Chapter 1

Introduction to Neural Networks

Let us start this dissertation by introducing the key components of an artificial neural network and discussing the way it can be configured for a specific application. Please note that this chapter is not meant to provide a comprehensive overview on neural networks, rather to investigate some aspects and concepts functional to the following chapters. For further reading, we refer the reader to, e.g., [1, 2, 3], from which we retrieved many of the informations provided in this chapter.

Throughout this work we confine the attention to the most-spread neural network paradigm - the *feedforward* neural network, presented in Section **??** - employing the well-known *backpropagation of error* as learning rule, derived in Section **??**. Actually, in the numerical experiments we carried out and whose results will be discussed in Chapter **??**, we mainly refer to a variant of backpropagation - the Levenberg-Marquardt algorithm [], shortly discussed in Section **??**.

Before moving to the description of technical neural networks, let us provide a brief excursus on their biological counterparts. The goal is to highlight the basic features of the human nervous system, focusing on the working principles of neurons and the way informations are processed, thus to extract the key concepts which should be taken over into a mathematical, simplified representation.

1.1 Biological motivation

The information processing system of a vertebrate can coarsely be divided into the *central nervous system* (CNS) and the *peripheral nervous system* (PNS). The former consists of the *brain* and the *spinal cord*, while the latter mainly comprises the *nerves*, which transmit informations from all other parts of the body to the CNS (*sensory nerves*) and viceversa (*motor nerves*). When an output stimulus hits the sensory cells of an organ sense, these generate an electric signal, called *sensory signal*, which is transfered to the central nervous system via the *sensory nerves*. Within the CNS, informations are stored and managed to provide the muscles with a suitable *motor signal*, broadcast through the *motor nerves* and finally converted by the effectors into a system output [2].

Hence, both the central and peripheral nervous system are directly involved in the information processing workflow. At the cellular level, this is accomplished through a huge amount of modified cells called *neurons*. These processing elements continuosly communicate each other by means of electric signals, traveling through a thick net of connections. For instance, in a human being each neuron is linked in average with $10^3 - 10^4$ other neurons. As detailed in the next paragraph, a neuron is characterized by a rather simple structure, specifically designed to rapidly collect input signals and generate an output pulse whenever the accumulated incoming signal exceeds a threshold - the *action potential*. In other terms, a neuron acts as a switch, establishing a typically nonlinear input-output mapping [3].

From a simplifying perspective, a neuron consists of three main components: the *dendrites*, the *nucleus* or *soma*, and the *axon*. Dendrites are tree-like networks of nerve fibers receiving input signals from many sources and conveying them directly to the nucleus of the neuron. Here, input signals are accumulated

and thresholded, as mentioned before. The possible output pulse is then broadcast to the cells contiguous the neuron through the axon - a unique, slender fiber constituing an extension of the soma and splitting in many branches at the opposite extremity [5]. To ease the electrical conduction of the signal, the axon is isolated through a myelin sheath which consists of Schwann cells (in the PNS) or oligodendrocytes (in the CNS). However, this insulating film is not continuous, rather presents gaps at regular intervals called *nodes of Ranvier*, which lets the signal be conducted in a saltatory way.

The signal coming from the axon of another neuron or from another cell is transfered to the dendrites of a neuron through a particular connection called *synapsis*[1]. A synaptic may be either electrical or chemical. In the former, the presynaptic side, i.e. the sender axon, and the postsynapic side, i.e. the receiver dendrite, are directly in contact, so that the potential can simply travel by electrical conduction. Conversely, a chemical synapsis consists of a synaptic *cleft*, physically separating the presynaptic side from the postsynaptic side. Then, to let the action potential reach the postsynaptic side, at the presynaptic side the electrical pulse is converted into a chemical signal. This is accomplished by releasing some chemical substances called *neurotransmitters*. These neurotransmitters then cross the cleft and bind to the receptors dislocated onto the membrane of the postsynaptic side, where the chemical signal is re-converted into an electrical potential. On the other hand, neurotransmitters do not simply broadcast the action potential. Indeed, we can distinguish between excitatory and inhibitory neurotransmitters, respectively amplifying or modulating the signal. Hence, the pulse outgoing a neuron is preprocessed within the synapsis before reaching the target cell. In other terms, a neuron gets in input many *weighted* signals, which should then be collected.

Different studies have unveiled the tight correlation between the synapses the neurons establish among each other, and the tasks a neural network can address [1]. That is, the set of interneuron connection strengths represent the information storage, i.e. the knowledge, of a neural network [3]. Knowledge is acquired through a *learning* or *training* process, entailing adjustments at the synaptic level to adapt to environmental situations. The adjustments may not only involve the modification of existing synapses, but also the creation of new synaptic connections. Hence, the nervous system is a distributed memory machine whose evolutionary structure is shaped by experience.

As mentioned above, a biological neural network acquaints itself with problems of a specific class through a learning procedure. During the learning, the network is exposed to a collection of situations, giving it the possiblity to derive a set of tools which will let it provide reasonable solutions in similar circumstances. In other terms, the cognitive system should be able to *generalize*. Furthermore, after a successfull training a neural network should also show a discrete level of *fault tolerance* against external errors, e.g. noisy inputs. It worths notice here that the nervous system is also naturally fault tolerant against *internal* errors. Indeed, in case a neuron or a (relatively small) group of neurons got damaged or died, the other processing nodes would take care of its tasks, so that the overall cognitive capabilities would be only slightly affected [3].

1.2 Artificial neural networks

Inspired by the biological information processing system discussed so far, an artificial neural network (ANN), usually simply referred to as "neural network", is a computational model capable to learn from observational data, i.e. by example, thus providing an alternative to the algorithmic programming paradigm [4]. Exactly as its original counterpart, it consists of a collection of processing units, called (artificial) neurons, and directed weighted synaptic connections between the neurons themselves. Data travel among neurons through the connections, following the direction imposed by the synapses themselves. Hence, an artificial neural network is an *oriented graph* to all intents and purposes, with the neurons as *nodes* and the synapses as oriented *edges*, whose weights are adjusted by means of a *training* process to configure the network for a specific application [5].

[1] For the sake of completeness, we mention that there exist synapses directly connecting the axon of the sender neuron with either the soma or the axon of the receiver. Actually, a synapsis may also connect the axon of a neuron with the dendrite or soma of the same neuron (autosynapsis). However, for our purposes we can confine the attention to the axon-dendrite synapsis.

Formally, a neural network is a sorted triple $(\mathcal{N}, \mathcal{V}, w)$, where $\mathcal{N}$ is the set of *neurons*, with cardinality $|\mathcal{N}|$, $\mathcal{V} = \{(i, j) : i, j = 1, \dots, |\mathcal{N}|\}$ is the set of *connections*, with (i, j) denoting the oriented connection linking the sending neuron i with the target neuron j, and $w : \mathcal{V} \to \mathbb{R}$ is the *weight function*, defining the weight $w_{i,j} = w((i, j))$ of the connection (i, j) [3]. We point out that a weight may be either positive or negative, making the underlying connection either excitatory or inhibitory, respectively. By convention, $w_{i,j} = 0$ means that neurons i and j are not directly connected.

In the following, we dive deeper into the structure and training of a neural network, starting by detailing the structure of an artificial neuron.

1.2.1 Neuronal model

As its name may suggest, an artificial neuron represents a simplified model of a biological neuron, retaining its main features discussed in Section **??**. To introduce the components of the model, let us consider the neuron j represented in Figure **??**. Suppose that it is connected with m sending neurons $s_1, \dots, s_m$, and n receiving (target) neurons $r_1, \dots, r_n$. Denoting by $y_\Omega(t) \in \mathbb{R}$ the scalar output fired by a generic neuron Ω at time t, neuron j gets the weighted inputs $w_{s_k,j} \cdot y_{s_k}(t)$, $k = 1, \dots, m$, at time t, and sends out the output $y_j(t + \Delta t)$ to the target neurons $r_1, \dots, r_n$ at time $t + \Delta t$. Please note that in the context of artificial neural networks the time is discretized by introducing the timestep Δt. This is clearly not plausible from a biological viewpoint; on the other hand, it dramatically eases the implementation. In the following, we will avoid to specify the dependence on time unless strictly necessary, thus to lighten the notation.

An artificial neuron j is completely characterized by three functions: the propagation function, the activation function, and the output function. These will be defined and detailed hereunder in the same order they get involved in the data flow.

Propagation function. The propagation function f_{prop} converts the vectorial input $\boldsymbol{p} = [y_{s_1}, \dots, y_{s_m}]^T \in \mathbb{R}^m$ into a scalar u_j often called *net input*, i.e.

$$u_j = f_{prop}(w_{s_1,j}, \dots, w_{s_m,j}, y_{s_1}, \dots, y_{s_m}). \tag{1.2.1}$$

A common choice for f_{prop} (used also in this work) is the weighted summer, adding up the scalar inputs multiplied by their respective weights:

$$f_{prop}(w_{s_1,j}, \dots, w_{s_m,j}, y_{s_1}, \dots, y_{s_m}) = \sum_{k=1}^{m} w_{s_k,j} \cdot y_{s_k}. \tag{1.2.2}$$

The function (1.2.2) provides a simple yet effective way of modeling the accumulation of different input electric signals within a biological neuron; this motivates its popularity.

Activation function. At each timestep, the *activation state* a_j, often shortly referred to as *activation*, quantifies at which extent neuron j is currently active or excited. It results from the activation function f_{act}, which combines the net input u_j with a threshold $\theta_j \in \mathbb{R}$ [3]:

$$a_j = f_{act}(u_j; \theta_j) = f_{act}(\sum_{k=1}^{m} w_{s_k,j} \cdot y_{s_k}; \theta_j), \tag{1.2.3}$$

where we have employed the weighted summer (1.2.2) as propagation function. From a biological perspective, the threshold θ_j is the analogous of the action potential mentioned in Section **??**. Mathematically, it represents the point where the absolute value $|f'_{act}|$ of the derivative of the activation function is maximum. Then, the activation function reacts particularly sensitive when the net input u_j hits the threshold value θ_j [3].
Furthermore, noting that θ_j is a parameter of the network, one may like to adapt it through a training process, exactly as can be done for the synaptic weights, as we shall see in Section **??**. However, θ_j is currently incorporated in the activation function, making its runtime access somehow cumbersome. This is typically

overcome by introducing a *bias neuron* in the network. A bias neuron is a continuously firing neuron, with constant output $y_b = 1$, which gets directly connected with neuron j, assigning the weight $w_{b,j} = -\theta_j$ to the connection. As can be deduced by Figure **??**, θ_j is now treated as a synaptic weight. Moreover, the net input becomes

$$u_j = \sum_{k=1}^{m} w_{s_k,j} \cdot y_{s_k} - \theta_j \,, \tag{1.2.4}$$

i.e. the threshold is now included in the propagation function rather than in the activation function, which we can now express in the form

$$a_j = f_{act}(\sum_{k=1}^{m} w_{s_k,j} \cdot y_{s_k} - \theta_j) \,. \tag{1.2.5}$$

Let us point out that this trick can be clearly applied to all neurons in the network which are characterized by a non-vanishing threshold: just connect the neuron with the bias, weighting the connection by the threshold value. However, for ease of illustration in the following we shall avoid to include the bias neuron in any graphical representation of a neural network.

Conversely to the propagation function, there exist various choices for the activation function, as the Heaviside function, which assumes only 0 or 1, according to whether the argument is negative or positive, respectively:

$$f_{act}(v) = \begin{cases} 0 \,, & \text{if } v < 0, \\ 1 \,, & \text{if } v \geq 0. \end{cases} \tag{1.2.6}$$

Neurons characterized by such an activation function are usually named McCulloch-Pitts neurons, after the seminal work of McCulloch and Pitts [2], and their employment is usually limited to single-layer perceptrons designed for classification (see Section **??**). In addition, note that (1.2.6) is discontinuous, with a vanishing derivative everywhere except that in the origin, thus not admissible for the backpropagation training algorithm presented in Section **??**.

Among continuous activation maps, sigmoid functions have been widely used for the realization of artificial neural networks due to their graceful combination of linear and nonlinear behaviour [2]. Sigmoid functions are s-shaped and monotically increasing, and assume values in a bounded interval, typically $[0,1]$, as the logistic function,

$$f_{act}(v) = \frac{1}{1 + e^{-v/T}} \quad \text{with } T > 0 \,, \tag{1.2.7}$$

or $[-1,1]$, as the hyperbolic tangent,

$$f_{act}(v) = \frac{e^{v} - e^{-v}}{e^{v} + e^{-v}} \,. \tag{1.2.8}$$

Output function. Finally, the output function f_{out} is in charge of calculating the scalar *output* $y_j \in \mathbb{R}$ based on the activation state a_j of the neuron:

$$y_j = f_{out}(a_j) \,; \tag{1.2.9}$$

typically, f_{out} is the identity function, so that activation and output of a neuron coincides, i.e. $y_j = f_{out}(a_j) = a_j$. Let us point out that while the input $\boldsymbol{p} = [y_{s_1}, \ldots, y_{s_m}]^T \in \mathbb{R}^m$ of the neuron is generally vectorial, i.e. $m > 1$, the output is scalar. The output y_j could then either be sent to other neurons, included the outputting neuron itself (autosynapsis), or constitute a component of the overall output vector of the network, as for the neurons in the output layer of a feedforward neural network (see Section **??**).

It worths mention here that, as the activation function, also the output function is usually *globally* defined, i.e. all neurons in the network (or at least a group of neurons) are equipped with the same output function.

The neural model presented so far actually refers to the so called *computing* neuron, i.e. a neuron processing input informations to provide a response. However, in a neural network one may also distinguish *source* neurons, supplying the network with the respective components of the activation pattern (input

vector) [2]. The role of source neurons will be clearer in the next section, where we will introduce the multilayer feedforward neural network. Here we just mention that such a neuron receives a scalar and unweighted input, which is simply forward to the connected neurons; no computations are performed.

1.2.2 Network topologies: the feedforward neural network

The way neurons are interconnected within a network defines the *topology* of the network itself, i.e. its design. In literature, many network architectures have been proposed, sometimes tailored to a specific application or task. In this section, we expand our investigation for the two most common network topologies: the feedforward and the recurrent neural network.

Feedforward neural network. In a feedforward neural network, also called *perceptron*, neurons are arranged into *layers*, with one *input layer* of $M_{\boldsymbol{p}}$ source neurons, K *hidden layers*, each one consisting of H_k computing neurons, $k = 1, \dots, K$, and an *output layer* of $M_{\boldsymbol{q}}$ computing neurons. As characteristic property, neurons in a layer can only be connected with neurons in the next layer towards the output layer. Then, an *activation pattern* $\boldsymbol{p} \in \mathbb{R}^{M_{\boldsymbol{p}}}$, supplied to the network through the source nodes in the first layer, provides the input signal for the neurons in the first hidden layer. For each hidden layer, its output signal gives the input pattern for the following layer. In this way, informations travel towards the last layer of the network, i.e. the output layer, whose outputs constitute the components of the overall output $\boldsymbol{q} \in \mathbb{R}^{M_{\boldsymbol{q}}}$ of the network, response to the input pattern $\boldsymbol{p}$. Hence, a feedforward network establish a map between the *input space* $\mathbb{R}^{M_{\boldsymbol{p}}}$ and the *output space* $\mathbb{R}^{M_{\boldsymbol{q}}}$. This makes this network architecture suitable for classification and continuous function regression tasks; the latter is the one we are concerned with.

Feedforward networks can be classified according to the number of hidden neurons they present, or, equivalently, the number of layers of trainable weights. Single-layer perceptrons (SLPs) just consist of the input and output layer; no hidden layers. Note that the layer which the name refers to is the output layer; the input layer is not accounted for since it does not perform any calculation. Despite their quite simple structure, their range of application is rather limited, since they can only represent linearly separable data [3]. Conversely, multi-layer perceptrons (MLPs) are universal function approximators, as stated by the Cybenko [**?**, **?**]. In detail:

- a multi-layer perceptron with one layer of *hidden neurons* and differentiable activation functions can approximate any *continuous* function [**?**];
- a multi-layer perceptron with two layers of *hidden neurons* and differentiable activation functions can approximate *any function* [**?**].

Hence, in many practical applications there is no reason to employ MLPs with more than two hidden layers; we experienced this fact in the numerical experiments we performed.

An instance of a three-layer (i.e. two hidden layer plus the output layer) feedforward network is offered in Figure **??**. In this case, we have $M_{\boldsymbol{p}} = 3$ input neurons (denoted with the letter i), $H_1 = H_2 = 6$ hidden neurons (letter h), and $M_{\boldsymbol{q}} = 4$ output neurons (letter o). In particular, we point out that it represents an instance of a *completely linked* perceptron, since each neuron is directly connected with all neurons in the following layer.

Finally, let us just mention that, although we have previously stated that in a feedforward neural network a synapses can only connect pairs of neurons in contiguous layers, recent years have seen the development of different variants. For instance, *shortcut connections* skip one or more layers, while *lateral connections* takes place between neurons within the same layer.

Recurrent neural network. In recurrent networks any neuron can bind with any other neuron, but autosynapses are forbidden, i.e. the output of a neuron can be input into the same neuron at the next time step. If each neuron is connected with all other neurons, then the network is said *completely linked*. As a consequence, one can not distinguish neither input or output neurons any more: neurons are all equivalent. Then, the input of the network is represented by the initial *network state*, which is the set of activation states

for all neurons in the network. Similarly, the overall network output is given by the final network state. So, communication between a recurrent neural network and the surrounding environment takes place through the states of the neurons. Examples of recurrent networks are the Hopfield networks [], inspired by the behaviour of electrically charged particles within a magnetic field, and the self-organizing maps by Kohonen [], highly suitable for cluster detection.

As mentioned in the introductory chapter, in this work we refer to neural networks for the approximation of the map between the parameters of a parametrized partial differential equation, and the coefficients of the relative reduced basis solution. Although a detailed explanation will be provided in the next chapter, what is worth notice here is that it represents a continuous function regression task. Then, motivated by the previously reported considerations, a multi-layer feedforward neural network turns out as the most suitable network architecture for our purposes.

We are now left to investigate the way the weights of a perceptron can be *trained* to meet our purposes.

1.2.3 Training a multilayer feedforward neural network

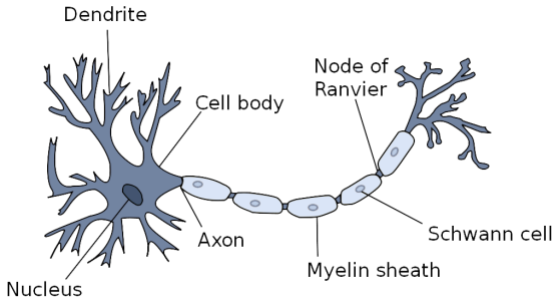

Figure 1.2.1

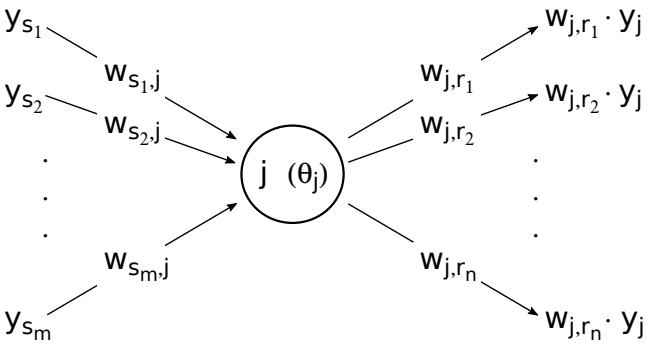

Figure 1.2.2

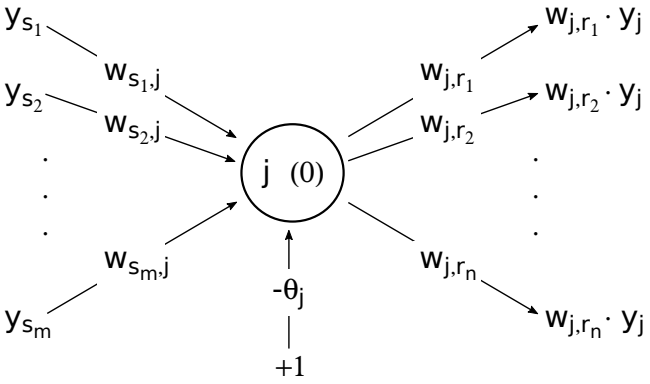

Figure 1.2.3

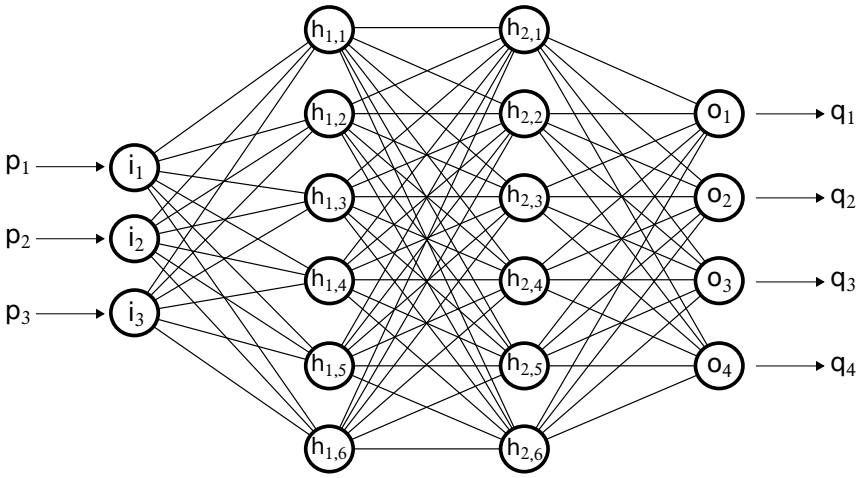

Figure 1.2.4

Chapter 2

Reduced Basis method for nonlinear elliptic PDEs

In this chapter, we derive a Reduced Basis (RB) approximation for a parametrized Boundary Value problem (BVP) involving a possibly nonlinear elliptic Partial Differential Equation (PDE). As a convenient test case, the nonlinear Poisson equation in one and two spatial dimensions is considered, with the nonlinearity stemming from a solution-dependent viscosity. In the one-dimensional case, the parameters may characterize the viscosity itself, the forcing term, or the boundary conditions. On the contrary, in the two-dimensional framework we are concerned with a parameter-free PDE defined on a domain undergoing geometrical transformations. In the RB context, when dealing with shape variations one needs to introduce a parametric transformation, mapping the whole domain to a parameter-independent reference configuration, e.g. a unit square [**?**]. For this aim, we refer to the boundary displacement-dependent transfinite map (BDD TM) proposed in [**?**], building a volumetric parametrization given the boundary parametrization of the domain.

For the sake of numerical efficiency, the RB procedure is usually carried out within an offline-online framework. The *offline* stage consists in generating the reduced basis out of an ensemble of *snapshots*, i.e. high-fidelity numerical solutions to the BVP for different realizations of the parameters. In this thesis, the well-known Proper Orthogonal Decomposition (POD) technique is used, relying on the Finite Element (FE) method for the computation of the snapshots. Then, given new values for the parameters, in the *online* phase one seeks an approximation to the high-fidelity solution in the reduced space, i.e. the linear space generated by the reduced basis. To retain the well-posedness of the problem, the system yielded by the FE method is projected onto the reduced space, thus enforcing the orthogonality to the reduced basis of the residual for each equation [**?**, **?**]. Hence, the computational cost associated with the *resolution* of the *reduced* model is *independent* of the size of the original, large-scale model.

The whole procedure sketched so far is detailed in the following sections. Section **??** and **??** discuss the Finite Element method applied to the one- and two-dimensional Poisson equation, respectively. The Reduced Basis technique is described in Section **??**, while in the final Section **??** we present an alternative, Neural Network-based approach for the computation of the reduced solution, which represents the actual novelty of the proposed reduced order modeling algorithm.

2.1 Finite Element method for one-dimensional Poisson

Bibliography

[1] Hagan M. T., Demuth H. B., Beale M. H., De Jesús O. *Neural Network Design, 2nd Edition*. eBook, 2014.

[2] Haykin S. *Neural Networks: A comprehensive foundation*. Prentice Hall, Upper Saddle River, NJ, 2004.

[3] Kriesel D. *A Brief Introduction to Neural Networks*. `http://www.dkriesel.com/en/science/neural_networks`.

[4] Nielsen M. A. *Neural Networkd and Deep Learning*. Determination Press, 2015.

[5] Stergiou C., Siganos D. *Neural Networks*. `https://www.doc.ic.ac.uk/~nd/surprise_96/journal/vol4/cs11/report.html#Introductiontoneuralnetworks`.