POLITECNICO DI MILANO

FACOLTÀ DI ING I - SCUOLA DI INGEGNERIA CIVILE, AMBIENTALE E TERRITORIALE
Corso di in Ingegneria Civile - Civil Engineering

Tesi di Laurea Magistrale

Artificial Neural Networks applied to problems in structural Mechanics and thermo-Mechanics

Relatore:
Prof. Alberto Corigliano

Correlatore:
Ing. Luca Rosafalco

Tesi di laurea di:
Stefano Gianbattista Fanizzi
Matricola 859265

Anno Accademico 2017-2018

CONTENTS

LIST OF FIGURES

LIST OF TABLES

INTRODUCTION

Replication of the human brain is one of the greatest challenge the human race has been working on. The first recorded reference to the brain date back to the 17th century B.C. , on an ancient Egyptian medical treatise. The great thinkers of history from Greece to Rome believed the brain *to be the seat of intelligence*. The modern study of the brain anatomy began in the Renaissance, from the work of Mondino de Luzzi, and they continue over the years until the invention of high technological tools, the *miscroscopes*. Camillo Golgi was able to show the structure of a *neuron*, and gradually the knowledge increases to this time, where quantum physic is able to push science beyond our wildest dream [19]. (From Wikipedia Contributors, *Human brain* [23]).

From '60s, mathematical models start developing while the step change coincides with the spread of personal computers. As a matter of fact, like in many other fields, the actual realization of theoretical concepts becomes real. *Artificial Neural Networks* are hidden in our everyday lives, as SPAM filter for email client or marketing strategies through *cookies*. Other application lies on objects recognition, and in general term of *features* extraction from large database of data.

Artificial Neural Networks are the *hot* topic of this historical period (i.e. the first quarter of the 20th century) and researchers from the all the countries are currently working on it and techniques are continually evolving.

The purpose of the thesis is to apply Artificial Neural Networks to the Mechanical engineering field. Experimental data are usually available but often it is not clear how to tackle them properly: ANNs are able to interpret them and to generate models for next insights. For particular applications, some versions of ANNs, namely *Recurrent Neural Networks*, can substitute determinist approaches, like the *finite element methods*, if sufficiently trained; this may give benefits when the algorithms demand high computational costs and resources.

Literature often addresses the problem from a theoretical point of view neglecting practical implications: the imprint of the work tries to answers to the call of a coder who needs efficient combination of theory and practice.

MATLAB®, and in particular the Deep Learning Toolbox™, is used for the ANN algorithms. The worked examples are developed from scratch, in detail:

- The analytical solution of 3.1, 3.2 and 3.3 is implemented in MATLAB®.
- The solution of 3.4 is achieved applying the FEM, coded in FORTRAN with The NAG® Fortran compiler.
- The solution of 3.5 is computed by a MATLAB® program given from the authors Confalonieri, Terraneo, and Corigliano of [3].

The analytical solutions are presented in the appendices to centre the attention on results, since the physical nature of the problem is not perceived by the network. In fact, what influences the most is the tuning of parameters defining the neural network. Results shows that small changes in the architecture for the same problem or equal architectures for systems apparently similar, lead to divergent output. It is not trivial to overcome this sensitivity, and a possible way is to find the solution in an iterative manner.

The thesis is divided in three parts:

FIRST CHAPTER describes the feed-forward backpropagation neural networks. It starts with an overview of the mathematical model, and it includes a series of introductory examples.

SECOND CHAPTER extends the concept of the first chapter to Recurrent Neural Networks. It contains differences and novelties of this architecture along with an example based on a dynamic response of a damped oscillator.

THIRD CHAPTER focuses and develops worked examples, based on structural Mechanics and thermo-Mechanics. In details:

i. Static deflection of a cantilever beam at page 39.

ii. Dynamic deflection of a simply supported beam at page 45, developed analytically through modal analysis.

iii. Dynamic deflection of a cantilever beam at page 50, developed numerically by the FEM method.

iv. Dynamic response of a portal frame at page 45, developed analytically through modal analysis.

v. Dynamic response of a plane resonator at page 52, coupled in the sense of thermo-Mechanics, developed numerically by the FEM method.

1 ARTIFICIAL NEURAL NETWORKS

Work on artificial neural network is been motivated from its inception by the recognition that the human brain computes in an entirely different way from the conventional digital computers. The brain has the capability to organize its structural constituents, known as *neurons*, to perform certain computations (for example pattern recognition, perception and motor control) many time faster than the fastest digital computer existing today.

To do all of this, at birth, a brain already has considerable structure and the ability to build its own rules of behaviour through what we usually refer to as "experience". Indeed, experience is built up over time, with much of the development of the human brain taking place during the first two years from birth, but the development continues well beyond that stage.

Very generally, a *neural network* is a machine that is designed to *model* the way in which the brain performs a particular task or function of interest; the network is usually implemented by using electronic components or is simulated in software on digital computer. To achieve good performance, neural networks employ a massive interconnection of simple computing cells referred to as "neurons" or "processing units".

The design of neural network is motivated by analogy with the brain, which is living proof that fault-tolerant parallel processing is not physically possible, but also fast and powerful. Engineers look to neurobiology for new ideas to solve problems more complex than those based on conventional hardwired design techniques. (From Haykin, *Neural Networks and Learning Machines* [7]).

1.1 PERCEPTRON

The first neuronal model is due to Rosenblatt, and it is call *Perceptron*. It is composed by:

INPUTS A series of inputs simulating the signals from the external environment.

SINAPTIC WEIGHTS Parameters that give a weight to each input signal, assigning an importance level.

BIAS Internal fixed input working as activation parameter for the neuron.

SUMMING JUNCTION The sum of all the weighted inputs coming to the neuron.

ACTIVATION FUNCTION A priori function that process the summing function, giving the output of the neuron.

Picture 1 depicts a schematic view of the perceptron.

The model for the k-th perceptron can be cast as:

$$\begin{cases} u_k = \sum_{j=1}^{m} \omega_{kj} x_j \\ y_k = \varphi(\underbrace{u_k + b_k}_{v_k}) \end{cases} \tag{1}$$

Where:

x_j Inputs.

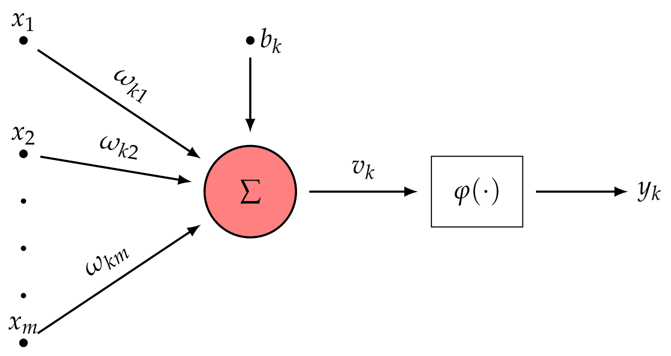


Figure 1: Perceptron Model.

ω_{kj} Synaptic weights.

u_k Linear Combiner.

b_k Bias.

v_k Induced local field or activation potential.

$\varphi(\cdot)$ Activation function.

The writing can be lighten considering the bias as an additional synapse setting:

$$\begin{cases} x_0 = 1 \\ \omega_{k0} = b_k \end{cases} \tag{2}$$

Hence,

$$\begin{cases} u_k = \sum_{j=0}^{m} \omega_{kj} x_j \\ y_k = \varphi(\underbrace{u_k + b_k}_{v_k}) \end{cases} \tag{3}$$

1.2 ARCHITECTURE

The architecture of a neural network defines its work flow. The single perceptron model, defined in 1.1, is the fundamental component of every neural network architecture.

Two basic types of network architectures can be distinguished:

SINGLE-LAYER/MULTILAYER FEEDFORWARD NETWORKS The work flow is one directional from the input to the output.

RECURRENT NETWORKS The work flow is bi-directional; the input is influenced by *feedback* of previous output.

Other typology are directly derived, improving the network for specific tasks.

Picture 2 depicts a schematic view of a multi-layer feed-forward network.

1.3 THE TRAINING CONCEPT

A neural network should *learn* from the environment in the same way a biological brain would do, by *experience*. Learning means understanding of concatenation of actions and reactions leading to a specific event, aiming the replication. The same

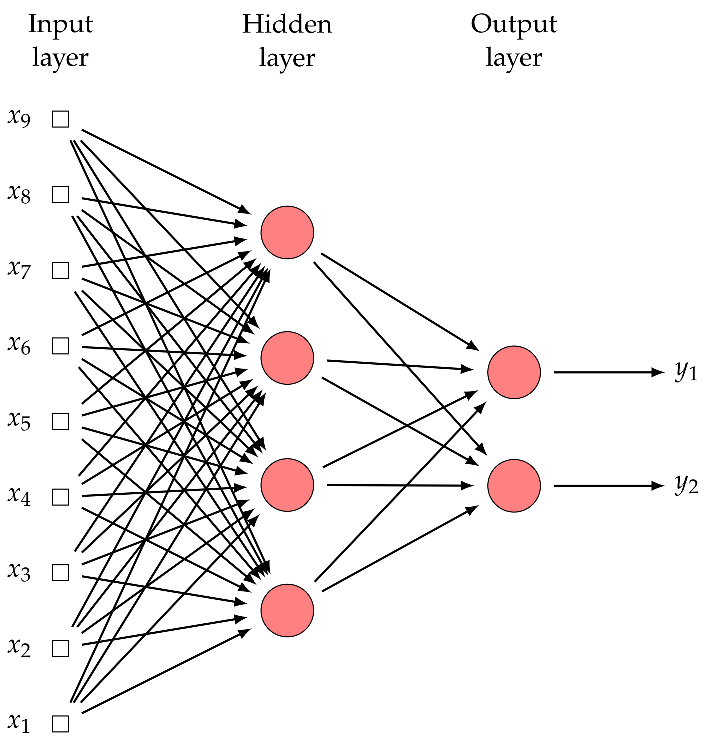


Figure 2: Multilayer feedforward networks scheme

concept is here applied: synaptic weights and bias are adjusted in such a way to reproduce a particular event, represented by known couples of input and expected output, forming a dataset.

Data can be distinguished in:

LABELED Where pairs of input-output data are available.

UNLABELED Where only input data are available.

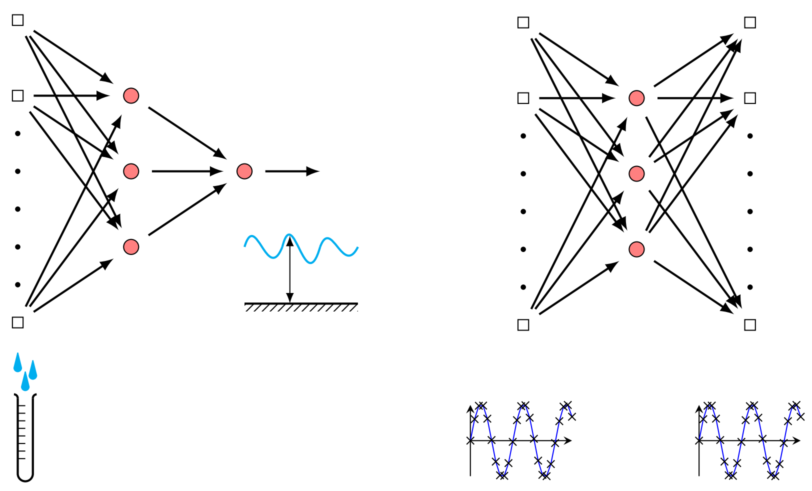
The training phase refers to *supervised* and *unsupervised* whether the data is labeled or unlabeled.

As an example of the first type, the input could be the rainfall in a particular area while the output the water level of a river: the aim of the network would be to predict the flood hazard. On the contrary, the second type could be tricky to be understand: in signal processing, inputs might be time instants where the aim is the data compression. The neural network would extract the meaningful data and reconstruct the original dataset, keeping input and output equals. The latter type of architecture is called auto-encoder.

Therefore, the dataset is part of the definition of a network, being strictly connected within its architecture.

During learning, replication of data, i.e outputs for a neural network, would generally not be equal to observation: closer is the reproduction, higher is the level of understanding. Thus, this process can be seen as minimizing the difference between observed and predicted data, which is the aim of a specific branch of mathematics, which belongs to a well known mathematical class of problems, called *optimization* problems. The minimization is set between the target data, i.e the response of the observation, and the output of the neural net, that depends on weights and biases as previously mentioned. Then a statistically meaningful function is defined, called loss function, and the minimum deviation computed. The typical one is the *mean square error* or MSE, defined as:

$$MSE = \frac{\sum_{n=0}^{n} \left(y_i - \hat{y}_i\right)^2}{n} \tag{4}$$



(a) Supervised (b) Unsupervised

Figure 3: Example of different architectures for supervised and unsupervised learning.

Where y_i and $\hat{y}_i$ are the i-th target and the output respectively.

Generally, the training phase is subdivided into three steps, at which different parts of the dataset are associated:

LEARNING Actual minimization of the loss function based on the largest part of the dataset.

TESTING Test, or validation, of the neural network over a small part of the dataset *during the learning*. It uses techniques useful to prevent over-fitting.

GENERALIZATION The neural network is asked to *generalize*, i.e to correctly predict target values.

The minimization is based on *backpropagation* of the error, i.e the error at the output layer is brought back through hidden layers, weights and biases adjusted and new outputs computed. This process is performed until convergence is reached, and each back-forward passage is called *epoch*.

1.4 BACKPROPAGATION ALGORITM

The backpropagation algorithm is at the base of the training of every neural network. The loss function is minimized computing the gradient related to each node of the network, that is function of weights and biases. The different ways in which the gradient is computed lead to algorithm specifically designed to increase the speed of convergence, optimization of the memory or attenuation of overfitting and convergence to local minima. Among them there are the ones based on the gradient descent and the quasi-Newton algorithm.

For a deep understanding of the backpropagation algorithm [see 7, pp. 129–141]. For the optimization problem [see 7, pp. 186–199] and [9, 14].

1.4.1 Gradient Descent Algorithm

Let us consider a differential function $f(x, y)$ in the two variables x and y, the gradient $\nabla f(x, y)$ is defined as:

$$\nabla f(\mathbf{x}) = \frac{\partial f(x,y)}{\partial x} \cdot \mathbf{i} + \frac{\partial f(x,y)}{\partial j} \cdot \mathbf{j} \tag{5}$$

It has the following properties:

i. The direction is normal to the contour lines in any point (a, b).

ii. In any point (a, b), $f(x, y)$ decreases more rapidly in the direction $-\nabla f(a, b)$.

The two properties can be used to find the minimum of $f(x, y)$ if f is differentiable and convex.

Rewriting the function f in a vectorial form, by setting:

$$(a, b) = \begin{bmatrix} a \\ b \end{bmatrix} = \mathbf{x} \tag{6}$$

The minimum can be found starting from an initial guess $\mathbf{x}_0$ and moving towards minimum by using i. and ii.:

$$\mathbf{x}_1 = \mathbf{x}_0 - \gamma \cdot \nabla f(\mathbf{x}_0) \tag{7}$$

γ is a parameter that governs the decreasing rate. The procedure is set in an iterative form as:

$$\mathbf{x}_{n+1} = \mathbf{x}_n - \gamma \cdot \nabla f(\mathbf{x}_n) \tag{8}$$

Until convergence is satisfied:

$$f(\mathbf{x}_n) \simeq \mathbf{0} \tag{9}$$

Alternative versions of the gradient descent algorithm compute the optimal direction at each step of the iteration, such as the scaled-conjugated gradient descent algorithm.

1.4.2 Quasi-Newton Methods

The Newton or Newton-Raphson methods are based on the computation of the Hessian matrix.

Considering a differentiable function $f(\mathbf{x})$. Its Taylor expansion reads:

$$f(\mathbf{x}_0 + \delta\mathbf{x}) = f(\mathbf{x}_0) + \nabla f(\mathbf{x}_0))\delta\mathbf{x} + \mathcal{O}(\delta\mathbf{x}^2) \tag{10}$$

Then, by setting:

$$f(\mathbf{x}_0 + \delta\mathbf{x}) = \mathbf{0} \tag{11}$$

one finds:

$$\delta\mathbf{x} = -J(\mathbf{x}_0)^{-1} f(\mathbf{x}_0) \tag{12}$$

Where $J(\mathbf{x})$ is the Jacobian of $f(\mathbf{x}_0)$:

$$J(\mathbf{x}) = \nabla f(\mathbf{x}_0) \tag{13}$$

If $\mathbf{x}$ is a root, $\delta\mathbf{x}$ measures how far is the solution from the vector $\mathbf{x}_0$. If $f(\mathbf{x})$ is non-linear, the vector:

$$\mathbf{x}_1 = \mathbf{x}_0 - J(\mathbf{x}_0)^{-1} f(\mathbf{x}_0) \tag{14}$$

si approximation of the root. The procedure can be set in an iterative way by setting:

$$\mathbf{x}_{n+1} = \mathbf{x}_n - J(\mathbf{x}_n)^{-1} f(\mathbf{x}_n) \tag{15}$$

checking the convergence on $f(\mathbf{x}_{n+1})$.

The latter procedure can be also used to find minima and maxima. The gradient of f is set equal to zero:

$$\nabla f(\mathbf{x}) = \mathbf{0} \tag{16}$$

Then, substituting into 10:

$$\nabla f(\mathbf{x} + \delta\mathbf{x}) = \nabla f(\mathbf{x}_0) + H(\mathbf{x_0})\delta\mathbf{x} = \mathbf{0} \tag{17}$$

Where $H(\cdot)$ is the Hessian operator. Rearranging some terms:

$$\delta\mathbf{x} = -H(\mathbf{x}_0)^{-1}\nabla f(\mathbf{x}_0) \tag{18}$$

Finally, the iterative procedure reads:

$$\mathbf{x}_{n+1} = \mathbf{x}_n - H(\mathbf{x}_n)^{-1}\nabla f(\mathbf{x}_n) \tag{19}$$

The drawback of this method is the computation of the inverse of the Hessian matrix, due to its high computational cost, especially if the problem has several variables.

The quasi-Newton method accounts for the problem of substituting the inverse of the Hessian matrix at each iteration step with an approximation of it, for instance by using the secant tangent stiffness.

1.4.3 Levemberg-Marquardt Method

The Levemberg-Marquardt method defines the step increment $\delta\mathbf{x}$ as a linear combination between the ones determined by the gradient descend and the quasi-Newton methods:

$$\delta\mathbf{x} = -[\mathbf{H}^{-1} + \lambda\mathbf{I}]\nabla f(\mathbf{x}_0) \tag{20}$$

The idea is to combine the robustness of the gradient descent method with the speed of convergence of the Quasi-Newton one.

For additional information regarding the implementation and the effects of λ on the stability of the algorithm [7, see].

1.5 FUNCTION APPROXIMATION PROBLEM

Considering the following problem:

$$\begin{cases} \mathbf{d} = f(\mathbf{x}) \\ \mathbf{d} = \text{Output} \\ \mathbf{x} = \text{Input} \\ \mathbf{f} = \text{Unknown model} \end{cases} \tag{21}$$

And a set of labeled data $\boldsymbol{\tau}$:

$$\boldsymbol{\tau} = \left\{(\mathbf{x}_i, \mathbf{d}_i)\right\}_{i=1}^{N} \tag{22}$$

The aim is to built up a neural network such that:

$$\|F(\mathbf{x}) - f(\mathbf{x})\| < \varepsilon \quad \text{for all } \mathbf{x} \tag{23}$$

With

$f(\mathbf{x})$ Unknown function to be evaluated.

$F(\mathbf{x})$ Function given by the neural network (due to the mapping between input and output).

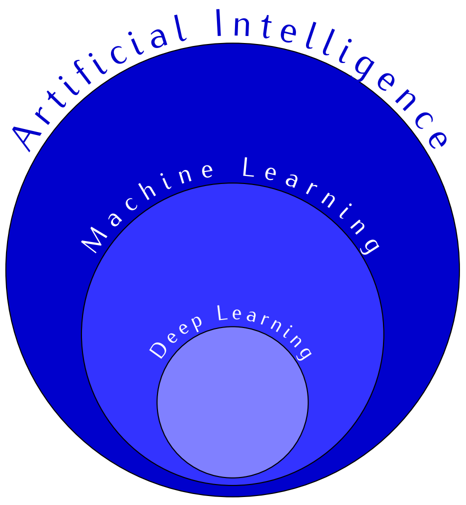


Figure 4: Subclasses of AI

1.6 DESIGN OF MULTILAYER FEEDFORWARD NETWORKS

The definition of the architecture of a neural network itself is characterized by the choice of number of perceptrons, layers, synaptic connections which allows to create several type of ANNs, still remaining in the chosen network category. Also, the backpropagation algorithm may converge, as it would probably do, to different local minima or in the best scenario to the absolute minimum. This means that there is not a unique correct configuration, but only the best one upon some a priori fixed thresholds: the design maybe be set up as find the *best configuration* among different architectures or find the *first* architecture that leads to suitable results. The downside is that there is not a theoretical approach to find out those configurations: the choices are based on the experience of the developer and the use of code in *trial and error* fashion.

In the following sections is explored how neural networks are influenced by the choice of parameters, studying the sensitivity to modifications, varying one by one the principal characteristics of an ANN.

1.6.1 Number of Layers

The number of layers is strictly connected to a specific branch of machine learning, generally referred as *deep learning*. The word *deep* is there to indicate how much the neural network is deep, i.e how many hidden layers are chosen.

This number is mainly defined depending on the aim of the ANN: considering the applications described in the introduction, the *sufficient* number of layers may varies from one for function approximation 1.5 and dozens for image recognition. This discussion is mainly focused on function approximation, hence it overlooks the second type.

The word *sufficient* is emphasised to highlight the considerations made in the introduction of the section. To this scope, it is helpful to remember that higher is the number of free parameters of the problem, higher is the difficulty to reach convergence: for instance, the number of local minima increases within free parameters and so the possibility to be stuck over one of them. An high number influences the overfitting as well.

These observations lead to the choice as first attempt of a single hidden layer. The presented examples show clearly that one hidden layer is generally sufficient to approximate any generic function, taking care of the choices of the other elements of the ANN.

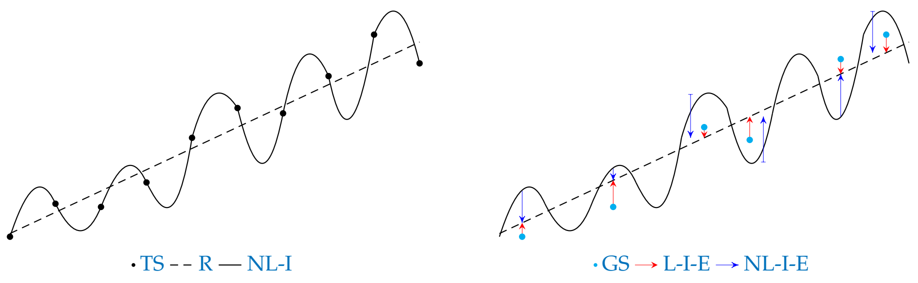


Figure 5: Graphical interpretation of the overfitting due to high number of perceptrons.

1.6.2 Number Of Hidden Neurons

Hidden neurons are the part of the neural network that actually performs the computations: they take inputs, analyse the importance of each of them according to the training (i.e weighed sum) and give a response. The responses are generally treated a second time in the output layer, which eventually gives the output of the ANN.

According to this, there should be a minimum value of neurons which does not produce an *underfitting* of data and no bounds for what concern the maximum value. On the contrary, the use of a too high number of neurons introduces too much flexibility in the network, and then the possibility to lead to the problem of *overfitting*.

Too many perceptrons would create a neural network which is not able to *generalize*. The problem of generalization refers to the case in which the training of the ANN performs well in the learning phase, i.e the minimization of the error is reached, but the response of the neural network is bad for data outside the training phase. This problem is usually clarified by the *Occam's razor*: the simplest solution tends to be the most correct.

Imaging to interpolate a set of experimental data, which can be the training one TS, with linear trend through linear regression R: by definition, only two free parameters define the interpolation, the slope m and the intercept of the y-axis q. Being the data experimental, they will be affected by noise. The use of complex functions for the interpolation is not forbidden: increasing the number of free parameters by using non-linear function it may decrease the MSE, since the interpolation better follows the series of points. Inquiring the solution in a new set of data, which can be the generalization set GS, the results may have a response totally different from the linear one of the training, because the interpolation tends to include the noise. Picture 5 shows that the error considering the linear interpolation $\rightarrow$ L-I-E is globally lower than the error of the non-linear interpolation $\rightarrow$ NL-I-E .

This effect is strictly related to the data on which the ANN is asked to work. As a consequence, there is not a priori number of perceptrons which is suitable for every type of problems.

A general indication of the upper bound can be computed by considering the ratio between the number of data in the training set and the total number of free parameters inside the model:

$$\alpha \cdot N_h \cdot (N_i + N_o) = N_s \tag{24}$$

Where

N_h Number of hidden neurons.

N_i Number of input neurons.

N_o Number of output neurons.

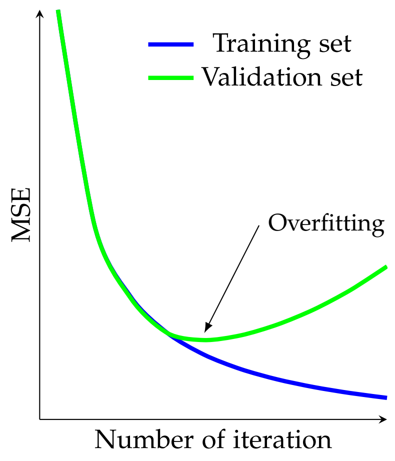


Figure 6: Overfitting due to data

N_s Number of samples in training dataset.

α Scaling factor, between two and ten.

The idea is to keep the number of free parameters always lower than the number of data for the training, introducing the parameter α. Then:

$$N_h = \frac{N_s}{\alpha \cdot (N_i + N_o)} \tag{25}$$

The problem of overfitting can be mitigated through techniques that work directly during the training and through the use of specific algorithms.

A widely used approach is the subdivision in different portions of the dataset as introduced in section 1.3. The definition of a validation dataset allows to train and test the ANN on different data: indeed, during the training, the neural network is tested on data not employed in the optimization procedure and checked. If the loss function decreases on the training set while it increases on the validation one overfitting is occurring and the algorithm stops. This method does not actually prevent overfitting but it is a warning, and it suggests other precautions. To have an effective and true vision over the whole set of data, the selection for each portion is based on random process instead of a continuous blocks subdivision. This approach may have some side effects for some typology of neural networks. Picture 6 shows the concept.

At the algorithm level, an usual effective procedure is to update weights and biases for a smaller part of the whole training dataset, in the order of 64 or 128, called *batches*. An *iteration* is when a batch is passed and the parameters updated, while an *epoch* is when the whole training dataset is passed from input to output.

Example 1.6.2.1 Variation of number of neurons
The example regards the built-up of a feedforward neural network with constant parameter varying the number of neurons. The aim of the ANN is to approximate a sine curve following the concept of section 1.5.

N_s	N_i	N_o	N_h	TA	LF	$\varphi(\cdot)$
25	1	1	-	GD	MSE	tanh

Table 1: ANN parameters.

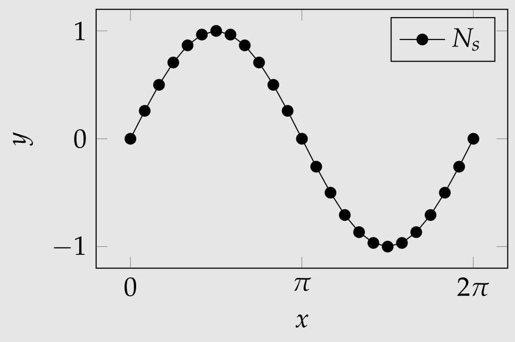


Figure 7: Training set.

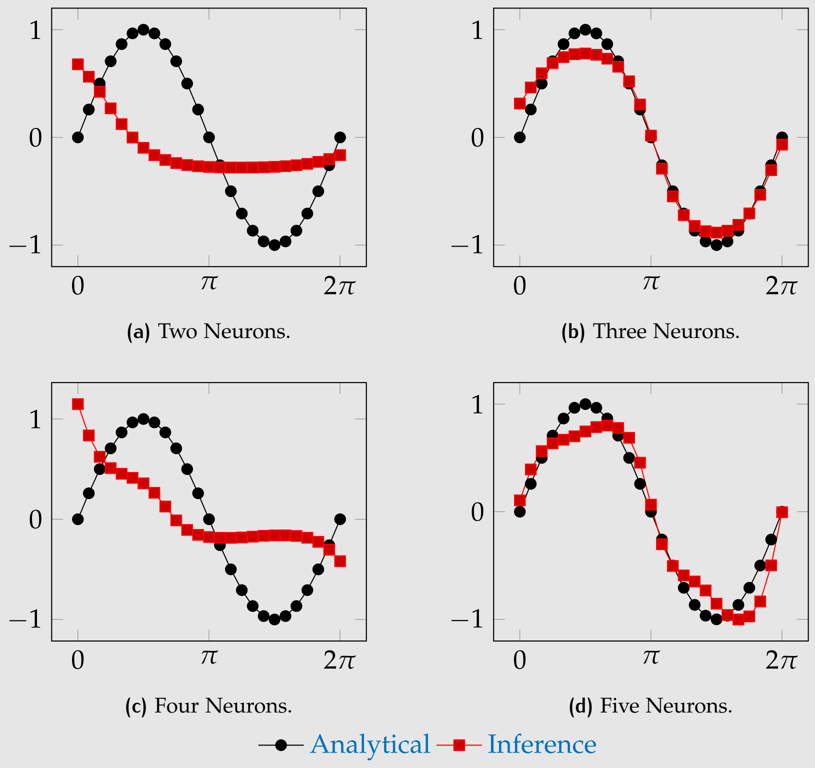


Figure 8: Inference variation due to the number of hidden neurons.

The figure shows how the net underfits the data (20a), inferences within good approximation (20b) and starts to overfit (20c) and (20d), increasing the oscillations at increasing number of hidden neurons. Passing from three (20b) to four (20c), the ANN completely corrupts the acceptable results; passing from four (20c) to five (20d) it restores the good predictions, although it shows the characteristics of overfitting.

1.6.3 Activation Function

The activation function has a pivotal role in the functioning of neural networks, having the role of *activator* of perceptrons. Giving a vector of inputs, each perceptron should be able to identify the relative importance of each one of its entries, in accordance with its role and infers the desired output. The idea is to find out a function that is able to distinguish and plotting different values, giving an univocal response in case some conditions are not fulfilled. From some points of view, the perceptron should work similarly to a Boolean function: giving a "positive" response if the

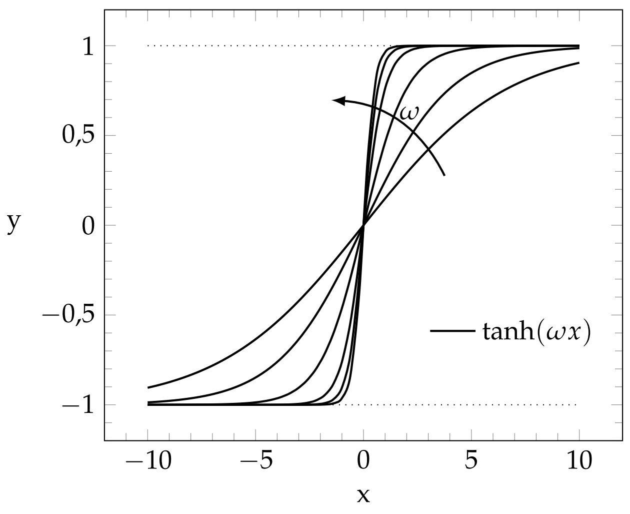


Figure 9: Hyperbolic tangent.

condition is true, i.e the inputs contribution falls in a specific range, actually giving different results, and a "negative" response, giving out a default number.

A family of functions that satisfies these conditions are the sigmoid functions[1]. A sigmoid function is a s-shaped function which is approximately linear in small ranges around zeros, and quickly goes towards two horizontal asymptotes. When the argument varies in the first range, the outputs are well separate, assimilating the "positive" response of a Boolean function, while outside they are closer and closer, assimilating the "negative" response of a Boolean function.

A well known sigmoid function is the hyperbolic tangent, which has the following definition:

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{26}$$

And properties:

$$\lim_{x\to\pm\infty} \tanh(x) = \pm 1 \tag{27}$$

$$\lim_{x\to 0} \tanh(x) = x \tag{28}$$

Moreover:

$$\tanh(x) \approx x \ , \ x = [-1; +1] \tag{29}$$

The latter property is also important with regard to the initialization of parameters (see 1.6.4).

As previously defined, the argument of the sigmoid is a weighted sum over the inputs plus the bias, which, in light of this working mechanisms, has the role of translating the ordinates, increasing or decreasing the trigger point. Considering the example 1.6.4.1 in section 1.6.4, the picture 10 shows when each perceptron is activated for $x = \{-1; 0.25, 0.75\}$.

An essential remark regards the condition required to fulfil (29). If the input vector ranges too big or too small values, the weights, which are the parameters controlling the width of the sigmoid argument, would be too small or too big respectively, in order to squash the domain in the sought range. This condition may arise numerical issues during the minimization procedure worked out by the back-propagation algorithm, increasing the computational time or not reaching any convergence. This problem is generally known as *vanishing of the gradient*, and it occurs when the weights are closer to zero.

1 12.

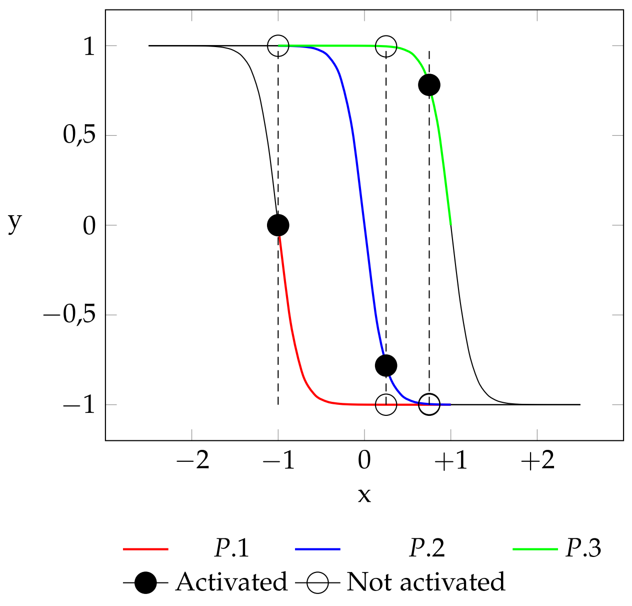


Figure 10: Activation function range

In order to avoid this issue, an effective procedure that should be always performed is to scale the input and the target vectors, using standardization or normalization. The first one scales numbers belonging to the range x_{Min}-x_{Max} and translates them in values belonging to the new range x_{min}-x_{max} :

$$\begin{aligned} x &\in \left[x_{Max}; x_{Min}\right] \rightarrow x_s \in \left[x_{max}; x_{min}\right] \\ x_s &= \frac{x - x_{Min}}{x_{Max} - x_{Min}}\left(x_{max} - x_{min}\right) + x_{min} \end{aligned} \tag{30}$$

While the second procedure normalizes the number with respect to mean $E[X]$ ad variance σ_X^2:

$$x_n = \frac{x - E[X]}{\sigma_X^2} \tag{31}$$

An activation function that is commonly used in the output layer is the linear activation function, which does not modify the activation potential (see the definition of the perceptron 1.1). The main role of the output layer is to filter the outputs of the hidden layer removing the saturation terms and leaving only the output in the active region. This behavior can be observed looking at the figure 13.

Considering for instance an input value:

$$x_s = -0.25 \Leftrightarrow x = \frac{5}{4}\pi \tag{32}$$

the activation potential in the output layer becomes:

$$u_k = \sum_{i=1}^{3} \omega_{io} S\left(\omega_{hi} x + b_{hi}\right) \tag{33}$$

Substituting the values of weights and biases:

$$\begin{aligned} u_k = & +0.76 \cdot S\big(+4.30 \cdot (-0.25) + (-4.11)\big) + \\ & + (-0.96) \cdot S\big(+4.22 \cdot (-0.25) + 0.01\big) + \\ & + (-0.99) \cdot S\big((-4.18) \cdot (-0.25) + (-4.24)\big) \\ & \underbrace{+0.76 \cdot S(-5.19)}_{\text{Saturation P.1}} + \underbrace{(-0.96) \cdot S(-1.05)}_{\text{Active region}} + \underbrace{(-0.99) \cdot S(-3.20)}_{\text{Saturation P.3}} \\ & \underbrace{+0.23}_{\text{Saturation sum}} \quad + \quad \underbrace{+0.75}_{\text{Active region}} \end{aligned} \tag{34}$$

In this way, the sum of the saturation terms is removed by the bias:

$$v_k = \sum_{i=1}^{3} \omega_{io} S\big(\omega_{hi} x + b_{hi}\big) - b_o = +0.23 + 0.75 - 0.23 = +0.75 \tag{35}$$

And:

$$v_k = 0.75 \simeq \sin\left(\frac{5}{4}\pi\right) = \frac{\sqrt{2}}{2} \tag{36}$$

1.6.4 The Problem of Random Initialization of Parameters

Traditionally, the minimimization problem starts with a random selection of weights and biases. The number of local minima increases with the number of variables the loss function depends on, and consequently the possibility to be stuck over one of them is increased. A proper selection of the first initial guesses reduces the risk.

A widely use algorithm for two layers neural networks is the Nguyen-Widrow algorithm. The algorithm selects the initial parameters such that each perceptron covers a specific part of the input space, without or with small overlapping domains. In this way all the perceptron are independently responsible of a portion of the training set. To improve the stability of the procedure the input space is generally standardized in the range of minus plus one. In section 1.6.3 other benefits of the pre-processing of data have already been discussed. For additional information [see 12].

Considers an ANN with the following features:

- Two layers feedforward network.
- Single input and single output.
- N number of hidden perceptron.
- Sigmoid activation function.
- Unitary weights in the output layer.

The output of such neural network can be written as:

$$y = \sum_{i=0}^{N-1} v_i S\big(\omega_i x + \omega_{bi}\big) \tag{37}$$

Considering to have standardized input vector in the range plus-minus one, the ω_i and ω_{bi} in the argument of the sigmoid function can be seen as the width and the center of the portion of the abscissa taken by the $i-th$ perceptron. The image of the function is almost linear in the range minus-plus one, with first derivative equal to one, while it tends quickly to two horizontal asymptotes outside. When the argument of the sigmoid falls into the latter range, it is said the perceptron *saturates*.

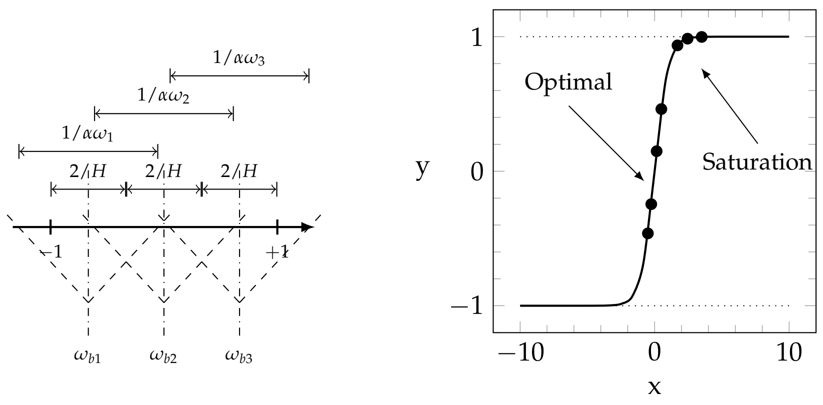


Figure 11: Optimal definition of parameters for random initialization.

This condition can be avoided limiting the domain of the sigmoid between minus and plus one.

$$-1 < w_i x + \omega_{bi} < 1 \tag{38}$$

Hence, the x interval can be written as:

$$-\frac{1}{\omega_i} - \omega_{bi} < x < \frac{1}{\omega_i} - \omega_{bi} \tag{39}$$

Thus, the length of the interval is:

$$\left|-\frac{1}{\omega_i}\right| + \frac{1}{\omega_i} = \frac{2}{\omega_i} \tag{40}$$

If the input domain is proportionally distributed among the number of perceptrons, the single interval turns to be the total length over N. Imposing the equality with expression (40):

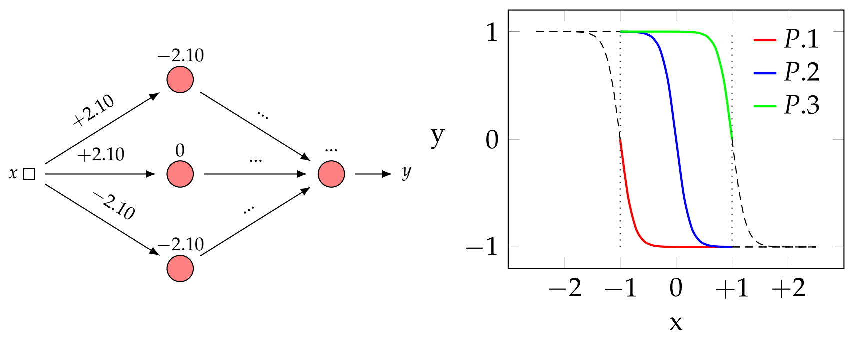
$$\frac{2}{\omega_i} = \frac{2}{H} \tag{41}$$

The computation of the initial weight ω_i is straightforward. However, it is better to have slightly overlapped domains, to be sure of covering all values over the intersection. an α parameter that guarantees the latter condition is therefore introduced:

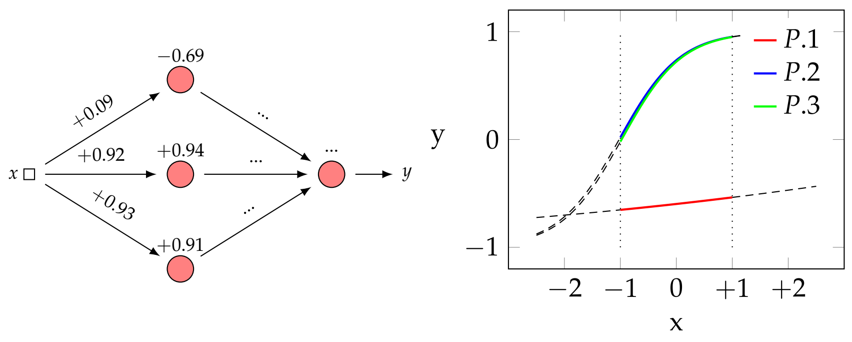
$$\frac{2}{\alpha\omega_i} = \frac{2}{H} \rightarrow \omega_i > \frac{H}{\alpha} \tag{42}$$

The authors in Nguyen and Widrow [12] suggest a value of $\alpha = 0.7$.

Then, the biases are randomly uniformly distributed such that the condition (38) is verified.



(a) Initial



(b) Final

Figure 12: Sigmoid domain for the three perceptron.

Picture 12 shows how the perceptrons initialized with the NW algorithm widely covers all the input space, compared to a generic random generation function. Weights and biases of the first layer are computed according to the previous equations for standardize input between minus and plus one.

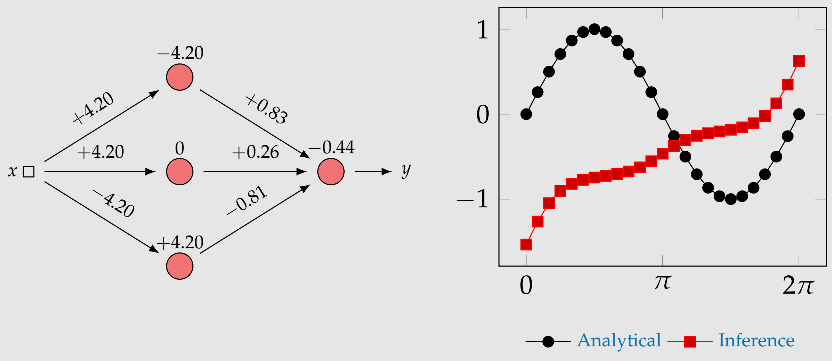
For the case of feedforward backpropagation neural networks applied to function approximation, another important remark that can be deduced from picture 12 regards the definition of the number of layers. The considerations presented in section 1.6.1 are strengthened by the fact that if the perceptrons are widely distributed across the input space, there is no need,in order to correctly approximate the function, to add a second hidden layer. As a matter of fact, by using the discussed initialization rule, the second layer would have an input space made mainly by saturated terms, i.e set of values close to minus-plus one.

On the contrary, the number of unknowns would increase, and consequently it increases the uncertainties and the complexity of the problem, especially to what concerns the minimization of the loss function. In deep learning for image recognition, for instance in convolutional neural networks, at each layer is assigned a portion the data, i.e a portion of an image identified by a matrix of pixels, that it is studied *independently*. In function approximation, there is only one output, which is the value of the function.

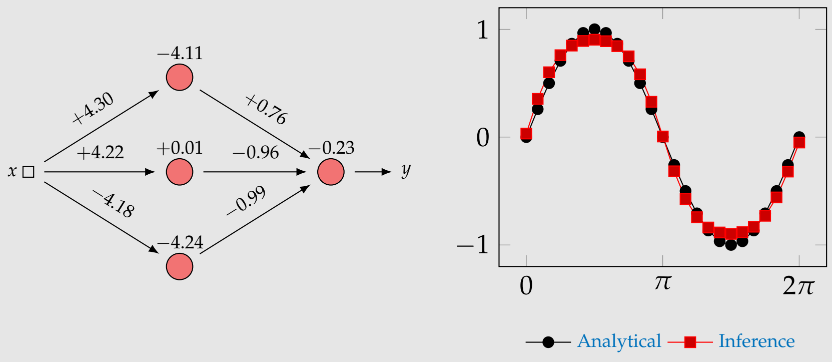
Example 1.6.4.1 Initialization function
The example regards how the random initialization function influences the convergence of the training, keeping the other parameters constant. In details, the Nguyen-Widrow algorithm is compared with an initialization based on a random number generator.

N_s	N_i	N_o	N_h	TA	LF	$\varphi(\cdot)$	NI
25	1	1	3	GD	MSE	Sig	1000

Table 2: ANN parameters.

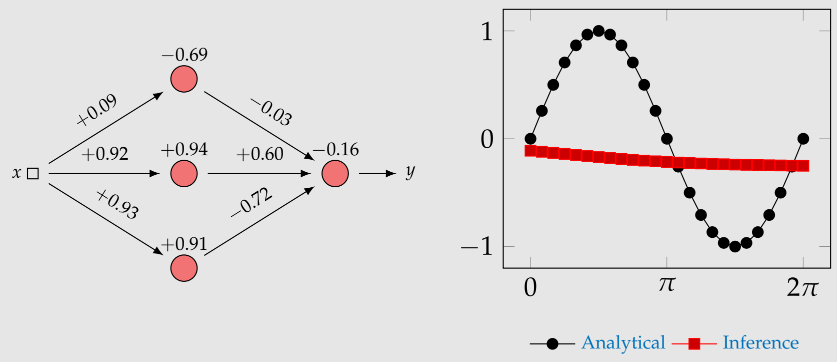


(a) Initial

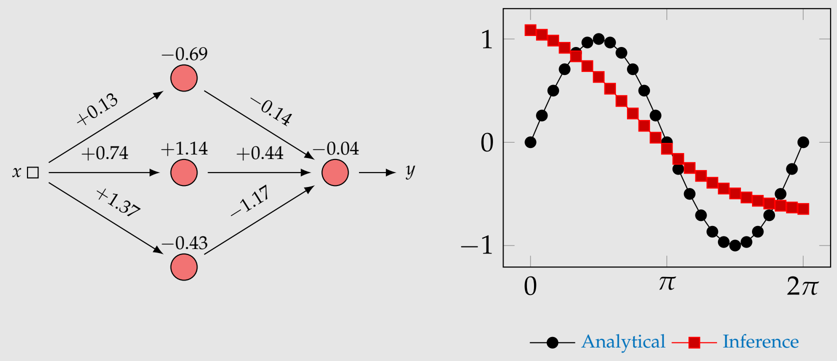


(b) Final

Figure 13: NW initialization



(a) Initial



(b) Final

Figure 14: Generic random initialization.

Picture 14 shows how big is the influence of the choice of the weight initialization strategy on the final results. On the contrary, the good behaviour of the NW algorithm can be seen comparing the variation of the weights and biases in the hidden layer before and after training: small differences are synonym of right initial guesses, while large variation for bad ones, like the second case.
One may think that the difference between the two initialization strategies is just in the number of iteration necessary to reach convergence, but this is not generally true: when a bad initialization is done, the training may stop due to validation check preventing overfitting.

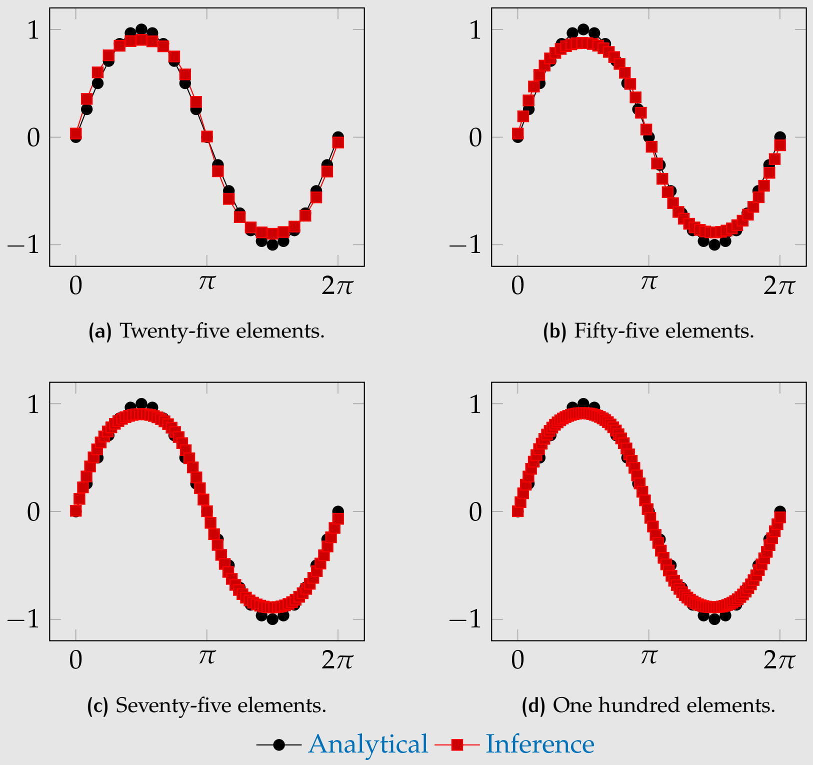
1.6.5 Data Distribution

The learning of any type of artificial intelligence is based on observation, as briefly introduced in section 1.3. It can be said that higher is the *experience* that it is fed, higher is the level of learning. For neural networks, this concept has a limitation, since after a suitable minimum is reached, the solution does not improve substantially, even increasing the number of data (see example 1.6.5.1). If anything, the computational time increases. A possible solution to optimize the ANN is to preprocess the data depending on the distribution. Again, there is not a theoretical way to find an optimal training set size, and the only reasons are based on experience.

Example 1.6.5.1 Variation of the size of the training set
The example treats the influence of the size of the training set in the learning of an ANN. Four increasing size dimensions are considered, starting from twenty-five to one hundred elements.

N_s	N_i	N_o	N_h	TA	LF	$\varphi(\cdot)$
-	1	1	3	GD	MSE	tanh

Table 3: ANN parameters.



(a) Twenty-five elements. (b) Fifty-five elements. (c) Seventy-five elements. (d) One hundred elements.

Figure 15: Inference variation due to the size of the training set.

The pictures show how increasing the input size does not produce any meaningful improvement . This result may be taken into account when the available data are unnecessary too close, for instance for a sensor continuously receiving data or dynamic finite elements solution where a small step dimension is necessary.

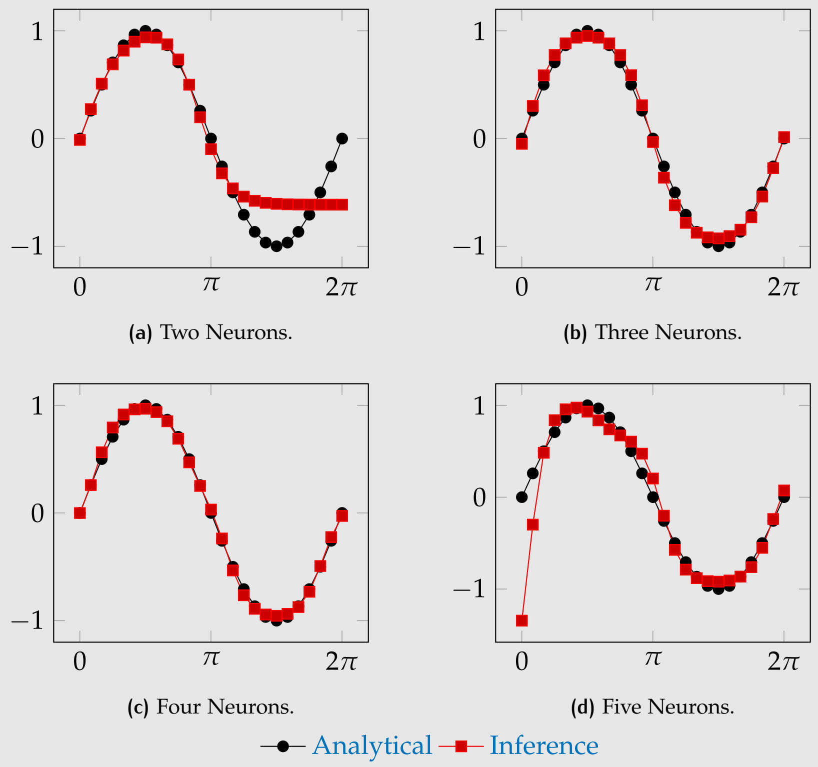
1.6.6 Training Algorithms

This section considers how much the chosen algorithm influences the outcomes of a feedforward backpropagation neural network. The examples 1.6.2.1 and 1.6.4.1 will analyse the effect of the application the quasi-Newton QN and the levemberg-Marquardt LM methods. The details about the methods are presented in section 1.4. The examples disregard the computational time and the RAM necessary to perform the computations, information that may influence the choice of the algorithm.

Example 1.6.6.1 Training algorithms - Number of perceptrons.
The example treats the influence of the training algorithm in the learning of an ANN.

N_s	N_i	N_o	N_h	TA	LF	$\varphi(\cdot)$
25	1	1	-	-	MSE	tanh

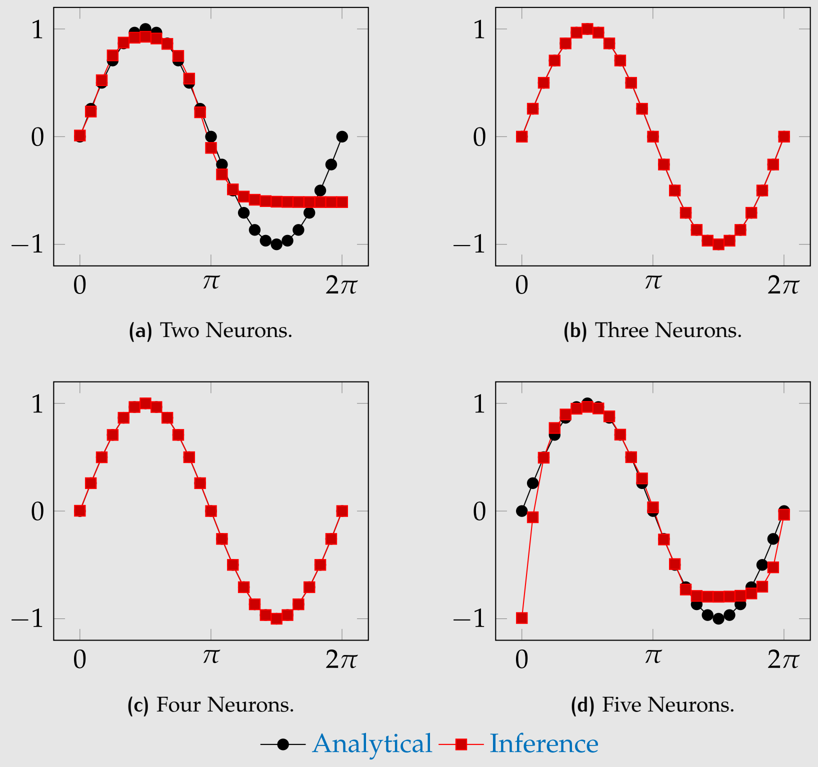
Table 4: ANN parameters.



(a) Two Neurons. (b) Three Neurons.

(c) Four Neurons. (d) Five Neurons.

Figure 16: Inference variation due to the number of hidden neurons - Quasi-Newton algorithm.



(a) Two Neurons. (b) Three Neurons.

(c) Four Neurons. (d) Five Neurons.

Figure 17: Inference variation due to the number of hidden neurons - Lavemberg Marquart algorithm.

It can be seen from figures 16 and 17 that for what concern the case with two neurons 16a and 17a, the response is good only for the half of the domain, even if better than the case with the gradient descent. For three perceptron, the good response with the gradient descent is reflected also with the QN and LM method. This is in accordance with the idea that optimized algorithms should always improve the solution, instead of worsen it. The response with four neurons is particularly interesting, since the solution for both GD and LM is good, while the interpolation fails with the GD. This example shows the effects of adopting optimal algorithms in the interpolation ability of an ANN. In other words, they stabilize the results in terms of number of perceptrons, reducing incertanties and simplifying the design. It should be noted that, regardless of the employed training algorithm, the results become again bad by increasing the number of perceptrons. Concluding, the use of optimal algorithms is crucial to have good responses from ANNs, allowing for solutions really close to the actual ones (see case 17c: it can be considered precise enough as the analytical) and increasing the numbers of suitable network's architecture to be employed.

Example 1.6.6.2 Training algorithms - Random initialization.
The example treats the influence of the random initialization of parameters in the learning of an ANN.

N_s	N_i	N_o	N_h	TA	LF	$\varphi(\cdot)$
25	1	1	-	-	MSE	tanh

Table 5: ANN parameters.

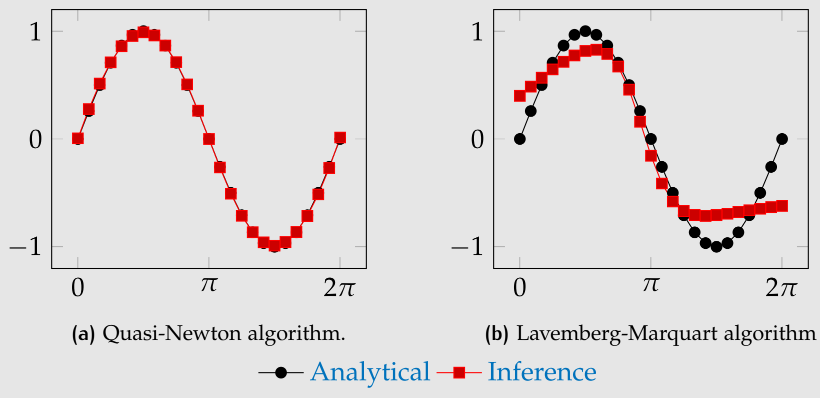


Figure 18: Inference variation due to the training algorithm for different random initialization of parameters.

Picture 18 shows the training algorithm sensitivity to the definition of the random initial parameters.
The result of the GD, visible in figure 14, are not satisfactory: changing to the LM, the solution improves, keeping in any case a certain level of error 18a, while choosing the QN, the solution follows strictly the analytical results, improving considerably the response 18b.
In conclusion, the optimal definition of the random parameters is definitely important, since even an optimal algorithm such as the LM, is not enough to overcome a bad initialization.

1.7 NOISE

The section explores what happens if the approximated function is affected by noise. It is artificially introduced by setting:

$$f(\mathbf{x}) = f(\mathbf{x})\big(1 + \alpha R(\mathbf{x})\big) \tag{43}$$

Where α is a scaling constant and $R(\mathbf{x})$ is a vector of normally distributed random numbers.

Example 1.7.0.1 Noise
The example treats the influence of noise in the response of an ANN.

N_s	N_i	N_o	N_h	α	TA	LF	$\varphi(\cdot)$
50	1	1	-	15%	LM	MSE	tanh

Table 6: ANN parameters.

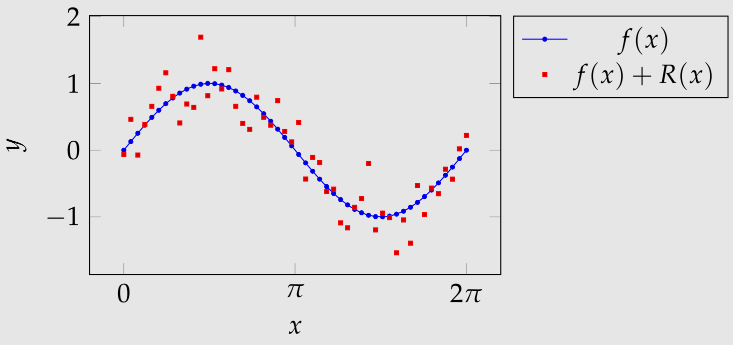


Figure 19: Training set.

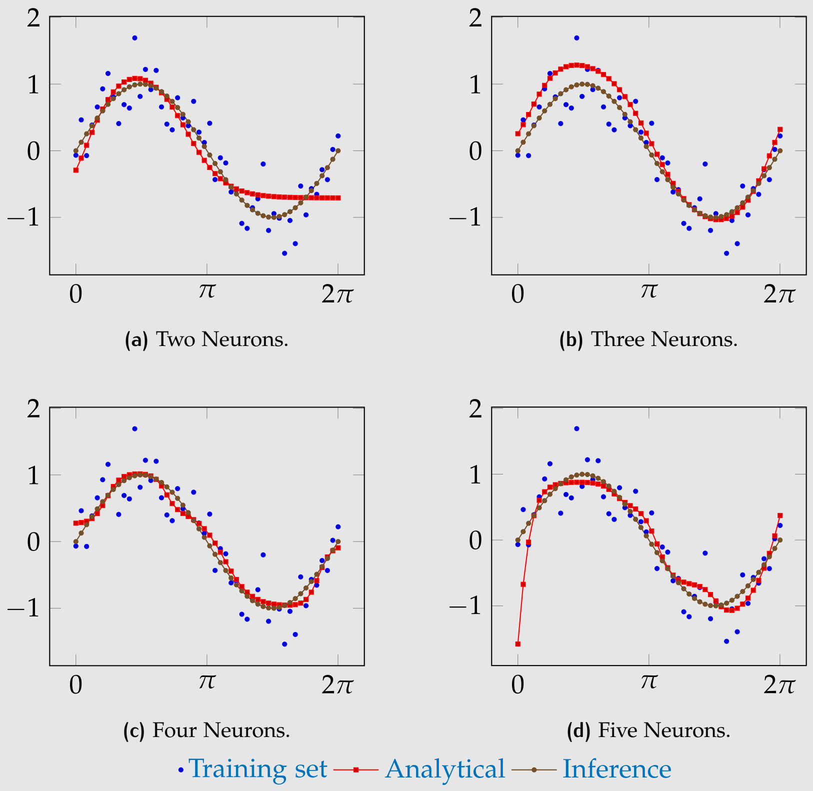


Figure 20: Inference variation due to the number of hidden neurons.

2 RECURRENT NEURAL NETWORKS

Recurrent neural networks or RNN are part of the neural network architecture family in which *feedback* is involved. Whereas a dynamic system is taken into consideration, the current response of the system depends on the previous behavior of the system. This may happens for instance for speech recognition: the dynamic system is made up by the sound flow and the interpretation of words depends on the current and previous captured letters.

An heavy constraint of the feedforward backpropagation neural network is the range of prediction domain, limited by the extremes of the training sets. In that case, the normalization affects the interpolation capability of the neural network: as a matter of fact, for values over the training maxima and minima, the response of all the perceptrons in the hidden layer would fall on the saturated trench of its sigmoid, if hyperbolic tangent are defined as activation function.

This problem is overcomed by RNN including feedbacks: the output at each time step is influenced by the previous, making a direct connection between past and current values. This allows to predict in *future* beyond the extremes of the training set, imposing a closed circle or recurrency between initial values and current output.

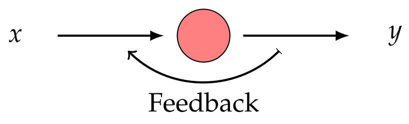


Figure 21: Feedback for RNN.

In the function approximation field, the dynamic term defined for RNN can be directly compared with the dynamics interpreted from a physical point of view, that is the variation of a system in time. Considering for instance a generic function f:

$$f = f(t) \quad , t \in \Omega = [0, +\infty) \tag{44}$$

The current value $f(t_n)$ at the time instant t_n can be evaluated suitably interpolating the previous values of f:

$$f(\mathbf{P}) : \mathbf{V} \to \mathbb{R} \tag{45}$$

With:

$$\mathbf{P} = \Big\{ f(t_{n-1}), f(t_{n-2}), \dots, f(t_{n-p}) \Big\} \quad , |\mathbf{P}| = p \tag{46}$$

$$\mathbf{V} = \Big\{ f(t_1), f(t_2), \dots, f(t_{n-1}) \Big\} \quad , |\mathbf{V}| = n - 1 \tag{47}$$

The dimension p of the domain $\mathbf{P}$ represents the number of previous outputs to be used for the current prediction, called *delays*. Increasing the number of delays increases the information given to the neural network, and the tuning of it highly depends on the function f: a small number would give not enough information for correctly access the next step, while the RNN can misunderstand the data for too many values.

For further information [see 7, chapter 15]

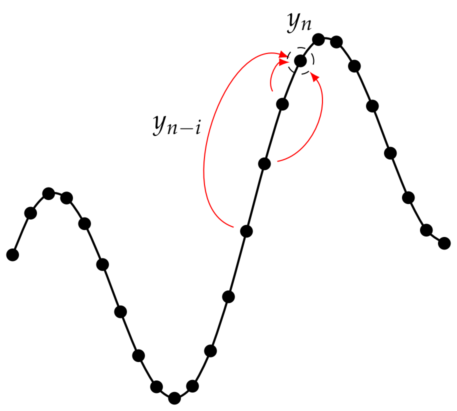


Figure 22: RNN for function approximation

2.1 NON-LINEAR AUTOREGRESSIVE NEURAL NETWORKS

Non-linear autoregressive neural networks or NAR networks[1] are based on an architecture directly derived from the feedforward backpropagation ANN introduced in chapter 1.

The skeleton, made of input, hidden and output layers, is shared: the recurrency is set by connecting the output layer and the input one; at each cycle, the neural network computes the n-th value starting from the vector of delays.

Then, the architecture is made out of a number of input nodes equal to the number of delays and a single output, while the hidden layer size is a chosen parameter.

Picture 23 shows the concept referring to the notation of section 2, by considering three delays and two perceptrons in the hidden layer.

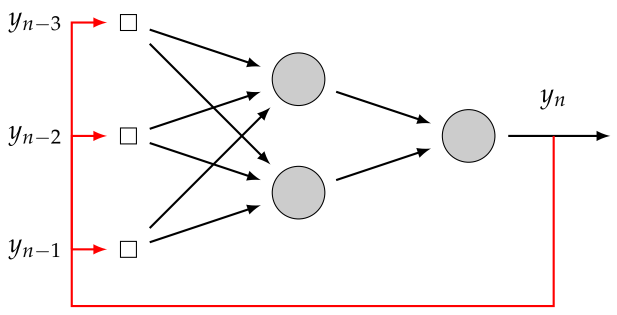


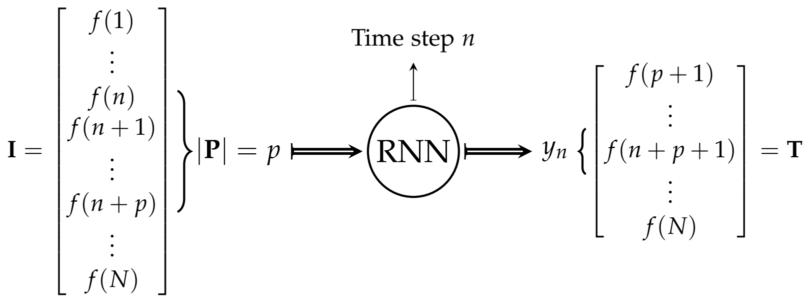
Figure 23: NAR network for function approximation.

The data input vector **I** for the training process contains the values of the function f:

$$\mathbf{I} = \begin{bmatrix} f(1) \\ f(2) \\ \vdots \\ f(i) \\ \vdots \\ f(N) \end{bmatrix} \tag{48}$$

Those values are taken in subvectors of dimension p, depending on the chosen number of delays. At each step, the output is the next element of the input vector at position $(n + p + 1)$:

1 see 7, p. 791.



This means that the input and the target vectors during training are equal, aside for the initial part corresponding to the number of delays, which are used to start the recurrent process. The backpropagation of the error is slightly different for RNNs, since the updating process runs over each time step, and it is called backpropagation through time or BPTT.

2.2 NON-LINEAR AUTOREGRESSIVE WITH EXOGENOUS INPUTS NEURAL NETWORKS

Non-Linear Autoregressive With Exogenous Inputs Neural Networks or NARX networks[2] are an extension of the NAR network including external inputs in the feed-forward network, at each time step.

This architecture allows to include additional information on the prediction from another time function which does not directly depends on the primary one. An example could be the prediction of the temperature in time of a room including the humidity variation, known from the beginning of the analysis or real-time acquired independently from other systems, like sensors.

The architecture is similar to the NAR one, and it includes dedicated input nodes for the external inputs. The concept here is the same as the feedback for the function f, and the number of input delays is decided on the base of external function and it is not correlated with the previuosly defined number of feedback delays. Figure 24 depicts the same generic neural network introduced in section 2.1 including two input feedbacks.

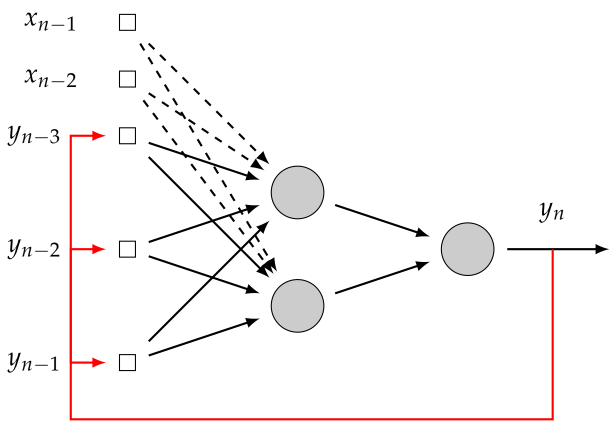


Figure 24: NARX network for function approximation.

2 see 7, p. 791.

The mathematical model is the same, and it is here reported to highlight how two different sets of weights are managed. The external input time series is indicated with the letter x. The output depends on the input and feedback delays as:

$$y = y(x(t), x(t-1), \dots | y(t-1), y(t-2), \dots) \tag{49}$$

In the hidden layer:

$$\begin{cases} v_k = \left(\sum_{i=1}^{n} \omega_{ki}^{x} x_i + \sum_{j=1}^{m} \omega_{kj}^{y} y_j + b_k \right) \\ y_k = \varphi(v_k) \end{cases} \tag{50}$$

Where:

x_i Inputs.

y_j Feedbacks.

ω_{ki}^{x} Input weights.

ω_{kj}^{y} Feedback weights.

b_k Bias.

n Number of external inputs.

m Number of delays.

2.3 BPTT - BACK-PROPAGATION THROUGH TIME

The backpropagation through time algorithm BPTT for training a recurrent network is an extension of the standard back propagation algorithm. It may be derived by unfolding the temporal operation of the network into a layered feedforward network, similarly of what depicted in figure 25.

For detailed information of the actual algorithm [see 7, p. 808].

2.4 OPEN AND CLOSED FORM

The supervised training of any neural networks is based, as already mentioned, upon couples of known input-output vectors. The same applies for RNN: once the number of delays is set, the loss function and weights and biases are updated at each time step, as indicated in section 23.

In the case of NAR networks, the input and target vector must be equal by definition. The target vector can be built-up a priori of the training by using the same values of the input one or built up in running using the output of the RNN. The first case is referred as training in open form, while the second in closed form. In the ideal case, they should be the same, but that does not always happen, because of the propagation of error of which the output of the network is affected. In any case, the RNN is able to make prediction only in closed form, when there is direct connection between input and output: after the training in open form, the network is converted in closed one imposing the recurrency.

This means a RNN trained in open form generally performs better than the closed one in the training range of values, but it may fail in the prediction since it has never considered the direct connection of the recurrency. A possible approach is to perform the training in open form first, and then repeat it in closed form not starting from scratch, but using the parameters, i.e weights and biases, computed in the first step. This procedure set the starting point to values closer to convergence

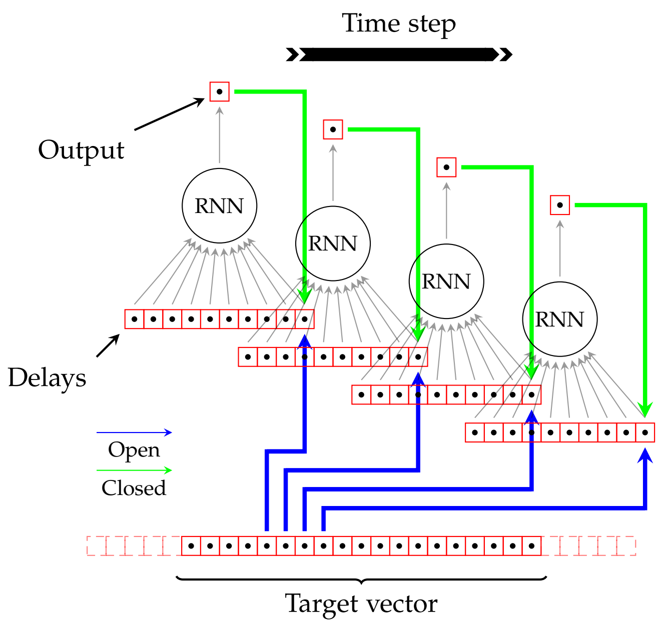


Figure 25: Training of RNN: open vs. closed form.

of the solution, for instance making possible to avoid early stops by the checking systems preventing overfitting.

Picture 25 depicts the concept.

2.5 DESIGN OF RECURRENT NEURAL NETWORKS

The section explores the design of RNNs similarly of what already presented for feedforward backpropagation networks. It is tested the sensitivity of the solution with respect to the number of perceptrons and the data distribution, including the additional parameters specific for this type of networks. In this analysis, the conclusions from the results obtained in chapter 1 about the influences of training algorithms and the random initialization of parameters are accepted for valid. This means the setting of the Lavemberg-Marquart algorithm for the training and the Nguyen-Widrow algorithm for the initialization of parameters.

2.5.1 Similarities With Feedforward Neural Network

The architecture of NAR and NARX networks are directly derived from the one of feedforward backpropagation networks, and they share some characteristics. Number of perceptrons, number of layers, activation function, training algorithms and initialization of parameters have the same definition, and the same dissertation is here still valid.

For instance the number of layers can be chosen arbitrarily between one to a greater or smaller number, but a reasonable choice is to built up the architecture as simple as possible: the definition of a sigmoid as activation function ensures that a single layer, providing a sufficient number of perceptrons, is enough to correctly approximate any function (See section 1.6.4 and 1.6.1 in chapter 1).

The problem of overfitting of data is still a critical element to be considered in the definition of the architecture: the type of training algorithm and a correct initialization of parameters mitigate its influences, but they may not totally.

Some examples are further presented showing the sensitivity of RNNs to those parameters. A small preview of examples 2.5.1.1 and 2.5.1.2: the training set must be properly selected, depending on the time series function. As matter of fact, for proper predictions the periodicity have to be seen by the network.

Example 2.5.1.1 sensitivity to number of perceptrons and delays
The example treats the influence of the number of perceptrons and delays in the response of a RNN.
The time series is the dynamics solution of a damped SDOF oscillator with linear stiffness in free motion, depicted in figure ??. F is a static force keeping the system in equilibrium. Removing the force F, the system starts to oscillates, until the new equilibrated configuration is reached.

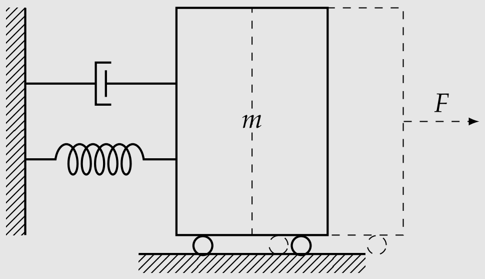


Figure 26: SDOF system.

N_d	N_h	t_f [s]	S	$T.A.$	$L.F.$	$\varphi(\cdot)$
-	-	0.5	0.005	L.M.	MSE	tanh

Table 7: RNN parameters.

The initial ANN configuration includes the minimum number of delays and perceptrons. The idea is to identify them starting from the ground, gradually enhancing the solution. Figure 27 shows the training set used for the first RNN. Its definition does not include on purpose the periodicity of the dynamic response, to observe that the neural networ is able to predict.
Each prediction is complement with the open and close form solution for the training. When the close solution is far from the open one, no correct prediction must be expected, according to what already said in section 2.4.
Figures 28b, 29b, 30b and 31b shows the results varying the number of perceptron from 2 to 4 and a number of delays equal to 2 and 3. No ANNs is able to outcome acceptable results, both in term of open vs. close and for predictions. The motivation lies on the training set, too poor of data for a good training.

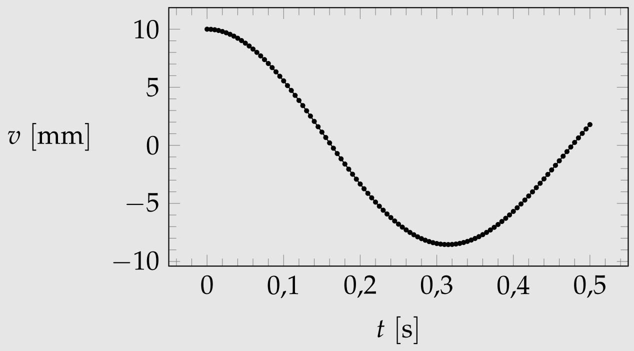
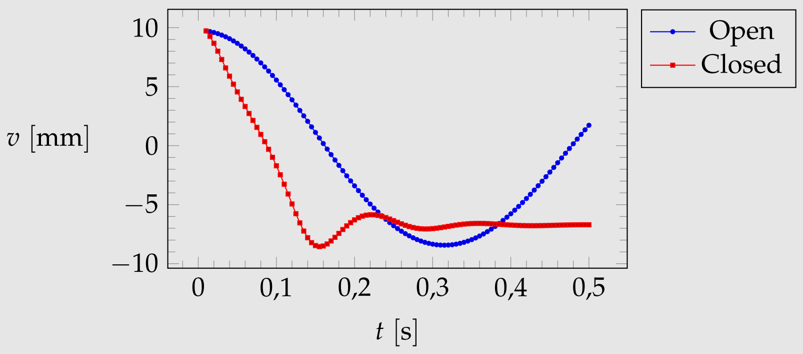
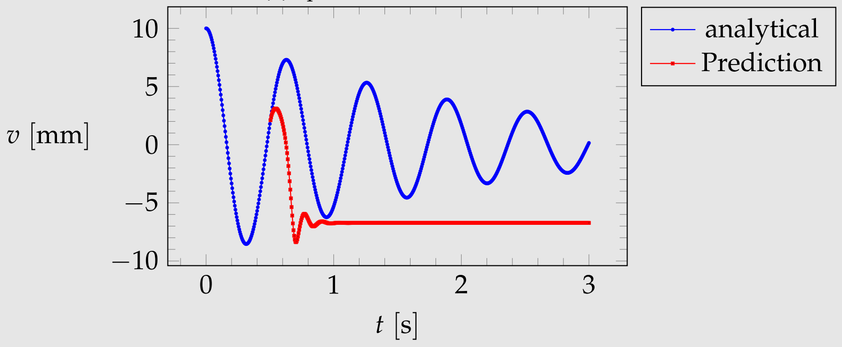


Figure 27: Training set.

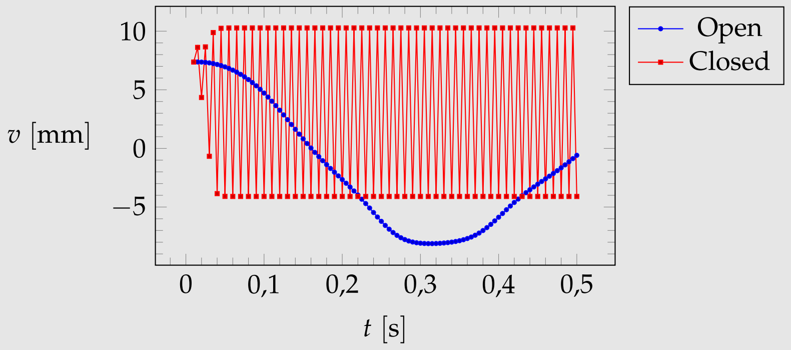


(a) open vs. closed solution.

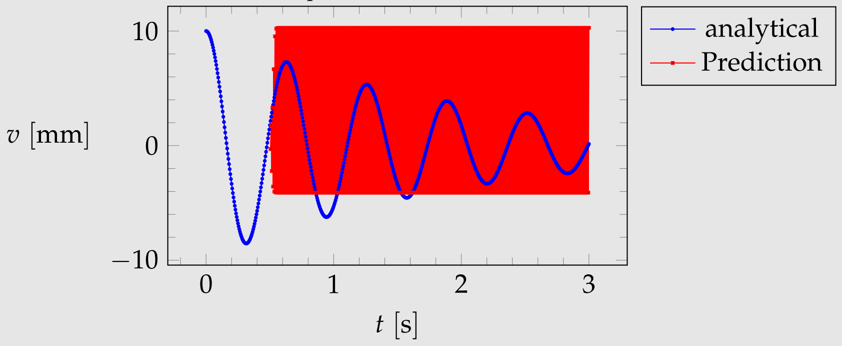


(b) Analytical vs. predicted solution.

Figure 28: Solution for 2 perceptrons and 2 delays.

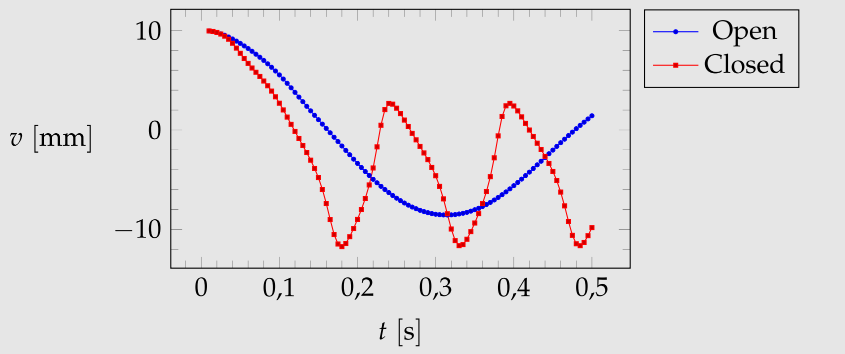


(a) open vs. closed solution.

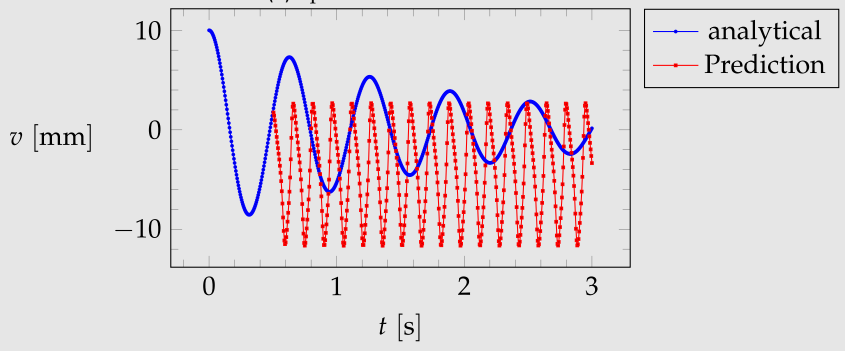


(b) Analytical vs. predicted solution.

Figure 29: Solution for 3 perceptrons and 2 delays.

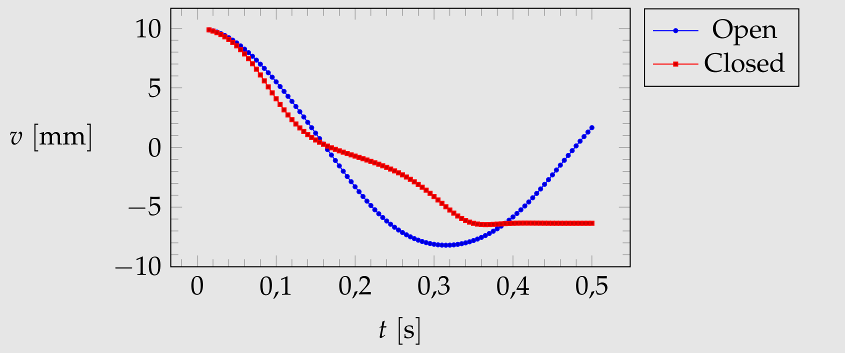


(a) open vs. closed solution.

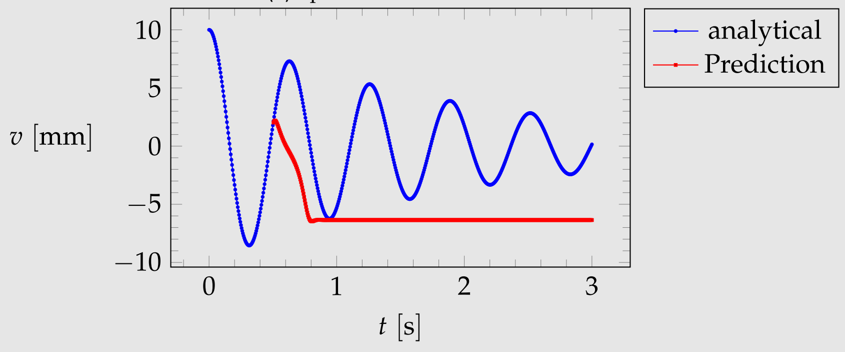


(b) Analytical vs. predicted solution.

Figure 30: Solution for 4 perceptrons and 2 delays.



(a) open vs. closed solution.



(b) Analytical vs. predicted solution.

Figure 31: Solution for 3 perceptrons and 3 delays.

Example 2.5.1.2 sensitivity to the training set data
The example 2.5.1.1 shows that the RNN is not able to reproduce and predict the

time series if the periodicity is not included in the training set. The prediction capability of the network will be tested for a new time interval length t_f.

N_d	N_h	t_f [s]	S	$T.A.$	$L.F.$	$\varphi(\cdot)$
-	-	-	0.005	L.M.	MSE	tanh

Table 8: RNN parameters.

As previously mentioned, the time interval is extended to 1 s. The ANN is able to predict acceptable values for the case of 2 perceptrons and delays, i.e the minimum values (figure 33). Increasing the complexy of the architecture, the response becomes worse, similar to what happens for the ANNs defined in examples of chapter 1. Increasing the training set increases the capability of the RNN to capture the trend of the function and make predictions.
Then, it is spontaneous wondering what happens for larger training sets. Figure 38b, 39b, 40b and 41b shows that the solution is worse: this is in accordance with the fact that the more information are fed to a neural networks, the more complex it must be to handle positively the problem.
Finally, it is worth noting that good results in term of open-closed form do not correspond to good predictions, as shown by figures 35a and 34a. The same happens in figure 40, where the predictions are acceptable but not good as the ones of figure 33.

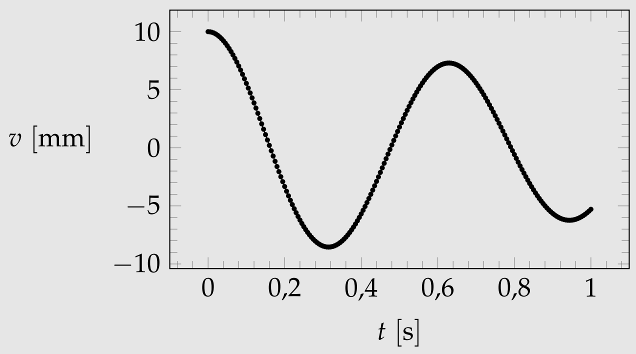
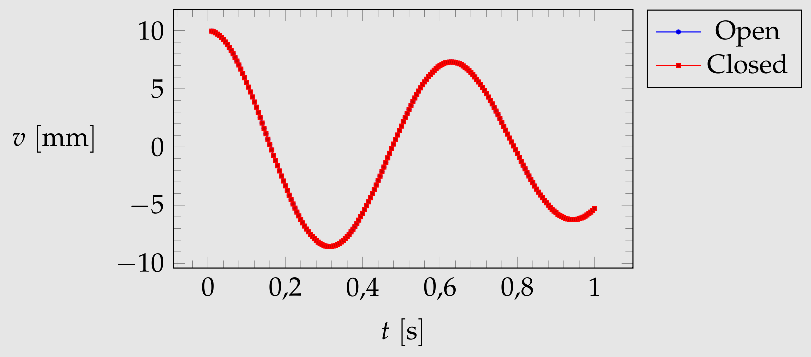
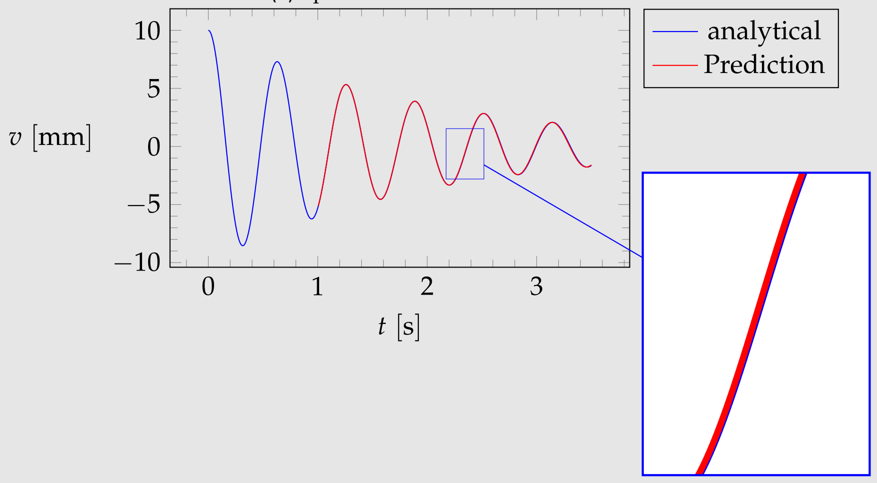


Figure 32: Training set for $t \in [0:1]$ [s]

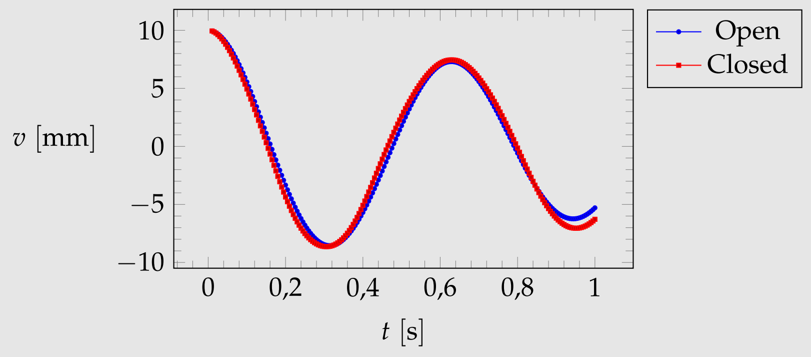


(a) open vs. closed solution.

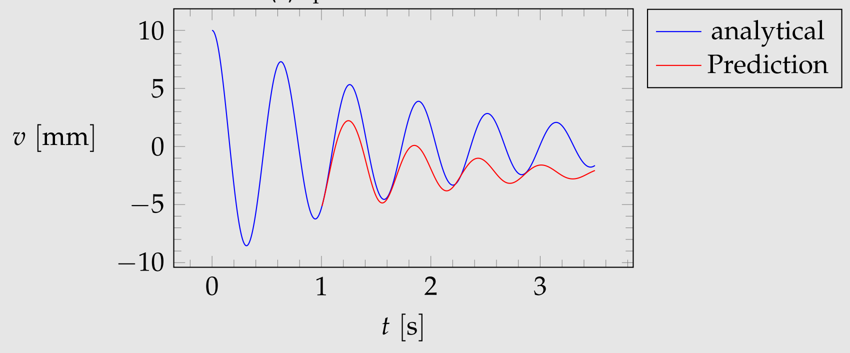


(b) Analytical vs. predicted solution.

Figure 33: Solution for 2 perceptrons and 2 delays.

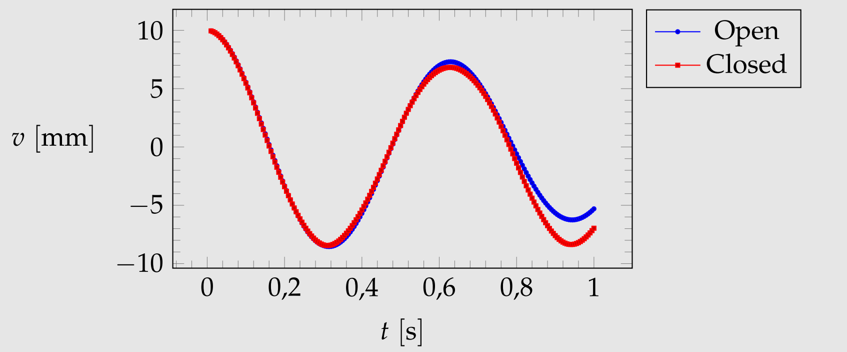


(a) open vs. closed solution.

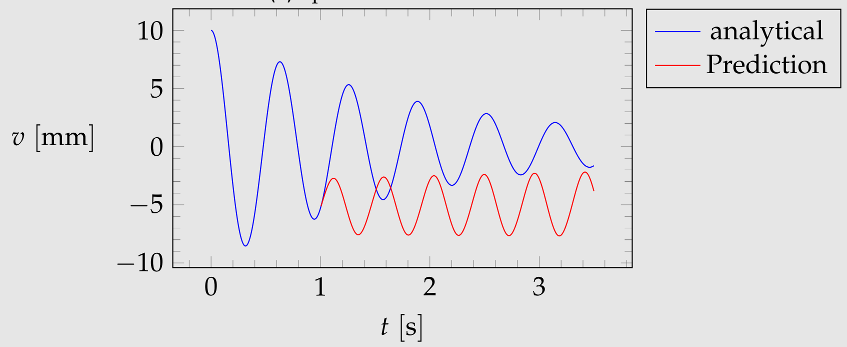


(b) Analytical vs. predicted solution.

Figure 34: Solution for 3 perceptrons and 2 delays.

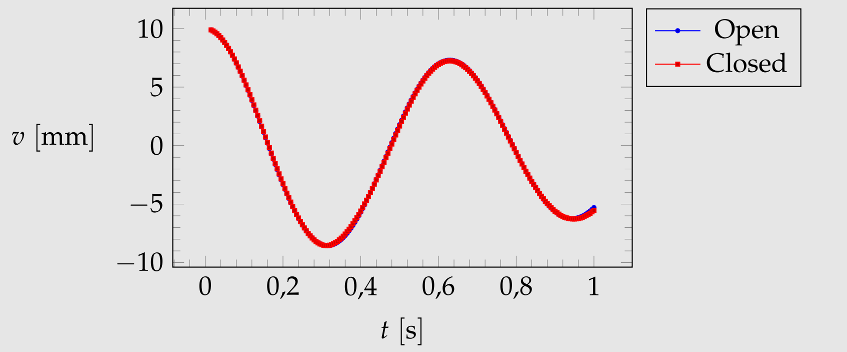


(a) open vs. closed solution.

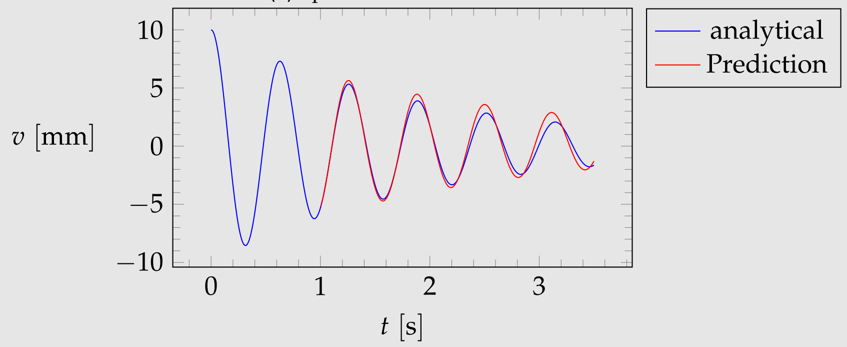


(b) Analytical vs. predicted solution.

Figure 35: Solution for 4 perceptrons and 2 delays.



(a) open vs. closed solution.



(b) Analytical vs. predicted solution.

Figure 36: Solution for 3 perceptrons and 3 delays.

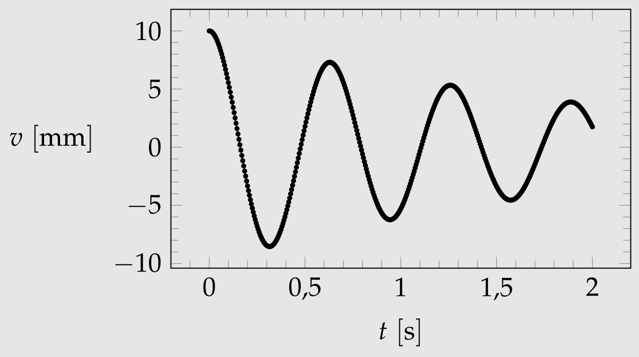
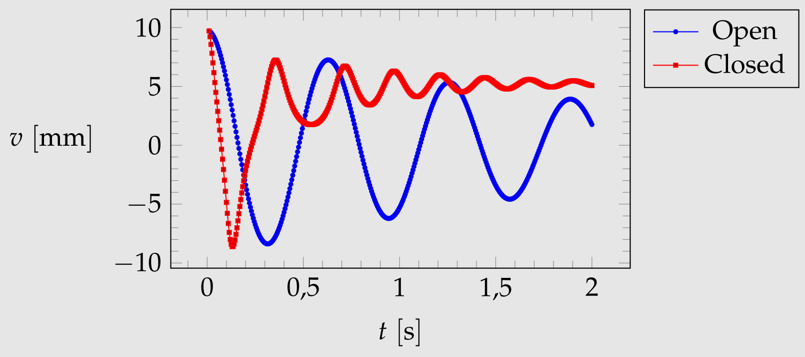
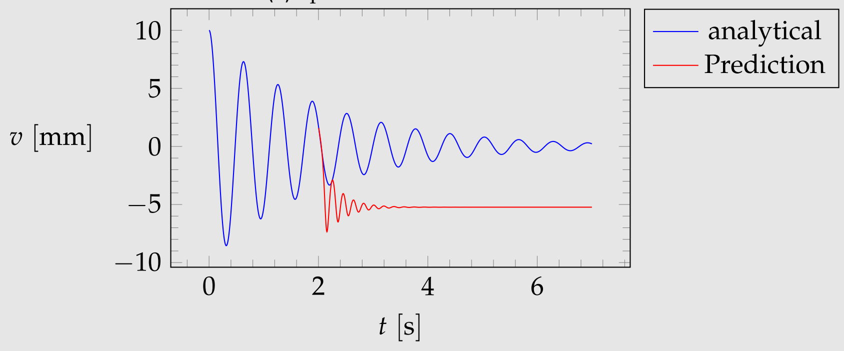


Figure 37: Training set for $t \in [0:2]$ [s]

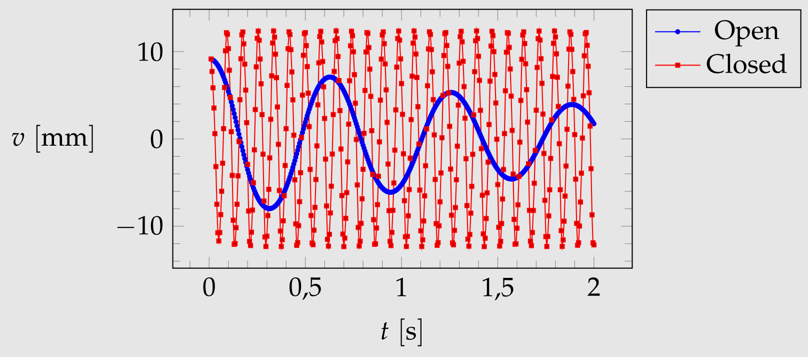


(a) open vs. closed solution.

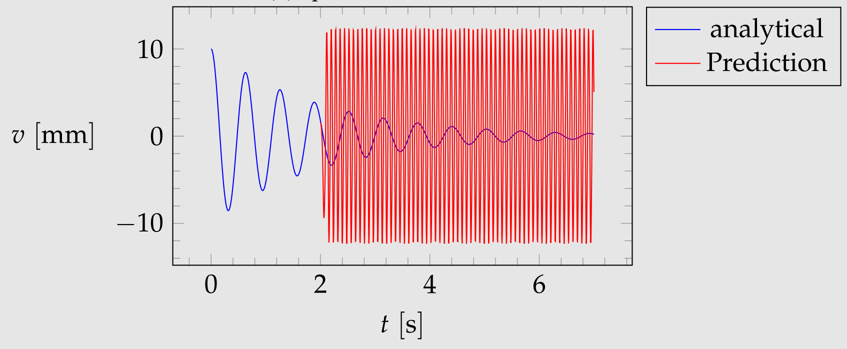


(b) Analytical vs. predicted solution.

Figure 38: Solution for 2 perceptrons and 2 delays.

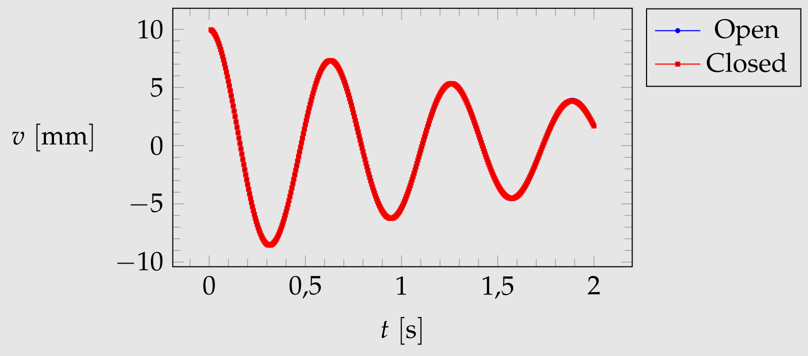


(a) open vs. closed solution.

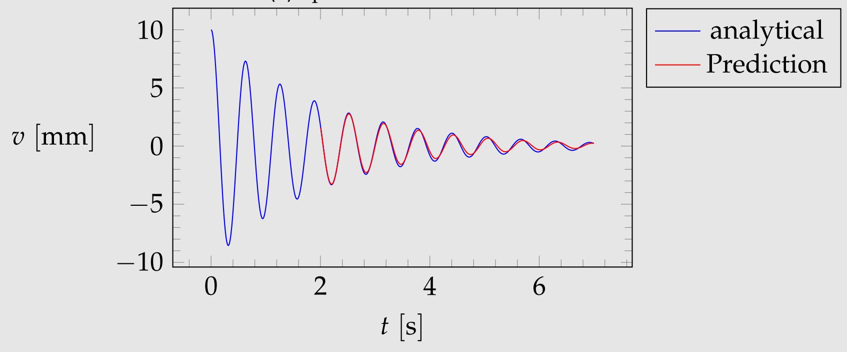


(b) Analytical vs. predicted solution.

Figure 39: Solution for 3 perceptrons and 2 delays.

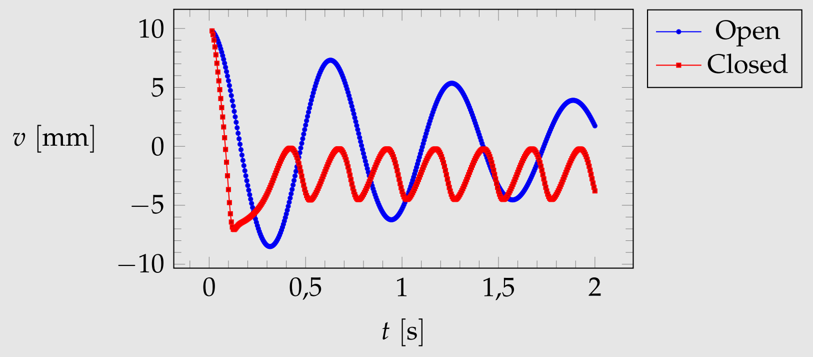


(a) open vs. closed solution.

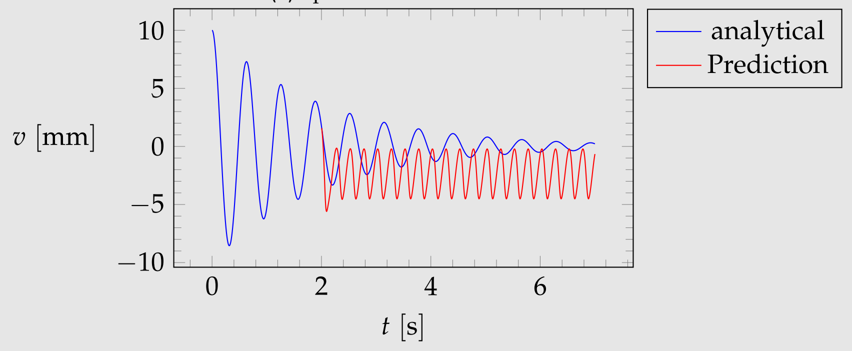


(b) Analytical vs. predicted solution.

Figure 40: Solution for 4 perceptrons and 2 delays.



(a) open vs. closed solution.



(b) Analytical vs. predicted solution.

Figure 41: Solution for 3 perceptrons and 3 delays.

2.5.2 Cross And Auto Correlation Function

The main feature of RNNs is the ability to acquire information from previous states of system. The optimal number depends on the function: smooth curves would need few delays to be correctly predicted, being small the variation between consecutive points while for convoluted trends more data should be necessary. In both cases, it is important to avoid the inclusion of repetitions, i.e. giving useless information already feed to the network.

An effective way is to evaluate the periodicity of the function: it can be done computing the auto-correlation for what concern the NAR network and the auto-correlation within the cross-correlation for the exogenous inputs for the NARX one[3].

Considering a function $f(t)$, the auto-correlation function $R_f(\tau)$ is defined as:

$$R_f(\tau) = \lim_{T\to\infty} \frac{1}{T} \int_{-T/2}^{T/2} f(t)f(t+\tau)dt \tag{51}$$

Where τ is the time delay and T the period. It measures the sum between the product of a function in a fixed time τ and the function itself over $t \in [a,b]$. R_f oscillates reaching the local maxima when τ is in correspondence of the maximum sum. Applying (51) to a sin function:

$$R_f(\tau) = \lim_{T\to\infty} \frac{1}{T} \int_{-T/2}^{T/2} sin(t)sin(t+\tau)dt \quad ,t \in [0,2\pi] \tag{52}$$

The time axis can be discretized in N_s elements depending on the time step S. The τ domain can be defined in function of the latter definition by setting the number of steps N_l, also called *lags*. Finally, the auto-correlation function is evaluated and plotted. Considering:

$$\Big\{ N_s = N_l = 100 \;\rightarrow\; S = 0.0635 \tag{53}$$

3 see 10, p. 408.

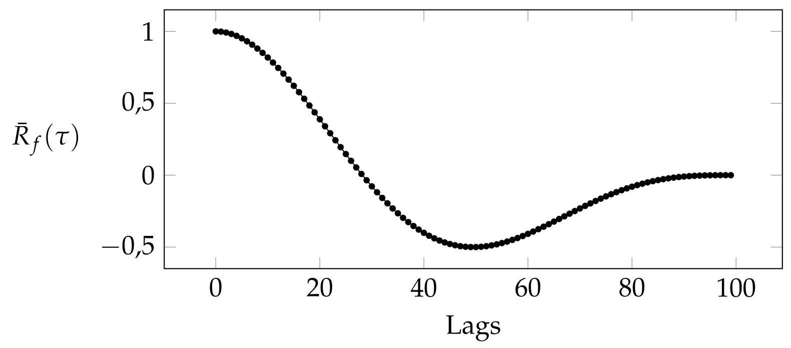


Figure 42: Normalized autocorrelation function $\bar{R}_f(\tau)$

The maximum value of $R_f(\tau)$ is at zero lags, when the two sine functions are perfectly overlapped at $\tau = 0$, corresponding to $N_l \simeq 50$. Multiplying the latter quantity for the time step, one finds:

$$N_l \cdot S \simeq 3.17 = \frac{T_f}{2} \tag{54}$$

Where T_f is the period of the sine function, equal to:

$$T_f = 2\pi \tag{55}$$

This means that for number of lags greater that this value the influence of additional time steps decreases. Picture 42 shows the results. In conclusion, the number of delays should be chosen close to the one of lags that maximises the autocorrelation function.

The same holds for the cross-correlation function:

$$R_{fx}(\tau) = \lim_{T\to\infty} \frac{1}{T} \int_{-T/2}^{T/2} f(t)x(t+\tau)dt \tag{56}$$

Where $x(t)$ is the exogenous input function. Again, the number of chosen lags should maximize the function $R_{fx}(\tau)$.

2.6 TRAINING SET

The introduction of delays and recurrency brings some effects on the definition of the set of data in the learning phase. For feedforward backpropagation neural network, the set of labeled data are subdivided randomly in training, validation and generalization by the algorithm, to reduce the possible overfitting.

The basic assumptions introduced in equation (45) assumes the previous outputs must be fed to the RNN in a precise sequence, keeping constant the time step. This is a crucial point, since weights are trained accordingly to that series.

Considering a random generation, the time step would never be constant and the periodicity of the function not detectable. Then, it is preferable to define the learning set in a sequential way, even if the possibilities that the network overfits data increases. A possibile solution is to pre-treat input.

Besides, the sequential definition of data wells up another problem about the pre-treating of inputs. The function generated from a complex problems, such as structural dynamic systems, inherits some characteristics. For instance, numerical solver, such as the Newmark algorithm, are stable, i.e they ensures convergence, only if the sampling step is *sufficiently* small; it is commonly necessary to use a time step in the order of $1 \times 10^{-6} \div 1 \times 10^{-7}$.

The higher number of steps has a counter part for the definition of the optimal number of delays, since it would be necessary a too large number of lags to exploit the periodicity of the function. For the case 42, increasing N_s in the order of 1×10^3, leads to a N_d equal to $50 \cdot 1 \times 10^3 = 50000$, which is not compatible with the type of problem RNNs are defined for. Instead, pre-treating data, sparsing the input values, would have the double benefits of lightening the network, without invalidate the prediction response, and mitigating overfitting.

3 WORKED EXAMPLES

3.1 STATIC BEAM DEFLECTION

The function approximation problem is suitable for a parametric analysis. Whereas a linear or non linear relation within multiple parameters has to be analyzed, an artificial neural network can be trained to store the solution. The possible reasons are the necessity to access data in a real time fashion, useful for non-linear relations, being the response of a trained ANN faster than any iterative algorithm, or to built up graphical or a digital abaqus useful for initial design.

An example regards a parametric analysis of the cantilever beam depicted in figure 43. The function to be approximated is the relation giving the vertical displacement v_B at the free edge within materials and geometric properties.

$$v_B = \frac{1}{3}\frac{Fl^3}{EI} \tag{57}$$

Adopting the same notation of section 1.5:

$$\begin{cases} d = v_B(\xi = \xi_B) = f(\mathbf{x}) \\ \mathbf{x} = \{F; l; E; b; h\} \end{cases} \tag{58}$$

With:

$$f : \mathbf{D} = \mathbb{R}^5 \to \mathbf{I} = \mathbb{R} \tag{59}$$

The set of labeled data $\boldsymbol{\tau}$ is then:

$$\boldsymbol{\tau} = \left\{ (\mathbf{x}_i, v_{Bi}) \right\}_{i=1}^{N} \tag{60}$$

Where N is the number of training data. The ANN is trained for different N to observe the sensitivity of the solution, similarly to what presented in example 1.6.5. The training set is built up considering fixed ranges of values and different steps defined by a number of subdivisions N_d

$$N_d = 2n \quad ,\text{for } n = 1, \ldots, 8 \tag{61}$$

Table 9 collects the domain of the parameters.

The number of training elements NT is chosen to range from a small value, on purpose not sufficient for a good training, to a suitable one. In details, 32 for the poorest one, while 1 048 576 for the richest one.

The number of perceptron NP is chosen to range from 4 to 15.

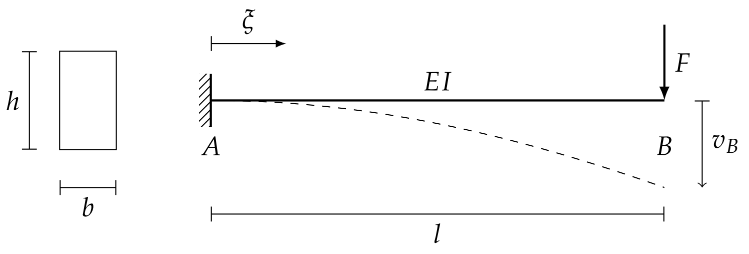


Figure 43: Clamped beam subjected to a fixed load

-	Min	Max	Unit
F	1	10	kN
l	0,5	2	m
E	28	33	GPa
a	20	40	cm
b	20	60	cm

Table 9: Input training set data - Clamped beam - Distributed data.

NP\NT	32	1024	7776	32 768	100 000	248 832	537 824	1 048 576
4	0,85	0,16	0,24	0,65	0,66	0,97	0,97	1,83
5	0,58	0,05	0,13	0,24	0,39	0,59	0,59	1,25
6	0,51	0,05	0,09	0,19	0,32	0,79	0,79	0,94
7	0,43	0,05	0,10	0,14	0,23	0,57	0,57	0,64
8	0,80	0,07	0,07	0,25	0,29	0,30	0,30	0,54
9	1,53	0,04	0,07	0,10	0,20	0,30	0,30	0,50
10	0,45	0,03	0,07	0,15	0,17	0,19	0,19	0,34
11	1,49	0,11	0,06	0,08	0,16	0,19	0,19	0,45
12	0,86	0,05	0,03	0,05	0,11	0,16	0,16	0,28
13	0,53	0,02	0,05	0,07	0,13	0,20	0,20	0,31
14	1,51	0,06	0,02	0,06	0,18	0,09	0,09	0,18
15	1,23	0,04	0,02	0,06	0,09	0,15	0,15	0,27

Table 10: Euclidian norm of the error vs. number of perceptrons and training data - Distributed training data.

The error is computed inquiring the ANN with the same values of the training set. Hence, the error does not measures the generalization capacity of the neural network.

The error e_2 is evaluated considering the Euclidian norm of the error vector $\mathbf{e}$, defined as the difference between the analytical and the exact solution, normalized by the maximum displacement:

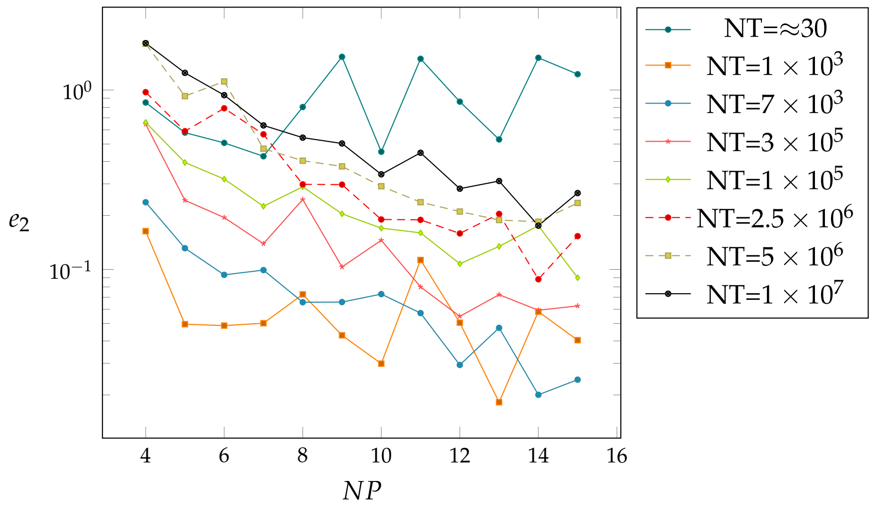
$$e_2 = \|\mathbf{e}\| = \left\| \frac{\mathbf{v}_{ana} - \mathbf{v}_{ANN}}{v_{max}} \right\| \tag{62}$$

Table 10 collects the errors e_2. It is not simple to guess the possible trends of the errors because the optimal definition of parameters of an ANN must be evaluated cosidering simultaneously NP and NT. Increasing the elements of the training set is useless, or it may have negative effects, if the number of perceptrons is not increased accordingly; but in the same time it increases also the possibility of having overfitting. The results show a consistent reduction of the error from 32 to 1024,7776 and 32 768 followed by a gain in the order or 25 %÷30 % (Figure 44a). The increase must be seen from a global point of view: indeed, focusing only on 100 000,248 832,537 824 and 1 048 576 the error keep decreasing, suggesting that a number of perceptrons higher the considered maximum 15 would lead to an improvement of the solution (Figure 44b).

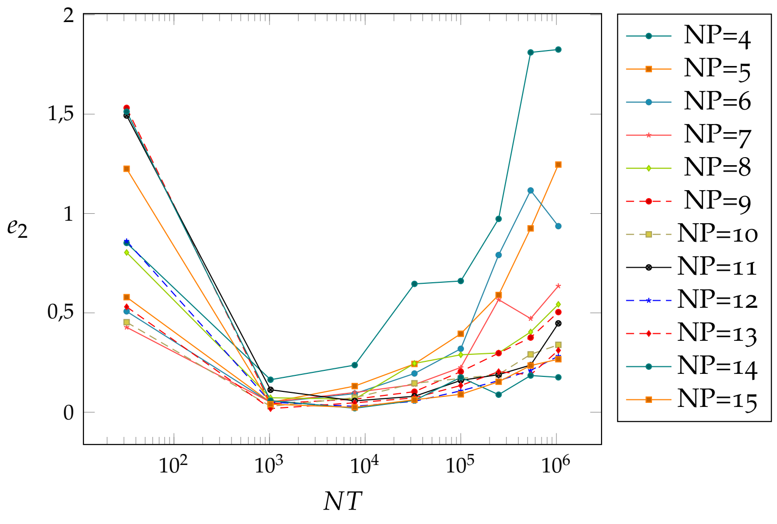
A second set of results is presented by changing the training dataset, defined using the same range of values of table 9 but keeping a fixed step; see table 11. Then, data sets are generated by picking up random combinations of values. In this case, it is possible to compute the number of combinations N_C by multiplying the number of possible values N_i assumed by each parameter:

$$N_C = \Pi_{i=1}^{5} N_i = 70960176 \tag{63}$$

The number of training elements is chosen proportionally to the total number N_C: 100 elements for 0.000 14 % of N_C and 1 000 000 for 1.4 % of N_C.



(a) e_2 vs. NP.



(b) e_2 vs. NT.

Figure 44: e_2 error trends - Distributed training data.

-	Min	Max	Step	Unit	Elements
F	1	0,1	10	kN	101
l	0,5		0,1	m	16
E	28	0,1	33	GPa	51
a	20	1	40	cm	21
b	20	1	60	cm	41

Table 11: Input training set data - Clamped beam - Distributed data.

NP\NT	100	1000	10000	100000	1000000
4	0,86	0,55	0,29	0,40	0,40
5	8,91	0,36	0,20	0,25	0,24
6	2,19	0,34	0,16	0,51	0,21
7	1,95	0,25	0,18	0,10	0,10
8	1,73	0,10	0,16	0,24	0,13
9	1,51	0,70	0,08	0,09	0,07
10	1,26	0,26	0,07	0,13	0,06
11	1,82	0,25	0,06	0,08	0,08
12	4,20	0,19	0,06	0,05	0,06
13	0,95	0,17	0,03	0,05	0,05
14	1,29	0,24	0,06	0,05	0,05
15	2,63	0,29	0,02	0,06	0,05

Table 12: Error vs. number of perceptrons and training data - Random TS.

The error is computed according to (62) inquiring the ANN for values picked up with the same reasons of the training sets. Hence, they could belong or not to the training set, and the error measures also the generalization capacity of the ANN. In particular, 10 elements for each parameters are chosen, leading to a number of combinations up to 100 000.

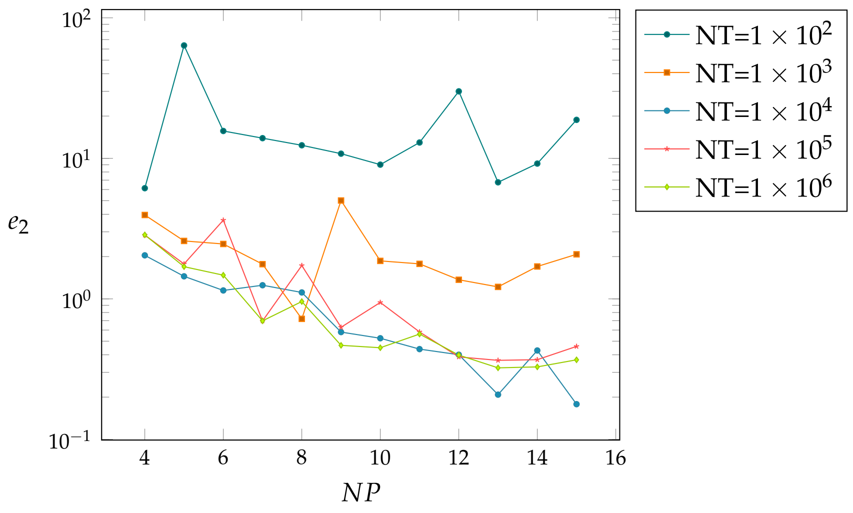
Table 12 collects the errors e_2 for the second training data definition. In this case the results agree with the expectation: the order of magnitude is similar first, indicating the goodness of the ANN architecture; the error trends decrease with respect both NT and NP, showing also the generalization capability of the neural network to interpolate data.

For both cases, a small number of training data is not sufficient to ensure a good training. This is in accordance to the theoretical concept introduced in chapter 1.

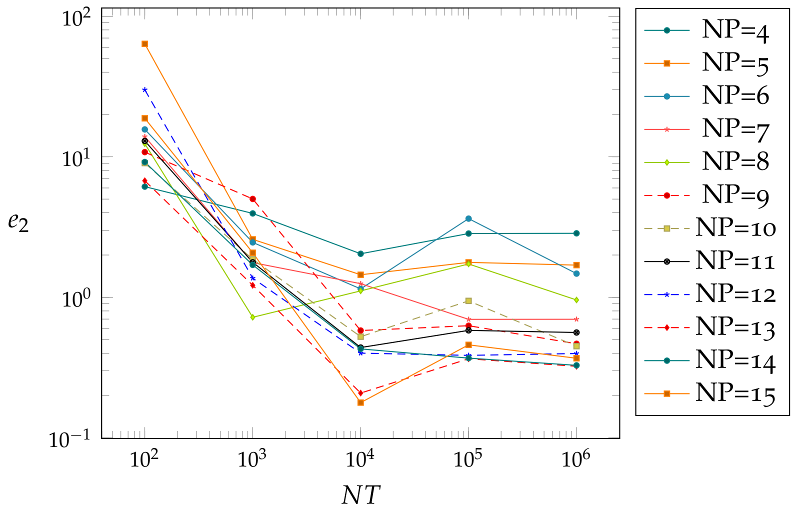
Concluding, picture **??** shows the displacement variation with respect the parameters of table 9, for an ANN built up with the largest number of perceptrons and training set, keeping mutually constant $P = 5$ kN, $l = 1$ m, $E = 30$ MPa, $b = 25$ cm and $h = 50$ cm.

As reference, the training of the two sets of ANN takes around 10 h, trained on a PC with the following specifications:

- Intel® i5200@2.2 GHz.
- 4 GB DDR3 RAM.

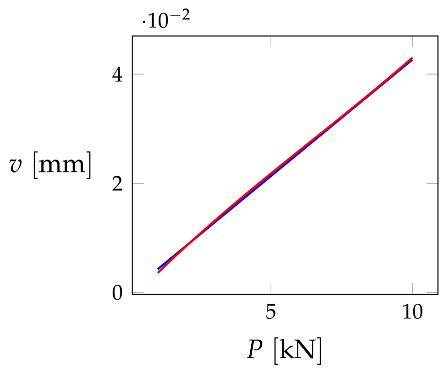


(a) e_2 vs. NP.

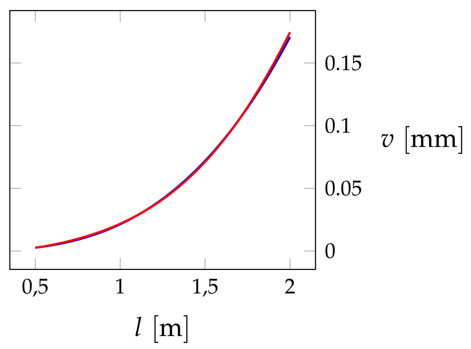


(b) e_2 vs. NT.

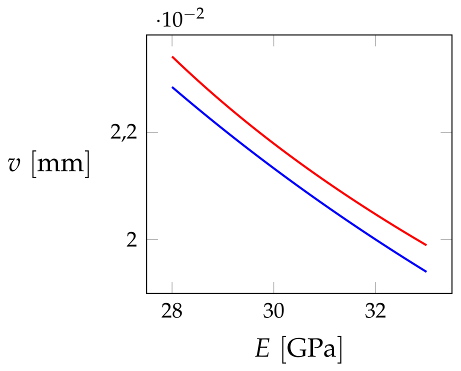
Figure 45: e_2 error trends - Random training data.



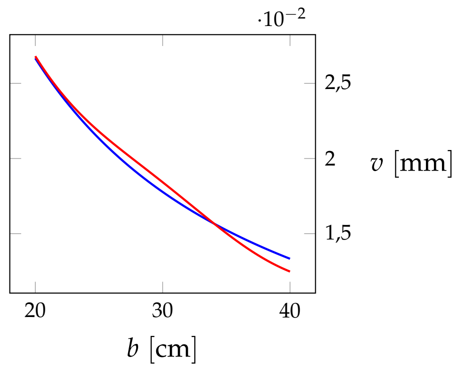
(a) v vs. P



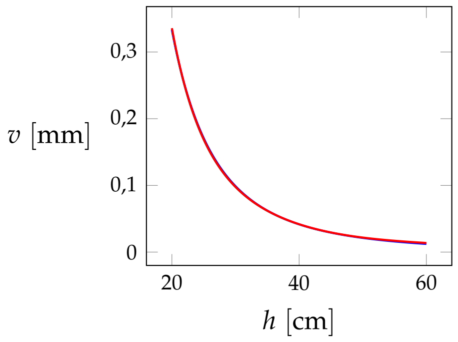
(b) v vs. l



(c) v vs. E



(d) v vs. b



(e) v vs. h

Figure 46: Analytical vs. ANN - Curve comparison.

-	Min	Max	N_d	Unit
F	1	10	15	kN
l	0,5	2	6	m
E	28	33	1	GPa
a	20	40	6	cm
b	20	60	6	cm
t	0	5	10	s

Table 13: Input training set data - Simply supported beam.

3.2 DYNAMIC BEAM DEFLECTION

This example is an extension of the static beam deflection of section 3.1. The function to be approximated is the vertical displacement at the middle of the simply supported beam depicted in figure **??**. The material and geometrical parameters are the kept the same, to make a direct comparison between the two. The analytical solution is presented in appendix 3.5.

The dynamic term is introduced in the analysis as an additional parameter t:

$$t \in [0 : 5\text{s}] \tag{64}$$

As a consequence, the analysis in time is defined only upon the training domain, and no reliable results must be expected outside that range of values.

Differently from the previous example, the training set is generated setting a number of divisions N_d for each parameter, to give preference at some of them. This trick is necessary, otherwise the total number of combination would be too large to be handled by the computer.

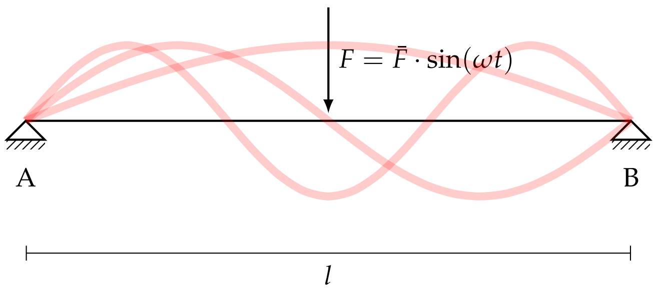


Figure 47: Simply supported beam - Harmonic excitation.

The solution has been searched varying the number of perceptron N_p and the dimension of the training set, that depends on N_d. Unfortunately, the results are not good as the ones of the static beam deflection. For this reason, it is reported only one solution for the data collected in table 13. Picture **??** shows the solution in term of displacement variation with respect force and time. The ANN is able to capture the linear variation within P, but not the harmonic trend with respect to t.

Summarizing, the response of an ANN feed with data generated from systems apparently similar, is not taken for granted. Even if from a mechanical point of view the physics is similar, the trend functions $f(\mathbf{x})$ are totally different.

3.3 PORTAL FRAME

The analysed time series function is taken from the dynamic modal solution of a two-story frame subjected to an harmonic force. The problem is solved under the hypothesis of axial inextensibility, which leads to the definition of two normal coor-

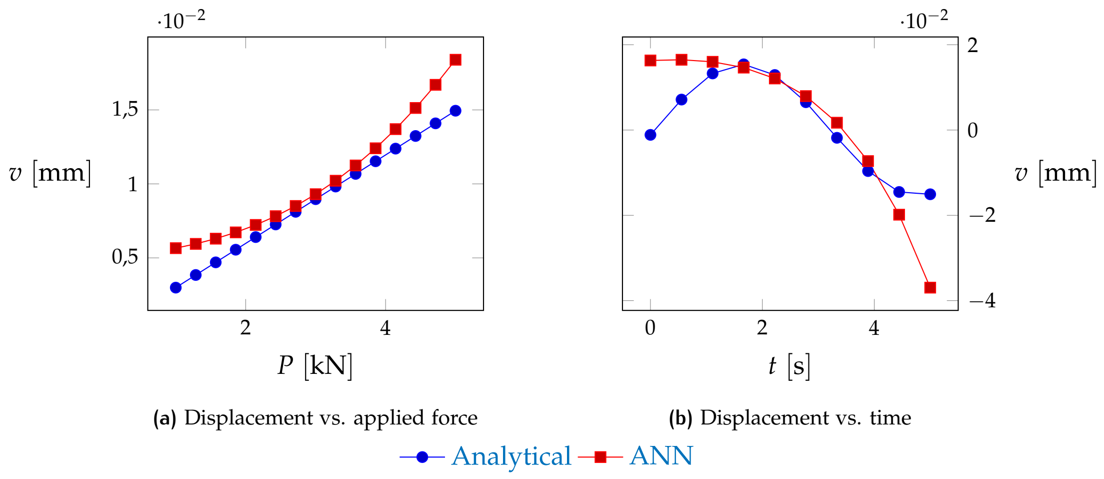


Figure 48: Curve comparison.

dinates parallel to the ground. Table **??** collects material and geometrical properties. The complete solution can be found in appendix 3.5.

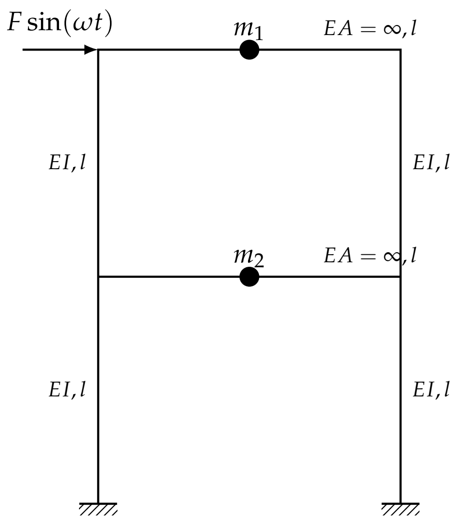


Figure 49: Two-storey frame subjected to an harmonic force.

The important characteristics for the definition of the RNN are the time steps and the final time, equal to 0.001 s and 0.5 s÷1 s respectively.

From the physical point of view, the time series function contains all the information about the response of the structure: if the parameters are kept constant to the whole analysis, there is no need to add additional information. Whereas there are parameters that varies in time, a possible modification is to include them as exogenous input to RNN. This may happen for instance if the frame system is subject to an acceleration coming from the ground motion, i.e an earthquake.

Being the example a fully-fledged multi-dof system, the possible solution methods is to built-up a single RNN feed with both input time series and two outputs, representing the solution at the time step n, or built up different RNNs.

In the first type, weights and biases will be shared within the functions: this means that the network would try to make the interpolation with the same parameters. If the trends are different, it could be difficult to find out acceptable predictions. Also the number of delays is shared, meaning that the autocorrelation function cannot be use any more. Moreover, the amount of RAM necessary to perform the computation increases significantly, with the risk to make unstable the software.

-	Value	Unit
m_1	5×10^3	m
m_2	10×10^3	m
E	30	GPa
l	3	m
a	50	cm
ξ	5%	–
ω	10	Hz
$\bar{F}$	10	kN
Δt	0,001	s

Table 14: Material and geometrical properties for the 2D beam problem.

The second type is more flexible, because it allows to tune differently the RNNs. The counter part is that, unless for specific hardware configuration, the training and the consequent predictions are performed sequentially, and this maybe a problem if the whole solution is needed at the same instant.

The analysis takes into consideration the top displacement.

S	$T.A.$	$L.F.$	$\varphi(\cdot)$
0.001 s	L.M.	MSE	tanh

Table 15: RNN parameters.

The first RNN is built up to see the response from an architecture defined by the lowest possible settings, that is 2 for both number of delays and perceptrons. Picture 51 shows the results.

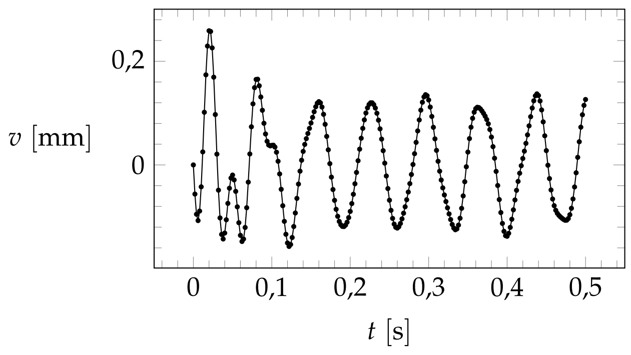


Figure 50: Training set for $t \in [0 : 0.5]$ [s]

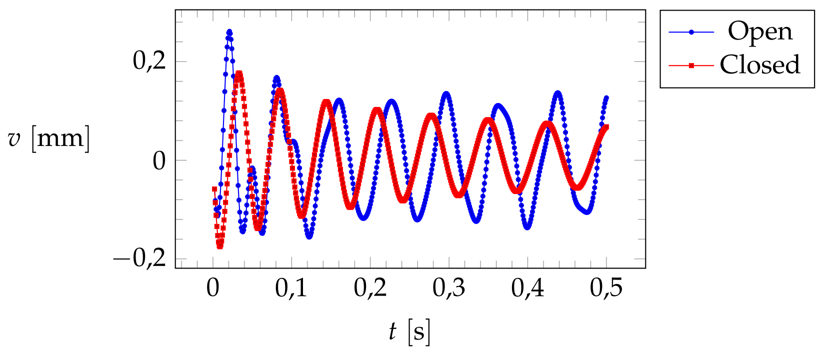


Figure 51: open vs. closed solution for 2 perceptrons and 2 delays.

The smooth result suggests that the time series is too complex for just two delays and it should be necessary to increase them. Then, the number is chosen looking at the auto-correlation function depicted in figure 52.

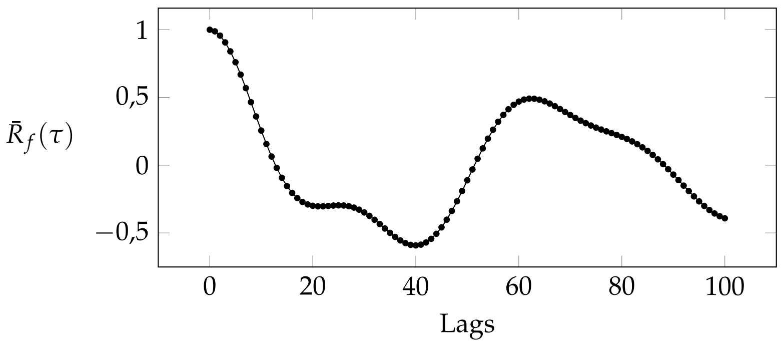
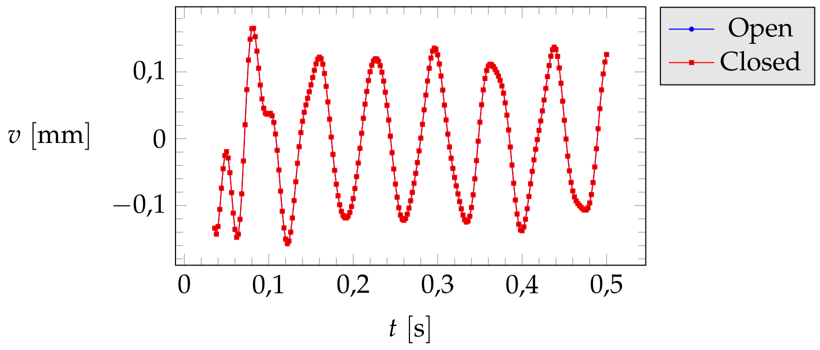


Figure 52: Autocorrelation function for the frame system response

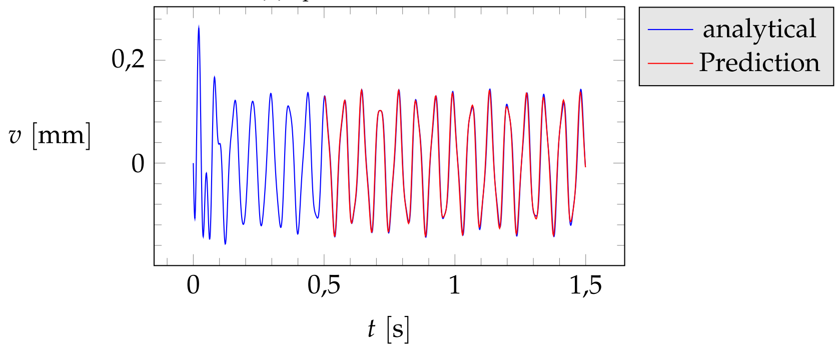
From the picture 52 it can be seen that the number of significant lags is around 40. A possible strategy is to sample the input at steps of two, following the suggestion of section 2.6. In this case, the number of lags is halved and chosen to be 18, a reasonable value considering the elements of the training set. The number of perceptrons increases accordingly, since the input layer is more complex to be analysed and so it requires more tunable parameters.

The solution is found in a trial and error fashion, by setting an iterative procedure, modifying the number of delays around the initial guess and the number of perceptron between 2 and 15. The best solution is found out for 8 perceptrons and 18 delays, visible in figure 53

To complete the discussion, figure 54 reports the solution for 19 delays, to realize how big it could be the misleading response by slightly changing some of the architecture parameters.

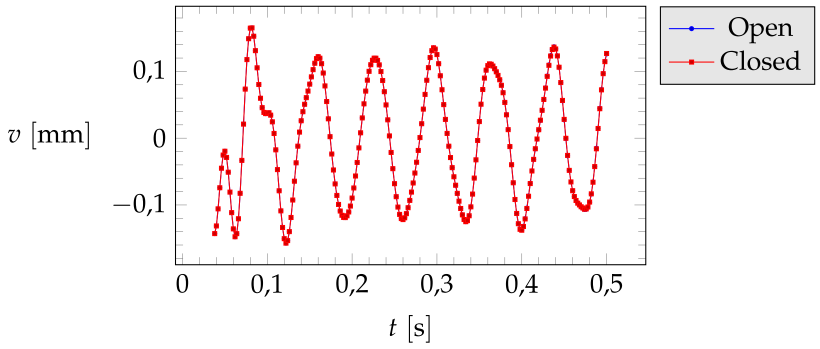


(a) open vs. closed solution.

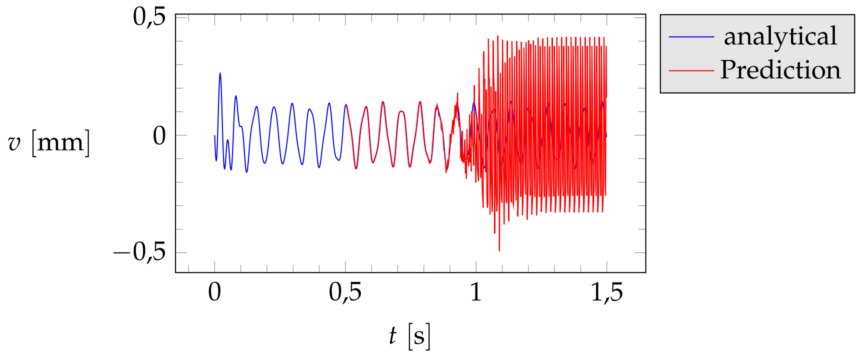


(b) Analytical vs. predicted solution.

Figure 53: Solution for 8 perceptrons and 18 delays.



(a) open vs. closed solution.



(b) Analytical vs. predicted solution.

Figure 54: Solution for 8 perceptrons and 19 delays.

To enhance the solution, the only way is to increase the training set range, if possible. Picture 57 shows the final results: improvements can be seen in the last part of the prediction, where solution 53 starts to get away from the analytical one.

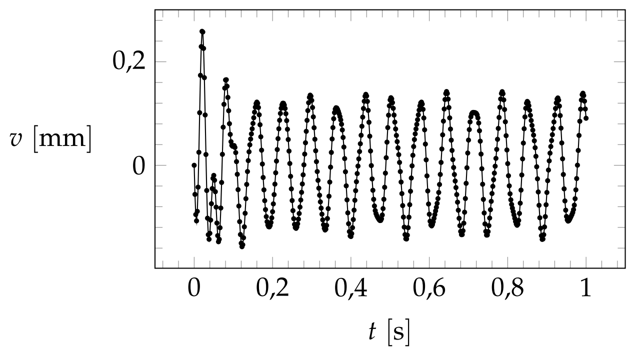
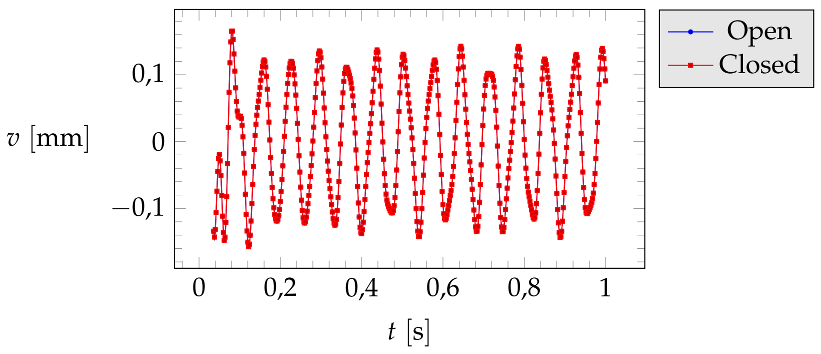
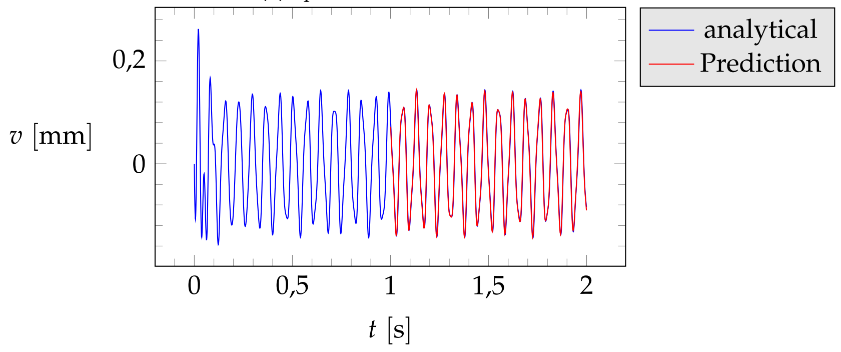


Figure 55: Training set for $t \in [0:1]$ [s]



(a) open vs. closed solution.



(b) Analytical vs. predicted solution.

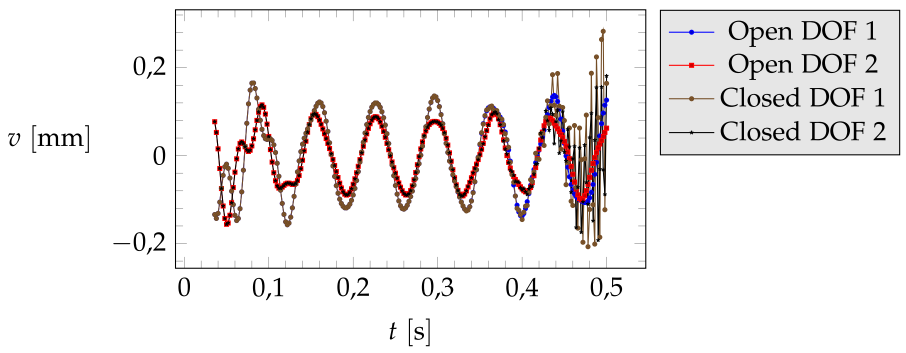
Figure 56: Solution for 8 perceptrons and 18 delays.

However, there is not generally a direct improvement of the solution: as a matter of facts, additional data would require additional parameters, and so a previously well trained RNN does not result in a better prediction capability. This means that the best way of thinking is to tune the RNN every time a new information is added.

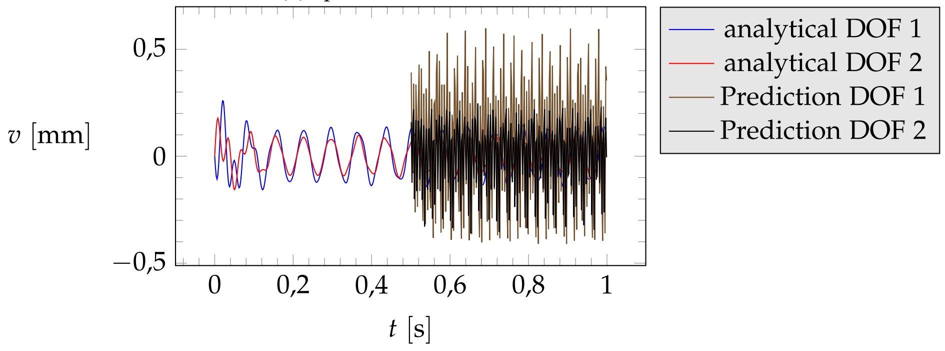
As corollary, picture shows the results of a RNN built up with the two DOFs. Considering 18 number of delays and 8 number of perceptrons as solution 53, the prediction fails: indeed, new design is always preferred when data changes, as indicated in several occasions.

3.4 2D PLANE PROBLEM

This example considers the dynamic solution of a 2D plane problem. In detail, the structure is a cantilever beam subjected to an harmonic force at the one extreme with linear elastic material properties.



(a) open vs. closed solution.



(b) Analytical vs. predicted solution.

Figure 57: Solution for 8 perceptrons and 18 delays.

The solution is numerically computed by the FEM, using 2D constant strain triangle and solved by the Newmark's algorithm. The structural scheme is the same of example 43 introduced in chapter 1. Picture **??** shows the adopted mesh, defined by 490 nodes and 802 elements.

The data of the problem are collected in table.

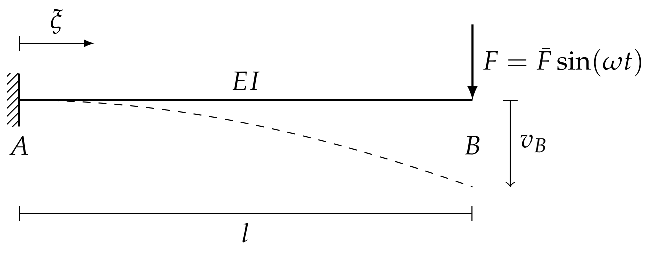


Figure 58: Clamped beam subjected to an harmonic load.

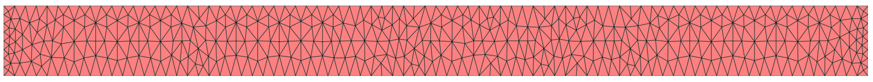


Figure 59: 2D mesh - CST elements.

The RNN can be created following the same reasons of the portal frame introduced in section 3.3. The main difference is the number of dofs: passing from 2 to 962, the built-up of a single RNN would require more than 40 GB of RAM, not compatible with standard computers. The alternative is a single RNN for each dof.

The latter approach reveals another benefit given by the use of a single dof response, that is the possibility to focus only on desired output. Whereas the deterministic approach on which the FEM is based needs all the displacements in time

-	Value	Unit
l	10	m
a	0,5	m
E	30	GPa
μ	0,18	–
γ_l	10	kN/m^2
t	0,15	m
a	0,5	m
Δt	0,1	s

Table 16: Material and geometrical properties for the 2D beam problem.

to continue the analysis, the RNN allows to extract and predict values only for characteristic points, such as the ones that expresses the maximum displacement.

For complex problems, as non-linear analysis, the computing time has a predominant role in the design, since the computational time may last several hours or days. Then, an artificial neural network could replace the full analysis after some training windows are available. This approach is similar to the model order reduction method, or MOR, widely used and based on features extraction for signals.

S	$T.A.$	$L.F.$	$\varphi(\cdot)$
0.001 s	L.M.	MSE	tanh

Table 17: RNN parameters.

The optimal solution is found in an iterative way, similarly to the portal frame example 3.3, ranging the value of perceptron between 2 and 15 and with the number of delays found through the autocorrelation function, reported in figure 60.

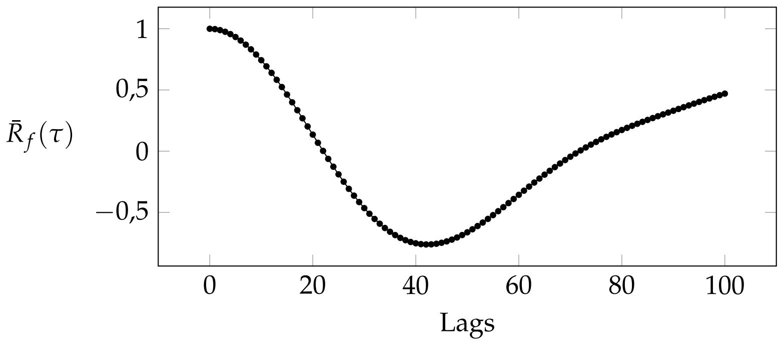


Figure 60: $\bar{R}_f(\tau)$ - 2D plane beam - 1 sampling step.

Also in this example the number of delays is too large, and an opportune sampling is required. In this case it is chosen to sample a step of three; picture 61 shows the updated autocorrelation function, showing an optimal number of lags equal to 18. It is reported directly the best solution found out from the iteration. It is characterize by a number of delays equal to 16 and a number of perceptrons 7.

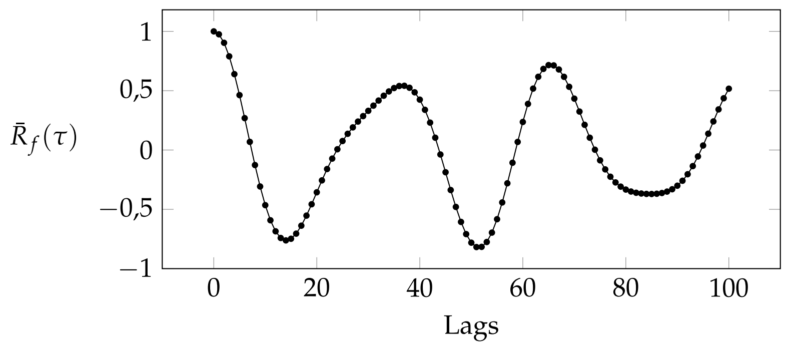


Figure 61: $\bar{R}_f(\tau)$ - 2D plane beam - 3 sampling step.

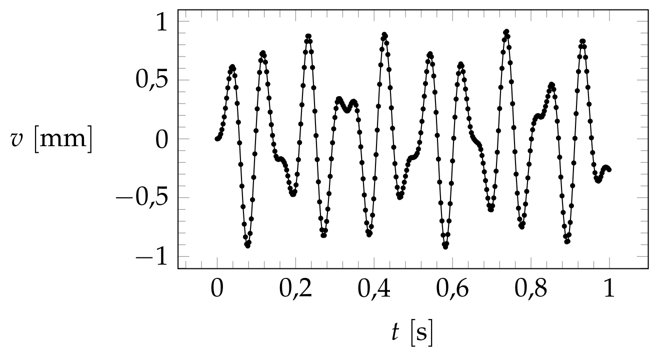


Figure 62: Training set for $t \in [0:1]$ [s] - 2D plane beam.

3.5 MULTI-PHYSICS - PLANE RESONATOR

The term multi-physics is used to define systems which involve different physical phenomena. In general, any problem is a multi-physical problem: when one physical aspect prevails upon the others, the secondary terms are disregarded from the analysis, simplifying the mathematical modelling. This simplification is not always applicable: for instance, a beam subjected to an high frequency pulsation generates a temperature differential which produces not negligible local compression and tension stresses. In these cases, the problem belongs to the family of thermo-mechanical problems, being coupled in the sense of mechanical and thermal effects.

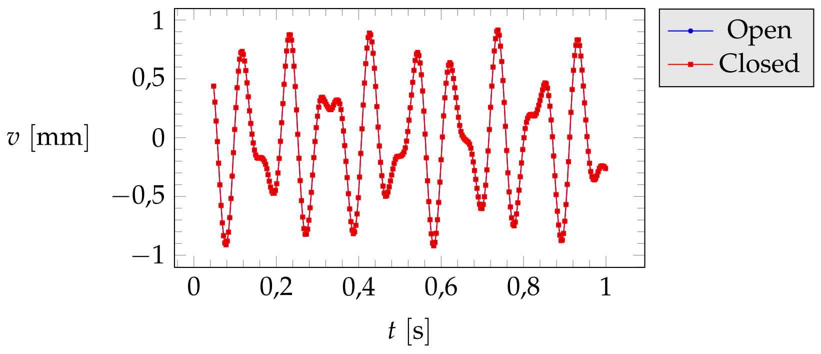
Similarly, the effect of temperature cannot be anyway disregarded if the spatial dimensions of the body of the system are very small, in the order of µm: this happens for instance for *micro electro-mechanical system* or MEMS.

This example considers the dynamic behaviour of a plane resonator, a system that is able to vibrate in resonance, preserving the frequency. The dissipation of energy in term of generated heat from the temperature differential, produces a not desired decrease in the vibration frequency. This phenomenon goes under the name of thermo-elastic-damping (TED), and an in-deep analysis of the problem is useful in the design process, due to the expensive production cost of the device. This example is part of thermo-mechanical problem, since the dynamic response must account for the additional term due to the temperature field.

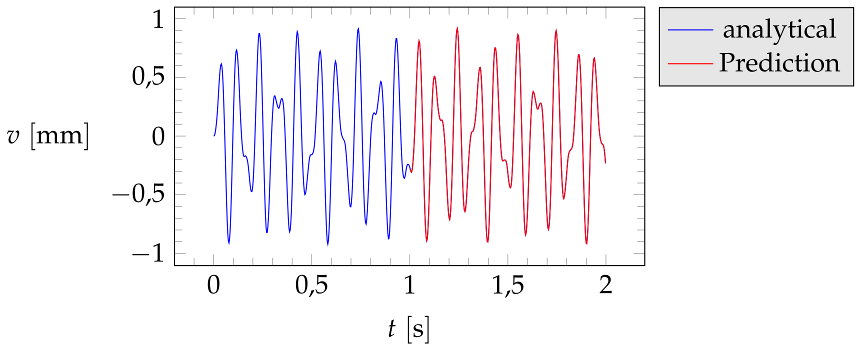
A basic introduction to the problem is reported in appendix 3.5. Technical details can be found in [3].

The solution is presented is a way similar to the previous examples, showing the results for the displacement in the y direction and then for the temperature. The time series functions belongs to the node at the center of the resonator.

First, the identification of the optimal number of delays from the autocorrelation function reported in figure 65. As it can be seen from the picture, the number of



(a) open vs. closed solution.



(b) Analytical vs. predicted solution.

Figure 63: Solution for 7 perceptrons and 16 delays - 2D plane beam.

delays is too large; then, a number of 2 step is chosen, leading to a value equal to 35 (see picture 66).

Looking at the displacement function in figure **??**, it can be seen that the trend is not as smooth as in the previous cases, and this is due to the high frequency of vibration and the coupling term with the temperature. Therefore, it cannot be increased too much the number of sampling steps, otherwise the real signal trend may be lost.

Then, the optimal result is found iteratively starting from 35 number of delays and the same number of perceptron of example 3.4 and 3.3. Picture 68 shows that the predicted solution does not follows the analytical curve as well as the previous examples. This is due to the complexity of the function, where the output must relies on the interpolation capability of the RNN. Again, the only possible strategy is to extend the trained set, if data are available.

The same operation is performed for what concern the temperature variation. The number of delays first: picture 69 shows a number of delays equal to 40 for a sampling step equal to 2.

The temperature function here is even more complex than the displacement one: it is expected acceptable results for the period but not so well for the amplitude. Picture shows the results 71.

Another approach is suggested by the staggered algorithm introduced in appendix 3.5. In that case, the solution is found jumping back and forth from one domain to another, until convergence is fulfilled. The same philosophy can be applied by using NARX architecture: considering for instance the displacement series, the RNN is trained giving as external input the temperature variation; separately is performed the same analysis by switching the time series.

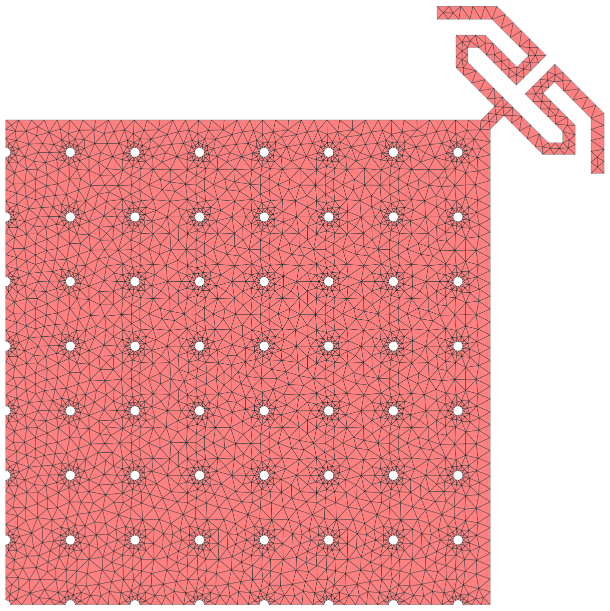


Figure 64: Multi-physics - Resonator - CST elements.

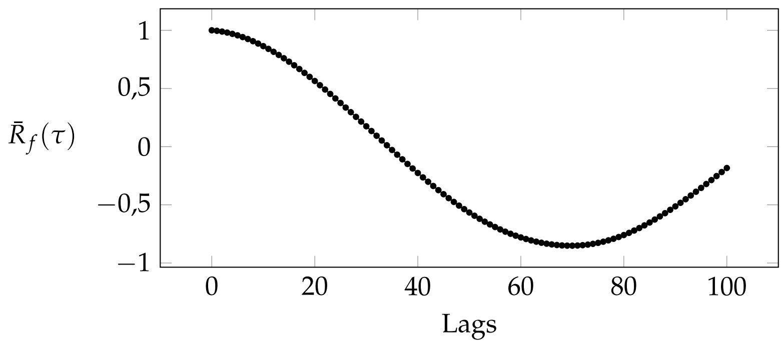


Figure 65: $\bar{R}_f(\tau)$ - Plane resonator - 1 sampling step - D.

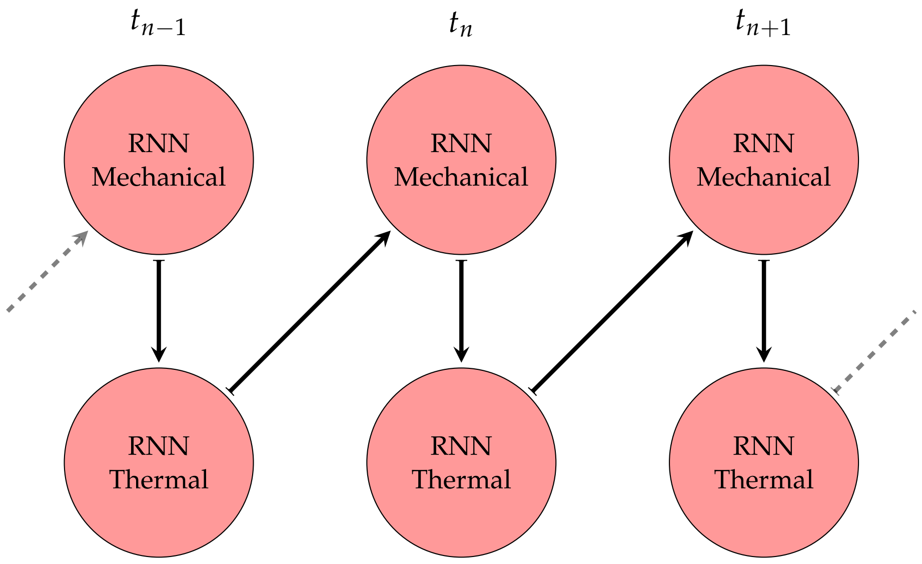


Figure 72: Passing information scheme from mechanical and thermal RNNs.

After the training, there would be two systems that can predict values giving as input the output of the other one: in a loop cycle, this procedure allows to make

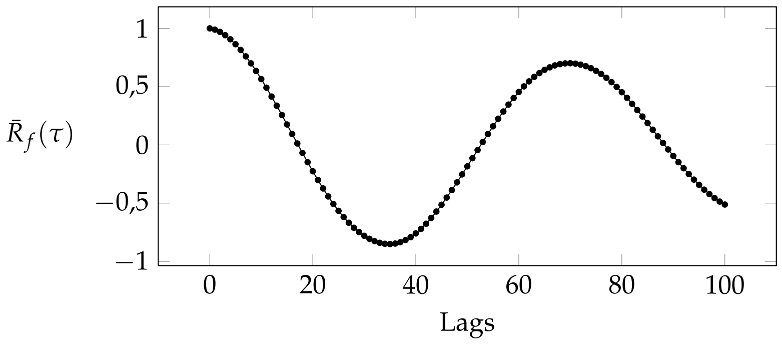


Figure 66: $\bar{R}_f(\tau)$ - Plane resonator - 2 sampling step - D.

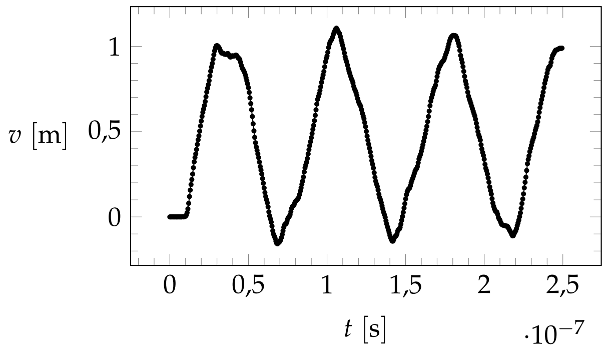


Figure 67: TS for $t \in [0 : 2.5 \times 10^{-7}]$ [s] - Plane resonator - D.

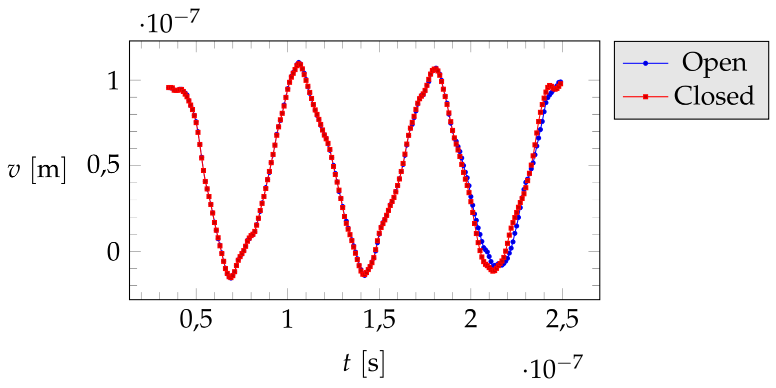
predictions until a desired final time. In the following it is reported the solution organized as before, indicating also the cross-correlation function.

Picture 73 shows the auto-correlation and cross-correlation for the two time series. It is chosen 35 feedback and input delays for the displacement function, while 40 and 35 for the temperature one respectively. Again the solution is found in an iterative way as before.

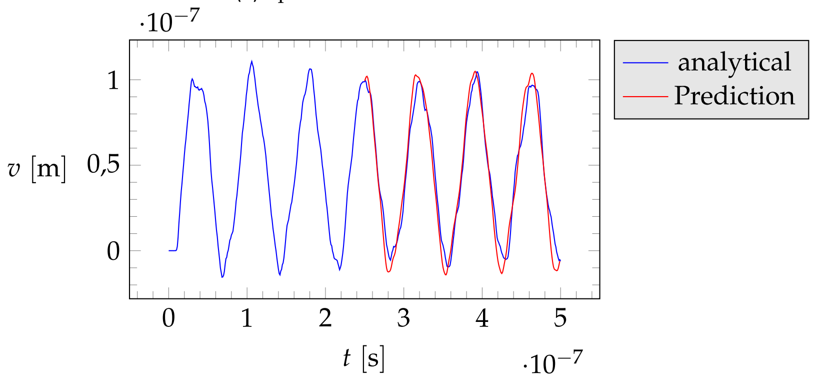
Picture 74 shows the results in terms of open and closed form: for what concern the displacement function, the closed trend is closer to the open one, similar to the NARN network interpolation, while for the temperature the effects of oscillation becomes higher and the RNN shows some difficulties in the reproduction of the curve.

Picture 75 reports the predictions of the system of RNNs: the fitting is not even closer for the temperature, while some the displacement it keeps the period but it exhibits spurious oscillations.

In conclusion, the RNN shows some limits whereas the series are not sufficiently smooth. A possibile strategy is to pre-treats inputs, not only for what regards the sampling step, but also the shape, depending on the scope of the analysis and whether the oscillations are relevant or not for the interpretation of data.



(a) open vs. closed solution.



(b) Analytical vs. predicted solution.

Figure 68: Solution for 10 NP and 35 delays - Plane resonator. - D.

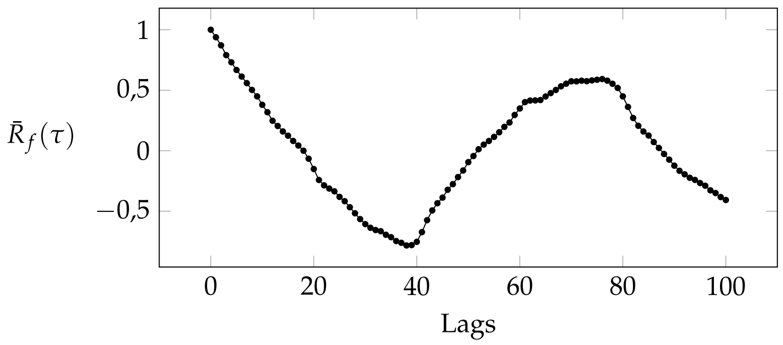


Figure 69: $\bar{R}_f(\tau)$ - Plane resonator - 2 sampling step. - T.

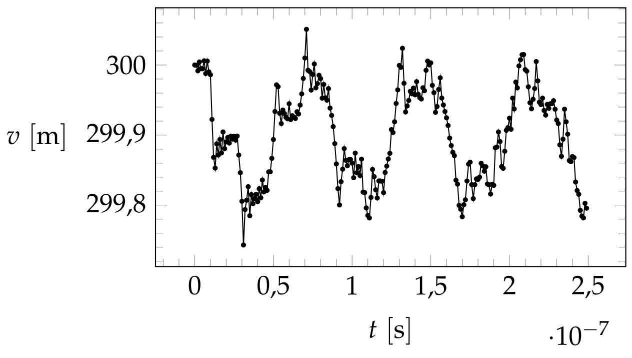
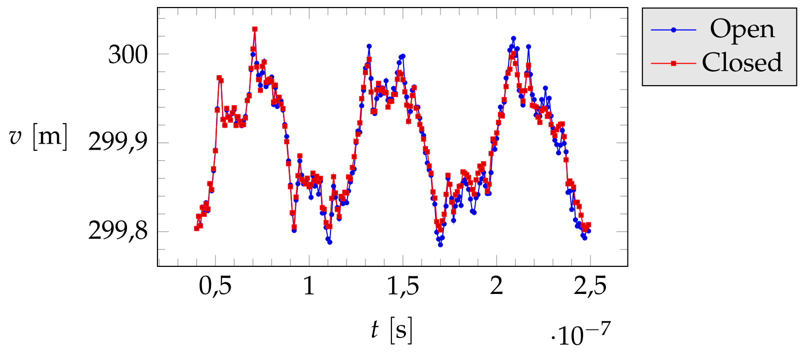
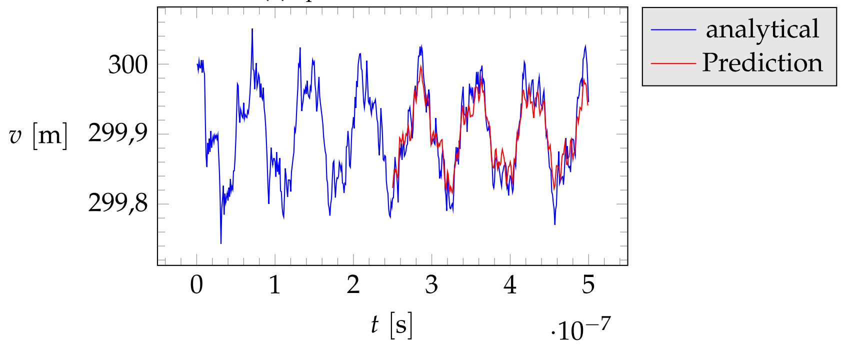


Figure 70: Training set for $t \in [0 : 2.5 \times 10^{-7}]$ [s] - Plane resonator - T.

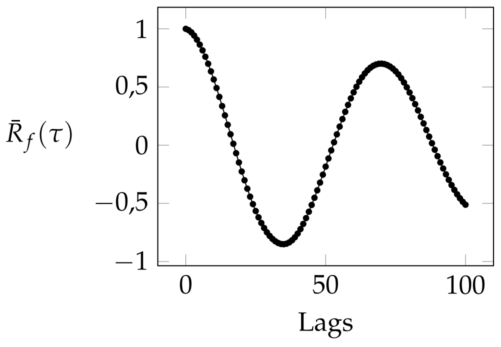


(a) open vs. closed solution.

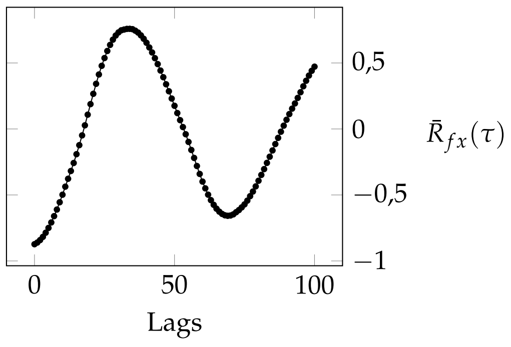


(b) Analytical vs. predicted solution.

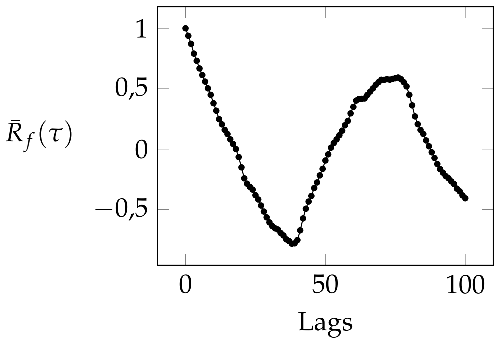
Figure 71: Solution for 10 NP and 40 delays - Plane resonator - T.



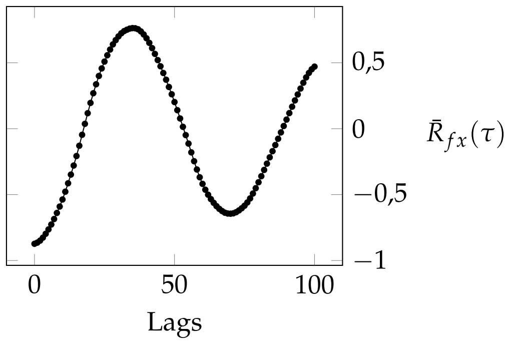
(a) $\bar{R}_f(\tau)$ - Displacement.



(b) $\bar{R}_{fx}(\tau)$ - Temperature.

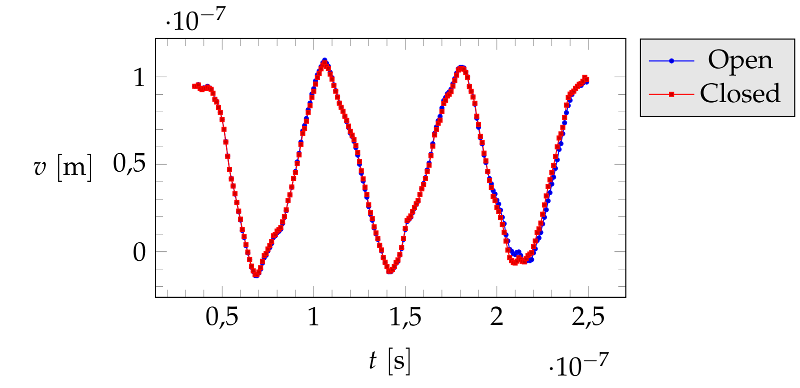


(c) $\bar{R}_f(\tau)$ - Temperature.

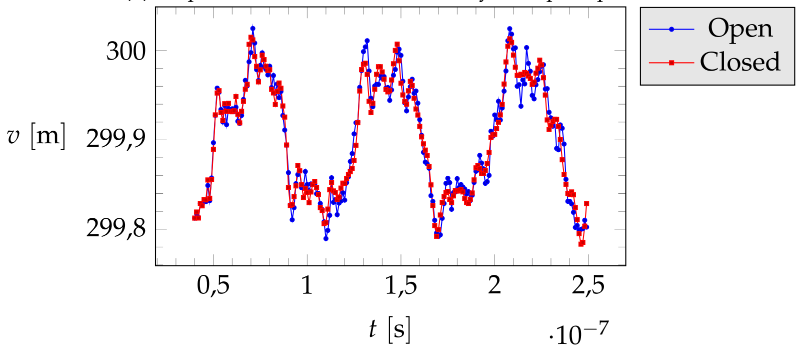


(d) $\bar{R}_{fx}(\tau)$ - Displacement.

Figure 73: Normalized auto-correlation and cross-correlation function - Plane resonator - 2 sampling step.

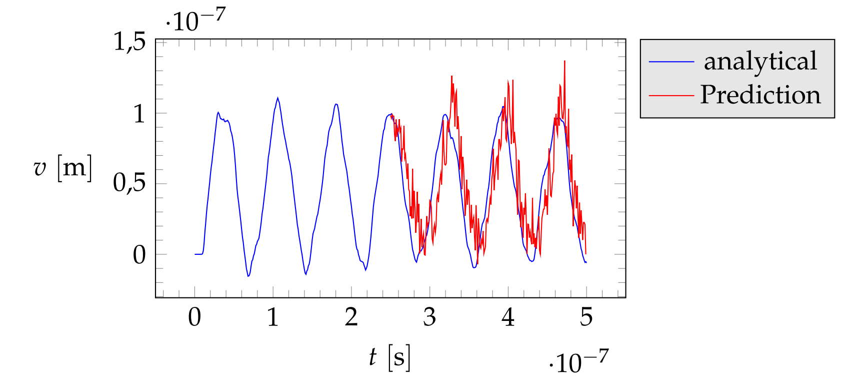


(a) Displacement time series - 35 delays - 10 perceptrons.

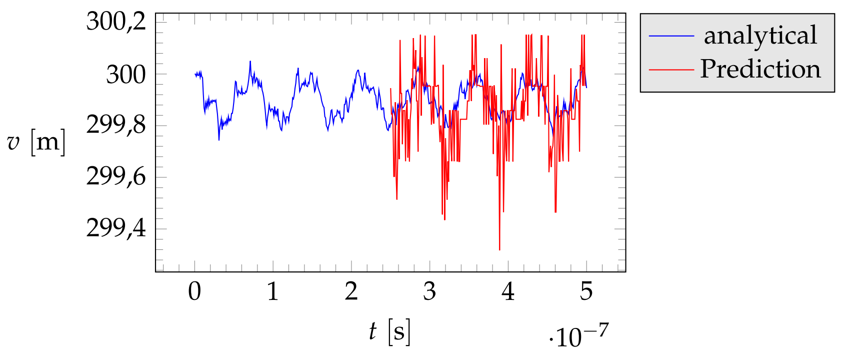


(b) Temperature time series - 40 delays - 13 perceptrons.

Figure 74: Open and closed solution - Plane resonator.



(a) Analytical vs. predicted solution - 10 perceptrons and 35 delays - Displacement.



(b) Analytical vs. predicted solution - 13 perceptrons and 40 delays - Temperature.

Figure 75: Solution for the coupled problem - Plane resonator

CONCLUSION

The aim of the thesis is to explore the possible use of ANN on engineering problems.

The first important observation coming from the starting chapter regards the non linearity: what ones is led to think is that the function interpolating the data points is non linear; actually what can be addressed as non linear is just at the global view: the domain is divided in sets, each of them interpolated by a linear function, defined by weights and biases.

The static 3.1 and the dynamic beam deflection 3.2 are used as examples of feed-forward backpropagation ANN. The former includes a study of the variation of errors with respect to the number of perceptrons and the size of the training set. A first type of error is evaluated on the training set and, on the contrary of the expectations, there is a small increase of the error as the size becomes larger. The second one includes a test on the generalization capability and it follows the expectations. In any case, the ANN expresses good interpolating capability, sustained by the small amount of resources necessary to store and perform the computation. The dynamic examples changes the function to be interpolated, and as expected, results are not good: in fact, the trend of the curve and the data distribution highly effects the outcome of an ANN. As conclusion, ANN are suitable for parametric analysis, taking care of the goodness of the training phase.

The application of RNN for the case of portal frame 3.3 and 2D beam case 3.4 follows the standard definition: giving a time series function, deciding how many points of it the user want to interpolate, RNN delivers a prediction. Acceptable results are obtained if a sufficient number of entry data are given to the RNN. The expectations are fulfilled: under some conditions, a working Recurrent Network can be built up and used to obtain reliable results.

Finally, an ambitious tentative follows the path of the staggered algorithm widely used for solving coupled physical systems. The aim is to make a direct comparison with MOR methods, where the solution is found on reduced basis decreasing the order of magnitude of the size of the problem. They are based on a training phase, fully in agreement with the one of neural networks, and performs the computation relying only on statistically meaningful variables. Two RNNs are built-up independently, with information coming from the different physics. Final predictions are obtained inquiring the RNNs step-by-step passing information from one to the other, similar of what the staggered algorithm is based on. Unfortunately, the results are not good as desired, and the mistake may lie on the chosen physical system: as a matter of fact, the thermo-Mechanical coupling of resonators occours only under high frequency excitations, leading to trends extensively affected by oscillations, not positively handled by RNNs. Applying the same procedure to simpler examples may have affermative conlusions.

The thesis can be enhanced following different reasons and attitude:

ARCHITECTURE Explore innovative architectures every day developed (Convolutional networks, Long-Short-Term-Memory (LSTM) networks, generative adversarial network (GAN), . . .).

ALGORITHM Improve the algorithm involved in the computation (Optimization algorithm, Random initialization algorithm).

ARCHITECTURE FUNCTIONS In-deep studies of pivotal nodes of the architecture (Activation function definition, weights and biases position).

MODAL ANALYSIS FOR BEAM

The indefinite dynamic equilibrium equation reads:

$$\begin{cases} EI\dfrac{\partial^4\omega}{\partial\xi^4} = -\varrho A\dfrac{\partial^2\omega}{\partial t^2} \\ \omega(0,t) = \omega(l,t) = 0 \\ EI\dfrac{\partial^2\omega}{\partial\xi}\Big|_{\xi=0} = EI\dfrac{\partial^2\omega}{\partial\xi}\Big|_{\xi=l} = 0 \end{cases} \tag{65}$$

Its solution leads to the definition of the modal shapes. For a simply supported beam it holds:

$$\phi_n(\xi) = \sin(\gamma\xi) = \sin\left(\frac{n\pi}{l}\xi\right) \tag{66}$$

Then, the solution can be written as:

$$\omega(\xi,t) = \sum_{n=1}^{\infty}\phi_n(\xi)q_n(t) = \boldsymbol{\phi}(\xi)^T\mathbf{q}(t) \tag{67}$$

Where $\phi_n(\xi)$ and $q_n(t)$ are the modal shapes and the principal coordinates respectively.

Introducing the Euler-Lagrangian equation for discrete dynamic system:

$$\frac{\partial}{\partial t}\left(\frac{\partial T}{\partial\dot{q}_i}\right) - \frac{\partial T}{\partial q_i} + \frac{\partial V}{\partial q_i} = Q_i \tag{68}$$

Where T is the kinetic energy, V is the potential, Q_i the active Lagrangian components and $q_i(t)$ the principal or generalized coordinates. By definition:

$$T = \frac{1}{2}\int_V \dot{\omega}^2 dm = \int_0^l \rho A\dot{\omega}^2(\xi,t)d\xi \tag{69}$$

$$E = \frac{1}{2}\int_{\bar{V}}\sigma_{ij}\varepsilon_{ij}dV = \int_0^l M\chi d\xi = \int_0^l EI\left(\frac{\partial^2\omega}{\partial\xi^2}\right)^2 d\xi \tag{70}$$

$$\delta W = \sum_i Q_i\delta\omega_i \tag{71}$$

Substituting (67) into (69), (70) and (71) one obtains for the kinetic energy:

$$\begin{aligned} T &= \frac{1}{2}\int_0^l \dot{\mathbf{q}}^T\boldsymbol{\phi}^T m\boldsymbol{\phi}\dot{\mathbf{q}}d\xi = \frac{1}{2}\dot{\mathbf{q}}^T\left[\int_0^l \boldsymbol{\phi}^T m\boldsymbol{\phi}d\xi\right]\dot{\mathbf{q}} = \\ &= \frac{1}{2}\dot{\mathbf{q}}^T\begin{bmatrix} \int_0^l m\phi_1^T\phi_1 d\xi & 0 & \dots & 0 \\ 0 & \int_0^l m\phi_2^T\phi_2 d\xi & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & \int_0^l m\phi_n^T\phi_n d\xi \end{bmatrix}\dot{\mathbf{q}} = \\ &= \frac{1}{2}\dot{\mathbf{q}}^T\begin{bmatrix} \frac{l}{2}m & 0 & \dots & 0 \\ 0 & \frac{l}{2}m & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & \frac{l}{2}m \end{bmatrix}\dot{\mathbf{q}} = \\ &= \frac{1}{2}\dot{\mathbf{q}}^T\underbrace{\frac{l}{2}m\mathbf{I}}_{M^*}\dot{\mathbf{q}} = \frac{1}{2}\dot{\mathbf{q}}^T\mathbf{M}^*\dot{\mathbf{q}} \end{aligned} \tag{72}$$

For the potential energy:

$$
\begin{aligned}
T &= \frac{1}{2}\int_0^l \mathbf{q}^T\ddot{\boldsymbol{\phi}}^T m\ddot{\boldsymbol{\phi}}\mathbf{q}d\xi = \frac{1}{2}\mathbf{q}^T\left[\int_0^l \ddot{\boldsymbol{\phi}}^T m\ddot{\boldsymbol{\phi}}d\xi\right]\mathbf{q} = \\
&= \frac{1}{2}\mathbf{q}^T\begin{bmatrix} \int_0^l m\ddot{\phi}_1{}^T\ddot{\phi}_{1_1}d\xi & 0 & \dots & 0 \\ 0 & \int_0^l m\ddot{\phi}_{1_2}^T\ddot{\phi}_{1_2}d\xi & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & \int_0^l m\ddot{\phi}_{1_n}^T\phi_n d\xi \end{bmatrix}\mathbf{q} = \\
&= \underbrace{\frac{1}{2}\mathbf{q}^T\frac{l}{2}EI\begin{bmatrix} \left(\frac{\pi}{l}\right)^4 & 0 & \dots & 0 \\ 0 & \left(\frac{2\pi}{l}\right)^4 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & \left(\frac{n\pi}{l}\right)^4 \end{bmatrix}}_{K^*}\mathbf{q} = \\
&= \frac{1}{2}\mathbf{q}^T\mathbf{K}^*\mathbf{q}
\end{aligned}
\tag{73}
$$

For the active component:

$$
\begin{aligned}
\delta W = F(t)\delta\omega(\xi, t) &= F(t)\boldsymbol{\phi}^T\delta\mathbf{q}(t) = \\
&= F(t)\begin{bmatrix} \sin\left(\frac{\pi}{l}a\right) \\ \sin\left(\frac{2\pi}{l}a\right) \\ \vdots \\ \sin\left(\frac{n\pi}{l}a\right) \end{bmatrix}\delta\mathbf{q}(t) = \mathbf{Q}^T\mathbf{q}(t)
\end{aligned}
\tag{74}
$$

Being all the matrices diagonal, the problem is decoupled and each principal coordinate can be studied as a single DOF system. Finally,

$$
m_i^*\ddot{q}_i + k_i^*\dot{q} = Q_iF(t) \rightarrow q_i = \frac{Q_i}{k}N(\beta)\sin\left(\bar{\omega}t - \zeta(\beta)\right) \tag{75}
$$

With:

$$
N(\beta) = \left|\frac{1}{1-\beta^2}\right| \tag{76}
$$

$$
\begin{cases} \zeta = 0, & \text{for} \quad 0 < \beta < 1 \\ \zeta = \pi, & \text{for} \quad \beta > 1 \end{cases} \tag{77}
$$

Where q_i gives back the shape of the response of each DOF in time.

As a final remark, the damping can be included easily in the formulation, if the problem can be considered decoupled in term of damping, by using the Rayleight formula:

$$
\mathbf{C}^* = \alpha\mathbf{M}^* + \beta\mathbf{K}^* \tag{78}
$$

Where α and β are constants experimentally defined. Being $\mathbf{C}^*$ a linear combination of $\mathbf{M}^*$ and $\mathbf{K}^*$, the system is still studied with an updated version of solution (75):

$$
N(\beta) = \frac{1}{\sqrt{\left(1-\beta^2\right)^2 + 4\xi^2\beta^2}} \tag{79}
$$

$$
\zeta = \arctan\left(\frac{2\xi\beta}{1-\beta^2}\right), \quad 0 \le \zeta \le \pi \tag{80}
$$

The complete solution can be found in [1].

SDOF OSCILLATOR - FREE MOTION

The basic SDOF oscillator is made up by a spring, a dumper and a punctual mass, representing stiffness, damping term and inertia of the system respectively. As depicted in figure 76, no external forces are applied under the free motion hypothesis. The equation of motion reads:

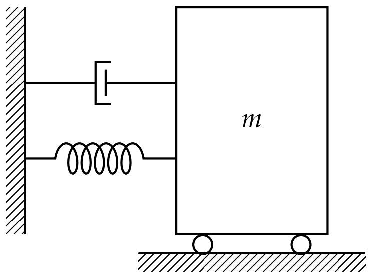


Figure 76: SDOF system.

$$\begin{cases} m\ddot{x}(t) + c\dot{x}(t) + kx(t) = 0 \\ \dot{x}(0) = a \\ x(0) = b \end{cases} \tag{81}$$

Rearranging some terms:

$$\begin{cases} \ddot{x}(t) + 2\xi\omega_1\dot{x}(t) + \omega_1^2 x(t) = 0 \\ \dot{x}(0) = a \\ x(0) = b \end{cases} \tag{82}$$

Where:

$$\omega_1^2 = \frac{k}{m} \tag{83}$$

$$\xi = \frac{c}{2\sqrt{km}} \tag{84}$$

The equilibrium equation is a second order linear differential equation with constant coefficients. The solution can be sought in the form:

$$x(t) = X_0 e^{\lambda t} \tag{85}$$

Substituing and manipulating some terms, the procedure will eventually lead to:

$$x(t) = e^{-\xi\omega_1 t}\left[A\cos(\omega_d t) + B\sin(\omega_d t)\right] \tag{86}$$

Where:

$$\begin{cases} A = x(0) \\ B = \dfrac{\dot{x}(0) + \xi\omega_1 x(0)}{\omega_d} \end{cases} \tag{87}$$

And

$$\omega_d = \omega_1 \cdot \sqrt{1 - \xi^2} \tag{88}$$

Considering the following initial condition:

$$\begin{cases} \dot{x}(0) = 0 \\ x(0) = \dfrac{F}{k} \end{cases} \tag{89}$$

The equilibrium equation reads:

$$\begin{cases} x(t) = e^{-\xi\omega_1 t}\left[A\cos(\omega_d t) + B\sin(\omega_d t)\right] \\ A = \dfrac{F}{k} \\ B = \dfrac{\xi\omega_1}{\omega_d}\dfrac{F}{k} \end{cases} \tag{90}$$

The complete solution can be found in [1, 2, 10]

PORTAL FRAME

The solution is computed through modal analysis.

The discretized equilibrium equation reads:

$$\mathbf{m}\ddot{\mathbf{x}} + \mathbf{c}\dot{\mathbf{x}} + \mathbf{k}\mathbf{x} = \mathbf{Q} \tag{91}$$

The displacement vector is manipulated by making the following transformation:

$$\mathbf{x} = \mathbf{\Phi}\mathbf{y} \tag{92}$$

Where $\mathbf{\Phi}$ is a matrix collecting the eigenvectors and $\mathbf{y}$ the generalized, or normal, coordinates. Substituting into (91), one finds:

$$\mathbf{x} = \mathbf{\Phi}\mathbf{y} \rightarrow \mathbf{\Phi}^T\mathbf{m}\mathbf{\Phi}\ddot{\mathbf{y}} + \mathbf{\Phi}^T\mathbf{c}\mathbf{\Phi}\dot{\mathbf{y}} + \mathbf{\Phi}^T\mathbf{k}\mathbf{\Phi}\mathbf{y} = \mathbf{\Phi}^T\mathbf{Q} \rightarrow \tag{93}$$

$$\rightarrow \mathbf{M}\ddot{\mathbf{y}} + \mathbf{C}\dot{\mathbf{y}} + \mathbf{K}\mathbf{y} = \mathbf{\Phi}^T\mathbf{Q} \tag{94}$$

It can be demonstrated that:

$$\mathbf{\Phi}^T(\cdot)\mathbf{\Phi} \tag{95}$$

is a diagonalization for the matrices $\mathbf{M}$ and $\mathbf{K}$, while for $\mathbf{C}$ it holds under specific conditions.

Then, the equilibrium system is decoupled and it can be solved thought the equations of the single dof system:

$$\ddot{y}_j + 2\xi\omega_j\dot{y}_j + \omega_j^2 y_j = \frac{\boldsymbol{\varphi}_j\mathbf{Q}}{M_j} \tag{96}$$

Eigenvectors and eigenvalues are computed solving the eigenvalue problem:

$$(\mathbf{k} - \omega^2\mathbf{m})\boldsymbol{\varphi} = \mathbf{0} \tag{97}$$

Substituting the data collected in table ??, the solution of (97) leads to:

$$\omega_1 = 90.20\ 1/s \tag{98}$$

$$\omega_2 = 217.76\ 1/s \tag{99}$$

for the natural frequencies and:

$$\boldsymbol{\varphi}_1 = \begin{bmatrix} 0.0100 \\ 0.0071 \end{bmatrix}; \boldsymbol{\varphi}_2 = \begin{bmatrix} 0.0100 \\ -0.0071 \end{bmatrix} \tag{100}$$

for the eigenvectors.

Instead, it can be demonstrated that the solution of the SDOF system reads:

$$\begin{cases} v(t) = e^{\xi\omega_1 t}\left[A\cos(\omega_d t) + B\sin(\omega_d t)\right] + \frac{F}{k}N\sin(\omega t - \zeta) \\ A = v(0) + \frac{F}{k}N\sin(\zeta) \\ B = \frac{1}{\omega_d}\left[\dot{v}(0) + \xi\omega_1\left[v(0) + \frac{F}{k}N\sin(\zeta)\right]\right] - \omega\frac{F}{k}N\cos(\zeta) \end{cases} \tag{101}$$

With:

$$\zeta = \arctan\left(\frac{2\xi\beta}{1-\beta^2}\right) \tag{102}$$

$$\beta = \frac{\omega}{\omega_1} \tag{103}$$

$$N = \frac{1}{\sqrt{(1-\beta^2)^2 + 4\xi^2\beta^2}} \tag{104}$$

The complete solution can be found in [1, 2, 10]

THERMO-MECHANICAL PROBLEMS

In the following are reported the indefinite equilibrium equation for the two physics and the coupling system. For the complete discussion [3, 15, 20].

1 THERMAL FIELD

The indefinite equilibrium equation is based on the energy balance and the Fourier's law and reads:

$$\nabla \cdot (-\mathbf{K} \cdot \nabla T) = g + c\varrho\dot{T} \tag{105}$$

Where

$\mathbf{K}$ Thermal conductivity tensor.

T Temperature.

$\nabla \cdot (\cdot)$ Divergence operator.

$\nabla(\cdot)$ Gradient operator.

$\dot{(\cdot)}$ Time derivative.

g Internal heat generation.

c Capacity.

ϱ Density.

The associated boundary conditions reads:

$$\begin{cases} \mathbf{T} = \bar{\mathbf{T}} & ,\text{on } \Gamma_T \\ \mathbf{q} \cdot \mathbf{n} = \bar{\mathbf{q}} & ,\text{on } \Gamma_q \end{cases} \tag{106}$$

Where $\bar{\mathbf{T}}$ and $\bar{\mathbf{q}}$ are fixed temperature and flux respectively.

The coupling term is introduced a posteriori supposing that the flux depends linearly on the strain velocity:

$$\mathbf{q}_t = \boldsymbol{\beta}\dot{\boldsymbol{\varepsilon}} = T_0 \cdot \mathbf{D}\boldsymbol{\alpha}^T\dot{\boldsymbol{\varepsilon}} \tag{107}$$

Where

T_0 Initial temperature.

$\mathbf{D}$ Elastic tensor.

$\boldsymbol{\alpha}$ Vector of coefficients of thermal expansion.

2 MECHANICAL FIELD

The coupling term is due to the temperature variation, and it has a component in the total elastic deformation $\boldsymbol{\varepsilon}$.

$$\boldsymbol{\varepsilon} = \boldsymbol{\varepsilon}_e - \boldsymbol{\varepsilon}_0 \tag{108}$$

Where

ε_e Elastic deformation.

ε_0 Inelastic deformation.

Consequently, the constitutive law reads:

$$\sigma = \mathbf{D} \cdot \varepsilon_e = \mathbf{D} \cdot (\varepsilon - \varepsilon_0) = \mathbf{D} \cdot \varepsilon \underbrace{-\mathbf{D} \cdot \varepsilon_0}_{\substack{\text{Additional} \\ \text{to the FE} \\ \text{formulation}}} \tag{109}$$

3 COUPLED INDEFINITE EQUILIBRIUM SYSTEM

Mechanical domain:

$$\begin{cases} \begin{cases} \nabla \cdot \sigma + \mathbb{F} + \varrho \ddot{\mathbf{u}} = \mathbf{0} \\ \varepsilon = \frac{1}{2}\left(\nabla(\mathbf{u}) + \nabla(\mathbf{u})^T\right) & \text{, on } \Omega \\ \sigma = \mathbf{D}\sigma - \varepsilon_0) \end{cases} \\ \mathbf{u} = \bar{\mathbf{u}} & \text{, on } \Gamma_u \\ n\sigma = \mathbf{f} & \text{, on } \Gamma_u \end{cases} \tag{110}$$

Thermal domain:

$$\begin{cases} \nabla \cdot (-\mathbf{K} \cdot \nabla T) = g + c\varrho\dot{T} + T_0 \cdot \mathbf{D}\boldsymbol{\alpha}^T\dot{\varepsilon} & \text{, on } \Omega \\ \mathbf{T} = \bar{\mathbf{T}} & \text{, on } \Gamma_T \\ \mathbf{q} \cdot \mathbf{n} = \bar{\mathbf{q}} & \text{, on } \Gamma_q \end{cases} \tag{111}$$

4 COUPLED DISCRETIZED EQUILIBRIUM SYSTEM

After the discretization by the FEM, the equilibrium equation system reads:

$$\begin{cases} \mathbf{M}_u\ddot{\mathbf{U}} + \mathbf{K}_u\mathbf{U} - \mathbf{G}_u\mathbf{T} = \mathbf{P}_u \\ \mathbf{M}_T\dot{\mathbf{T}} + \mathbf{K}_T\mathbf{T} + \mathbf{G}_T^T\dot{\mathbf{U}} = \mathbf{P}_T \end{cases} \tag{112}$$

Where the products $\mathbf{G}_u\mathbf{T}$ and $\mathbf{G}_T^T\dot{\mathbf{U}}$ represents the coupling terms for the mechanical and thermal problem respectively.

5 MONOLITHIC APPROACH

The equilibrium system is cast in the form:

$$\begin{bmatrix} \mathbf{M}_u & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{bmatrix} \begin{bmatrix} \ddot{\mathbf{U}} \\ \mathbf{0} \end{bmatrix} + \begin{bmatrix} \mathbf{0} & \mathbf{0} \\ \mathbf{M}_T^T & \mathbf{M}_T \end{bmatrix} \begin{bmatrix} \mathbf{0} \\ \dot{\mathbf{T}} \end{bmatrix} + \begin{bmatrix} \mathbf{K}_u & -\mathbf{G}_u \\ \mathbf{0} & \mathbf{K}_T \end{bmatrix} \begin{bmatrix} \mathbf{U} \\ \mathbf{T} \end{bmatrix} = \begin{bmatrix} \mathbf{P}_u \\ \mathbf{P}_T \end{bmatrix} \rightarrow \tag{113}$$
$$\rightarrow \mathbf{M}_s\ddot{\mathbf{s}} + \mathbf{C}_s\dot{\mathbf{s}} + \mathbf{K}_s\mathbf{s} = \mathbf{P}_s$$

Then, the solution vector **S** is computed accordingly to the chosen algorithm. The simplicity of this approach is counterbalanced by the computational cost, since $\mathbf{M}_s$,$\mathbf{C}_s$ and $\mathbf{K}_s$ are sparse matrices.

6 STAGGERED APPROACH

The equilibrium equation system is solved separately for the mechanical and thermal part. At the time step *n*, it is computed the temperature distribution first, by using an initial guess vector for the velocity $\dot{\mathbf{U}}_0$. Then, it is computed the displacement field, substituting the temperature term previously obtained. Finally, a new velocity vector $\dot{\mathbf{U}}_1$ is computed and the convergence checked. This procedure is applied iteratively until the error is less than a fixed threshold. When the convergence is fulfilled, the algorithm goes on time step $n+1$.

ACRONYM

ANN	Artificial neural network.
RNN	Recurrent neural network.
MSE	Mean Square Error.
TS	Training set.
R	Regression.
NL-I	Non-linear interpolation.
GS	Generalization set.
L-I-E	Linear interpolation.
NL-I-E	Non-linear interpolation error.
TA	Training algorithm.
LF	Loss function.
NI	Number of iterations.
GD	Gradient descend.
QN	Quasi-Newton.
LM	levemberg-Marquardt.
DOF	Degree Of Freedom.
NT	Numer of training elements.
NP	Numer of perceptron.
BPTT	Back Propagation Through Time.
FEM	Finite Element Method.
MOR	Model Order Reduction.
LSTM	Long-Short-Term-Memory
GAN	generative adversarial network
AI	Artificial Intelligence
TED	Thermo-Elastic-Damping.
D	Displacement.
T	Temperature.

BIBLIOGRAPHY

[1] Federico Cheli and Giorgio Diana. *Advanced dynamics of mechanical systems*. 2015, pp. 1–818.

[2] Ray W. Clough and Joseph Penzien. *Dynamics of Structures, Third Edition*. Third Edit. Berkeley, CA: Computers and Structures, Inc., 1995.

[3] F Confalonieri, M Terraneo, and A Corigliano. "Domain decomposition and model order reduction approaches applied to the solution of the fully coupled thermo-mechanical problem in vibrating microsystems". In: (2013), pp. 1–23.

[4] Alberto Corigliano and Alberto Taliercio. *Meccanica Computazionale - Soluzione del problema elastico lineare*. Società Editrice Esculapio, 2005.

[5] S. Focardi, I. Massa, and A. Uguzzoni. *Fisica Generale - Termodinamica e fluidi*. II Edizion. C.E.A. Casa Editrice Ambrosiana, 2004.

[6] Crescenzio Gallo, Michelangelo De Bonis, and Albert Einstein. "Reti Neurali Artificiali Tutorial". In: (), pp. 1–29.

[7] Simon Haykin. *Neural Networks and Learning Machines*. Third edit. Ontario: Pearson Education, 2009.

[8] J. S. Hesthaven and S. Ubbiali. "Non-intrusive reduced order modeling of non-linear problems using neural networks". In: *Journal of Computational Physics* 363 (2018), pp. 55–78. URL: https://doi.org/10.1016/j.jcp.2018.02.037.

[9] Nam Ho Kim. *Introduction to nonlinear finite element analysis*. 2015, pp. 1–430.

[10] Leonard Meirovitch. *Elements of Vibration Analysis*. 1986.

[11] Azam Moosavi, Răzvan Ştefănescu, and Adrian Sandu. "Multivariate predictions of local reduced-order-model errors and dimensions". In: *International Journal for Numerical Methods in Engineering* 113.3 (2018), pp. 512–533.

[12] Derrick Nguyen and Bernard Widrow. "Improving the learning speed of 2 - layer neural networks by choosing initial values of the adaptive weights". In: *Joint conference on neural networks*. Stanford, 1990, 3:21–26.

[13] A Pavelka and A. Procházka. "Algorithms for initialization of neural network weights". In: *Sbornik prispevku 12. rocniku konference MATLAB 2004* 2 (2004), pp. 453–459.

[14] Robert A. Adams. *Calcolo Differenziale 2 - Funzioni di più variabili*. Quarta edi. Milano: C.E.A. Casa Editrice Ambrosiana, 2007.

[15] Steffen Rothe, Jan Henrik Schmidt, and Stefan Hartmann. "Analytical and numerical treatment of electro-thermo-mechanical coupling". In: *Archive of Applied Mechanics* 85.9-10 (2015), pp. 1245–1264.

[16] Riccardo Sacco. "Numerical methods for civil engineering - Notes of the course". In: (2013).

[17] Steven W. Smith. *Neural Networks*. 2013, pp. 451–480. ISBN: 9780131471399. DOI: 10.1016/b978-0-7506-7444-7/50063-7.

[18] Einar Strommen. *Structural Dynamics (Springer Series in Solid and Structural Mechanics, Vol.2)*. 2014, p. 521. URL: http://www.springer.com/gp/book/9783319018010.

[19] Ron Tenne et al. "Super-resolution enhancement by quantum image scanning microscopy". In: *Nature Photonics* 13.2 (2019), pp. 116–122. ISSN: 1749-4893. DOI: 10.1038/s41566-018-0324-z. URL: https://doi.org/10.1038/s41566-018-0324-z.

[20] M. Vaz and M. R. Lange. "Thermo-mechanical coupling strategies in elastic–plastic problems". In: *Continuum Mechanics and Thermodynamics* 29.2 (2017), pp. 373–383.

[21] Cima Vojtech. *Deep Learning - Concept and use cases*. Tech. rep. London: Bayncore - Intel A.I. Academy.

[22] Wikipedia Contributors. *Artificial neural network*. URL: https://en.wikipedia.org/w/index.php?title=Artificial%7B%5C_%7Dneural%7B%5C_%7Dnetwork%7B%5C&%7Doldid=889571712.

[23] Wikipedia Contributors. *Human brain*. 2019. URL: https://en.wikipedia.org/w/index.php?title=Human%7B%5C_%7Dbrain%7B%5C&%7Doldid=890268987.

[24] Wikipedia Contributors. *Machine learning*. URL: https://en.wikipedia.org/w/index.php?title=Machine%7B%5C_%7Dlearning%7B%5C&%7Doldid=889950774.

[25] Wikipedia Contributors. *Recurrent neural network*. URL: https://en.wikipedia.org/w/index.php?title=Recurrent%7B%5C_%7Dneural%7B%5C_%7Dnetwork%7B%5C&%7Doldid=889817533.