+1}}$, $\ell = 0, \dots, L$, such that with

$$\boldsymbol{x}^{(0)} := \boldsymbol{x} \tag{2.1.1a}$$

$$\boldsymbol{x}^{(\ell)} := \sigma(\boldsymbol{W}^{(\ell-1)}\boldsymbol{x}^{(\ell-1)} + \boldsymbol{b}^{(\ell-1)}) \qquad \text{for } \ell \in \{1, \dots, L\} \tag{2.1.1b}$$

$$\boldsymbol{x}^{(L+1)} := \boldsymbol{W}^{(L)}\boldsymbol{x}^{(L)} + \boldsymbol{b}^{(L)} \tag{2.1.1c}$$

holds

$$\Phi(\boldsymbol{x}) = \boldsymbol{x}^{(L+1)} \qquad \text{for all } \boldsymbol{x} \in \mathbb{R}^{d_0}.$$

We call L the **depth**, $d_{\max} = \max_{\ell=1,\dots,L} d_\ell$ the **width**, σ the **activation function**, and $(\sigma; d_0, \dots, d_{L+1})$ the **architecture** of the neural network Φ. Moreover, $\boldsymbol{W}^{(\ell)} \in \mathbb{R}^{d_{\ell+1} \times d_\ell}$ are the **weight matrices** and $\boldsymbol{b}^{(\ell)} \in \mathbb{R}^{d_{\ell+1}}$ the **bias vectors** of Φ for $\ell = 0, \dots L$.

Remark 2.2. Typically, there exist different choices of architectures, weights, and biases yielding the same function $\Phi : \mathbb{R}^{d_0} \to \mathbb{R}^{d_{L+1}}$. For this reason we cannot associate a unique meaning to these notions solely based on the *function* realized by Φ. In the following, when we refer to the properties

of a neural network Φ, it is always understood to mean that there exists at least one construction as in Definition 2.1, which realizes the function Φ and uses parameters that satisfy those properties.

The architecture of a neural network is often depicted as a connected graph, as illustrated in Figure 2.1. The **nodes** in such graphs represent (the output of) the neurons. They are arranged in **layers**, with $\boldsymbol{x}^{(\ell)}$ in Definition 2.1 corresponding to the neurons in layer ℓ. We also refer to $\boldsymbol{x}^{(0)}$ in (2.1.1a) as the **input layer** and to $\boldsymbol{x}^{(L+1)}$ in (2.1.1c) as the **output layer**. All layers in between are referred to as the **hidden layers** and their output is given by (2.1.1b). The number of hidden layers corresponds to the depth. For the correct interpretation of such graphs, we note that by our conventions in Definition 2.1, the activation function is applied after each affine transformation, except in the final layer.

With our convention, the depth of the neural network corresponds to the number of applications of the activation function. Neural networks of depth one are called **shallow**, if the depth is larger than one they are called **deep**. The notion of deep neural networks is not used entirely consistently in the literature, and some authors use the word deep only in case the depth is much larger than one, where the precise meaning of "much larger" depends on the application.

Throughout, we only consider neural networks in the sense of Definition 2.1. We emphasize however, that this is just one (simple but very common) type of neural network. Many adjustments to this construction are possible and also widely used. For example:

- We may use **different activation functions** σ_ℓ in each layer ℓ or we may even use a different activation function for each node.
- **Residual** neural networks allow "skip connections". This means that information is allowed to skip layers in the sense that the nodes in layer ℓ may have $\boldsymbol{x}^{(0)}, \dots, \boldsymbol{x}^{(\ell-1)}$ as their input (and not just $\boldsymbol{x}^{(\ell-1)}$), cf. (2.1.1).
- In contrast to feedforward neural networks, **recurrent** neural networks allow information to flow backward, in the sense that $\boldsymbol{x}^{(\ell-1)}, \dots, \boldsymbol{x}^{(L+1)}$ may serve as input for the nodes in layer ℓ (and not just $\boldsymbol{x}^{(\ell-1)}$). This creates loops in the flow of information, and one has to introduce a time index $t \in \mathbb{N}$, as the output of a node in time step t might be different from the output in time step $t+1$.

Let us clarify some further common terminology used in the context of neural networks:

- **parameters**: The parameters of a neural network refer to the set of all entries of the weight matrices and bias vectors. These are often collected in a single vector

$$\boldsymbol{w} = ((\boldsymbol{W}^{(0)}, \boldsymbol{b}^{(0)}), \dots, (\boldsymbol{W}^{(L)}, \boldsymbol{b}^{(L)})). \tag{2.1.2}$$

  These parameters are adjustable and are learned during the training process, determining the specific function realized by the network.

- **hyperparameters**: Hyperparameters are settings that define the network's architecture (and training process), but are not directly learned during training. Examples include the depth, the number of neurons in each layer, and the choice of activation function. They are typically set before training begins.
- **weights**: The term "weights" is often used broadly to refer to *all* parameters of a neural network, including both the weight matrices and bias vectors.

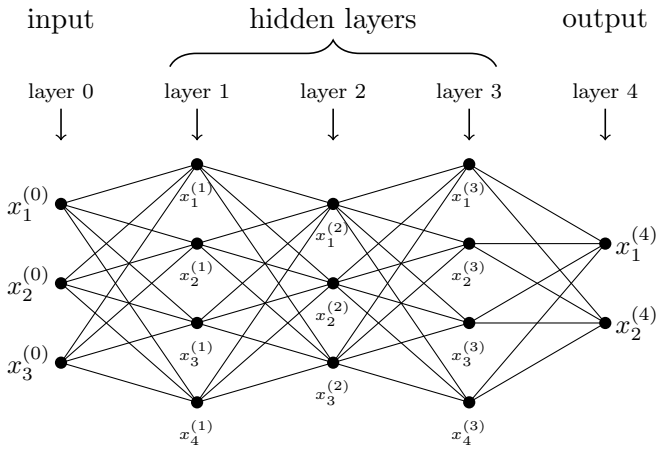


Figure 2.1: Sketch of a neural network with three hidden layers, and $d_0 = 3$, $d_1 = 4$, $d_2 = 3$, $d_3 = 4$, $d_4 = 2$. The neural network has depth three and width four.

- **model**: For a fixed architecture, every choice of network parameters $\boldsymbol{w}$ as in (2.1.2) defines a specific function $\boldsymbol{x} \mapsto \Phi_{\boldsymbol{w}}(\boldsymbol{x})$. In deep learning this function is often referred to as a model. More generally, "model" can be used to describe any function parameterization by a set of parameters $\boldsymbol{w} \in \mathbb{R}^n$, $n \in \mathbb{N}$.

2.1.1 Basic operations on neural networks

There are various ways how neural networks can be combined with one another. The next proposition addresses this for linear combinations, compositions, and parallelization. The formal proof, which is a good exercise to familiarize onself with neural networks, is left as Exercise 2.5.

Proposition 2.3. *For two neural networks Φ_1, Φ_2, with architectures*

$$(\sigma; d_0^1, d_1^1, \dots, d_{L_1+1}^1) \quad \text{and} \quad (\sigma; d_0^2, d_1^2, \dots, d_{L_2+1}^2)$$

respectively, it holds that

(i) for all $\alpha \in \mathbb{R}$ exists a neural network Φ_α with architecture $(\sigma; d_0^1, d_1^1, \dots, d_{L_1+1}^1)$ such that

$$\Phi_\alpha(\boldsymbol{x}) = \alpha \Phi_1(\boldsymbol{x}) \qquad \text{for all } \boldsymbol{x} \in \mathbb{R}^{d_0^1},$$

(ii) if $d_0^1 = d_0^2 =: d_0$ and $L_1 = L_2 =: L$, then there exists a neural network Φ_{parallel} with architecture $(\sigma; d_0, d_1^1 + d_1^2, \dots, d_{L+1}^1 + d_{L+1}^2)$ such that

$$\Phi_{\text{parallel}}(\boldsymbol{x}) = (\Phi_1(\boldsymbol{x}), \Phi_2(\boldsymbol{x})) \qquad \text{for all } \boldsymbol{x} \in \mathbb{R}^{d_0},$$

(iii) if $d_0^1 = d_0^2 =: d_0$, $L_1 = L_2 =: L$, and $d_{L+1}^1 = d_{L+1}^2 =: d_{L+1}$, then there exists a neural network Φ_{sum} with architecture $(\sigma; d_0, d_1^1 + d_1^2, \dots, d_L^1 + d_L^2, d_{L+1})$ such that

$$\Phi_{\text{sum}}(\boldsymbol{x}) = \Phi_1(\boldsymbol{x}) + \Phi_2(\boldsymbol{x}) \qquad \text{for all } \boldsymbol{x} \in \mathbb{R}^{d_0},$$

(iv) if $d^1_{L_1+1} = d^2_0$, *then there exists a neural network* Φ_{comp} *with architecture* $(\sigma; d^1_0, d^1_1, \ldots, d^1_{L_1}, d^2_1, \ldots, d^2_{L_2+1})$ *such that*

$$\Phi_{\text{comp}}(\boldsymbol{x}) = \Phi_2 \circ \Phi_1(\boldsymbol{x}) \qquad \text{for all } \boldsymbol{x} \in \mathbb{R}^{d^1_0}.$$

2.2 Notion of size

Neural networks provide a framework to parametrize functions. Ultimately, our goal is to find a neural network that fits some underlying input-output relation. As mentioned above, the architecture (depth, width and activation function) is typically chosen apriori and considered fixed. During training of the neural network, its parameters (weights and biases) are suitably adapted by some algorithm. Depending on the application, on top of the stated architecture choices, further restrictions on the weights and biases can be desirable. For example, the following two appear frequently:

- **weight sharing**: This is a technique where specific entries of the weight matrices (or bias vectors) are constrained to be equal. Formally, this means imposing conditions of the form $W^{(i)}_{k,l} = W^{(j)}_{s,t}$, i.e. the entry (k,l) of the ith weight matrix is equal to the entry at position (s,t) of weight matrix j. We denote this assumption by $(i,k,l) \sim (j,s,t)$, paying tribute to the trivial fact that "$\sim$" is an equivalence relation. During training, shared weights are updated jointly, meaning that any change to one weight is simultaneously applied to all other weights of this class. Weight sharing can also be applied to the entries of bias vectors.

- **sparsity**: This refers to imposing a sparsity structure on the weight matrices (or bias vectors). Specifically, we apriorily set $W^{(i)}_{k,l} = 0$ for certain (k,l,i), i.e. we impose entry (k,l) of the ith weight matrix to be 0. These zero-valued entries are considered fixed, and are not adjusted during training. The condition $W^{(i)}_{k,l} = 0$ corresponds to node l of layer $i-1$ *not* serving as an input to node k in layer i. If we represent the neural network as a graph, this is indicated by not connecting the corresponding nodes. Sparsity can also be imposed on the bias vectors.

Both of these restrictions decrease the number of learnable parameters in the neural network. The number of parameters can be seen as a measure of the complexity of the represented function class. For this reason, we introduce $\text{size}(\Phi)$ as a notion for the number of learnable parameters. Formally (with $|S|$ denoting the cardinality of a set S):

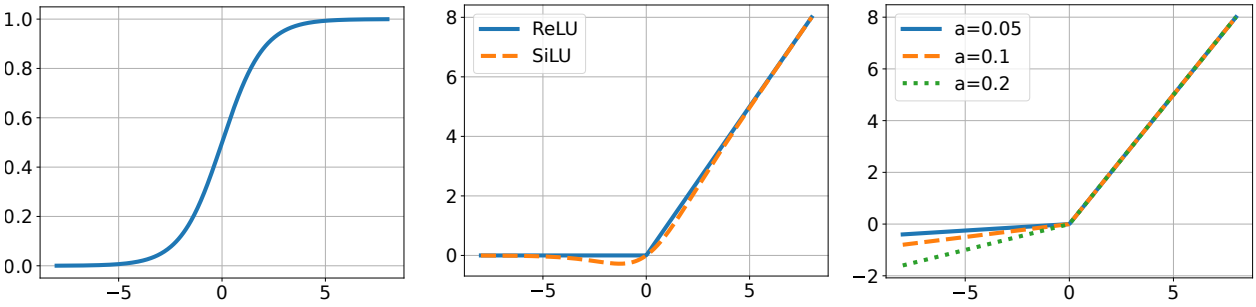
Definition 2.4. Let Φ be as in Definition 2.1. Then the **size** of Φ is

$$\text{size}(\Phi) := \left| \left(\{(i,k,l) \,|\, W^{(i)}_{k,l} \neq 0\} \cup \{(i,k) \,|\, b^{(i)}_k \neq 0\} \right) / \sim \right|. \tag{2.2.1}$$

2.3 Activation functions

Activation functions are a crucial part of neural networks, as they introduce nonlinearity into the model. If an affine activation function were used, the resulting neural network function would also be affine and hence very restricted in what it can represent.

The choice of activation function can have a significant impact on the performance, but there does not seem to be a universally optimal one. We next discuss a few important activation functions and highlight some common issues associated with them.



(a) Sigmoid (b) ReLU and SiLU (c) Leaky ReLU

Figure 2.2: Different activation functions.

Sigmoid: The sigmoid activation function is given by

$$\sigma_{\text{sig}}(x) = \frac{1}{1+e^{-x}} \qquad \text{for } x \in \mathbb{R},$$

and depicted in Figure 2.2 (a). Its output ranges between zero and one, making it interpretable as a probability. The sigmoid is a smooth function, which allows the application of gradient-based training.

It has the disadvantage that its derivative becomes very small if $|x| \to \infty$. This can affect learning due to the so-called vanishing gradient problem. Consider the simple neural network $\Phi_n(x) = \sigma \circ \cdots \circ \sigma(x+b)$ defined with $n \in \mathbb{N}$ compositions of σ, and where $b \in \mathbb{R}$ is a bias. Its derivative with respect to b is

$$\frac{\mathrm{d}}{\mathrm{d}b}\Phi_n(x) = \sigma'(\Phi_{n-1}(x))\frac{\mathrm{d}}{\mathrm{d}b}\Phi_{n-1}(x).$$

If $\sup_{x\in\mathbb{R}}|\sigma'(x)| \leq 1-\delta$, then by induction, $|\frac{\mathrm{d}}{\mathrm{d}b}\Phi_n(x)| \leq (1-\delta)^n$. The opposite effect happens for activation functions with derivatives uniformly larger than one. This argument shows that the derivative of $\Phi_n(x,b)$ with respect to b can become exponentially small or exponentially large when propagated through the layers. This effect, known as the *vanishing- or exploding gradient effect*, also occurs for activation functions which do not admit the uniform bounds assumed above. However, since the sigmoid activation function exhibits areas with extremely small gradients, the vanishing gradient effect can be strongly exacerbated.

ReLU (Rectified Linear Unit): The ReLU is defined as

$$\sigma_{\text{ReLU}}(x) = \max\{x, 0\} \qquad \text{for } x \in \mathbb{R},$$

and depicted in Figure 2.2 (b). It is piecewise linear, and due to its simplicity its evaluation is computationally very efficient. It is one of the most popular activation functions in practice. Since its derivative is always zero or one, it does not suffer from the vanishing gradient problem to the same extent as the sigmoid function. However, ReLU can suffer from the so-called *dead neurons* problem. Consider the neural network

$$\Phi(x) = \sigma_{\mathrm{ReLU}}(b - \sigma_{\mathrm{ReLU}}(x)) \qquad \text{for } x \in \mathbb{R}$$

depending on the bias $b \in \mathbb{R}$. If $b < 0$, then $\Phi(x) = 0$ for all $x \in \mathbb{R}$. The neuron corresponding to the second application of σ_{ReLU} thus produces a constant signal. Moreover, if $b < 0$, $\frac{\mathrm{d}}{\mathrm{d}b}\Phi(x) = 0$ for all $x \in \mathbb{R}$. As a result, every negative value of b yields a stationary point of the empirical risk. A gradient-based method will not be able to further train the parameter b. We thus refer to this neuron as a dead neuron.

SiLU (Sigmoid Linear Unit): An important difference between the ReLU and the Sigmoid is that the ReLU is not differentiable at 0. The SiLU activation function (also referred to as "swish") can be interpreted as a smooth approximation to the ReLU. It is defined as

$$\sigma_{\mathrm{SiLU}}(x) := x\sigma_{\mathrm{sig}}(x) = \frac{x}{1 + e^{-x}} \qquad \text{for } x \in \mathbb{R},$$

and is depicted in Figure 2.2 (b). There exist various other smooth activation functions that mimic the ReLU, including the Softplus $x \mapsto \log(1 + \exp(x))$, the GELU (Gaussian Error Linear Unit) $x \mapsto xF(x)$ where $F(x)$ denotes the cumulative distribution function of the standard normal distribution, and the Mish $x \mapsto x \tanh(\log(1 + \exp(x)))$.

Parametric ReLU or Leaky ReLU: This variant of the ReLU addresses the dead neuron problem. For some $a \in (0, 1)$, the parametric ReLU is defined as

$$\sigma_a(x) = \max\{x, ax\} \qquad \text{for } x \in \mathbb{R},$$

and is depicted in Figure 2.2 (c) for three different values of a. Since the output of σ does not have flat regions like the ReLU, the dying ReLU problem is mitigated. If a is not chosen too small, then there is less of a vanishing gradient problem than for the Sigmoid. In practice the additional additional parameter a has to be fine-tuned depending on the application. Like the ReLU, the parametric ReLU is not differentiable at 0.

Bibliography and further reading

The concept of neural networks was first introduced by McCulloch and Pitts in [140]. Later Rosenblatt [190] introduced the perceptron, an artificial neuron with adjustable weights that forms the basis of the multilayer perceptron (a fully connected feedforward neural network). The vanishing gradient problem shortly addressed in Section 2.3 was discussed by Hochreiter in his diploma thesis [90] and later in [17, 92].

For a historical survey on neural networks see [201] and also [125]. For general textbooks on neural networks we refer to [83, 3], with the latter focusing on theoretical aspects. Also see [71, 180] for more recent monographs. For the implementation of neural networks we refer for example to [66, 37].

Exercises

Exercise 2.5. Prove Proposition 2.3.

Exercise 2.6. In this exercise, we show that ReLU and parametric ReLU create similar sets of neural network functions. Fix $a > 0$.

(i) Find a set of weight matrices and biases vectors, such that the associated neural network Φ_1, with the ReLU activation function σ_{ReLU} satisfies $\Phi_1(x) = \sigma_a(x)$ for all $x \in \mathbb{R}$.

(ii) Find a set of weight matrices and biases vectors, such that the associated neural network Φ_2, with the parametric ReLU activation function σ_a satisfies $\Phi_2(x) = \sigma_{\mathrm{ReLU}}(x)$ for all $x \in \mathbb{R}$.

(iii) Conclude that every ReLU neural network can be expressed as a leaky ReLU neural network and vice versa.

Exercise 2.7. Let $d \in \mathbb{N}$, and let Φ_1 be a neural network with the ReLU as activation function, input dimension d, and output dimension 1. Moreover, let Φ_2 be a neural network with the sigmoid activation function, input dimension d, and output dimension 1. Show that, if $\Phi_1 = \Phi_2$, then Φ_1 is a constant function.

Exercise 2.8. In this exercise, we show that for the sigmoid activation functions, dead-neuron-like behavior is very rare. Let Φ be a neural network with the sigmoid activation function. Assume that Φ is a constant function. Show that for every $\varepsilon > 0$ there is a non-constant neural network $\widetilde{\Phi}$ with the same architecture as Φ such that for all $\ell = 0, \dots L$,

$$\|\boldsymbol{W}^{(\ell)} - \widetilde{\boldsymbol{W}}^{(\ell)}\| \leq \varepsilon \text{ and } \|\boldsymbol{b}^{(\ell)} - \widetilde{\boldsymbol{b}}^{(\ell)}\| \leq \varepsilon$$

where $\boldsymbol{W}^{(\ell)}$, $\boldsymbol{b}^{(\ell)}$ are the weights and biases of Φ and $\widetilde{\boldsymbol{W}}^{(\ell)}$, $\widetilde{\boldsymbol{b}}^{(\ell)}$ are the biases of $\widetilde{\Phi}$.

Show that such a statement does not hold for ReLU neural networks. What about leaky ReLU?