Politecnico di Milano

Mathematical Engeneering

Computational Science and Engeneering

Broadband Earthquake Ground Motion from Physics-Based Numerical Simulations Using Artificial Neural Networks

Supervisor
Prof. Paola Francesca ANTONIETTI

Candidate
Enrico MANUZZI

Co-Supervisor
Prof. Chiara SMERZINI
Prof. Ilario MAZZIERI

Academic Year 2018-2019

Abstract

In this work we present a strategy to generate broadband earthquake ground motions from the results of three-dimensional physics-based numerical simulations (PBNS) based on employing Artificial Neural Networks (ANNs) algorithms.
Physics-based simulated ground motions embody a rigorous seismic-wave propagation model (i.e., including source, path, and site effects), which is however reliable only in the long-period range (typically above 0.75–1 s), owing to the limitations posed both by computational constraints and by insufficient knowledge of the medium at short wavelengths. To cope with these limitations, the proposed approach makes use of ANNs. These networks are trained on a set of strong-motion records, in order to predict the response spectral ordinates at short periods using as input the spectral ordinates at long periods obtained by the PBNS.
This work focuses on an in-depth sensitivity and robustness analysis of the proposed algorithm with respect to the input selection, hyperparameters analysis, the use of optimization algorithms to set parameters, dataset handling and the "bootstrap aggregating" technique.
The capability of the final model to reproduce in a realistic way the engineering features of earthquake ground motion is successfully proven on real study cases, namely the L'Aquila (2009), Po Plain (2012) and Norcia (2016) earthquakes.

Acknowledgments

I would like to express my gratitude to my supervisor, Professor Paola Francesca Antonietti, for the guidance, the trust and her human and motivating attitude toward me. I wish to thank also Professor Chiara Smerizini and Professor Ilario Mazzieri for their support and valuable advice.
I thank my friends and my girlfriend, that in person or from far helped me out every day, listening to my ideas and complaints, making me laugh and sometimes making me mad.
Last but not least, my deepest gratitude to my family, who always unconditionally supported me.

Contents

List of Figures

List of Tables

Introduction

Numerical simulations of seismic wave propagation phenomena are invaluable for many geophysical applications: to quantify the ground motion of potential earthquakes [3, 16], to understand the seismic response of hydrocarbon reservoirs [46, 72], and to estimate the unknown elastic properties of a medium given its seismic response [70], for example. In particular, this thesis focuses on an application in earthquake engineering, namely the prediction of ground motion following an earthquake for a broad range of wavelenghts. Boosted by the ever-increasing availability of parallel high-performance computing, three-dimensional (3D) physics-based numerical simulations (PBNS) are becoming one of the leading tools to obtain synthetic ground-motion time histories, for which the use for seismic hazard and engineering applications is subject to a growing attention [10, 54].
Physics-based numerical modelling already proved in the recent past to be well suited for global [56, 43, 44, 32, 77] and regional-scale simulations [18, 1, 5, 53, 29, 23, 73], making potentially feasible the challenging problem of a multiscale simulation from the seismic source to the structural response within a single computational model [48, 38].

Being based on a detailed spatial discretization of the continuum and on the numerical approximation of the elastodynamics equation, carried out according to a numerical method (such as finite differences, finite element, or spectral element methods), PBNS require a sufficiently detailed model of the seismic source, of the fault rupture dynamics and of the Earth's crustal layers.
The accuracy of the PBNS in the high-frequency range is limited, by both the increased computational burden as the mesh gets finer and by the lack of detailed knowledge to construct a geological model with sufficient details for a broad range of wavelengths. Hence, accuracy achieved by PBNS is usu-

ally limited up to 1–1.5 Hz [54] (for more details see the end of Section 1.3.3).

In recent years, deep learning models have achieved considerable success in various tasks, such as computer vision, natural language processing and reinforcement learning [45].
Positive results have been achieved by applying these techniques also to wave-dynamics, such as using deep neural networks (DNNs) to generate realistic wave fields in homogeneous and complex media [81], starting from ground motion data. In this fashion DNNs could be used to provide less accurate but faster solutions, which might be crucial also for model-based decision making and planning in complex physical environments.
Speed-ups of these algorithms have been implemented for some specific cases, such as horizontally layered media [52]. However, in general, the training phase continues to be a bottleneck, since the training data must be processed serially over thousands of individual training runs.
Hybrid approaches [54] have been explored, in order to compensate the respective limitations of PBNS and NNs. In fact, since PBS are reliable only in the long-period range, a neural network has been used to predict the response spectral ordinates at short periods. This allows to portray in a more realistic way the wave-field at broad-band frequencies, even though the efficient coupling procedure of the two models presents some open questions that need to be addressed.
Finally, DNN and convolutional neural networks (CNNs) have been also used to solve directly partial differential equations (PDEs) [79, 69, 35]. CNNs and DNNs can be indeed viewed as piecewise linear functions [35][36], which makes them suitable for approximating PDEs in the same fashion of finite elements methods (FEMs). These networks can indeed represent complex-shaped functions, without the need of basis functions like in the FEMs, making them potentially well suited to represent complex solutions with an affordable computational cost.
However, a solid mathematical workground on this topic is still under development. A little light has been shed, understanding the expressive power of these models based on analysing their approximation properties [36].
Very recently a unified model that simultaneously recovers some convolutional neural networks (CNN) for image classification and multigrid (MG) methods for solving PDEs has been proposed by [35]. As a result, how and why CNN models work can be mathematically understood in a similar fashion as for multigrid methods for which a much more mature and better

developed theory [78]. Motivated from various known techniques from multigrid method, many variants and improvements of CNN can then be naturally obtained.
In a different perspective, new multigrid methods for numerical PDEs can be motivated by deep learning [39].
Moreover MG methods, which rely on the construction of a hierarchy of finer grids in order to accelerate the convergence of the underlying iterative scheme, have a massively parallel intrinsic structure that can be exploited within HPC architectures [69]. Exploiting this dualism might allow to cope better with the bottleneck of the training phase in NNs and with the limitations of PBSs in the high frequency range.

The novel approach firstly proposed by Paolucci et al. [54] and further expanded and analyzed inn this thesis, consists in generating broadband (BB) ground motions, which couples the results of PBNS for a specific earthquake ground-motion scenario with the predictions of an Artificial Neural Network (ANN). Specifically we are interested in the spectral acceleration (SA) at high frequencies, i.e. for short periods.
SA is the maximum acceleration during an earthquake of an object, modelled as a damped, harmonic oscillator moving with one degree of freedom. The SA as a function of the frequency is called response spectrum. Spectral acceleration is related to the natural frequency of vibration of the building, and it is used in earthquake engineering to have a more reliable approximation of the motion of a building during an earthquake than the peak ground acceleration (PGA) value.
The basic steps of the algorithm can be summarized as follows:

1. the ANN is trained on a strong-motion dataset, to correlate short period ($T \leq T^*$) spectral ordinates with the long-period ones ($T > T^*$), being T^* the threshold period beyond which the results of the PBNS are supposed to be accurate;

2. the trained ANN is used to obtain the short-period spectral ordinates of the physics-based earthquake ground motion for periods below T^*;

3. the PBNS long-period time histories are enriched at high frequencies with an iterative spectral matching approach, until the response spectrum matches the short-period part obtained by the ANN.

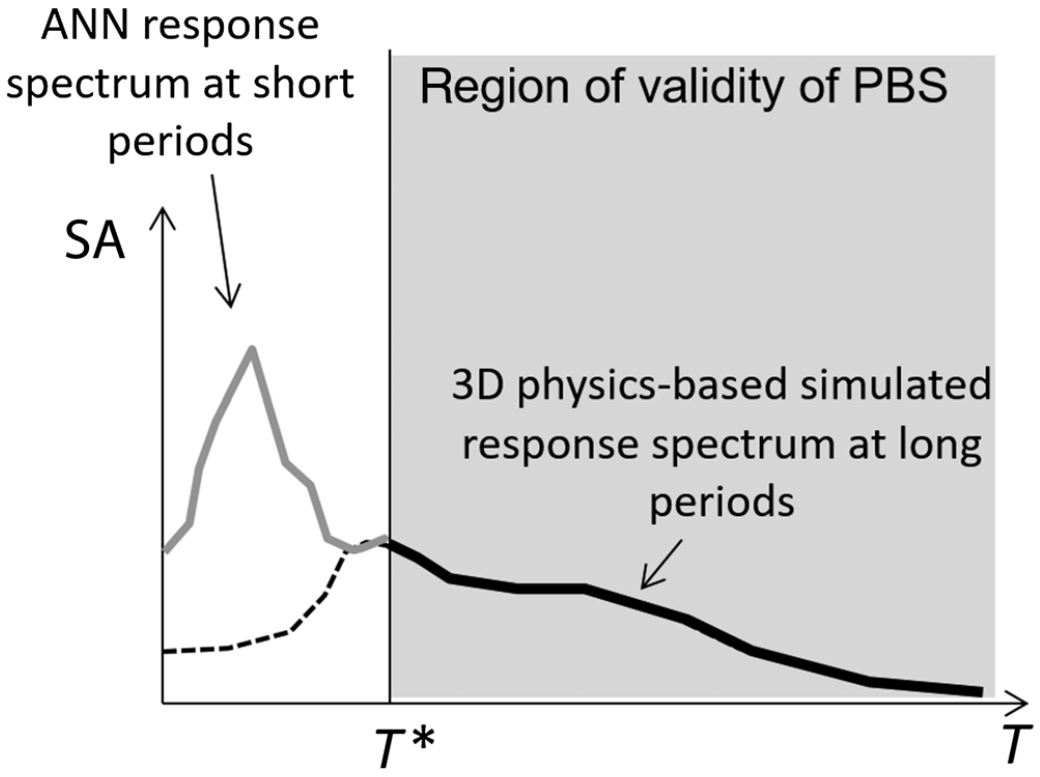

Figure 1: Figure taken from [54]. Sketch illustrating the idea at the basis of the Artificial Neural Network-based algorithm to generate broadband ground motions. For a given ground motion, response spectral ordinates at short periods, that is, for periods $T \leq T^*$, where T^* is the minimum period of validity of the physics-based numerical model, are computed from the 3D physics-based simulated response spectral ordinates at long periods. SA, spectral acceleration; PBNS, physics-based numerical simulation.

In this thesis we will focus on a detailed analysis of the first two steps of the algorithm, leaving the implementation of new matching approaches for future developments.

Using as a starting point the workflow proposed by [54], in this thesis we will propose and analyze new algorithms. More precisely, we will address the following points:

1. improve the performance, to prove the validity of the proposed algorithm in terms of practical results;

2. understand the key features of the model and why they are crucial, in order to prove its generality.

The thesis consists of 9 chapters, that are briefly described in the following:

- Chapter 1: Principles of earthquake ground motion. In this chapter we present the key notions of earthquake engineering, starting from the definitions (such as peak ground acceleration (PGA), spectral acceleration (SA), shear waves, ...), then focusing on the governing equations of the physical problem and how to solve them numerically.

- Chapter 2: The Artificial Neural Network model. In this chapter we recall the theoretical background on the ANN model, with focus on the model structure, how to train it, i.e. how to set its parameters, and how to evaluate performance, with focus on its application to earthquake prediction.

- Chapter 3: Input Selection. Here, new variants of the original ANN model of [54] are presented, specifically the use of additional information concerning epicentral distance, magnitude, shear waves velocity and soil conditions are discussed.

- Chapter 4: Optimization Algorithms. In this chapter we recall the main algorithms for optimization, i.e. revising a set of methods to find numerically the minimum of a function.

- Chapter 5: Training Algorithm Selection. In this chapter we report a detailed analysis of the best suited optimization algorithms in terms of CPU time and performance, for the selected ANNs.

- Chapter 6: Training, Validation and Test Sets. This chapter deals with the analysis of the optimal way to partition the dataset into training, validation and test sets, in order to improve performance.

- Chapter 7: Predictions Using Data from Numerical Simulations. In this chapter we employ the ANNs to real study cases, specifically Po Plain (2012), L'Aquila (2009) and Norcia (2016), to test the effectiveness and robustness of the proposed method.

- Chapter 8: Bootstrap Aggregating. Here the possibility of averaging the output of more networks is explored, by using a technique called bootstrap aggregating, in order to further improve stability and performance.

- Chapter 9: Conclusions and Further Developments. In this chapter we report a summary of the main results provided, and we discuss possible developments, in line with the most recent approaches concerning the solution of partial differential equations using ANNs.

Chapter 1

Principles of Earthquake Ground Motion

In this chapter we review the main ingredients in the field of earthquake engineering that will be needed in the forthcoming chapters. Starting from the main definitions, we then focus on the governing equations of the physical problem and how to solve them numerically [3]. In the following we adopt the notation and the presentation structure of [3, 25].

1.1 Main Definitions

An earthquake is the result of a sudden release of energy in the outer layers of the Earth's crust. The strain energy slowly accumulated over the time is transformed into kinetic energy and radiated through the Earth's layers. By studying the main properties of motion, using suitable numerical techniques, it is possible characterize those features of ground shaking, that can be employed to design seismic risk protection strategies.
In the following we report some useful definitions:

- hypocenter: also known as focus, is the point within the Earth where an earthquake rupture starts (see Figure 1.1);
- epicenter: is the point directly above it at the surface of the Earth (see Figure 1.1);
- epicentral distance (R_{epi}): is the distance, along the Earth surface, of a point of interest form the epicenter location;

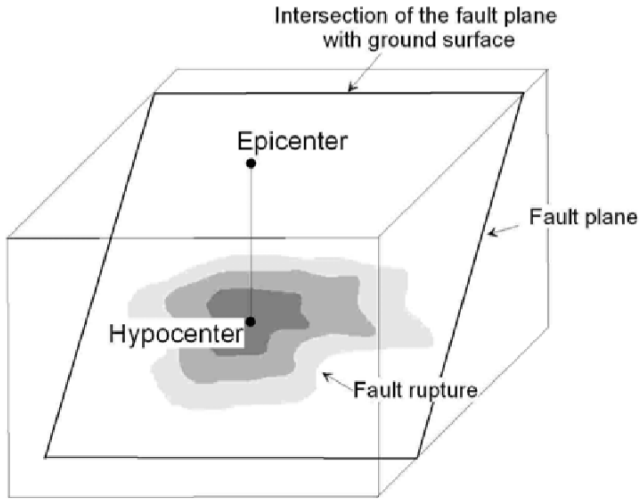

Figure 1.1: Figure taken from [25]. Hypocenter and epicenter of an earthquake. The fault plane is the plane along which the rupture occurs.

- peak ground acceleration (PGA): is the maximum ground acceleration that occurred during an earthquake at a fixed location on the earth surface, often split into its horizontal and vertical components.

1.1.1 Magnitude

The magnitude is a measure of intensity of an earthquake, and can be quantified with different scales. Let M_0 be the seismic moment defined as

$$M_0 = G\overline{\Delta u}A, \tag{1.1}$$

where G is the shear modulus of the crustal rocks of the fault where the rupture occurs, $\overline{\Delta u}$ is the average slip (displacement offset between the two sides of the fault) and A is the fault area. The moment magnitude M_w, is then defined as

$$M_w = \frac{2}{3}log_{10}(M_0) - 6, \tag{1.2}$$

where M_0 is measured in $[Nm]$.

site class	V_{s30} interval $[m/s]$
A	800 >
B	360 - 800
C	180 - 360
D	0 - 180

Table 1.1: Soil classification table using velocity of the shear waves in the first 30 meters (V_{s30}).

1.1.2 Seismic Waves

Seismic waves can be classified into body and surface waves. Body waves travel through the interior of the Earth and they are further split into pressure (P) and shear (S) waves. P-waves generate a ground motion that is aligned with the direction of the wave field, they can travel through all media (solids and liquids) and are characterized by the highest wave velocities, typically ranging from 0.5 to 6 km/s. S-waves are transversal waves that induce a ground motion perpendicular to the direction of the wave field and, differently from pressure waves, they can travel only through solid media. S-waves are slower than P-waves, with a wider range of propagation velocities, typically ranging from about 100 m/s to 3.5 km/s. The mean shear-wave velocity in the top 30 m is known as V_{S30}. This value can be used for soil classification, as reported in Table 1.1. When body waves reach the Earth's surface, they generate surface waves, such as Love or Rayleigh waves, that induce a rolling and a circular ground motion, respectively. Surface waves are characterized by the slowest wave velocities and have a longer period of oscillation and a larger amplitude with respect to body waves (see Figure 1.2).

1.1.3 Spectral Acceleration

Spectral acceleration (SA) is the maximum acceleration of an object during an earthquake. It is usually modelled as a damped, harmonic oscillator moving with one degree of freedom (see Figure 1.3). Specifically, let

- m be the mass of the oscillator;
- $x(t)$ be the absolute displacement of the oscillator at time t;

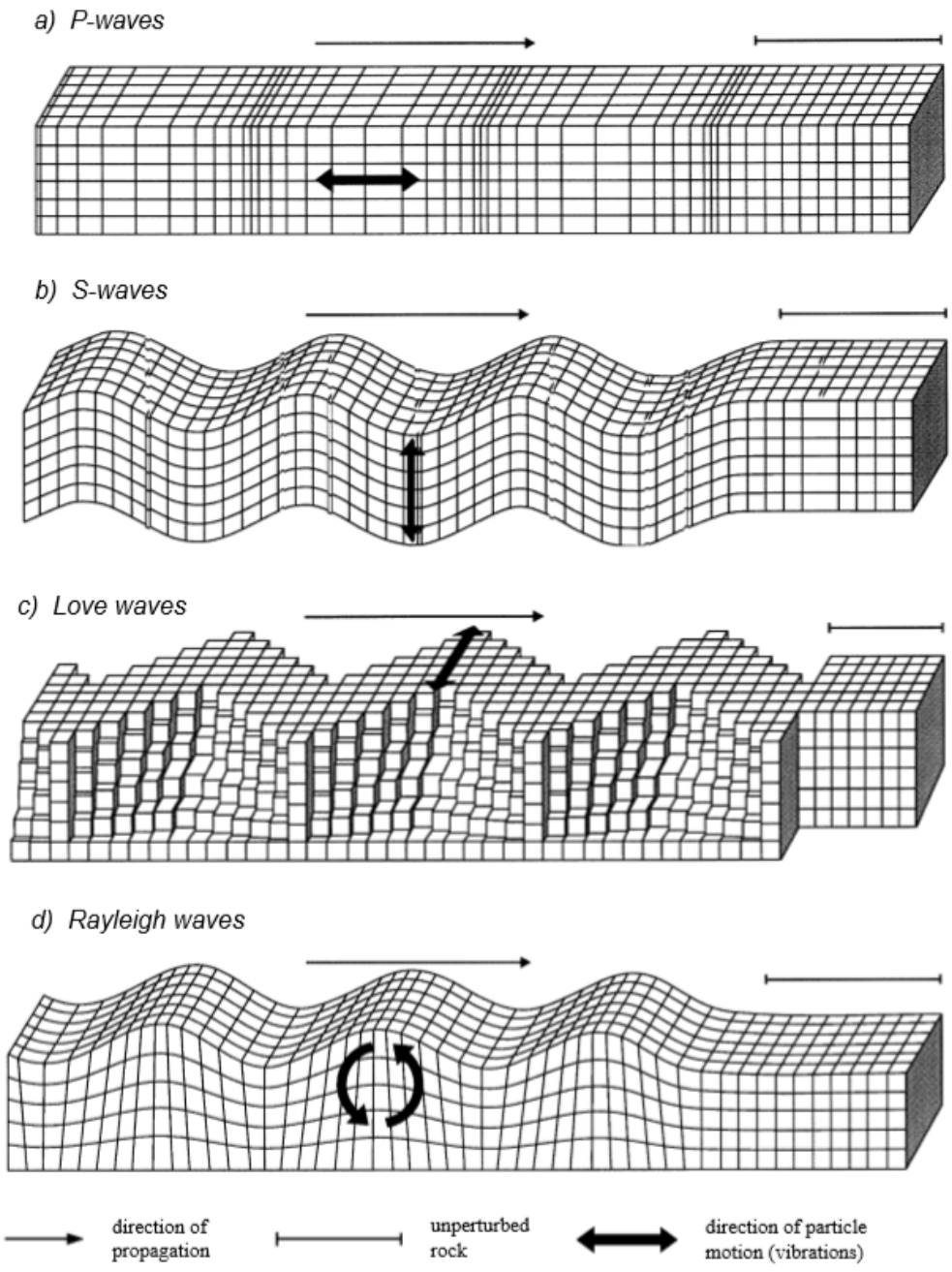

Figure 1.2: Figure taken from [25]. Graphical representation of the deformations induced by the travelling of different types of elastic waves propagating: a) longitudinal wave (or P-wave); b) transversal wave (or S-wave); c) surface wave, Love type; d) surface wave, Rayleigh type. P and S waves are the only ones that can propagate in an homogeneous, elastic, unbounded medium.

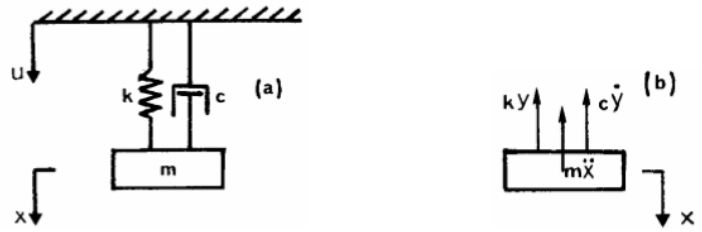

Figure 1.3: Figure taken from [25]. a) Scheme of an oscillator with one degree of freedom. b) Forces acting on mass m.

- $u(t)$ be the absolute displacement of the support, representing the ground, at time t;
- k be the stiffness constant of the spring (measured in [F]/[L]);
- c be the viscosity constant of the dumper (measured in [F][T]/[L]).

The dynamic equilibrium equation reads

$$m\ddot{x}(t) + c\dot{y}(t) + ky(t) = 0, \tag{1.3}$$

where $y(t) = x(t) - u(t)$ is the relative displacement of the oscillator w.r.t. the support. It is possible to rearrange the terms to obtain

$$m\ddot{y}(t) + c\dot{y}(t) + ky(t) = -m\ddot{u}(t), \tag{1.4}$$

and then

$$\ddot{y}(t) + 2\omega_n\zeta\dot{y}(t) + \omega_n^2 y(t) = -\ddot{u}(t), \tag{1.5}$$

where

$$\omega_n = \sqrt{\frac{k}{m}} \tag{1.6}$$

is the circular frequency of the oscillator and

$$\zeta = \frac{c}{2m\omega_n} = \frac{c}{c_{cr}} \tag{1.7}$$

is the damping ratio. For a given period $T = \frac{1}{\omega}$ the spectral acceleration is defined as

$$SA\,(T, \zeta) = \max_t |\ddot{x}(t)|. \tag{1.8}$$

SA as a function of the frequency is called acceleration response spectrum. SA is related to the natural frequency of vibration of the building, and it is used in earthquake engineering to have a more reliable approximation of the motion of a building during an earthquake than the peak ground acceleration (PGA) value. By definition, in this case

$$PGA = \max_t |\ddot{u}(t)|. \tag{1.9}$$

For very stiff structures we have $T \to 0$, hence we get

$$SA\,(T \to 0, \zeta) \cong \max_t |\ddot{u}(t)| = PGA. \tag{1.10}$$

1.2 Physical Problem and Governing Equations

Consider a bounded domain $\Omega \subset \mathbb{R}^3$ (representing the portion of the ground where we study wave propagation), and assume that its boundary is decomposed into three disjoint portions Γ_D, Γ_N and Γ_{NR}, where we impose the values of displacement, of tractions, and of the fictitious tractions introduced to avoid unphysical reflections, respectively. Considering a temporal interval $(0, T]$, with $T > 0$, the dynamic equilibrium equation for a viscoelastic medium subject to an external force can be modelled with the equations

$$\begin{cases} \rho\ddot{\boldsymbol{u}} + 2\rho\zeta\dot{\boldsymbol{u}} - \nabla \cdot \underline{\sigma}(\boldsymbol{u}) + \rho\zeta^2\boldsymbol{u} = \boldsymbol{f} & \text{in } \Omega \times (0, T], \\ \boldsymbol{u} = \boldsymbol{0} & \text{on } \Gamma_D \times (0, T], \\ \underline{\sigma}(\boldsymbol{u})\,\boldsymbol{n} = \boldsymbol{t} & \text{on } \Gamma_N \times (0, T], \\ \underline{\sigma}(\boldsymbol{u})\,\boldsymbol{n} = \boldsymbol{t}^* & \text{on } \Gamma_{NR} \times (0, T], \\ \boldsymbol{u} = \boldsymbol{u}_0 & \text{in } \Omega \times \{0\}, \\ \dot{\boldsymbol{u}} = \boldsymbol{v}_0 & \text{on } \Omega \times \{0\}, \end{cases} \tag{1.11}$$

where $\boldsymbol{u} = \boldsymbol{u}(\boldsymbol{x}, t)$ is the displacement field, $\underline{\sigma}(\boldsymbol{u})$ is the stress tensor, $\boldsymbol{f} = \boldsymbol{f}(\boldsymbol{x}, t)$ is a given external load (representing e.g. a seismic source) and ρ is the medium density. $\boldsymbol{u}_0$ and $\boldsymbol{v}_0$ are smooth initial values for the displacement and the velocity field, respectively. On the boundary we impose a rigidity condition on Γ_D, a traction $\boldsymbol{t} = \boldsymbol{t}(\boldsymbol{x}, t)$ on Γ_N and a fictitious traction $\boldsymbol{t}^* = \boldsymbol{t}^*(\boldsymbol{x}, t)$ on Γ_{NR}. We assume that ρ is a uniformly bounded, strictly positive function. In (1.11) the parameter $\zeta \geq 0$ (usually referred to as *damping*

factor) is a decay factor with the dimension of the inverse of time and it is supposed to be piecewise constant in order to model media with sharp elastic impedance and damping behaviors.
For the stress tensor $\underline{\sigma}$ we use the following constitutive equation (Hooke's law):

$$\underline{\sigma}(\boldsymbol{u}) = \lambda \mathrm{tr}(\underline{\epsilon}(\boldsymbol{u}))\underline{I} + 2\mu\underline{\epsilon}(\boldsymbol{u}), \tag{1.12}$$

where $\underline{\epsilon}(\boldsymbol{u}) = (\nabla\boldsymbol{u} + \nabla^T\boldsymbol{u})/2$ is the strain tensor, $\underline{I}$ is the identity tensor, $\lambda = \lambda(\boldsymbol{x}) \in L^\infty(\Omega)$ and $\mu = \mu(\boldsymbol{x}) \in L^\infty(\Omega)$ are the first and second Lamé elastic coefficients, respectively, and $\mathrm{tr}(\cdot)$ is the trace operator. By introducing the fourth order Hooke's tensor $\underline{D}$, relation (1.12) can also be written as

$$\underline{\sigma}(\boldsymbol{u}) = \underline{D}\,\underline{\varepsilon}(\boldsymbol{u}), \tag{1.13}$$

with $\underline{D}$ a uniformly bounded and symmetric and positive definite tensor.

1.3 Numerical Discretization

1.3.1 Weak Formulation

For an open, bounded, polygonal domain $D \subset \mathbb{R}^3$ and for a non-negative integer s we denote by $H^s(D)$ the L^2 Sobolev space of order s and by $\|\cdot\|_{s,D}$ and $|\cdot|_{s,D}$ the corresponding norm and semi-norm. For $s = 0$, we write $L^2(D)$ in place of $H^0(D)$. We set $\boldsymbol{H}^s(D) = [H^s(D)]^3$ and $\mathcal{H}^s(D) = [H^s(D)]^{3\times 3}$, and denote by $(\cdot,\cdot)_D$ and $\langle\cdot,\cdot\rangle_{\partial D}$ the corresponding $\mathbf{L}^2$ and $\mathcal{L}^2$ inner products.

Let $\boldsymbol{V} = \{\boldsymbol{v} \in \boldsymbol{H}^1(\Omega),\ \boldsymbol{v} = \boldsymbol{0} \text{ on } \Gamma_D\}$; the weak formulation of problem (1.11) read as: $\forall t \in (0,T]$ find $\boldsymbol{u}(t) \in V$ such that

$$(\rho\ddot{\boldsymbol{u}}(t),\boldsymbol{v})_\Omega + (2\rho\zeta\dot{\boldsymbol{u}}(t),\boldsymbol{v})_\Omega + (\rho\zeta^2\boldsymbol{u},\boldsymbol{v})_\Omega + \mathcal{A}(\boldsymbol{u}(t),\boldsymbol{v}) = \mathcal{F}(\boldsymbol{v}) \qquad \forall \boldsymbol{v} \in V, \tag{1.14}$$

supplemented with the initial conditions $\boldsymbol{u}(0) = \boldsymbol{u}_0$ and $\dot{\boldsymbol{u}}(0) = \boldsymbol{v}_0$, where

$$\mathcal{A}(\boldsymbol{u},\boldsymbol{v}) = (\underline{\sigma}(\boldsymbol{u}),\underline{\varepsilon}(\boldsymbol{v}))_\Omega, \quad \mathcal{F}(\boldsymbol{v}) = (\boldsymbol{f},\boldsymbol{v})_\Omega + \langle\boldsymbol{t},\boldsymbol{v}\rangle_{\Gamma_N} + \langle\boldsymbol{t}^*,\boldsymbol{v}\rangle_{\Gamma_{NR}}.$$

If $\Gamma_{NR} = \emptyset$ and $\zeta = 0$, the above problem is well-posed, see for instance [63, Theorem 8.3-1].

1.3.2 Partitions and trace operators

We consider a (not necessarily conforming) decomposition $\mathcal{T}_\Omega$ of Ω into L nonoverlapping polyhedral sub-domains Ω_ℓ, i.e., $\overline{\Omega} = \cup_\ell \overline{\Omega}_\ell$, $\Omega_\ell \cap \Omega_{\ell'} = \emptyset$ for $\ell \neq \ell'$. On each Ω_ℓ, we built a *conforming*, *quasi-uniform* computational mesh $\mathcal{T}_{h_\ell}$ of granularity $h_\ell > 0$ made by open disjoint elements $\mathcal{K}_\ell^j$, and suppose that each $\mathcal{K}_\ell^j \in \Omega_\ell$ is the affine image through the map $F_\ell^j : \widehat{\mathcal{K}} \longrightarrow \mathcal{K}_\ell^j$ of either the unit reference hexahedron or the unit reference tetrahedron $\widehat{\mathcal{K}}$. Given two adjacent regions $\Omega_{\ell^\pm}$, we define an interior face F as the non-empty interior of $\partial\overline{\mathcal{K}^+} \cap \partial\overline{\mathcal{K}^-}$, for some $\mathcal{K}^\pm \in \mathcal{T}_{h_{\ell^\pm}}, \mathcal{K}^\pm \subset \Omega_{\ell^\pm}$, and collect all the interior faces in the set $\mathcal{F}_h^I$. We also define $\mathcal{F}_h^D$, $\mathcal{F}_h^N$ and $\mathcal{F}_h^{NR}$ as the sets of all boundary faces where displacement, traction or fictitious tractions are imposed, respectively. In this definition, it is implicit the assumption that each boundary face can belong to exactly one of the sets $\mathcal{F}_h^D$, $\mathcal{F}_h^N$, $\mathcal{F}_h^{NR}$. We collect all the boundary faces in the set $\mathcal{F}_h^b$. We now assume that for any element $K \in \mathcal{T}_h$ and for any face $F \subset \partial K$, it holds $h_\mathcal{K} \lesssim h_F$ (see [55, 30, 21, 22, 11, 12, 4] for details and variants).
Let $\mathcal{K}^\pm \in \mathcal{T}_{h_{\ell^\pm}}, \mathcal{K}^\pm \subset \Omega_{\ell^\pm}$ be two elements sharing a face $F \in \mathcal{F}_h^I$, and let $\boldsymbol{n}^\pm$ be the unit normal vectors to F pointing outward to $\mathcal{K}^\pm$, respectively. For (regular enough) vector and tensor-valued functions $\boldsymbol{v}$ and $\underline{\tau}$, we denote by $\boldsymbol{v}^\pm$ and $\underline{\tau}^\pm$ the traces of $\boldsymbol{v}$ and $\underline{\tau}$ on F, taken within the interior of $\mathcal{K}^\pm$, respectively, and set

$$\begin{aligned} &[\![\boldsymbol{v}]\!] = \boldsymbol{v}^+ \odot \boldsymbol{n}^+ + \boldsymbol{v}^- \odot \boldsymbol{n}^-, \quad [\![\underline{\tau}]\!] = \underline{\tau}^+ \boldsymbol{n}^+ + \underline{\tau}^- \boldsymbol{n}^-, \\ &\{\boldsymbol{v}\} = \frac{\boldsymbol{v}^+ + \boldsymbol{v}^-}{2}, \qquad\qquad \{\underline{\tau}\} = \frac{\underline{\tau}^+ + \underline{\tau}^-}{2}, \end{aligned} \tag{1.15}$$

where $\boldsymbol{v} \odot \boldsymbol{n} = (\boldsymbol{v}^{\mathrm{T}}\boldsymbol{n} + \boldsymbol{n}^{\mathrm{T}}\boldsymbol{v})/2$. On $F \in \mathcal{F}_h^b$, we set $\{\boldsymbol{v}\} = \boldsymbol{v}$, $\{\underline{\tau}\} = \underline{\tau}$, $[\![\boldsymbol{v}]\!] = \boldsymbol{v} \odot \boldsymbol{n}$, $[\![\underline{\tau}]\!] = \underline{\tau}\boldsymbol{n}$.

1.3.3 Discrete Formulation

We describe now the numerical approximation of the (weak formulation) of (1.11) through a Discontinuous Spectral Element method, that couples Discountinous Galerkin methods [66, 37, 20] with Spectral Elements [24, 24, 40, 71, 26, 60, 62, 41, 42, 13, 76, 14]. For time integration, we use leap-frog time marching scheme [61, 5, 40, 15, 49].

To each subdomain Ω_ℓ we assign a nonnegative integer N_ℓ, and introduce the finite dimensional space

$$\mathbf{V}_{h_\ell}^{N_\ell}(\Omega_\ell) = \{\boldsymbol{v} \in \mathbf{C}^0(\overline{\Omega}_\ell) \,:\, \boldsymbol{v}\big|_{\mathcal{K}_\ell^j} \circ F_\ell^j \in [\mathbb{M}^{N_\ell}(\widehat{\mathcal{K}})]^3 \quad \forall\, \mathcal{K}_\ell^j \in \mathcal{T}_{h_\ell}\}, \tag{1.16}$$

where $\mathbb{M}^{N_k}(\widehat{\mathcal{K}})$ is either the space $\mathbb{P}^{N_k}(\widehat{\mathcal{K}})$ of polynomials of total degree at most N_k on $\widehat{\mathcal{K}}$, if $\widehat{\mathcal{K}}$ is the reference tetrahedron, or the space $\mathbb{Q}^{N_k}(\widehat{\mathcal{K}})$ of polynomials of degree N_k in each coordinate direction on $\widehat{\mathcal{K}}$, if $\widehat{\mathcal{K}}$ is the unit reference hexahedron in $\mathbb{R}^3$. We then define the space $\boldsymbol{V}_{\rm DG}$ as $\boldsymbol{V}_{\rm DG} = \prod_\ell \mathbf{V}_{h_\ell}^{N_\ell}(\Omega_\ell)$.
Let $\mathcal{F}_h(\cdot)$ be defined as

$$\mathcal{F}_h(\boldsymbol{v}) = \sum_{\Omega_\ell} (\mathbf{f}, \boldsymbol{v})_{\Omega_\ell} + \langle \boldsymbol{t}, \boldsymbol{v}\rangle_{\mathcal{F}_h^N} + \langle \boldsymbol{t}^*, \boldsymbol{v}\rangle_{\mathcal{F}_h^{NR}} \qquad \boldsymbol{v} \in \boldsymbol{V}_{\rm DG}, \tag{1.17}$$

where we have used the short-hand notation $\langle \boldsymbol{w}, \boldsymbol{v}\rangle_{\mathcal{F}_h^N} = \sum_{F\in\mathcal{F}_h^N} \langle \boldsymbol{w}, \boldsymbol{v}\rangle_F$ and $\langle \boldsymbol{w}, \boldsymbol{v}\rangle_{\mathcal{F}_h^{NR}} = \sum_{F\in\mathcal{F}_h^{NR}} \langle \boldsymbol{w}, \boldsymbol{v}\rangle_F$.
Let the bilinear form $\mathcal{A}_h(\cdot,\cdot)$ be defined as

$$\mathcal{A}_h(\boldsymbol{u}, \boldsymbol{v}) = \sum_{\Omega_\ell} (\underline{\sigma}(\boldsymbol{u}), \underline{\varepsilon}(\boldsymbol{v}))_{\Omega_\ell} - \langle \{\underline{\sigma}(\boldsymbol{u})\}, [\![\boldsymbol{v}]\!]\rangle_{\mathcal{F}_h^I} - \langle [\![\boldsymbol{u}]\!], \{\underline{\sigma}(\boldsymbol{v})\}\rangle_{\mathcal{F}_h^I} + \langle \eta [\![\boldsymbol{u}]\!], [\![\boldsymbol{v}]\!]\rangle_{\mathcal{F}_h^I}, \tag{1.18}$$

where, as before, $\langle \boldsymbol{w}, \boldsymbol{v}\rangle_{\mathcal{F}_h^I} = \sum_{F\in\mathcal{F}_h^I} \langle \boldsymbol{w}, \boldsymbol{v}\rangle_F$.
Denoting by $N_{\rm dof}$ be total number of degrees of freedom, the vector $\mathsf{U} = \mathsf{U}(t) \in \mathbb{R}^{N_{\rm dof}}$ contains, for any time t, the expansion coefficients of the semi-discrete solution $\boldsymbol{u}_h(t) \in \boldsymbol{V}_{\rm DG}$ in the chosen set of basis functions. Analogously, M and A are the matrices representations of the bilinear forms

$$\sum_{\Omega_\ell} (\rho \ddot{\boldsymbol{u}}_h(t), \boldsymbol{v})_{\Omega_\ell}, \quad \mathcal{A}_h(\boldsymbol{u}_h(t), \boldsymbol{v}),$$

respectively. Finally, F is the vector representation of the linear functional $\mathcal{F}_h(\cdot)$.
We subdivide the time interval $(0, T]$ into N_T subintervals of amplitude $\Delta t = T/N_T$ and we denote by $\mathsf{U}^i \approx \mathsf{U}(t_i)$ the approximation of U at time $t_i = i\Delta t$, $i = 1, 2, \ldots, N_T$. Whenever $\zeta = 0$, i.e. no dumping is present and $\Gamma_{NR} = \emptyset$, then the leap-frog method reads as

$$\begin{aligned} \mathsf{M}\mathsf{U}^1 &= (\mathsf{M} - \frac{\Delta t^2}{2}\mathsf{A})\mathsf{U}^0 + \Delta t \mathsf{M}\mathsf{V}^0 + \frac{\Delta t^2}{2}\mathsf{F}^0, \\ \mathsf{M}\mathsf{U}^{n+1} &= (2\mathsf{M} - \Delta t^2 \mathsf{A})\mathsf{U}^n - \mathsf{M}\mathsf{U}^{n-1} + \Delta t^2 \mathsf{F}^n, \quad n = 1, ..., N_T - 1, \end{aligned} \tag{1.19}$$

where $\mathsf{U}(0) = \mathsf{U}^0$, $\dot{\mathsf{U}}(0) = \mathsf{V}^0$ and $\mathsf{F}(0) = \mathsf{F}^0$ are suitable initial conditions. We notice that (1.19) involves a linear system with the matrix M to be solved at each time step.
According to the Courant, Friedrichs and Lewy (CFL) condition, the leap-frog scheme is stable provided that the time step Δt satisfies

$$\Delta t \leq C_{CFL} h, \tag{1.20}$$

where h is the granularity of the computational grid. It can be proven that

$$C_{CFL} \lesssim \frac{1}{\sqrt{\omega_h}} \tag{1.21}$$

where ω_h be the numerical angular frequency representing the best approximations of the angular frequencies of the a travelling wave [3]. Hence, at high frequencies, or equivalently at long periods, the computational cost increases.
If the numerical wave shows a phase leg with respect to the physical one we have a dispersion effect. It can also be shown that the error due to dispersion is proportional to ω_h [3].

Chapter 2

The Artificial Neural Network Model

2.1 Introduction

The purpose of this chapter is to give a detailed description of the artificial neural network model, how to handle the data at our disposal, and how to how to measure the quality of the results. In the following, we adopt the notation and presentation structure of [34] and [54].

2.2 Model Structure

Let $p \in \mathbb{R}^R$ be a set of inputs , $W \in \mathbb{R}^{S \times R}$ be the weights, $a \in \mathbb{R}^S$ be the output, $b \in \mathbb{R}^S$, $f : \mathbb{R}^S \to \mathbb{R}^S$ be the transfer function. The transfer function may be a linear or a nonlinear function. A particular transfer function is chosen to satisfy some specification of the problem that the neuron is attempting to solve.
A layer of neurons is simply a function of the form

$$a = f(Wp + b). \tag{2.1}$$

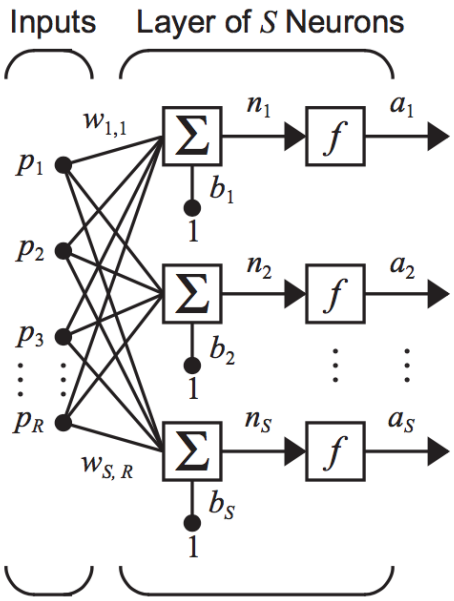

Figure 2.1: Figure taken from [34] representing a layer of neurons. A neural network can be composed of many layers, at least one.

If $S = 1$ we have a multiple-input neuron.

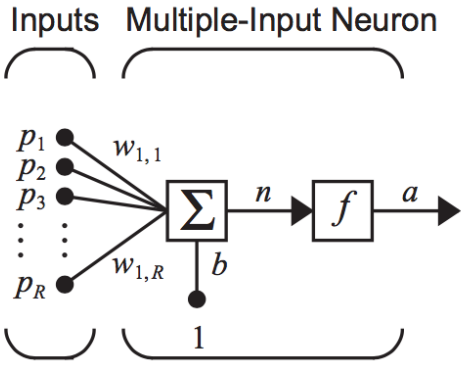

Figure 2.2: Figure taken from [34] representing multiple-input neuron. It is the fundamental unit of a layer.

If we compare this model with a biological neuron, the weights corresponds to the strength of a synapse, the cell body is represented by the sum

and the transfer function, and the neuron output represents the signal on the axon.
Finally a feed forward artificial neural network is the composition of more layer functions of the form:

$$a^i = f^i(W^i a^{i-1} + b^i) \quad i = 1, ...N \tag{2.2}$$

$$a^0 = p. \tag{2.3}$$

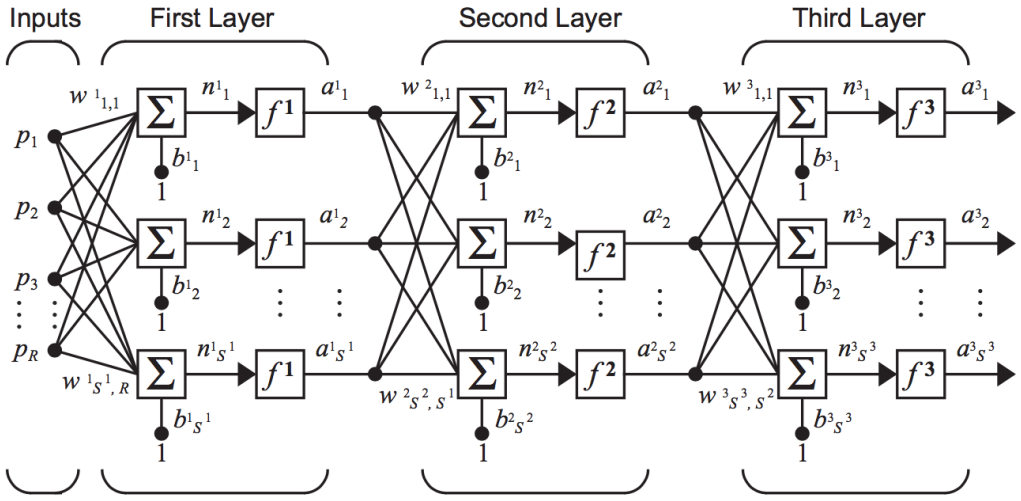

Figure 2.3: Figure taken from [34] representing a multiple-layer neural network, i.e. the composition of more layer functions.

It is called "feed forward" because the arcs joining nodes are unidirectional and there are no cycles. A layer whose output is the network output is called an output layer. The other layers are called hidden layers.

ANNs are generally used to estimate the nonlinear relationship between a highly populated vector of input variables and a vector of output unknowns, for the correlation of which fast and closed form rules cannot easily be applied. In fact, under mild mathematical conditions, any problem involving a continuous mapping between vector spaces can be approximated to arbitrary precision (i.e., within an error tolerance) by feed-forward ANNS, which are the most often used type [17].

Our purpose is to establish a correlation between long-period response spectral ordinates, specifically the spectral acceleration (SA), and short-period ones. Hence, we first need to define what we mean with long and short periods. The idea is to selcet a threshold period T^* and declare periods T such that $T \geq T^*$ "long periods", while if $T < T^*$ we declare them as "short periods". Since in practice there is a sampling of the response spectral ordinates for some specific periods, the cardinality of periods we consider is finite. Specifically the cardinality of the long-periods SA is indicated as N_{SA}^{LP}, while the cardinality of the short-periods SA is indicated asN_{SA}^{SP}. Since to make our predictions in practice we will use the long-periods SA coming from numerical simulations, T^* should be chosen such that the selected long-periods SA estimates are reliable. The value proposed in [54] is $T^* = 0.75$.
In the work of Paolucci et al. [54], the neural network is designed as a two-layer feed-forward neural network with N_n^h sigmoid hidden neurons and a linear output neuron. In particular, the hidden layer activation function is

$$a^h = f^h(n) = tansig(n) = \frac{2}{1 + e^{-2n}} - 1, \tag{2.4}$$

while the output activation function is the identity

$$a^o = f^o(n) = n, \tag{2.5}$$

where n is the output a neuron before applying the activation function. Using the structure defined before and these activation functions, the ANN takes the name of multilayer perceptron [7, 6].
The number of nodes in the input layer, defined to be N_n^i, equals the number of input variables N_{SA}^{LP}. The number of nodes in the output layer, called N_n^o, equals the number of target values N_{SA}^{SP}.

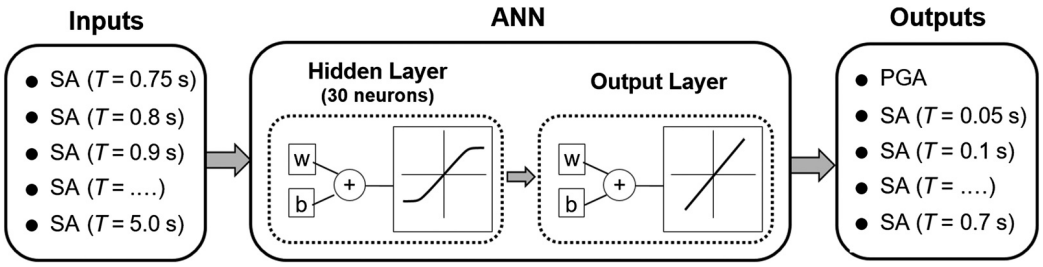

Figure 2.4: Figure taken from [54]. Logic scheme of the ANN training patterns: the long-period spectral ordinates (in this case $T^* = 0.75$ s) represent the teaching inputs, whereas the short-period ones are the outputs predicted by the ANN. The number of neurons in the hidden layer is $N_n^h = 30$. PGA, peak ground acceleration.

Due to the wide range of variability on the data, a logarithmic transformation of the SA is considered, as in [54]. Specifically, the N_{SA}^{LP} input parameters are $\{\log_{10}[\mathrm{SA}(T_k)]\}_{k=1}^{N_{\mathrm{SA}}^{\mathrm{LP}}}$, where SA is the acceleration response spectral ordinates at period T_k, ranging from the threshold period T^* to 5 s. The outputs are N_{SA}^{SP} ground-motion parameters, specifically, $\{\log_{10}[\mathrm{SA}(T_k)]\}_{k=1}^{N_{\mathrm{SA}}^{\mathrm{SP}}}$, ranging from zero up to T^*. SA is the maximum acceleration of an object during an earthquake, modelled as a damped harmonic oscillator with one degree of freedom, cf Chapter 1. As the oscillation period tends to zero, the object displacement tends to the displacement of the forcing term, i.e. the ground displacement. For this reason, while the concept of SA needs to be related to a period $T > 0$, in the limit as $T \to 0$ we recover the peak ground acceleration (PGA). This is why as a convention we will use $SA(T = 0) = PGA$. The ANN is designed to predict multiple outputs given multiple inputs: specifically, considering $T^* = 0.75s$, as in this study, the number of outputs and inputs is 20 and 9, respectively. The input periods selected are 0.75s, 0.8s, 0.9s, 1.0s and from 1.25s up to 5.0s inclusive with step 0.25s. The output periods are 0s, 0.05s and from 0.1s up to 7.0s inclusive with step 0.1s.
In the following we will keep this net structure, with the possibility of adding some input neurons. Indeed, in this work the possibility of adding up to three input neurons will be considered, in order to take into account also epicentral distance (R_{epi}), magnitude of the earthquake (M_w) and velocity of the shear waves in the first 30 meters (V_{s30}).

2.3 Training

2.3.1 Objective Function

To train a neural network means to set its parameters in order to obtain the desired input-output relation. This is done by minimizing an objective function, using a suitable optimization algorithm. Roughly speaking, the objective function represents the distance of model from the data. A common choice, which is the one considered in this work, is the mean squared error (MSE). Specifically, let $f : \mathbb{R}^n \to \mathbb{R}^m$ be our ANN model. Let $\mu \in \mathbb{R}^l$ be the set of parameters of the ANN, so that our response variable is $y = f(x; \mu)$. Given the training set $\{(x^i, y^i)\}_{i=1}^N$ our goal is to minimize a performance measure:

$$\min_{\mu} MSE(\mu) = \min_{\mu} \frac{1}{N} \sum_{i=1}^{N} \sum_{j=1}^{m} (y_j^i - f(x^i; \mu)_j)^2 \tag{2.6}$$

where the subscript j indicates the j-th element of a vector.
Since many optimization algorithms make use of derivatives, it is useful to have differentiable activation functions. When derivatives are computed using the chain rule we are talking about Backpropagation [67]: the idea is to propagate the error signal, computed in single teaching step, back to all connected neurons. In this work, the training will be done using a built-in neural network fitting tool available in MATLAB, namely the package nftool.

2.3.2 Training, Validation and Test Set

When inputs and desired outputs are given as data, we are talking about supervised learning. In our case, we dispose of a high-quality strong ground motion dataset, denoted in the following by SIMBAD (Selected Input Motions for displacement-Based Assessment and Design) [74]. The SIMBAD database presently consists of 467 three-component acceleration time histories from 130 earthquakes worldwide. The main objective is to provide records of engineering relevance for the most frequent design conditions in Italy. For this reason, only records from shallow crustal earthquakes, at epicentral distance R_{epi} approximately less than 35 km, with moment magnitude M_W ranging from 5 to 7.3, are considered. These are the conditions generally governing seismic hazard throughout Italy, for most return periods

of practical interest.
Many information about each earthquake are present within the dataset, among which we report the most significant for our analysis: event name and date, area, M_W , event latitude and longitude, station code and name, station latitude and longitude, elevation, site class according to EC8, V_{S30} measurements, R_{epi}.
Sites were classified into five ground categories according to the European and Italian seismic norms: A ($V_{S30} \geq 800ms$), B ($360 \leq V_{S30} < 800ms$), C ($180 \leq V_{S30} < 360ms$), D ($V_{S30} < 180ms$) and E (site C or D with thickness smaller than 20 m over rigid rock).
Usually the dataset is divided into three subsets: training, validation and test. Their roles are:

- **Training set**: it is used to set the model parameters through optimization algorithms.
- **Validation set**: it is supposed to be made of patterns different from those of the training set and thus can be used to monitor the accuracy of the ANN model during the training procedure.
- **Test set**: it is not used during the ANN training and validation, and it is needed to evaluate the network capability of generalization in the presence of new data.

This distinction helps limiting the problem of overfitting, which is a well known shortcoming of ANN design. In fact, even though the error on the training set is driven to a very small value, the network may fail in generalizing the learned training patterns if the patterns of the training set do not sufficiently cover the variety of possible realisations. A stopping criterion during the training phase can be whenever the error on the validation set starts growing.

2.3.3 Bootstrapping

According the data available in the training set and according other factors, such as the starting point of the optimization algorithm, we might end up in a local minimum of the objective function. Hence, it is necessary to train multiple times the network with different settings. To obtain a new training set we use the so called "Bootstrapping" technique. In general terms, it is

an algorithm that allows to simulate a random sampling when no new data are available. It consists in resampling with replacement from the original dataset, until the size of the new sample is equal to the one of the original dataset.
The idea is that if the dataset is representative of the population, its distribution should be close to the real one. Hence, a sample from the dataset should be, with a certain degree of approximation, analogous to a new sample from the original population.
This algorithm has several advantages:

- it is simple to be implemented;
- allows to check the stability of the results;
- it does not require to know the original population distribution;
- increases the number of samples when the dataset is small.

2.4 Model Evaluation

A key point in the forthcoming analysis is to define suitable criteria to evaluate the goodness of our model.
The procedure described in [54] consists in training 50 different networks, randomly splitting each time the SIMBAD dataset into training, validation and test set, with ratios respectively 70%, 20% and 10% from 95% of the total records. Then, on the remaining 5% set, which we will refer to as TST, the MSE is calculated for each of the 50 networks. Finally the one attaining the lowest MSE is selected. The same set TST can be used to evaluate the goodness of all networks because it has not been used during the training phase of any of them.
However, if we repeat the whole process multiple times, the MSE of the network selected with this method is experimentally highly variable. Hence, using the minimum MSE to estimate the goodness of our model produces unstable, and hence unreliable, results. The reason is that we are testing all of the networks against the same set TST, and this set has small size. Indeed, due to its reduced dimension, it is not representative of the phenomenon. With such small dimensions, there are considerable chances that over 50 networks one of them has a training set similar to TST, leading to

a low MSE value of equation (2.6). The more similar the training set is, the lower the MSE is.
The values of the minimum MSE are probably reflecting this phenomenon, rather than the actual goodness the neural network structure. Hence, either we enlarge the TST set, or we change evaluation method. Unfortunately the former is not possible in the immediate.
First of all, the TST set has been included in the training phase, meaning 100% of the dataset is now split into training, validation and test set. The MSE on the test set of each network is a good performance indicator of its generalization capabilities. Not only the test set is 10% of the dataset, i.e. the size of the TST set is doubled, but it also changes each time for each network, so that the probability of obtaining a test set similar to the training set are lower. However, the test set of one network cannot be used to test another network. This is because these sets are randomly extracted each time, hence data used as test for one net might used as training for another one. Not having a common meter of evaluation for all nets does not constitute an issue, since they will be later compared on specific study cases.
At this point, the minimum MSE could still be the one of net with similar training and test set, even though it is way less likely to happen. For this reason, it is preferable to use a more stable indicator of goodness of the model, that represents more the information coming from all of the trained networks. Hence, we decided to use the mean μ_{MSE} and the standard deviation σ_{MSE} of the MSE over the test set of all the 50 networks:

$$\mu_{MSE} = \sum_{i=1}^{N} MSE_i, \tag{2.7}$$

$$\sigma_{MSE} = \sqrt{\frac{1}{N-1}\sum_{i=1}^{N}(\mu_{MSE} - MSE_i)^2}, \tag{2.8}$$

where $N = 50$ is the number of networks, and MSE_i is the MSE of the i-th network on the its test set.
A low μ_{MSE}, which we will refer to also as bias, indicates the model is performing well and that it is complex enough to capture the phenomenon we are considering. A low σ_{MSE} indicates the model is stable and it is not overfitting the original data, hence it is capable to generalize w.r.t a new dataset.
For our analysis only the networks regarding the horizontal components of

the SA will be considered, since long and short periods SA are more strongly correlated than for the vertical component (for details see Paolucci et al. (2018) [54]).

Chapter 3

Input Selection

3.1 Introduction

The goal of our feedforward neural network is to estimate the spectral acceleration (SA) for periods below $T^* = 0.75s$ and the peak ground acceleration (PGA). The information provided as input are the SA above T^*, which are strictly case specific and hence essential to produce meaningful outputs. However we have at our disposal many other information like the epicentral distance of each station (R_{epi}), the magnitude of the earthquake (M_w) and velocity of the shear waves in the first 30 meters of soil (V_{s30}). It is not so clear whether these information can help to obtain better estimates, and the purpose of this chapter is to investigate which inputs should be selected.

3.2 Evaluation Principles

Hyperparameters are parameters whose value is set before the learning process, such as the number of neurons in the hidden layer. To interpret the way bias and variance change when varying the hyperparameters, we will rely on the following principles:

1. An input which carries a lot of information about the output might decrease bias and/or variance, while on the contrary if it carries a few information we might have an increase of bias/variance.

2. There is a bias-variance tradeoff: augmenting the number of hyperparameters leads to lower bias and higher variance. On the other hand,

decreasing the number of hyperparameters can have the opposite effect.

3. The so called capacity is the ability of a model to fit a wide variety of functions and it is determined by the number and the quality of the hyperparameters. The optimal model capacity increases with the amount of data.

3.3 Hyperparameters Analysis

To carry out an hyperparameter analysis, it is possible to proceed in three different ways:

1. adding a combination of new inputs among R_{epi}, M_w and V_{s30};
2. training the net for soft and stiff soil separately, since very different SA behaviours arise in the two cases;
3. varying the number neurons in the hidden layer.

Since the number of possible combinations grows too rapidly, we will proceed heuristically, using the three principles stated in the previous section to guide our analysis.

3.3.1 Inputs Analysis

As a first analysis we test all of the possible combinations of input. A standard value of 30 neurons in the hidden layer, suggested by Paolucci et al. (2018) [54], was kept in all cases. In Table (3.1) we report the mean (defined in (2.7)), also called bias, and the standard deviation (defined as in (2.8)) for the mean squared error (MSE) (defined as in (2.6)) of the 50 networks generated with the procedure described in the Chapter 2, with all of the possible combinations of additional inputs.

additional input	bias	std
none	0.0668	0.0079
R_{epi}	0.0633	0.0091
M_w	0.0659	0.0097
V_{s30}	0.0635	0.0093
all	0.0573	0.0083
M_w and V_{s30}	0.0607	0.0084
R_{epi} and V_{s30}	0.0609	0.0088
R_{epi} and M_w	0.0580	0.0083

Table 3.1: Mean squared error mean (bias) and standard deviation (std) for networks generated with all of the possible combinations of additional inputs, using 30 neurons.

Adding only one input results in a slight bias decrease and a slight variance increase, as expected from the bias-variance tradeoff (principle 2), compared to the standard case with no additional inputs.
Adding all of the inputs results instead in a considerable decrease of the bias while the standard deviation does not vary significantly. This is possibly due to the fact that, although variance should increase with the number of hyperparameters (principle 2), the added informative content is high enough to balance this effect and improve the performance (principle 1). As expected, these numerical results suggest that the three familes of inputs carry complementary information. This hypothesis is also confirmed by the fact that adding only two of them further decreases the bias, with respect to the one input cases, without increasing significantly the variance.
Adding all of the three inputs produces the same results as adding only R_{epi} and M_w. It might look like the information contained in V_{s30} is already contained in the other two variables but this is not true: adding one extra variable should by itself increase the variance (principle 2) so keeping the same bias and variance means that the input carries at least a little information (principle 1). Also it is well known that soil condition strongly affects the output so the fact there is no gain adding V_{s30} is rather due to the fact that there are no enough data for a model of such capacity (principle 3). This hypothesis is further supported by the fact that V_{s30} has a wide range of variability (it is indeed used to classify the soil in macro categories), hence a large amount of data are needed to exploit it properly.

3.3.2 Soil Classification Analysis

Since V_{s30} is used to classify the soil as stiff or soft, in principle it does not make sense to explore the possibility of both training the network only on a specific site class dataset and adding V_{s30} as input.
The possibility of training the net for soft and stiff soil separately was explored by Gatti (2017) [28] using 30 neurons, with no significant variations due to fact that the amount of data reduces too much (principle 3). However we cannot a priori exclude that changing the inputs or the number of neurons could indeed work better.
All of the combinations with R_{epi} and M_w were considered for a fixed number of 30 neurons.
In Table 3.2 we show the MSE bias and standard deviation for networks trained on data relative to different site classes (soft, stiff or all data), and with either the classical set of inputs (denoted by none) and additional inputs(R_{epi}, M_w and R_{epi} and M_w).

additional input	soft soil data		stiff soil data		all data	
	bias	std	bias	std	bias	std
none	0.0655	0.0101	0.0673	0.0140	0.0678	0.0079
R_{epi}	0.0642	0.0129	0.0646	0.0101	0.0633	0.0091
M_w	0.0629	0.0125	0.0653	0.0104	0.0659	0.0097
R_{epi} and M_w	0.0584	0.0120	0.0634	0.0107	0.0580	0.0083

Table 3.2: Mean squared error mean (bias) and standard deviation (std) for networks trained on data relative to different site classes (soft soil, stiff soil or all data), and with either the classical set of inputs (denoted by none) and additional inputs(R_{epi}, M_w and R_{epi} and M_w), using 30 neurons.

In all cases the variance increases, leading to a higher risk of overfitting. The available dataset is too poor to balance the number of hyperparameters (principle 3) leading to worse performance, which is what has been reported also by Gatti (2017) [28]. Unexpectedly, the bias for the stiff soil in the two inputs case increases significantly, possibly because:

- it is not a deterministic procedure;
- averaging on 50 networks might not be enough for a detailed analysis;

- there are complex interaction we did not or cannot account for.

However for our heuristics considerations the above problem is of minor interest. In conclusion, without additional data this option does not provide good results.

3.3.3 Number of Neurons Exploration

Setting the number of neurons for the hidden layer is a procedure to tune the model capacity. This is usually performed as the last part of the analysis because:

- a reference number of 30 neurons is already available, cf [54];
- the number of all possible combinations varying this hyperparameter is to high and hence it is better to select first which inputs to use;
- we expect that the performance improvement is less significant compared to the variation attained choosing different inputs.

The most promising networks were: i) the one with all inputs; ii) the one with only R_{epi} and M_w. What we expect is that by increasing the number of neurons the bias decreases and variance increases, while by decreasing the number of neurons the bias increases and variance decreases (principle 2). However it is not obvious how bias and variance change as functions of the number of neurons and hence both directions should be considered. In Table 3.3 we show the mean squared error mean (bias) and standard deviation (std) for different numbers of neurons, for a net with magnitude (M_w) and epicentral distance (R_{epi}) as input, and a net with in addition the velocity of the shear waves in the first 30 meters (V_{s30}) (all inputs).

neurons	all inputs		R_{epi} and M_w	
	bias	std	bias	std
20	0.0594	0.0073	0.0566	0.0079
25	0.0569	0.0068	0.0595	0.0075
30	0.0573	0.0083	0.0580	0.0083
35	0.0587	0.0079	0.0575	0.0070
40	0.0579	0.0069	0.0602	0.0060

Table 3.3: Mean squared error mean (bias) and standard deviation (std) for different numbers of neurons, for a net with magnitude (M_w) and epicentral distance (R_{epi}) as input, and a net with in addition the velocity of the shear waves in the first 30 meters (V_{s30}) (all inputs).

As expected the variations are much smaller then observed when selecting the additional inputs. It seems that the optimum is attained with 25 and 35 neurons for the network with all the inputs and the one with only R_{epi} and M_w respectively. We know that the error as a function of the model capacity should have a minimum which depends on the amount of data (principle 3). Indeed three inputs might imply a too high capacity, which is why 25 neurons appear to work better in this case, while two inputs might imply a too low capacity, which is why 35 neurons appear to work better in the other case.

3.4 Conclusions

Training nets separately for different class sites appears not to work because of a not rich enough dataset.
Two possible nets have been select, taking into account inputs significance and amount of data available:

1. Net ALL: with 25 neurons in the hidden layer, using R_{epi}, M_w and V_{s30} as additional inputs.

2. Net RM: with 35 neurons in the hidden layer, using R_{epi} and M_w as additional inputs.

In Table 3.4 we show a summary of the elements characterizing net ALL, net RM and net OLD, the latter being the net with the original settings (no additional inputs, 30 neurons) described in Chapter 2.

net	inputs	neurons	bias	std
OLD	SA	30	0.0683	0.0101
RM	SA, M_w, R_{epi}	35	0.0575	0.0070
ALL	SA, M_w, R_{epi}, V_{s30}	25	0.0569	0.0068

Table 3.4: Summary of the elements characterizing net ALL, net RM and net OLD, the latter being the net with the original settings (no additional inputs, 30 neurons). More precisely we report the inputs used to train the network (inputs), where SA is the spectral acceleration defined in Chapter 1, the mean squared error mean (bias) and standard deviation (std), and the number of neurons (neurons).

Both the nets ALL and RM attain very similar performance and training computational costs since net ALL has less neurons but one extra input. Moreover having V_{s30} as extra input in the net ALL looks in principle correct, since soil conditions should be taken into account. However, simpler models are usually more robust. Moreover, collecting precised enough information might be problematic, since often even in the SIMBAD dataset only the site class is provided, which corresponds to a range for the variable V_{s30} and not to its precised value. In our case, when only the site class was provided the conversion in the table 3.5 was applied.

site class	V_{s30} interval $[m/s]$	representative value $[m/s]$
A	800 >	900
B	360 - 800	580
C	180 - 360	270
D	0 - 180	90

Table 3.5: Average shear-waves velocity in first 30 meters (V_{s30}) representative values chosen for each site class.

Even though such approximation had to be applied for 16.58 % of the data, the results obtained were satisfactory.
In conclusion both ANNs should be kept for further testings on meaningful cases like L'Aquila, Po plain and Norcia, which will be done in the following.

Chapter 4

Optimization Algorithms

4.1 Introduction

To train a neural network means to set its parameters in order to obtain the desired input-output relation. This is done by minimizing an objective function $f(x)$, using an optimization algorithm. The purpose of this chapter is to provide a brief description of some optimization algorithms that can be used to train neural networks. In the following we adopt the notation and the presentation structure of [34] and [50]. We will consider an objective function $f : \mathbb{R}^n \to \mathbb{R}$, $f \in C^2(\mathbb{R}^n)$ to be minimized, i.e. we look for $x^* \in \mathbb{R}^n$ such that $f(x^*) = \min_{x \in \mathbb{R}^n} f(x)$.

4.2 Line Search Methods

In the line search strategy, the algorithm chooses a direction p_k and searches along this direction from the current iterate x_k for a new iterate with a lower function value. Given an initial guess $x_0 \in \mathbb{R}^n$, the step k has the form:

$$x_{k+1} = x_k + \alpha_k p_k, \tag{4.1}$$

where $x_k, p_k \in \mathbb{R}^n$ and $\alpha_k > 0$. The distance to move along p_k can be found by approximately solving the following one dimensional minimization problem to find a step length α_k:

$$\min_{\alpha_k > 0} f\left(x_k + \alpha_k p_k\right), \tag{4.2}$$

By solving (4.2) exactly (exact linear search), we would derive the maximum benefit from the direction p_k, but an exact minimization may be expensive and is usually unnecessary. Another possibility is to generate a limited number of trial step lengths until a step that loosely attained the minimum of (4.2) is found (inexact linear search). At the new point, a new search direction and step length are computed, and the process is repeated.
Popular inexact line search conditions, called *Wolfe conditions*, state that α_k should give sufficient decrease in the objective function f without being too small, as measured by the following inequalities:

$$\begin{aligned} f\left(x_k + \alpha_k p_k\right) &\leq f\left(x_k\right) + c_1 \alpha_k \nabla f_k^T p_k \\ \nabla f\left(x_k + \alpha_k p_k\right)^T p_k &\geq c_2 \nabla f_k^T p_k \end{aligned} \tag{4.3}$$

for some constant $c_1 \in (0,1)$, $c_2 \in (C_1,1)$. Here, we adopt the short-hand notation $\nabla f_k = \nabla f(x_k)$. In other words, the reduction in f should be proportional to both the step length α_k and the directional derivative $\nabla f_k^T p_k$.
Regarding the convergence of line search methods we have the following theorem [50].

Theorem 1. *Consider any iteration of the form (4.1), where p_k is a descent direction and α_k s atisfies the Wolfe conditions (4.3). Suppose that f is bounded below in $\mathbb{R}^n$ and that f is continuously differentiable in an open set $\mathcal{N}$ containing the level set $\mathcal{L} \stackrel{\text{def}}{=} \{x : f(x) \leq f\left(x_0\right)\}$, where $x_0 \in \mathbb{R}^n$ is the starting point of the iteration. Assume also that the gradient ∇f is Lipschitz continuous on $\mathcal{N}$, that is, there exists a constant $L > 0$ such that*

$$\|\nabla f(x) - \nabla f(\tilde{x})\| \leq L\|x - \tilde{x}\|, \quad \text{for all } x, |\tilde{x} \in \mathcal{N} \tag{4.4}$$

then

$$\sum_{k \geq 0} \cos^2 \theta_k \left\|\nabla f_k\right\|^2 < \infty, \tag{4.5}$$

where

$$\cos \theta_k = \frac{-\nabla f_k^T p_k}{\|\nabla f_k\| \, \|p_k\|}. \tag{4.6}$$

4.3 The Steepest Descent Method

We consider the first-order Taylor series expansion of $f(x)$ centered in x_k, i.e.

$$f\left(x_{k+1}\right) = f\left(x_k + \alpha_k p_k\right) \approx f\left(x_k\right) + \nabla f(x_k)^T \alpha_k p_k, \tag{4.7}$$

For $f(x_{k+1})$ to be less than $f(x_k)$ it must hold

$$\nabla f(x_k)^T \alpha_k p_k < 0. \tag{4.8}$$

To satisfy the above condition we can set $p_k = -\nabla f(x_k)$.
For quadratic functions of the form $f(x) = \frac{1}{2}x^T Q x - b^T x$ we then immediately have

$$\alpha_k = \frac{\nabla {f_k}^T \nabla f_k}{\nabla f_k^T Q \nabla f_k}. \tag{4.9}$$

The following theorem concerns the convergence rate of a steepest descent method.

Theorem 2. *Suppose that $f : \mathbb{R}^n \to \mathbb{R}$ is twice continuously differentiable, and that the iterates generated by the steepest-descent method with exact line searches converge to a point x^* at which the Hessian matrix $\nabla^2 f(x^*)$ is positive definite. Let r be any scalar satisfying*

$$r \in \left(\frac{\lambda_n - \lambda_1}{\lambda_n + \lambda_1}, 1 \right), \tag{4.10}$$

where $\lambda_1 \leq \lambda_2 \leq \cdots \leq \lambda_n$ are the eigenvalues of $\nabla^2 f(x^*)$. Then for all k sufficiently large, we have

$$f\left(x_{k+1}\right) - f\left(x^*\right) \leq r^2 \left[f\left(x_k\right) - f\left(x^*\right) \right]. \tag{4.11}$$

In general, we cannot expect the rate of convergence to improve if an inexact line search is used. Therefore, the above theorem shows that the steepest descent method can have a very slow rate of convergence, even when the Hessian is reasonably well conditioned.

4.4 The Newton's Method

The Newton's method is based on the second-order Taylor expansion of f near the point $x_k + \Delta X_k$:

$$f\left(x_{k+1}\right) = f\left(x_k + \Delta x_k\right) \approx f\left(x_k\right) + \nabla f_k^T \Delta x_k + \frac{1}{2} \Delta x_k^T \nabla^2 f_k \Delta x_k, \tag{4.12}$$

where $\nabla^2 f_k = \nabla^2 f(x_k)$. The quantity Δx_k minimizing the expansion is given by

$$p_k^{\mathrm{N}} = -\nabla^2 f_k^{-1} \nabla f_k \tag{4.13}$$

Since the Hessian matrix $\nabla^2 f_k$ might not always be positive definite, p_k^{N} might not always be a descent direction.
The following theorem provides sufficient conditions to ensure the local convergence of the Newton's method.

Theorem 3. *Suppose that f is twice differentiable and that the Hessian matrix $\nabla^2 f(x)$ is Lipschitz continuous in a neighborhood of a solution x^* at which $\nabla f(x^*) = 0$, and that $\nabla^2 f(x^*)$ is positive definite. Consider the Newton iteration $x_{k+1} = x_k + p_k$, where $p_k = -\nabla^2 f_k^{-1} \nabla f_k$. Then*

1. *if the starting point x_0 is sufficiently close to x^*, the sequence of iterates given by the Newton's method converges to x^*;*
2. *the rate of convergence of $\{x_k\}$ is quadratic;*
3. *the sequence of gradient norms $\{||\nabla f_k||\}$ converges quadratically to zero.*

While Newton's method usually produces faster convergence than steepest descent, it is possible for the algorithm to oscillate or diverge.
Another problem with Newton's method is that it requires the computation and storage of the Hessian matrix, as well as its inverse.

4.5 The BFGS Method

The Broyden - Fletcher - Goldfarb - Shanno (BFGS) method [31] is a quasi-Newton method, meaning the following quadratic model m_k of the objective function at the current iterate x_k is used:

$$m_k(p) = f_k + \nabla f_k^T p + \frac{1}{2} p^T B_k p, \tag{4.14}$$

where B_k is an $n \times n$ symmetric positive definite matrix that approximates the Hessian matrix. Note that the function value and gradient of this model at $p = 0$ match f_k and ∇f_k, respectively. The minimizer p_k of this convex quadratic model can be written explicitly as

$$p_k = -B_k^{-1} \nabla f_k. \tag{4.15}$$

Instead of computing B_k at every iteration, the idea is to update it in a simple manner to account for the curvature measured during the most recent step. Suppose we have generated a new iterate x_{k+1} and want to construct a new quadratic model m_{k+1}, of the form

$$m_{k+1}(p) = f_{k+1} + \nabla f_{k+1}^T p + \frac{1}{2} p^T B_{k+1} p. \tag{4.16}$$

One reasonable requirement is that the gradient of m_{k+1} should match the gradient of the objective function $f(\cdot)$ at the last two iterates x_k and x_{k+1}. Since $\nabla m_{k+1}(0)$ is precisely ∇f_{k+1}, the second of these conditions is satisfied automatically. The first condition can be written as

$$\nabla m_{k+1}\left(-\alpha_k p_k\right) = \nabla f_{k+1} - \alpha_k B_{k+1} p_k = \nabla f_k. \tag{4.17}$$

By rearranging the terms, we obtain

$$B_{k+1}\alpha_k p_k = \nabla f_{k+1} - \nabla f_k. \tag{4.18}$$

To simplify the notation it is useful to define the vectors

$$s_k = x_{k+1} - x_k = \alpha_k p_k, \quad y_k = \nabla f_{k+1} - \nabla f_k, \tag{4.19}$$

so that we get

$$H_{k+1} y_k = s_k. \tag{4.20}$$

where we have $H_{k+1} = B_{k+1}^{-1}$. To identify H_{k+1} uniquely we solve

$$\begin{aligned} &\min_H \|H - H_k\| \\ &\text{subject to} \quad H = H^T, \quad H y_k = s_k \end{aligned} \tag{4.21}$$

meaning that we look for the matrix H_{k+1} closest to H_k. Choosing a suitable norm (a weighted Frobenius norm) the solution is easily computable. Indeed we get

$$B_{k+1} = B_k - \frac{B_k s_k s_k^T B_k}{s_k^T B_k s_k} + \frac{y_k y_k^T}{y_k^T s_k}. \tag{4.22}$$

The next theorem concerns the convergence rate of the BFGS method, cf [50].

Theorem 4. *Suppose that $f : \mathbb{R}^n \to \mathbb{R}$ is twice continuously differentiable. Given an initial guess $x^0 \in \mathbb{R}^n$, consider the iteration $x_{k+1} = x_k + p_k$, where $p_k = -B_k^{-1}\nabla f_k$. Let us assume also that $\{x_k\}_{k\geq 0}$ converges to a point x^* such that $\nabla f(x^*) = 0$ and $\nabla^2 f(x^*)$ is positive definite. Then the sequence $\{x_k\}_{k\geq 0}$ converges superlinearly if and only if*

$$\lim_{k\to\infty} \frac{\|(B_k - \nabla^2 f\,(x^*))\, p_k\|}{\|p_k\|} = 0. \tag{4.23}$$

The BFGS method is robust, and its rate of convergence is superlinear, which is fast enough for most practical purposes. Even though Newton's method converges more rapidly (that is, quadratically), its cost per iteration usually is higher, because of the need for second derivatives and the solution of a linear system.

4.6 The Levenberg-Marquardt Method

In least-squares problems, the objective function f has the following special form:

$$f(x) = \frac{1}{2}\sum_{j=1}^{m} r_j^2(x), \tag{4.24}$$

where each r_j is a smooth function from $\mathbb{R}^n$ to $\mathbb{R}$. We refer to each r_j as a residual. This is often the case satisfied by the training step in ANNs, where the residual is a measure of the distance to the data.
Let

$$r(x) = (r_1(x), r_2(x), \ldots, r_m(x))^T, \tag{4.25}$$

and

$$J(x) = \left[\frac{\partial r_j}{\partial x_i}\right]_{j=1,2,\ldots,m \;\; i=1,2,\ldots,n}. \tag{4.26}$$

Then, it immediately follows

$$\nabla f(x) = \sum_{j=1}^{m} r_j(x)\nabla r_j(x) = J(x)^T r(x), \tag{4.27}$$

$$\begin{aligned}\nabla^2 f(x) &= \sum_{j=1}^{m} \nabla r_j(x) \nabla r_j(x)^T + \sum_{j=1}^{m} r_j(x) \nabla^2 r_j(x) \\ &= J(x)^T J(x) + \sum_{j=1}^{m} r_j(x) \nabla^2 r_j(x).\end{aligned} \tag{4.28}$$

In many applications, the first partial derivatives of the residuals and hence the Jacobian matrix $J(x)$ are relatively easy or inexpensive to be computed. We can thus easily obtain the gradient $\nabla f(x)$ and the first part of the Hessian, using the above formulas (4.27) (4.28). Moreover, the second term of the Hessian in (4.28) is often less important than the first one, either because the residuals r_j are close to affine near the solution (that is, $\nabla^2 r_j(x)$ are relatively small) or because of small residuals (that is, $r_j(x)$ are relatively small).
The Gauss-Newton method has the general form

$$x_{k+1} = x_k + \alpha_k p_k^{GN}, \tag{4.29}$$

where in order to define p_k^{GN} we make use of this approximation solving the following system to obtain the search direction p_k^{GN}:

$$J_k^T J_k p_k^{\mathrm{GN}} = -J_k^T r_k, \tag{4.30}$$

instead of solving the standard Newton equations $\nabla^2 f(x_k)\, p = -\nabla f(x_k)$. One problem with the Gauss-Newton method is that the matrix $H_k = J^T(x_k) J(x_k)$ might not be invertible. This can be overcome by using the following modification to the approximate Hessian matrix

$$G_k = H_k + \mu_k I. \tag{4.31}$$

These G_k has the same eigenvalues of H_k shifted by μ_k. With a large enough $\mu_k > 0$, G_k is positive definite, hence invertible. Moreover, if G_k can therefore be made positive definite, the solution p_k^{LM} of the following linear system

$$G_k p_k^{LM} = -J_k^T r_k, \tag{4.32}$$

is a descent direction. The Levenberg-Marquardt algorithm is therefore

$$x_{k+1} = x_k - [J^T(x_k) J(x_k) + \mu_k I]^{-1} J(x_k) r(x_k) \tag{4.33}$$

This algorithm has a number of useful features. Indeed as μ_k increases, it approaches the steepest descent algorithm with small learning rate, i.e.

$$x_{k+1} \cong x_k - \frac{1}{\mu_k} J^T(x_k)\, v(x_k) = x_k - \frac{1}{2\mu_k} \nabla f(x), \text{ for large } \mu_k. \tag{4.34}$$

If μ_k tends to zero the algorithm becomes the Gauss-Newton method. The algorithm starts with μ_k set to some small value. If a step does not yield a smaller value for f, i.e. $f(x_{k+1}) \geq f(x_k)$, then the step is repeated with μ_k multiplied by some factor $\vartheta > 1$. As observed before, as μ_k gets larger this algorithm approaches the steepest descent algorithm, which guarantees to decrease the value of $f(x^{k+1})$. On the contrary, if $f(x_{k+1}) < f(x_k)$, then μ_k is divided by ϑ for the next step, so that the algorithm will approach Gauss-Newton algorithm, which features higher convergence rate.

4.7 The Conjugate Gradient Method

The conjugate gradient method is an iterative method for solving a linear system of equations

$$Ax^* = b \tag{4.35}$$

where A is an $n \times n$ symmetric positive definite matrix. This problem can be stated equivalently as the following minimization problem:

$$\min_{x \in \mathbb{R}^n} \phi(x), \tag{4.36}$$

where

$$\phi(x) = \frac{1}{2} x^T A x - b^T x. \tag{4.37}$$

We recall that a set of nonzero vectors $p_0, p_1, \ldots, p_l \in \mathbb{R}^n$ is said to be conjugate with respect to the symmetric positive definite matrix A if

$$p_i^T A p_j = 0, \qquad \text{for all } i \neq j. \tag{4.38}$$

Given a starting vector $x_0 \in \mathbb{R}^n$ and a set of conjugate directions $\{p_0, p_1, \ldots, p_{n-1}\}$, consider the sequence

$$x_{k+1} = x_k + \alpha_k p_k, \quad k \geq 0 \tag{4.39}$$

where

$$\alpha_k = -\frac{r_k^T p_k}{p_k^T A p_k}, \tag{4.40}$$

minimizes the quadratic functional ϕ defined in 4.37, and

$$r_k = Ax_k - b. \tag{4.41}$$

The following results ensure convergence and key properties of the algorithm.

Theorem 5. *For any $x_0 \in \mathbb{R}^n$ the sequence $\{x_k\}$ generated by the conjugate direction algorithm (4.39), converges to the solution x^* of the linear system (4.35) in at most n steps.*

Theorem 6. *Let $x_0 \in \mathbb{R}^n$ be any starting vector and suppose that the sequence $\{x_k\}$ is generated by the conjugate direction algorithm. Then*

$$r_k^T p_i = 0, \quad \text{for } i = 0, 1, \ldots, k-1. \tag{4.42}$$

Moreover x_k is the minimizer of $\phi(x) = \frac{1}{2}x^T Ax - b^T x$ over the set

$$\{x | x = x_0 + \text{span}\{p_0, p_1, \ldots, p_{k-1}\}\}. \tag{4.43}$$

The conjugate gradient method is a conjugate direction method with a particular property: in generating its set of conjugate vectors, it can compute a new vector p_k by using only the previous vector p_{k-1}. It does not need all the previous elements $p_0, p_1, ..., p_{k-2}$ of the conjugate set; p_k is automatically conjugate to these vectors. This remarkable property implies that the method requires little storage and computation.
In the conjugate gradient method, each direction p_k is chosen to be a linear combination of the negative residual $-r_k$ (which is the steepest descent direction for the function ϕ) and the previous direction p_{k-1}. We write

$$p_k = -r_k + \beta_k p_{k-1}, \tag{4.44}$$

By premultiplying by $p_{k-1}^T A$ and imposing the condition $p_{k-1}^T A p_k = 0$, we find that

$$\beta_k = \frac{r_k^T A p_{k-1}}{p_{k-1}^T A p_{k-1}}. \tag{4.45}$$

In practice, the conjugate gradient algorithm can be formulated as reported in Algorithm 1.

Given x_0;
Set $r0 \leftarrow Ax_0$;
while $r_k \neq 0$ **do**

$$\alpha_k \leftarrow \frac{r_k^T r_k}{p_k^T A p_k}; \tag{4.46}$$

$$x_{k+1} \leftarrow x_k + \alpha_k p_k; \tag{4.47}$$

$$r_{k+1} \leftarrow r_k + \alpha_k A p_k; \tag{4.48}$$

$$\beta_{k+1} \leftarrow \frac{r_{k+1}^T r_{k+1}}{r_k^T r_k}; \tag{4.49}$$

$$p_{k+1} \leftarrow -r_{k+1} + \beta_{k+1} p_k; \tag{4.50}$$

$$k \leftarrow k + 1; \tag{4.51}$$

end

Algorithm 1: The Conjugate Gradient Method.

4.7.1 The Fletcher-Reeves Conjugate Gradient Method

An extension of the conjugate gradient method to nonlinear functions can be provided by making simple changes in Algorithm 1. We first need to perform a line search that identifies an approximate minimum of the non linear function f along the direction p_k. Second, the residual r_k, which is simply the gradient of $\phi(x_k)$, must be replaced by the gradient of the nonlinear objective f. These changes give rise to the Fletcher-Reeves Conjugate Gradient (FRCG) algorithm for nonlinear optimization, which is described in Algorithm 2.

Given x_0;
Evaluate $f_0 = f(x_0), \nabla f_0 = \nabla f(x_0)$;
Set $p_0 \leftarrow -\nabla f_0, k \leftarrow 0$;
while $\nabla f_k \neq 0$ **do**
Compute α_k and set $x_{k+1} = x_k + \alpha_k p_k$;
Evaluate ∇f_{k+1};

$$\beta_{k+1}^{\mathrm{FR}} \leftarrow \frac{\nabla f_{k+1}^T \nabla f_{k+1}}{\nabla f_k^T \nabla f_k}; \tag{4.52}$$

$$p_{k+1} \leftarrow -\nabla f_{k+1} + \beta_{k+1}^{\mathrm{FR}} p_k; \tag{4.53}$$

$$k \leftarrow k + 1; \tag{4.54}$$

end

Algorithm 2: Fletcher-Reeves Conjugate Gradient (FRCG) Method.

In FRCG algorithm is appealing for large nonlinear optimization problems because each iteration requires only the evaluation of the objective function and its gradient. No matrix operations are required for the steps (4.52)-(4.54), and just a few vectors need to be stored.
The search direction p_k may fail to be a descent direction unless α_k satisfies the strong Wolfe conditions:

$$\begin{aligned} f(x_k + \alpha_k p_k) &\leq f(x_k) + c_1 \alpha_k \nabla f_k^T p_k, \\ \left| \nabla f(x_k + \alpha_k p_k)^T p_k \right| &\leq -c_2 \nabla f_k^T p_k, \end{aligned} \tag{4.55}$$

where $0 < c_1 < c_2 < \frac{1}{2}$.

4.7.2 The Polak-Ribière Conjugate Gradient Method

Many variants of the Fletcher–Reeves CG method exist. They differ from each other mainly in the choice of the parameter β_k. An important variant, proposed by Polak and Ribière, defines this parameter as follows:

$$\beta_{k+1}^{\mathrm{PR}} = \frac{\nabla f_{k+1}^T (\nabla f_{k+1} - \nabla f_k)}{\|\nabla f_k\|^2} \tag{4.56}$$

It is identical to the FRCG algotirhm when f is a strongly convex quadratic function and the line search is exact. When applied to general nonlinear

functions with inexact line search, however, the behavior of the two algorithms differs. Numerical experiments indicate that the Polak and Ribière CG algorithm tends to be the more robust and efficient than the FRCG one.

4.7.3 The Scaled Conjugate Gradient Method

The Scaled Conjugate Gradient method [2] considers an update of the direction p_k of the form

$$p_{k+1} = -\theta_{k+1} g_{k+1} + \beta_k s_k, \quad k \geq 0 \tag{4.57}$$

where $g_k = \nabla f_k$, $s_k = x_{k+1} - x_k = \alpha_k p_k$ and

$$\beta_k = \frac{(\theta_{k+1} y_k - s_k)^T g_{k+1}}{y_k^T s_k}. \tag{4.58}$$

Observe that if $\theta_{k+1} = 1$, then we get the classical conjugate gradient algorithms according to the value of the scalar parameter β_k. On the other hand, if $\beta_k = 0$, then we get another class of algorithms according to the selection of the parameter θ_{k+1}. There are two possibilities for θ_{k+1}: a positive scalar or a positive definite matrix. If $\theta_{k+1} = 1$ we have the steepest descent algorithm. If $\theta_{k+1} = (\nabla^2 f_k)^{-1}$, or an approximation of it, then we get the Newton or the quasi-Newton algorithms, respectively. Therefore, we can see that in the general case, when $\theta_{k+1} \neq 0$ is selected in a quasi-Newton manner and $\beta_{k+1} \neq 0$, then we have a combination between the quasi-Newton and the conjugate gradient methods.
If $s_j^T \nabla f_{j+1} = 0$, $j = 0, 1, \ldots, k$ we get

$$p_{k+1} = -\theta_{k+1} g_{k+1} + \frac{\theta_{k+1} y_k^T g_{k+1}}{\alpha_k \theta_k g_k^T g_k} s_k, \tag{4.59}$$

which is the direction corresponding to a generalization of the Polak and Ribière formula. If additionally the successive gradients are orthogonal, then

$$p_{k+1} = -\theta_{k+1} g_{k+1} + \frac{\theta_{k+1} g_{k+1}^T g_{k+1}}{\alpha_k \theta_k g_k^T g_k} s_k, \tag{4.60}$$

which is the direction corresponding to a generalization of the Fletcher and Reeves formula. Therefore, (4.60) is a general formula for direction computation in a conjugate gradient manner including the classical Fletcher and Reeves, and Polak and Ribière algorithms.

4.8 The Resilient Backpropagation Method

In this section we desribe the so called "resilient backpropagation method". Let $\Delta_{k+1} \in \mathbb{R}^n$ be a vector of step lengths for every coordinate direction

$$\Delta^i_{k+1} = \begin{cases} \eta^+ \Delta^i_k & \text{if } \nabla f^i_k \nabla f^i_{k-1} > 0 \\ \eta^- \Delta^i_k & \text{if } \nabla f^i_k \nabla f^i_{k-1} < 0 \\ \Delta^i_k & \text{else} \end{cases} \tag{4.61}$$

where $0 < \eta^- < 1 < \eta^+$ and the apex i indicates the i-th element of a vector. The adaptation rule works as follows: every time the partial derivative of the corresponding x^i changes its sign, which indicates that the last update was too big and the algorithm has jumped over a local minimum, the update value Δ^i is decreased by the factor η^-. If the derivative retains its sign, the update value is slightly increased in order to accelerate convergence in shallow regions.
If the partial derivative changes sign, i.e. the previous step was too large and the minimum was missed, the previous weight update is reverted, i.e.

$$\Delta^i_{k+1} = -\Delta^i_k \quad \text{if } \nabla f^i_k \nabla f^i_{k-1} < 0. \tag{4.62}$$

The derivative is supposed to change its sign once again in the following step. In order to avoid a double punishment of the update value, there should be no adaptation of the update value in the succeeding step. In practice this can be done by setting $\nabla f^i_k = 0$ in (4.61).
If $\nabla f^i_k \nabla f^i_{k-1} \geq 0$, the update is analogous to the steepest descent method but only the sign of the gradient is taken into account, while the step length is the one determined above:

$$x^i_{k+1} = x^i_k + sign(-\nabla f^i_k)\Delta^i_k \quad \text{if } \nabla f^i_k \nabla f^i_{k-1} \geq 0. \tag{4.63}$$

Multilayer networks typically use sigmoid transfer functions in the hidden layers. Sigmoid functions are characterized by the fact that their slopes must approach zero as the input gets large. Training these network with a steepest descent algorithm causes small changes in their weights and biases, even though the weights and biases are far from their optimal values, because the gradient can have a very small magnitude.
The resilient backpropagation training algorithm [65] is particularly well suited in these cases, since only the sign of the derivative can determine the direction of the weight update and not their magnitude.

Chapter 5

Training Algorithm Selection

5.1 Introduction

The best suited algorithm to train a neural network depends on many factors, such as the number of data in the training set, the number of weights and biases in the network and the error goal. The aim of this chapter is to investigate which algorithm is best suited for net ALL and net RM.

5.2 Performance Measure and Overfitting

Let $f : \mathbb{R}^n \to \mathbb{R}^m$ be our ANN model. Let $\mu \in \mathbb{R}^l$ be the set of parameters of the ANN, so that our response variable is $y = f(x; \mu)$. Given the training set $\{(x^i, y^i)\}_{i=1}^N$ for $N \gg n$ our goal is to minimize a performance measure, representing the distance of the prediction to the real data, which we recall that in our case is the mean squared error (MSE):

$$\min_{\mu} \frac{1}{N} \sum_{i=1}^{N} \sum_{j=1}^{m} (y_j^i - f(x^i; \mu)_j)^2 \tag{5.1}$$

where the subscript j indicates the j-th element of the vector.
In order to avoid overfitting, a regularization term is usually added by penalizing large parameter values, i.e. (5.1) is replaced by

$$\min_{\mu} \frac{1}{N} \sum_{i=1}^{N} \sum_{j=1}^{m} (y_j^i - f(x^i; \mu)_j)^2 + \gamma ||\mu||_2^2, \tag{5.2}$$

where $\gamma > 0$.
Another procedure to avoid overfitting is called "early stopping" and consists in stopping the training step when the error on the validation set starts growing. In particular, the process stops whenever the error increases a fixed amount of times N_{fail}.

5.3 Workflow Motivation

The reason why this kind of analysis is done only after fixing the net structure is because the optimal net structure should be rather independent from the training algorithm, while the opposite is not true. Indeed the information of an input about the output is independent from the way you set the weights of the net, even though the way you exploit this information depends on the weights. Also, the number of neurons in the hidden layer should have a stronger impact than the techniques in the training phase to improve generalization, such as regularization and early stopping, even though all of these have an impact on overfitting. In fact, the number of neurons acts macroscopically on the complexity of the functions the model can explain and can both higher and lower the net capacity, while regularization and early stopping are finer tuning procedures and can only lower capacity.
Finally, we point out that will not focus on setting the parameters of the specific algorithm because:

- we expect in general the algorithm type to have a greater impact than the parameters tuning;
- in case the number of data increase or the network structure changes, this parameters should be reset.

5.4 Evaluation Principles

Since the minimum of the performance measure function is determined only by the data, all of the optimization algorithms to find such minimum should lead to setting very similar weights. Hence, we should be looking only for the fastest algorithm. However, different algorithms handle overfitting in different ways. For example, early stopping might not be effective with algorithms converging in few iterations. This is why in [75] networks were trained using the Levenberg-Marquardt algorithm [33] with parameters values to slow

down convergence.
Since net ALL and net RM have different structures from the standard net trained by Gatti [75], we retrained our networks using the Levenberg-Marquardt algorithm with its default parameters, that are reported in Table 5.1.

net	parameters of [75]		default parameters	
	bias	std	bias	std
ALL	0.0569	0.0068	0.0478	0.0077
RM	0.0575	0.0070	0.0486	0.0079

Table 5.1: Bias and standard deviation (std) for net ALL and net RM. Left: parameters proposed in [**tab:gatti param**]; Right: default parameters for the Levenberg-Marquardt algorithm.

In this case we have chosen the default parameters since the bias decreases while the variance seems to remain constant, compared to the settings of 5.1. This is possibly due to the fact that the two networks have a structure less prone to overfitting, thanks to a careful setting of the model capacity, hence diminishing the importance of early stopping in the procedure.
As it can be seen from Table 5.1, the performance can see performance can change significantly using different parameter setting for the same algorithm. Hence, to select the best suited algorithm, we will not only look at the execution time but also at bias and variance.
Moreover, there are different stopping conditions for every algorithm since not always the desired error threshold is reached. Hence, looking at bias and variance is imperative to see whether the training had success or not. Then, for a more detailed analysis it is possible to look at which stopping criterion has been satisfied.

5.5 Performance Analysis

As a first step, different training algorithms were tested in Matlab for both net ALL and net RM. The execution (CPU) time are relative to a run of the whole procedure, meaning the training of 50 networks with random initialization, which is indeed a fair choice in order to have a more stable estimate of the real

average execution time. In tables 5.2 and 5.3 are reported the CPU times for training with different algorithms respectively net ALL and net RM. Since these CPU times depend on the implementation and on the machine they are run on, they should be used only for rough comparisons between each other.

algorithm	time [s]	bias	std
Scaled Conjugate Gradient	16,34	0.0490	0.0067
Resilient Backpropagation	20,21	0.0562	0.0069
Fletcher-Reeves Conjugate Gradient	29,94	0.0474	0.0057
Polak-Ribiére Conjugate Gradient	30,75	0.0502	0.0069
Gradient Descent	35,64	0.1816	0.0259
Levenberg-Marquardt	37,77	0.0478	0.0077
BFGS Quasi-Newton	193,24	0.0492	0.0075

Table 5.2: Execution time for different algorithms used to train net ALL with error goal 10^{-3}.

algorithm	time [s]	bias	std
Scaled Conjugate Gradient	19.08	0.0504	0.0076
Resilient Backpropagation	23.02	0.0535	0.0065
Polak-Ribiére Conjugate Gradient	33.68	0.0499	0.0059
Fletcher-Reeves Conjugate Gradient	34.14	0.0496	0.0061
Gradient Descent	38.15	0.1876	0.0311
Levenberg-Marquardt	69.69	0.0486	0.0079
BFGS Quasi-Newton	447.94	0.0495	0.0061

Table 5.3: Execution time for different algorithms used to train net RM with error goal 10^{-3}.

We notice that the execution times reported in tables 5.2 and 5.3 are higher in general for net RM than for net ALL: this is because net RM has more neurons and hence more parameters to be set. In particular net ALL has 634 parameters, while net RM has 849 parameters. Since the dimensions of the two nets do not differ dramatically, the execution times are quite similar in the two cases.

Given the results reported in tables 5.2 and 5.3 we can draw the following conclusions:

- **Gradient Descent** [34] attains very high values of bias and variance, meaning the training was not successful. Indeed, the maximum number of iterations was reached, signaling that the algorithm is simply too slow. This is reasonable, since gradient descent is a very simple approach.

- **BFGS Quasi-Newton** [31, 19, 64, 9, 80, 8] is the slowest since the approximate Hessian must be stored, and its dimension is $n \times n$, where n is equal to the number of weights and biases in the network. Few iterations with high accuracy, hence computationally expensive, are performed.

- **Levenberg-Marquardt** [33, 47, 34, 8] algorithm is also slow. It behaves similarly to a Newton's method near a minimum. Hence, when the number of parameters increases, the computations required increase geometrically. Indeed, it performs better for net ALL, where the execution time is comparable with the one of the other algorithms, than for net RM.

- **Fletcher-Reeves** [27, 34, 68] and **Polak-Ribiéret** [57, 59, 58] Conjugate Gradient appear to be equivalent, with execution times in the middle range since they require a moderate memory storage.

- **Scaled Conjugate Gradient** [51, 2] and **Resilient Backpropagation** [65] are the fastest, since many iterations with inexact line search are performed, hence less computationally expensive, especially in terms of memory storage.
  Scaled Conjugate Gradient is the most performing because takes more into account information about the gradient magnitude, which Resilient Backpropagation does not account for, and the second derivative in Newton-like way, contrary to the other conjugate gradient algorithms.

The networks have been trained once more by setting the same initial values for the weights and the biases. Their initialization is indeed by default random, in order to avoid converging to the same local minimum each time the network is trained. This has been done in order to offer a more standardized

comparison of the algorithms in terms of time, since the same initial conditions are used. The results of this second set of experiments are reported in Table 5.4.

algorithm	time net ALL [s]	time net RM [s]
Scaled Conjugate Gradient	20.5003	21.6899
Resilient Backpropagation	23.3590	28.619
Fletcher-Reeves Conjugate Gradient	29.0517	33.6469
Polak-Ribiére Conjugate Gradient	29.1224	33.7605
Levenberg-Marquardt	43.7960	68.2432
BFGS Quasi-Newton	218.9319	504.3658

Table 5.4: The execution time for different algorithms used to train net ALL and net RM with non random initialization and error goal 10^{-3}.

As we can see by comparing the results of tables 5.2, 5.3 and 5.4, the execution times seems to be almost constant, suggesting that all of the above considerations are independent from the initial set chosen.
In conclusion, since the number of net parameters is large, algorithms that feature low memory storage and cheap iterations seems to perform better.

5.5.1 Test Set Analysis

Earthquake ground motion is a complex phenomenon and the associated data can be highly variable. Minima of the ANN objective function are determined from those data and hence they are variable as well. During the ANN training we look for these "data driven" minima. However, what we are really looking for, i.e. the true minima, are the minima we would obtain by using data from all the possible realization of the phenomenon. This discrepancy depends on the data variability.
It makes sense to look for the minima of the objective function, since it is the best guess of the true minima we can make. However, getting too close to these minima might result in overfitting. Hence, when the variability is high, it makes sense to sacrifice accuracy in the approximation of the objective function, in favour of other properties such as a solution with a reduced computational cost. This is why Scaled Conjugate Gradient (SCG) performs better than other algorithms, not only in terms of CPU time, but also in

terms of bias and standard deviation.
Fletcher-Reeves and Levenberg-Marquardt appear to perform better than SCG in terms of bias and standard deviation for net ALL and net RM respectively. However, in this case the moderate gain in performance is not comparable with the loss in terms of CPU time, w.r.t. SCG.
All the algorithms in general perform quite well on the test set. However, we notice that even though the error goal has been set to 10^{-3} the computed μ is always above this value. Indeed, the training phase always halts because of early stopping. Possible reasons are:

- not enough data are available;
- a better tuning of the ANN model is needed;
- some data preprocessing is needed.

The above hypothesis do not exclude each other. The simplest approach is probably some finer tuning of the model, which will be analyzed in the following chapters.

Chapter 6

Training, Validation and Test Set

6.1 Minimum Error Goal

From now on we will use Scaled Conjugate Gradient, since, given the results of Chapter 5, it appears to be the best suited training algorithm for our application.
During the training, the bias is always above 10^{-2}, due to early stopping. As a further confirmation we propose Table 6.1, containing bias and standard deviation for different error goals using Scaled Conjugate Gradient as training algorithm.

error goal	net ALL		net RM	
	bias	std	bias	std
10^{-1}	0.1040	0.0130	0.1028	0.0087
10^{-2}	0.0492	0.0071	0.0487	0.0066
10^{-3}	0.0490	0.0067	0.0504	0.0076
10^{-4}	0.0487	0.0077	0.0483	0.0068

Table 6.1: Bias and standard deviation (std) for different error goals, using Scaled Conjugate Gradient as training algorithms for net ALL and net RM.

In Table 6.2 we show the execution times for different error goals using Scaled Conjugate Gradient as training algorithms for net ALL and net RM.

error goal	net ALL time $[s]$	net RM time $[s]$
10^{-1}	9.73	12.47
10^{-2}	16.38	19.12
10^{-3}	17.63	19.17
10^{-4}	17.70	18.61

Table 6.2: Execution time for different error goals, using Scaled Conjugate Gradient as training algorithms for net ALL and net RM.

As we can see, from 10^{-2} and below bias and variance do not appear to have decreasing trend. Moreover, the execution time stops increasing at the same threshold.

6.2 Dataset Sliptting Strategy

The roles of training, validation and test set are:

- **Training set**: it is used to set the model parameters. Increasing its size allows more precised predictions, also by highering model capacity. We set it to be the 70% of the whole dataset.
- **Validation set**: it is supposed to be made of patterns different from those of the training set and thus used to monitor the accuracy of the ANN model during the training procedure. Increasing its size with respect to the training set might lead to a more severe but precised judgment of the model performance, causing early stopping to occur sooner. A too small size would make this performance check highly variable, hence unreliable. We set it to be the 20% of the whole dataset.
- **Test set**: a set, not used during ANN training and validation, but needed to evaluate the network capability of generalization in the presence of new data. Increasing its size leads to more reliable estimates. We set it to be the 10% of the whole dataset.

Since to decrease the bias we cannot increase our dataset, we can look for the best strategy to split our set of data it into training, validation and test sets. In fact, the training procedure depends on their relative dimensions, thus we

can use them to tune our model. One could argue that it would be more important to decide the relative dimensions of the training, validation and test sets when deciding the net structure, since the model capacity depends on the training set size. This is true, but, since by the training set is already 70% of the overall data we do not expect to obtain significant results varying it. On the contrary, the validation set is 20%, thus it is possible to vary it significantly with respect to its current size. This is exactly what we will test in the following.

Before proceeding with some numerical experiments, we made the following heuristic considerations:

1. **Training set**: it should be as big as possible, since precision keeps improving, however only as far as validation and test set are big enough to play their role properly.

2. **Validation set**: since it performs a check analogous to the one of the test set, in principle it should not be smaller than the latter. It should be also big enough to have early stopping working properly. Since early stopping appears already to dominate the learning process, we will attempt lowering it. Our hope is to lower the bias, since the validation process would be less precised. Another option is to lower the size of the validation set below the one of the test set, while increasing N_{fail}, i.e. the number of times that the error on the validation set needs to grow before halting the training in the early stopping procedure.

3. **Test set**: it should be big enough to give reliable estimates of the ANN performance but not more, not to waste resources that could be used in the other sets. It should be small compared to the training set.

In order to tune the relative dimensions of training, validation and test sets, we will proceed in the following order:

1. find the minimum size of the test set;

2. tune the validation set size;

3. use all of the remaining data in the training set.

The reason to follow the above criterion is that we expect the minimum size of the test set to be rather independent from the sizes of the training and validation sets.

6.3 Numerical Results

In Table 6.3 we report the computed bias and standard deviation for net ALL and net RM, when varying the relative sizes of training, validation and test sets.

training	validation	test	net ALL		net RM	
[%]	[%]	[%]	bias	std	bias	std
70	20	10	0.0490	0.0067	0.0504	0.0076
70	15	15	0.0493	0.0049	0.0509	0.0046
60	20	20	0.0502	0.0050	0.0508	0.0047
85	10	5	0.0506	0.0101	0.0507	0.0095

Table 6.3: Bias and standard deviation for net ALL and net RM, when varying the relative sizes of training, validation and test sets, with $N_{fail} = 6$.

The ratios proposed in [75] are 70-20-10 % for the training, validation and test set, respectively.
We can observe a significant decrease of the standard deviation when the size of the test set varies from 10% to 15%, while it appears to remain constant from 15% to 20%. Hence, 15% seems to be the optimal size.
The ratios 85-10-5 % reported in [54], designed for a network with no additional inputs, were also tested. However, this approach did not yield good results.
Since the bias did not decrease when the validation size was reduced to 15%, as mentioned before we are left with the option of lowering the validation set size and increment N_{fail}. The results of such experiments are reported in Table 6.4, where bias and standard deviation for net ALL and net RM, when varying N_{fail} and the relative sizes of training, validation and test sets.

N_{fail}	training	validation	test	net ALL		net RM	
	[%]	[%]	[%]	bias	std	bias	std
6	70	15	15	0.0493	0.0049	0.0509	0.0046
6	75	10	15	0.0503	0.060	0.0518	0.0050
12	75	10	15	0.0484	0.048	0.0489	0.044
18	80	5	15	0.0481	0.051	0.0494	0.047

Table 6.4: Bias and standard deviation for net ALL and net RM, when varying N_{fail} and the relative sizes of training, validation and test sets.

As clear from Table 6.4, decreasing the validation size to 10% and increasing N_{fail} till 12 is the best compromise since the standard deviation is not affected and the bias decreases. The decrease is not large, however at least we expect the above results to be reliable, since the test ratio is now 15%.

Since the number of net parameters is large, algorithms that sacrifice accuracy in favour of less memory storage and cheaper computations perform better. Scaled Conjgated Gradient is the fastest, with great comprimse in terms of performance on the test set.
An optimal ratio of 75-10-15 % for respectively the training, validation and test set was found. A high number of failures on the validation test, in order to halt the training in the early stopping procedure, was found optimal with this kind of ratio. With the current dataset size, it does not look possible improve performance of the ANN model in order to reach a bias on the test set below 10^{-2}.
In Table 6.5 we report bias, standard deviation and execution time of net ALL and RM with respect to net OLD, whose settings are defined in [75].

net	time [s]	bias	std
OLD	178.30	0.0683	0.0101
ALL	16,34	0.0484	0.0048
RM	19.08	0.0489	0.0044

Table 6.5: Comparison in terms of bias, standard deviation and execution time of net ALL and RM with respect to net OLD.

In Table 6.6 we report the main settings of nets ALL, RM and OLD.

net	input	neurons	N_{fail}	dataset ratios [%]	training
OLD	SA	30	6	70-20-10	LM
ALL	SA, R_{epi}, M_w, V_{s30}	25	12	75-10-15	SCG
RM	SA, R_{epi}, M_w	35	12	75-10-15	SCG

Table 6.6: The main settings of nets ALL, RM and OLD. LM is the shorthand for Levenberg-Marquard, SCG for Scaled Conjugate Gradient.

As visible from Table 6.5, with respect to net OLD the new nets are approximately 10 times more fast to be trained, 1.4 times more accurate, i.e. in terms of bias, and 2 times more precised, i.e. in terms of standard deviation.

Chapter 7

Predictions Using Data From Numerical Simulations

7.1 Introduction

So far we obtained an estimate of net performance using a test set. However, what we are interested in is using the proposed ANN-based approach starting from the data obtained from numerical simulations obtained with the code SPEED [48] (SPectral Elements in Elastodynamics with Discontinuous Galerkin - http://speed.mox.polimi.it). SPEED is an open-source code for the simulation of large-scale seismic events in three-dimensional complex media, using discontinuous Galerkin Spectral Element methods for the approximate solution of the differential problem.
The networks input data we have at our disposal are not the real ones but those obtained from the numerical simulations. To test the proposed approach for the generation of broad band ground motions and to verify the accuracy of results comparing them with the data registered during recent earthquakes, we consider in this chapter three real cases of interest. Specifically we will consider three earthquakes in Italy: L'Aquila 2009 ($M_w = 6.3$), the Po Plain 2012 ($M_w = 6.0$) and the Norcia 2016 ($M_w = 6.5$) earthquakes. These earthquakes are meaningful for validation purposes, because of the availability of a significant number of near-source strong motion records, some of which obtained at very short inter-station distances, as well as of the good knowledge on the complex geologic setting, which enabled the construction of a robust 3D numerical model.

7.2 Quantification of the Error Induced by Synthetic Ground-Motion Scenarios

What we are essentially interested in is to quantify the effect on our predictions of the error introduced by synthetic ground-motion data. In Table 7.1 we report the mean squared error (MSE) for synthetic ground-motion data generated with the code SPEED, calculated only for the SAs used as input in the three study cases, and the number of stations.

case	SPEED MSE	number of stations
Po Plain	0.1961	11
L'Aquila	0.0813	6
Norcia	0.0628	18

Table 7.1: The mean squared error (MSE) for synthetic ground-motion data generated with the code SPEED, calculated only for the SAs used as input in the three study cases, and the respective number of stations.

As we can see from the results reported in Table 7.1, in all cases the synthetic ground-motion MSE is higher than the MSE on the test set of our nets that is reported in Table 6.5. While the synthetic ground-motion MSE is comparable with the bias for L'Aquila and Norcia, it is much higher for the Po Plain test case. Norcia test case appears to be the most reliable study case, not only because it has the lowest MSE, but also because it has a much greater number of stations and hence records.
Since neural networks are black box models, it is hard to predict the sensitivity of the network, i.e. the error on the output knowing the error on the input. Hence, we only reasonably expect to have higher errors in the Po Plain test case. This is interesting for checking different levels of robustness of our networks.
It is also important to consider that the error introduced by the synthetic ground-motion is always due to an underestimate at short periods. This might lead also to consistent underestimates of the net predictions. For future works, this might suggest strategies to tackle the problem of coupling

numerical simulations and neural networks in a more natural way: for instance, it could be possible to penalize more underestimates in the training phase of the networks, through a customized objective function.

While other earthquakes in the Po Plain and L'Aquila are still present in the dataset this does not happen for the Norcia test case. The reasons why we did not eliminate from the dataset all of the other records concerning the Po Plain and L'Aquila cases are:

- these data represent approximately 11% of the data, hence the performance decrease would be too large, especially in our case where more data would be needed;
- we can observe better performance when data about previous earthquakes are available for a location.

Since no data at all about the earthquakes in Norcia have been used in the training phase, we will consider it as the most representative of the generalization capabilities of the broad band procedure. Moreover, this is also the most recent case among the three.

7.3 Best Network Approach

The approach adopted by Paolucci et al. [54] consists in selecting, among the 50 networks trained, the net attaining the lowest bias on the remaining 5% of the dataset. Indeed, that 5% has not been used in the training of the nets, hence it can be used to evaluate these nets with a common meter. However, as we have seen when setting the test set ratio to 15%, selecting a net on the basis of 5% of the whole dataset might lead to unstable results. Hence, we decided to select the net attaining the lowest bias on the test set, which has indeed a reasonable dimension, and to use all of the available data in the training phase. In other words we select the network with the best generalization properties with respect to the test set, as it reasonably can be considered to be representative of the phenomena.
We first apply our networks to the three study cases using the real data, as a further confirm of the test set performance and in order to have a closer idea of what to expect when using the synthetic ground-motion data.
In Table 7.2 performance on different study cases when using as input the

real data, where the net attaining the lowest bias on the test set over 50 networks has been used.

net	bias	L'Aquila MSE	Po Plain MSE	Norcia MSE
ALL	0.0484	0.0532	0.0334	0.0848
RM	0.0489	0.0358	0.0345	0.0353
OLD	0.0683	0.0377	0.0359	0.0410

Table 7.2: The net attaining the lowest bias, reported in the table, on the test set over 50 networks has been selected. In the table, performance on different study cases when using as input the real data.

The analogous results based on employing synthetic ground-motion data are repeated in Table 7.3.
As we can see from the results in Table 7.2, net RM and net OLD perform well, with MSE below the one on the test set. Net RM appears to perform slightly better in all cases. Net ALL instead performed better for Po Plain and worse for L'Aquila and Norcia. This could be due to the fact that net ALL makes use of more different inputs, hence could be less robust.
Notice that the MSE is higher for all nets in the Norcia test case, probably because no data about that site were used in the training. The error difference is evident for net ALL, which might lead us to think of an overfitting of the data.
We now have a look at what happens when using synthetic ground-motion data obtained by SPEED as input.

net	L'Aquila MSE	Po Plain MSE	Norcia MSE
ALL	0.1367	0.2147	0.1859
RM	0.0695	0.2529	0.0992
OLD	0.0864	0.2736	0.1120

Table 7.3: The net attaining the lowest bias on the test set over 50 networks has been selected. In the table, performance on different study cases when using as input the synthetic ground-motion data.

These results are repoterd in Table 7.3. As expected the error grows considerably in all case when using the data obtained by SPEED, in particular

for Po Plain since the SPEED MSE is greater. The error grows in a super additive way, with respect to the SPEED MSE and the MSE using the real data. However, it remains difficult to estimate a trend.
This time the better performance of net RM are amplified by the error introduced by SPEED. In conclusion, net RM appears to be not only more performing than the other nets, but also more robust when considering consistently biased input data.

Chapter 8

Bootstrap Aggregating Algorithms

Selecting the network attaining the lowest bias on the test set ensures that it generalizes well with respect to that set. The test set is supposed to be quite representative of the phenomena. Hence, if the difference of bias between nets is considerable, it is reasonable to select the one that attains the lowest bias. When they are comparable however, there is no reason why we should prefer the one with lowest bias, which might be actually overfitting the test set. Moreover, since in any case many runs of training are needed to avoid local minima, selecting only one network might not be the most effective choice.
A common strategy that exploits more networks is called "bagging". It stands for "bootstrap aggregating" and consists in creating multiple nets through bootstrapping, i.e. performing a random resampling of the data, and then averaging their outputs. It improves precision and stability, since averaging reduces the variability of the output, and allows to filter out the presence of gaussian noise in the data. Since in our case the synthetic ground-motion data obtained based one employing SPEED underestimate the real data, we are interested in observing the effect it has on bagging and how relevant it is with respect to gaussian noise.

8.1 Net Selection

Selecting networks with low a test set error allows, in principle, to pick those nets whose parameter values are as close as possible to the global ones. At the same time averaging more nets further improves stability. We are interested in observing the impact of the above effects on our nets, and analyze whether there is an optimal trade off between the two. To check whether actually nets with a low bias on the test set perform better than the ones with high bias, we proceed as follows. We first measure the behaviour of each net in all of the study cases. The vector of 50 nets was sorted according to the standardized error on the test set. Given a vector $x \in \mathbb{R}^n$ its standardized vector $z \in \mathbb{R}^n$ is biven by

$$z = \frac{x - \mathbf{1}\overline{x}}{s}, \tag{8.1}$$

where

$$\overline{x} = \frac{1}{n}\sum_{i=1}^{n} x_i, \tag{8.2}$$

$$s = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(\overline{x} - x_i)^2}, \tag{8.3}$$

and $\mathbf{1} = (1, 1, ..., 1) \in \mathbb{R}^n$.
To highlight possible trends, we consider also the moving average m_i with $k = 5$ elements of the standardized vector:

$$m_i = \frac{1}{|\mathcal{M}_i|}\sum_{j \in \mathcal{M}_i} z_j, \quad i = 1...50, \tag{8.4}$$

where z_j is the j-th entry of the vector z defined in (8.1) and $\mathcal{M}_i$ is a set defined as

$$\mathcal{M}_i = \{i - \frac{k-1}{2}, ..., i + \frac{k-1}{2}\} \cap \{1...50\}, \quad i = 1...50. \tag{8.5}$$

In Figure 8.1 and 8.2 we report the MSE and 5 elements moving average in L'Aquila, Po Plain and Norcia test cases for net RM, using real and synthetic input data, respectively. When using real input data (figure Figure 8.1) in all of the three cases the MSE appears to be lower and less variable for nets with a standardized error below -1. The same appears to

be valid when using instead synthetic input data (Figure 8.2), even though the trend is not so clear anymore because of the additional error introduced that produces higher spikes.
We now investigate the MSE of the average output of the nets. We expect that the MSE computed staring from the average output to be lower than the average MSE of the nets since the MSE is a convex function.
We will gradually average from 1 to 50 networks staring from the ones with a low bias on the test set, since as observed they seem to more robust when performing on new datasets. In Figure 8.3 we plot the cumulative MSE as a function of the error of the standardized error z defined in (8.1), i.e. the MSE of the mean output of nets with standardized test set error smaller or equal to z, both when using real and synthetic input data.
As expected, when using real input data, the minimum is clearly around -1 in all cases and the MSE values are lower than before. When using synthetic input data the error grows and the trend is more hardly observable. The minimum error appers to be reached around -1, more evidently for Po Plain and L'Aquila test case, while the error looks rather constant for Norcia test case.

We repeated the same analysis using net ALL and OLD. These results are reported in figures 8.4-8.6 and 8.7-8.9, respectively. Concerning net ALL we have very similar results. The only difference lies in the fact that the minimum appears to be around -1.5 instead of -1.
Concerning net OLD instead, there appears to be minimum around -1.5 or -1. However, in this case it looks reasonable to average all of the 50 nets, with a further gain in stability.

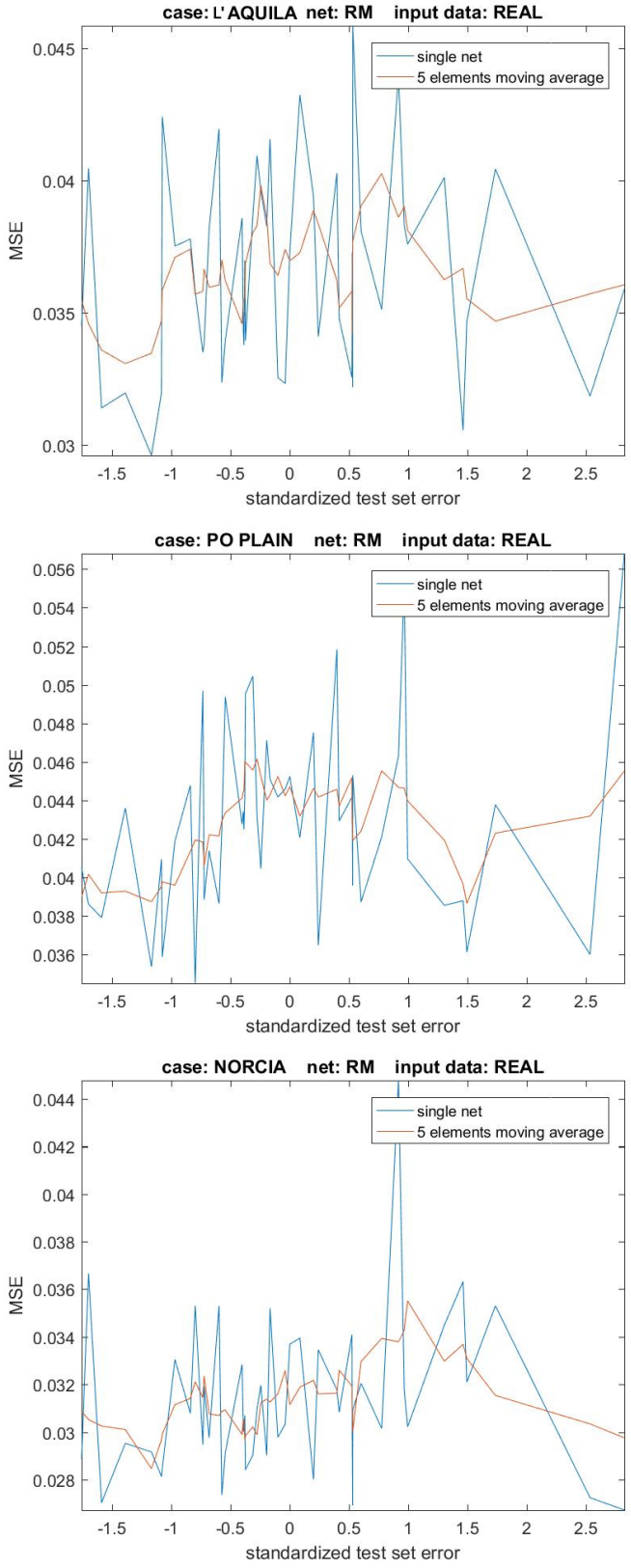

Figure 8.1: MSE and 5 elements moving average in L'Aquila, Po Plain and Norcia test cases for net RM using real input data.

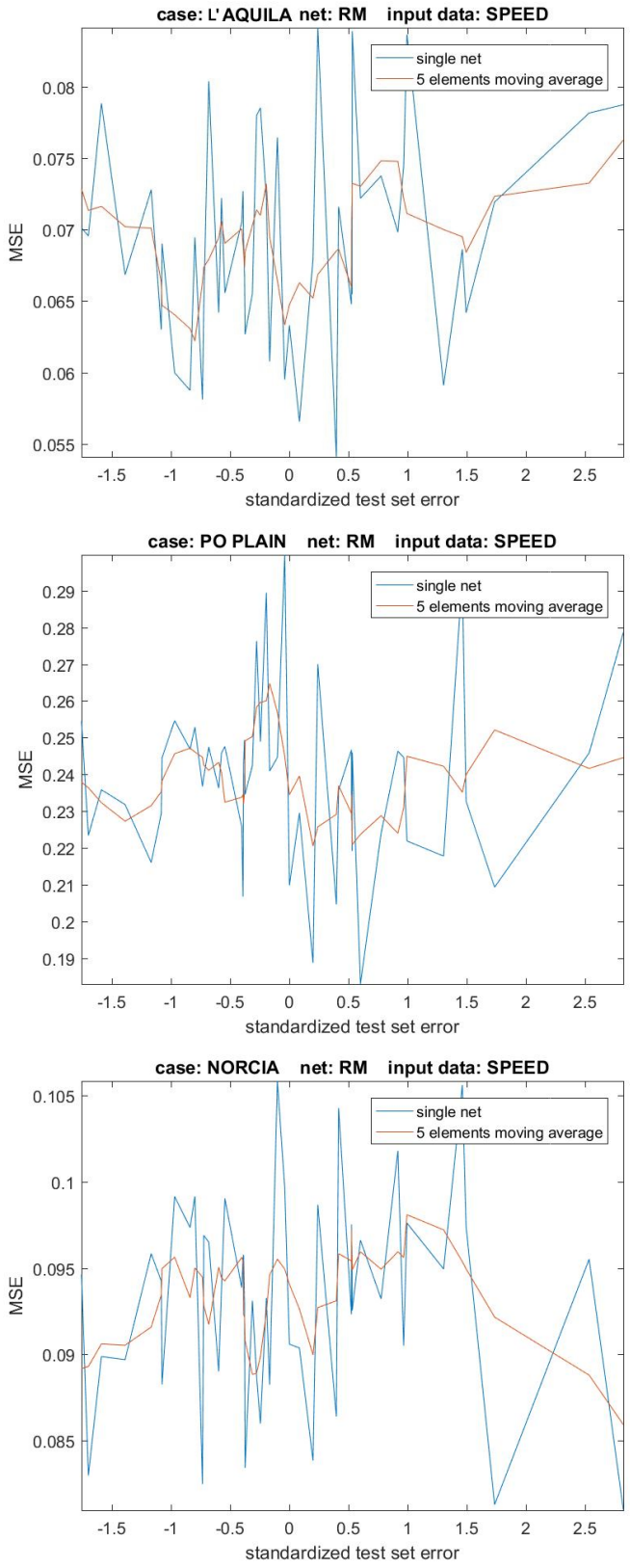

Figure 8.2: MSE and 5 elements moving average in L'Aquila, Po Plain and Norcia test cases for net RM using synthetic input data.

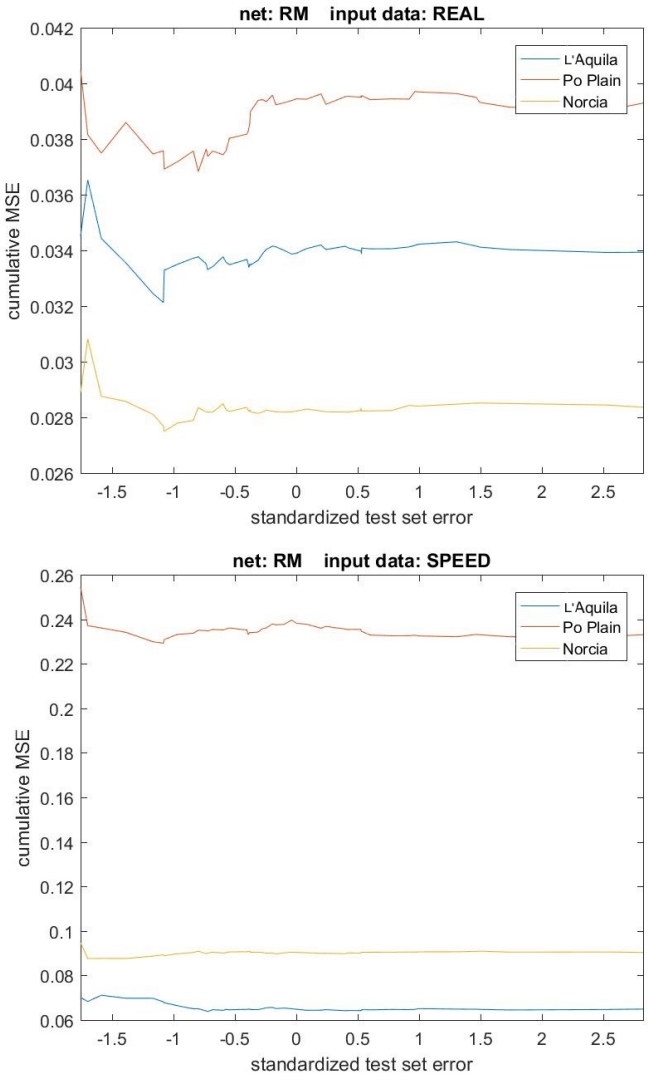

Figure 8.3: Cumulative $MSE(z)$, i.e. the MSE of the mean output of nets with standardized test set error smaller or equal to z, for net RM using real and synthetic data.

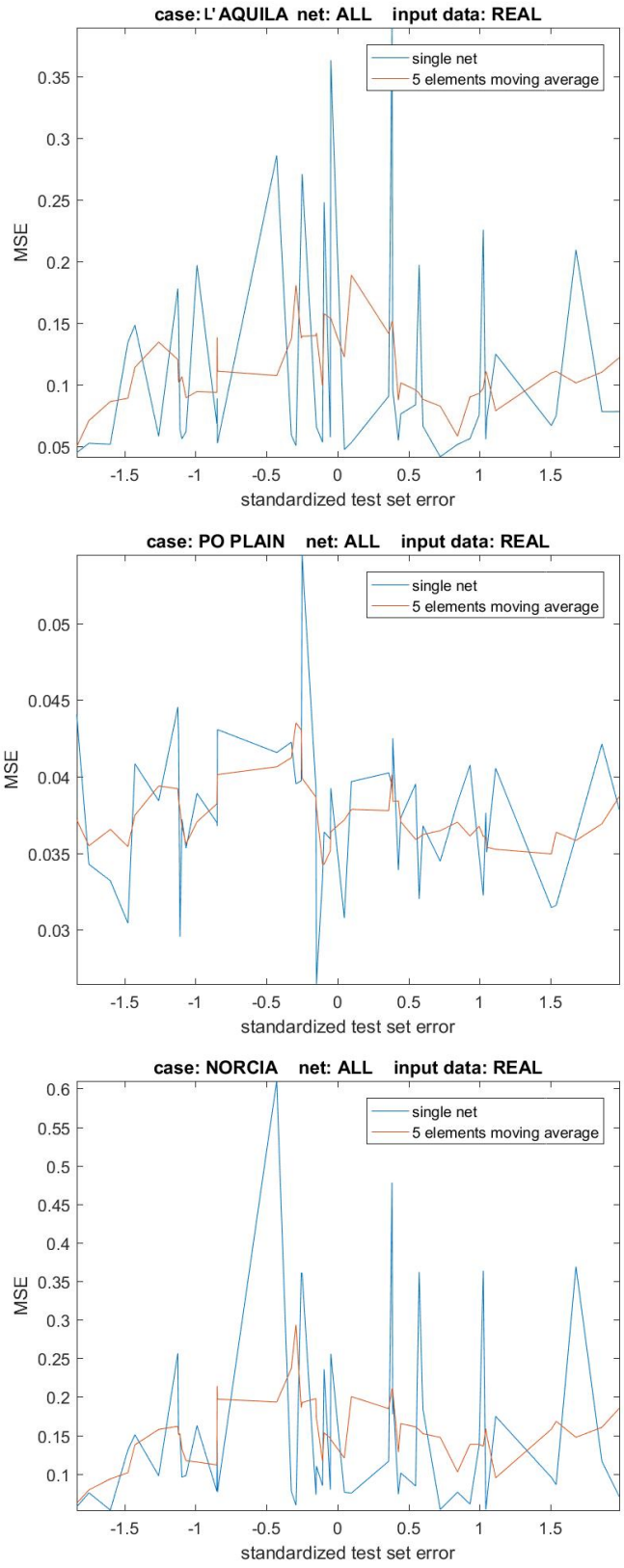

Figure 8.4: MSE and 5 elements moving average in L'Aquila, Po Plain and Norcia test cases for net ALL using real input data.

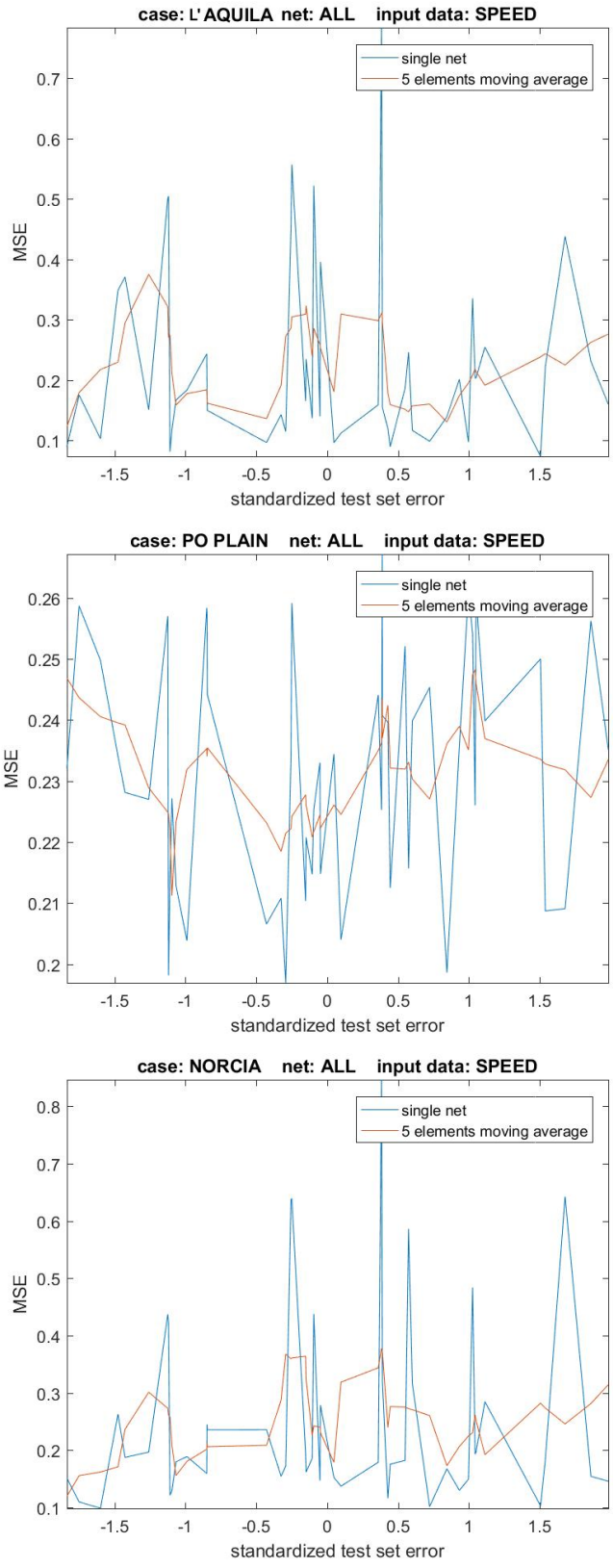

Figure 8.5: MSE and 5 elements moving average in L'Aquila, Po Plain and Norcia test cases for net ALL using synthetic input data.

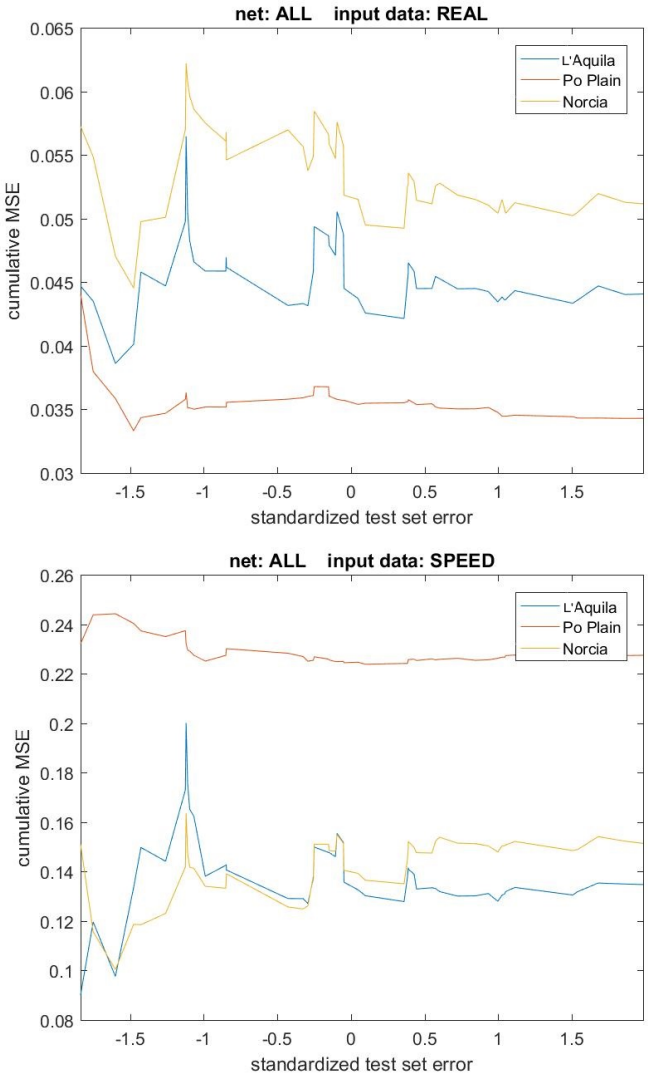

Figure 8.6: Cumulative $MSE(z)$, i.e. the MSE of the mean output of nets with standardized test set error smaller or equal to z, for net ALL using real and synthetic data.

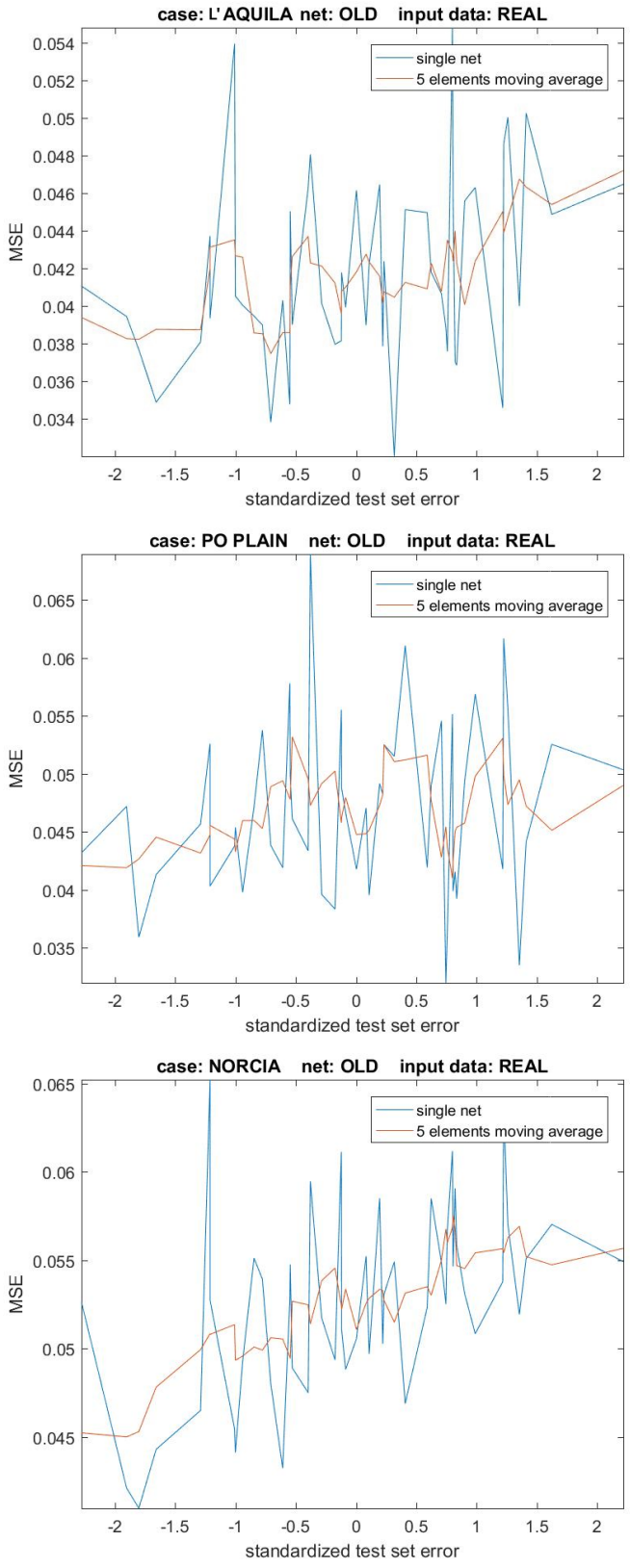

Figure 8.7: MSE and 5 elements moving average in L'Aquila, Po Plain and Norcia test cases for net OLD using real input data.

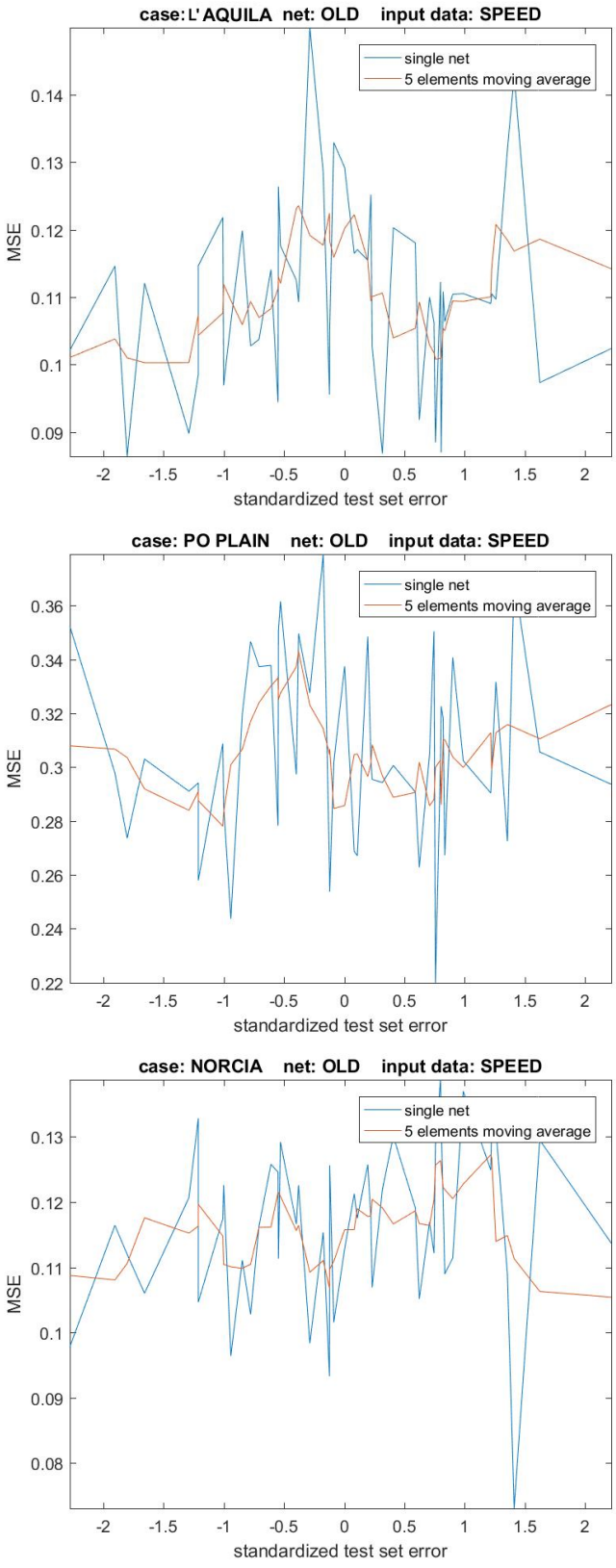

Figure 8.8: MSE and 5 elements moving average in L'Aquila, Po Plain and Norcia test cases for net OLD using synthetic input data.

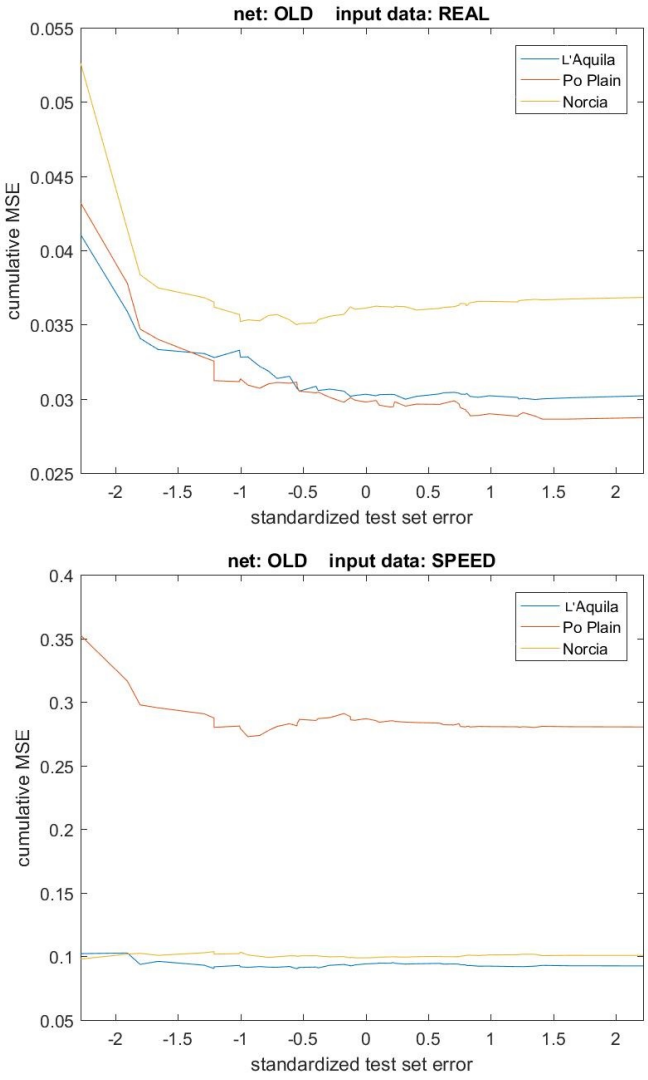

Figure 8.9: Cumulative $MSE(z)$, i.e. the MSE of the mean output of nets with standardized test set error smaller or equal to z, for net OLD using real and synthetic data.

8.2 Conclusions

In the following we report a summary of the effect of bagging on our nets for the various test cases. In Tables 8.1 and 8.2 we report the effect of bagging on nets of type RM when using real and synthetic input data, respectively. The maximum standardized test set error is equal to -1, and 7 nets out of 50 are below this threshold.

MSE L'Aquila	MSE Po Plain	MSE Norcia	bagging nets
0.0345	0.0405	0.0289	1
0.0333	0.0369	0.0275	7
0.0339	0.0393	0.0284	50

Table 8.1: Bagging effect on nets of type RM when using real input data. Maximum standardized test set error equal to -1, selecting 7 nets out of 50.

MSE L'Aquila	MSE Po Plain	MSE Norcia	bagging nets
0.0701	0.2545	0.0946	1
0.0677	0.2309	0.0888	7
0.0649	0.2331	0.0903	50

Table 8.2: Bagging effect on nets of type RM when using synthetic input data. Maximum standardized test set error equal to -1, selecting 7 nets out of 50.

We have repeated the same analysis for the nets of type ALL, whose results are reported in Tables 8.3 and 8.4, and for nets of type OLD, whose results are reported in Tables 8.5 and 8.6.
Concerning net RM we can observe in general an improvement by using bagging. The gain is not particularly large when using real input data, however constant across the three study cases when using the 7 selected nets. The fact that few nets in this case would be used for bagging may give worse stability properties with respect to using 50 nets. However, this is just an issue in terms of computational costs, since experimentally around 14% of the nets of type RM trained have a test set error smaller than -1.

The gain is more considerable when using synthetic input data. Using selected nets appears to perform better, especially in the Norcia test case for the reasons already discussed before.

MSE L'Aquila	MSE Po Plain	MSE Norcia	bagging nets
0.0447	0.0440	0.0573	1
0.0386	0.0358	0.0470	3
0.0441	0.0343	0.0512	50

Table 8.3: Bagging effect on nets of type ALL when using real input data. Maximum standardized test set error equal to -1.5, selecting 3 nets out of 50.

MSE L'Aquila	MSE Po Plain	MSE Norcia	bagging nets
0.0901	0.2321	0.1510	1
0.0976	0.2442	0.1004	3
0.1347	0.2274	0.1514	50

Table 8.4: Bagging effect on nets of type ALL when using synthetic input data. Maximum standardized test set error equal to -1.5, selecting 3 nets out of 50.

Concerning net ALL, we have a more evident improvement form the use of bagging, and in particular from the selection of nets. This not so valid anymore when using synthetic input data. However, the performance seem to be much better for the Norcia test case, which we consider more meaningful. Another issue is the fact that this time the net selection is very small, only 3 nets. Hence, in this case it might be really helpful to train more nets in order to enlarge the net selection.

MSE L'Aquila	MSE Po Plain	MSE Norcia	bagging nets
0.0411	0.0432	0.0526	1
0.0328	0.0313	0.0352	9
0.0302	0.0287	0.0368	50

Table 8.5: Bagging effect on nets of type OLD when using real input data. There is no maximum standardized test set error.

MSE L'Aquila	MSE Po Plain	MSE Norcia	bagging nets
0.1021	0.3524	0.0978	1
0.0918	0.2790	0.1033	9
0.0925	0.2803	0.1007	50

Table 8.6: Bagging effect on nets of type RM when using synthetic input data. There is no maximum standardized test set error.

Concerning net OLD we have in a general a considerable gain in using bagging, which is more evident when using real input data. As already observed there is no gain in making a net selection, which in this case was tested with a maximum standardized test set error of -1, and 9 nets out of 50 are below this threshold.

In Tables 8.7 and 8.8 we report a comparison of nets ALL, RM and OLD in terms MSE using real and synthetic input data respectively, and using the optimal number of nets in the bagging procedure for each net type. In conclusion using bagging seems to improve performance and, when possible, a further smaller gain can be obtained by using a selected nets.

net	MSE L'Aquila	MSE Po Plain	MSE Norcia	bagging nets	max std error
ALL	0.0386	0.0358	0.0470	3	-1.5
RM	0.0333	0.0369	0.0275	7	-1
OLD	0.0302	0.0287	0.0368	50	$+\infty$

Table 8.7: Nets comparison using bagging with real input data.

When using real data, net ALL appears to generalize worse than net RM and net OLD, and net RM and net ALL appear to be comparable. Even though net OLD performs better in the L'Aquila and Po Plain case, we prefer net RM since it performs considerably better in the Norcia case.

net	MSE L'Aquila	MSE Po Plain	MSE Norcia	bagging nets	max std error
ALL	0.0976	0.2442	0.1004	3	-1.5
RM	0.0677	0.2309	0.0888	7	-1
OLD	0.0925	0.2803	0.1007	50	$+\infty$

Table 8.8: Nets comparison using bagging with synthetic input data.

When using synthetic input data, net ALL seems to perform slightly better than net OLD, meaning the latter is less robust with respect to the error due to synthetic data.
Net RM seems to be the most robust and it performs better in all cases. Hence, we can conclude that using net RM would be the best choice to generate broadband earthquake ground motions scenarios.

Chapter 9

Conclusions and Further Developments

9.1 Results

In this work we proposed and analyzed a family of extensions of the ANN model proposed by [54], that is able to predict SA in the high frequency range. The ultimate goal is to use this model to correct PBNS in the high frequency range, since their accuracy is usually limited up to 1–1.5 Hz. This would allow to obtain broad band ground motions, that is of primary importance in earthquake engineering.

Having to tune many ingredients of the model, in order to keep constrained the exponential growth of combinations to be tested a downscale approach was followed: elements with a greater impact on the others were tested first, such as the hyperparameters for instance, although it should be kept in mind that often all of the elements influence each other. Moreover, we did not focus on a fine tuning of all the parameters. This because in case of future developments and enlarging of the dataset, only the macroscopic settings would still remain effective.
At first, the net structure was chosen, by setting the hyperparameters:

1. epicentral distance, magnitude of the earthquake were found meaningful for predictions, with the possibility of using also velocity of the shear waves in the first 30 meters;

2. training different networks for different soil types was found ineffective,

due to a a too large reduction of the dataset;

3. the optimal model capacity was found by adjusting the number of neurons in the hidden layer.

As a second step, different training algorithms were then tested. Since the number of net parameters is large, Scaled Conjugate Gradient was found to be the fastest due to its compromise between accuracy, low memory storage and inexpensive iterations. This algorithm turned out to be 10 times faster than the Levenberg-Marquardt algorithm, employed in [75].
In order to further improve performance, it would be need to enlarge the dataset. However, being this not possible in the immediate, a tuning of the optimal dataset splitting into training, validation and test set was done. The validation set was reduced, slightly increasing the risk of overfitting, in favour of a larger training set, to push more on performance, and a larger test set, to asses better the network generalization capabilities. Test set performance are a key information when having to select which networks to use for predictions. When using real input data, the threshold 10^{-2} appears to be a lower bound for the MSE.
L'Aquila, Po Plain and Norcia test cases were used for validation purposes, because of the availability of a robust 3D numerical model.
Nets were shown to perform worse, due the error introduced on the data by the numerical simulations. Net RM, using epicentral distance and earthquake magnitude, was shown to be the most robust.
Finally bagging was considered, further improving performance and stability. Using the test set performance as a selection criterion, in order to identify nets to average was found helpful in some of the analyzed cases.

In this work we have shown a possible way to improve performance and efficiency of the training step, with respect to the previous approach. However, the main contribution of this work is to shed a light on some key aspects that could be reused for further developments, such as inputs significance, dataset dimensions, effects of net dimensions on the training algorithm, robustness when using data from numerical simulations.

Recent discoveries about neural networks could shed a light on how to obtain broad-band numerical methods by coupling in a different and more natural way PBNS with NNs algorithms.

9.2 Further Developments

In order to produce reliable broad-band ground motion simulations, it is important to be able to solve the elastodynamics equation and hence to deal with its computational burden. However even disregarding this issue, due to the complexity of the underlying phenomena, limitations are present because of the lack of detailed knowledge of the site specific case. Thanks to the recent discoveries, CNNs could potentially deal with both of these problems. By using them to solve PDEs a lower computational cost might be attained, and the same structure has shown to be able to perform well also when dealing with uncertainty. In this sense there could be a natural hybridization: instead of gluing NNs simulations and PBSs, it could be possible to use only one model, for example simply by considering a CNN loss function that takes into account simultaneously both the data and the wave equation.
Further developments of the present work include:

1. Explore if a further step in the hybridization of the procedure is possible. This would be realized by considering a CNN to solve the elastodynamics equation that takes into account in its loss function also the site specific the data, to better deal with the lack of detailed knowledge.The aim is to obtain a simpler model and a more realistic wave field.

2. Further explore the parallelism between CNN and MG methods, as its importance is relevant not only applied to seismology, with a focus in solving the elastodynamics equation. This would allow to possibly build ad hoc procedures to solve the elastodynamics equation, such as considering a suitable net structure for the application and specific algorithms to fasten the training phase.

3. To use a CNN to solve the elastodynamic equation in real cases, that could be benchmarks like L'Aquila, Po Plain and Norcia sites. This in order to test the performance of the CNN, and observe possible advantages and disadvantages compared to other approaches, such as discontinuous Galerkin and spectral finite element methods [48]. Moreover, it would be interesting to see if it is possible to have reliable simulations also at the high frequency range.

This topic could bring important insights in both the fields of Artificial Intelligence and Computational Sciences, especially considering the recent

growing interest. The impact could be both from a theoretical point of view, in the understanding of the current algorithms in order to build better ones, and a practical one, which might be particularly well suited for Engineering applications in the field of Seismology and Earthquake Engineering.

Bibliography

[1] "3D Physics-Based Numerical Simulations: Advantages and Current Limitations of a New Frontier to Earthquake Ground Motion Prediction. The Istanbul Case Study". en. In: Geotechnical, Geological and Earthquake Engineering.

[2] Neculai Andrei. "Scaled conjugate gradient algorithms for unconstrained optimization". In: *Computational Optimization and Applications* 38.3 (2007), pp. 401–416.

[3] Paola F Antonietti et al. "Numerical modeling of seismic waves by discontinuous Spectral Element methods". In: *ESAIM: Proceedings and Surveys* 61 (2018), pp. 1–37.

[4] Paola F. Antonietti et al. "Review of Discontinuous Galerkin Finite Element Methods for Partial Differential Equations on Complicated Domains". G.R. Barrenechea et al. (eds.), Building Bridges: Connections and Challenges in Modern Approaches to Numerical Partial Differential Equations, *Lecture Notes in Computational Science and Engineering* 114:279–307. 2016.

[5] Hesheng Bao et al. "Large-scale simulation of elastic wave propagation in heterogeneous media on parallel computers". In: *Computer methods in applied mechanics and engineering* 152.1-2 (1998), pp. 85–102.

[6] Christopher M Bishop and CM Roach. "Fast curve fitting using neural networks". In: *Review of scientific instruments* 63.10 (1992), pp. 4450–4456.

[7] Christopher M Bishop et al. *Neural networks for pattern recognition*. Oxford university press, 1995.

[8] Åke Björck. *Numerical methods for least squares problems*. SIAM, 1996.

[9] Paul T Boggs and Jon W Tolle. "Convergence properties of a class of rank-two updates". In: *SIAM Journal on Optimization* 4.2 (1994), pp. 262–287.

[10] Brendon A Bradley et al. "Guidance on the utilization of earthquake-induced ground motion simulations in engineering practice". In: *Earthquake Spectra* 33.3 (2017), pp. 809–835.

[11] Andrea Cangiani, Emmanuil H. Georgoulis, and Paul Houston. "*hp*-version discontinuous Galerkin methods on polygonal and polyhedral meshes". In: *Math. Models Methods Appl. Sci.* 24.10 (2014), pp. 2009–2041.

[12] Andrea Cangiani et al. "*hp*-version discontinuous Galerkin methods for advection-diffusion-reaction problems on polytopic meshes". In: *ESAIM Math. Model. Numer. Anal.* 50.3 (2016), pp. 699–725.

[13] Emanuele Casarotti et al. "CUBIT and seismic wave propagation based upon the spectral-element method: An advanced unstructured mesher for complex 3D geological media". In: *Proceedings of the 16th International Meshing Roundtable*. Springer. 2008, pp. 579–597.

[14] Emmanuel Chaljub et al. "Quantitative comparison of four numerical predictions of 3D ground motion in the Grenoble valley, France". In: *Bulletin of the Seismological Society of America* 100.4 (2010), pp. 1427–1455.

[15] Emmanuel Chaljub et al. "Spectral-element analysis in seismology". In: *Advances in Geophysics* 48 (2007), pp. 365–419.

[16] Yifeng Cui et al. "Scalable earthquake simulation on petascale supercomputers". In: *SC'10: Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE. 2010, pp. 1–20.

[17] George Cybenko. "Approximation by superpositions of a sigmoidal function". In: *Mathematics of control, signals and systems* 2.4 (1989), pp. 303–314.

[18] Steven M Day et al. "Model for basin effects on long-period response spectra in southern California". In: *Earthquake Spectra* 24.1 (2008), pp. 257–277.

[19] JE Dennis Jr and RB Schnabel. "Numerical methods for unconstrained optimization and nonlinear equations prentice-hall series in computation". In: *Mathematics, Englewood Cliffs, NJ* (1983).

[20] Daniele Antonio Di Pietro and Alexandre Ern. *Mathematical Aspects of Discontinuous Galerkin Methods*. Vol. 69. Springer Science & Business Media, 2011.

[21] M. Dryja. "On Discontinuous Galerkin Methods For Elliptic Problems with Discontinuous Coefficients". In: *Comput. Methods in Appl. Math.* 3.1 (2003), 76–85.

[22] Maksymilian Dryja, Juan Galvis, and Marcus Sarkis. "BDDC methods for discontinuous Galerkin discretization of elliptic problems". In: *J. Complexity* 23.4-6 (2007), pp. 715–739.

[23] Michael Dumbser and Martin Käser. "An arbitrary high-order discontinuous Galerkin method for elastic waves on unstructured meshes—II. The three-dimensional isotropic case". In: *Geophysical Journal International* 167.1 (2006), pp. 319–336.

[24] E Faccioli et al. "2D and 3D elastic wave propagation by a pseudo-spectral domain decomposition method". In: *Journal of Seismology* 1.3 (1997), pp. 237–251.

[25] Ezio Faccioli and Roberto Paolucci. "Elementi di sismologia applicata all'ingegneria". In: (2005).

[26] Ezio Faccioli et al. "Spectral-domain decomposition methods for the solution of acoustic and elastic wave equations". In: *Geophysics* 61.4 (1996), pp. 1160–1174.

[27] Reeves Fletcher and Colin M Reeves. "Function minimization by conjugate gradients". In: *The computer journal* 7.2 (1964), pp. 149–154.

[28] Filippo Gatti. "Analyse physics-based de scénarios sismiques «de la faille au site» : prédiction de mouvement sismique fort pour l'étude de vulnérabilité sismique de structures critiques." fr. PhD thesis. Université Paris-Saclay, Sept. 2017.

[29] L. De Carvalho Paludo A. Svay F. Lopez-Caballero R. Cottereau and D. Clouteau Gatti F. "Investigation of the earthquake ground motion coherence in heterogeneous non-linear soil deposits". In: 2017.

[30] Emmanuil H. Georgoulis, Edward Hall, and Paul Houston. "Discontinuous Galerkin methods for advection-diffusion-reaction problems on anisotropically refined meshes". In: *SIAM Journal on Scientific Computing* 30.1 (2007/08), pp. 246–271.

[31] Philip E Gill, Walter Murray, and Margaret H Wright. "Practical optimization". In: (1981).

[32] Robert W Graves. "Simulating seismic wave propagation in 3D elastic media using staggered-grid finite differences". In: *Bulletin of the Seismological Society of America* 86.4 (1996), pp. 1091–1106.

[33] Martin T Hagan and Mohammad B Menhaj. "Training feedforward networks with the Marquardt algorithm". In: *IEEE transactions on Neural Networks* 5.6 (1994), pp. 989–993.

[34] Martin T Hagan et al. *Neural network design*. Vol. 20. Pws Pub. Boston, 1996.

[35] Juncai He and Jinchao Xu. "MgNet: A unified framework of multigrid and convolutional neural network". In: *Science China Mathematics* (2019), pp. 1–24.

[36] Juncai He et al. "Relu deep neural networks and linear finite elements". In: *arXiv preprint arXiv:1807.03973* (2018).

[37] Jan S Hesthaven and Tim Warburton. *Nodal Discontinuous Galerkin Methods: Algorithms, Analysis, and Applications*. Springer Science & Business Media, 2007.

[38] Yigit Isbiliroglu, Ricardo Taborda, and Jacobo Bielak. "Coupled soil-structure interaction effects of building clusters during earthquakes". In: *Earthquake Spectra* 31.1 (2015), pp. 463–500.

[39] Alexandr Katrutsa, Talgat Daulbaev, and Ivan Oseledets. "Deep multigrid: learning prolongation and restriction matrices". In: *arXiv preprint arXiv:1711.03825* (2017).

[40] Dimitri Komatitsch and Jeroen Tromp. "Introduction to the spectral element method for three-dimensional seismic wave propagation". In: *Geophysical Journal International* 139.3 (1999), pp. 806–822.

[41] Dimitri Komatitsch and Jeroen Tromp. "Spectral-element simulations of global seismic wave propagation-I. Validation". In: *Geophysical Journal International* 149.2 (2002), pp. 390–412.

[42] Dimitri Komatitsch and Jeroen Tromp. "Spectral-element simulations of global seismic wave propagation-II. Three-dimensional models, oceans, rotation and self-gravitation". In: *Geophysical Journal International* 150.1 (2002), pp. 303–318.

[43] Dimitri Komatitsch and Jeroen Tromp. "Spectral-element simulations of global seismic wave propagation—I. Validation". In: *Geophysical Journal International* 149.2 (2002), pp. 390–412.

[44] Dimitri Komatitsch and Jeroen Tromp. "Spectral-element simulations of global seismic wave propagation—II. Three-dimensional models, oceans, rotation and self-gravitation". In: *Geophysical Journal International* 150.1 (2002), pp. 303–318.

[45] Y LeCun, Y Bengio, and G Hinton. "Deep learning. nature 521". In: (2015).

[46] P Lubrano-Lavadera et al. "Seismic modelling: 4D capabilities for CO2 injection". In: *Energy Procedia* 114 (2017), pp. 3432–3444.

[47] Donald W Marquardt. "An algorithm for least-squares estimation of nonlinear parameters". In: *Journal of the society for Industrial and Applied Mathematics* 11.2 (1963), pp. 431–441.

[48] Ilario Mazzieri et al. "SPEED: SPectral Elements in Elastodynamics with Discontinuous Galerkin: A non-conforming approach for 3D multi-scale problems". In: *International Journal for Numerical Methods in Engineering* 95.12 (2013), pp. 991–1010.

[49] E Diego Mercerat and Nathalie Glinsky. "A nodal high-order discontinuous Galerkin method for elastic wave propagation in arbitrary heterogeneous media". In: *Geophysical Journal International* 201.2 (2015), pp. 1101–1118.

[50] Thomas V Mikosch, Sidney I Resnick, and Stephen M Robinson. *Springer Series in Operations Research and Financial Engineering*. 2006.

[51] Martin Fodslette Møller. "A scaled conjugate gradient algorithm for fast supervised learning". In: *Neural networks* 6.4 (1993), pp. 525–533.

[52] Benjamin Moseley, Andrew Markham, and Tarje Nissen-Meyer. "Fast approximate simulation of seismic waves with deep learning". In: *arXiv preprint arXiv:1807.06873* (2018).

[53] KB Olsen. "Site amplification in the Los Angeles basin from three-dimensional modeling of ground motion". In: *Bulletin of the Seismological Society of America* 90.6B (2000), S77–S94.

[54] Roberto Paolucci et al. "Broadband Ground Motions from 3D Physics-Based Numerical Simulations Using Artificial Neural Networks". en. In: *Bulletin of the Seismological Society of America* 108.3A (June 2018), pp. 1272–1286.

[55] Ilaria Perugia and Dominik Schötzau. "An *hp*-analysis of the local discontinuous Galerkin method for diffusion problems". In: *Journal of Scientific Computing* 17.1-4 (2002), pp. 561–571.

[56] Arben Pitarka et al. "Three-dimensional simulation of the near-fault ground motion for the 1995 Hyogo-ken Nanbu (Kobe), Japan, earthquake". In: *Bulletin of the Seismological Society of America* 88.2 (1998), pp. 428–440.

[57] Elijah Polak and Gerard Ribiere. "Note sur la convergence de méthodes de directions conjuguées". In: *ESAIM: Mathematical Modelling and Numerical Analysis-Modélisation Mathématique et Analyse Numérique* 3.R1 (1969), pp. 35–43.

[58] Michael James David Powell. "Restart procedures for the conjugate gradient method". In: *Mathematical programming* 12.1 (1977), pp. 241–254.

[59] Michael JD Powell. "Nonconvex minimization calculations and the conjugate gradient method". In: *Numerical analysis*. Springer, 1984, pp. 122–141.

[60] A. Quarteroni, A. Tagliani, and E. Zampieri. "Generalized Galerkin approximations of elastic waves with absorbing boundary conditions". In: *Computer Methods in Applied Mechanics and Engineering* 163.1–4 (1998), pp. 323 –341.

[61] Alfio Quarteroni and Alberto Valli. *Numerical Approximation of Partial Differential Equations*. Vol. 23. Springer Science & Business Media, 2008.

[62] Alfio Quarteroni and Elena Zampieri. "Finite Element Preconditioning for Legendre Spectral Collocation Approximations to Elliptic Equations and Systems". In: *SIAM Journal on Numerical Analysis* 29.4 (1992), pp. 917–936.

[63] P A Raviart and J M Thomas. *Introduction à l'analyse numérique des équations aux dérivées partielles*. Masson, 1983.

[64] Ge Ren-Pu and Michael JD Powell. "The convergence of variable metric matrices in unconstrained optimization". In: *Mathematical programming* 27.2 (1983), p. 123.

[65] Martin Riedmiller and Heinrich Braun. "A direct adaptive method for faster backpropagation learning: The RPROP algorithm". In: *Proceedings of the IEEE international conference on neural networks*. Vol. 1993. San Francisco. 1993, pp. 586–591.

[66] Béatrice Rivière. *Discontinuous Galerkin Methods for Solving Elliptic and Parabolic Equations: Theory and Implementation*. Society for Industrial and Applied Mathematics, 2008.

[67] David E Rumelhart et al. "Sequential thought processes in PDP models". In: *Parallel distributed processing: explorations in the microstructures of cognition* 2 (1986), pp. 3–57.

[68] L.E. Scales. "Introduction to Non-Linear Optimization". In: (1985).

[69] Jacob B Schroder. "Parallelizing over artificial neural network training runs with multigrid". In: *arXiv preprint arXiv:1708.02276* (2017).

[70] Gerard T Schuster. *Seismic inversion*. Society of Exploration Geophysicists, 2017.

[71] Geza Seriani, E Priolo, and A Pregarz. "Modelling waves in anisotropic media by a spectral element method". In: *Proceedings of the third international conference on mathematical and numerical aspects of wave propagation*. SIAM. 1995, pp. 289–298.

[72] Numair A Siddiqui et al. "2D and 3D seismic simulation for fault modeling: exploratory revision from the Gullfaks field". In: *Journal of Petroleum Exploration and Production Technology* 7.2 (2017), pp. 417–432.

[73] Chiara Smerzini and Manuela Villani. "Broadband numerical simulations in complex near-field geological configurations: The case of the 2009 M w 6.3 L'Aquila earthquake". In: *Bulletin of the Seismological Society of America* 102.6 (2012), pp. 2436–2451.

[74] Chiara Smerzini et al. “Ground motion record selection based on broadband spectral compatibility”. In: *Earthquake Spectra* 30.4 (2014), pp. 1427–1448.

[75] Marco Stupazzini. “A spectral element approach for 3D dynamic soil-structure interaction problems”. PhD thesis. Politecnico di Milano, 2004.

[76] Marco Stupazzini, Roberto Paolucci, and Heiner Igel. “Near-fault earthquake ground-motion simulation in the Grenoble valley by a high-performance spectral element code”. In: *Bulletin of the Seismological Society of America* 99.1 (2009), pp. 286–301.

[77] David J Wald and Robert W Graves. “The seismic response of the Los Angeles basin, California”. In: *Bulletin of the Seismological Society of America* 88.2 (1998), pp. 337–356.

[78] Jinchao Xu and Ludmil Zikatanov. “Algebraic multigrid methods”. In: *Acta Numerica* 26 (2017), pp. 591–721.

[79] Kailai Xu Bella Shi Shuyi Yin. “Deep Learning for Partial Differential Equations (PDEs)”. In: (2018).

[80] Ya-xiang Yuan. “A modified BFGS algorithm for unconstrained optimization”. In: *IMA Journal of Numerical Analysis* 11.3 (1991), pp. 325–332.

[81] Weiqiang Zhu, Yixiao Sheng, and Yi Sun. “Wave-dynamics simulation using deep neural networks”. In: (2017).