FACULTY
OF MATHEMATICS
AND PHYSICS
Charles University

BACHELOR THESIS

Ildefonso Padrón Peña

Artificial neural networks for macroeconomic data analysis

Department of Theoretical Computer Science and Mathematical Logic

Supervisor of the bachelor thesis:	doc. RNDr. Iveta Mrázová, CSc.
Study programme:	Computer Science
Study branch:	General Computer Science

Prague 2018

I would like to thank all who have supported me during the preparation of this work. Special thanks to my supervisor doc. RNDr. Iveta Mrázová, CSc for her guidance, inspiration and time spent on consultations.

Title: Artificial neural networks for macroeconomic data analysis

Author: Ildefonso Padrón Peña

Department: Department of Theoretical Computer Science and Mathematical Logic

Supervisor: doc. RNDr. Iveta Mrázová, CSc., Department of Theoretical Computer Science and Mathematical Logic

Abstract: The analysis and prediction of macroeconomic time-series is a factor of great interest to national policymakers. However, economic analysis and forecasting are not simple tasks due to the lack of a precise model for the economy and the influence of external factors, such as weather changes or political decisions. Our research is focused on Spanish speaking countries. In this thesis, we study different types of neural networks and their applicability for various analysis tasks, including GDP prediction as well as assessing major trends in the development of the countries. The studied models include multilayered neural networks, recursive neural networks, and Kohonen maps. Historical macroeconomic data across 17 Spanish speaking countries, together with France and Germany, over the time period of 1980-2015 is analyzed. This work then compares the performances of various algorithms for training neural networks, and demonstrates the revealed changes in the state of the countries' economies. Further, we provide possible reasons that explain the found trends in the data.

Keywords: clustering, classification, prediction, artificial neural networks, economic data.

Contents

Introduction **3**

1 Artificial neural networks **5**
- 1.1 Overview and motivation 5
- 1.2 Basic notions 5
- 1.3 Transfer functions 7
  - 1.3.1 Sigmoidal 7
  - 1.3.2 Tanh 8
  - 1.3.3 ReLU 8
  - 1.3.4 Linear transfer function 9
  - 1.3.5 Softmax 9
- 1.4 Notation 10
- 1.5 Feed forward algorithm 10
  - 1.5.1 Algorithm 11
- 1.6 Supervised learning and error function 11
- 1.7 Overfitting 12
  - 1.7.1 Cross-validation 12
  - 1.7.2 Confidence intervals 12
  - 1.7.3 Early stopping 12

2 Learning algorithms for MLP **14**
- 2.1 Overview 14
- 2.2 First order algorithms 14
  - 2.2.1 Back propagation 14
  - 2.2.2 Algorithm 15
  - 2.2.3 Back propagation with momentum 16
- 2.3 Second-order algorithms 16
  - 2.3.1 Levenberg-Marquardt 18
  - 2.3.2 Scaled Conjugate Gradient 18
- 2.4 Comparison of the algorithms 18

3 Recurrent neural networks **19**
- 3.1 Overview 19
- 3.2 Forward pass 20
- 3.3 Back propagation through time 21
  - 3.3.1 **V** deltas 21
  - 3.3.2 **W** deltas 22
  - 3.3.3 **U** deltas 23
- 3.4 Long-term dependencies problem 23
- 3.5 Long Short Term Memory 23
  - 3.5.1 LSTM Forward pass 25

4 Clustering **26**
4.1 Overview 26
4.2 Kohonen's neural network 26
4.2.1 The model 26
4.2.2 The training algorithm for Kohonen networks 27
4.3 Fuzzy C-means clustering 29
4.3.1 Algorithm 29
4.3.2 Number of clusters 29
4.4 A two-phase clustering approach 30

5 Case studies **31**
5.1 Function approximation 31
5.2 Clustering 32

6 Experiments **35**
6.1 Data 35
6.1.1 Introduction 35
6.1.2 Data description 36
6.1.3 Data preprocessing 41
6.1.4 Input & output 42
6.2 Data visualization 42
6.2.1 Set up 43
6.2.2 Analysis of the results 44
6.3 GDP Prediction 48
6.3.1 Criteria for picking the best model 48
6.3.2 Training algorithms 48
6.3.3 MLP network 48
6.3.4 RNN network 49
6.4 Analysis of the obtained results 53

Conclusion **55**

Bibliography **57**

List of Figures **59**

A Implementation and documentation **60**
A.1 Project overview 60
A.2 Requirements and installation 60
A.3 Running the data visualization experiments 61
A.4 Running the GDP forecasting experiments 62

Introduction

The analysis and prediction of macroeconomic time-series is a factor of great interest to politicians and policymakers. It is also important for the understanding and comparison of economic evolution of different countries over the history. However, economic forecasting is not a simple task because the models at hand may be inappropriate in unknown ways as the precise models of the economy itself are still not known, and some variables might have been measured inaccurately. It is thus usually infeasible to create a precise analytical model for these tasks. For these purposes, it is customary to resort to black-box models where the internal workings of the system is rather disregarded, and the aim is only to reproduce the systems behaviour. In this work we analyze and use such models; classic and recurrent neural networks, together with unsupervised neural networks focused on unsupervised learning and clustering.

A significant part of this work is thus devoted to the analysis and comparison of different neural network models and their respective training. Both the analysis and mutual comparison of the investigated approaches are oriented towards the task of time-series forecasting. We consider multilayer perceptrons (MLP) with various architectures, together with several pre-selected first and second order training algorithms, and compare their performance with those of the recurrent neural networks (RNN).

An essential part of this work is dedicated also to visualization of the time-series development. For this purpose we employ a special type of unsupervised neural network, the self-organizing map (SOM), together with another known clustering method, the fuzzy c-means clustering algorithm.

The data that we have used in these experiments is publicly available from the World Bank organization. The main focus of this work was put on the analysis of Spanish speaking countries[1], but we also considered France and Germany as a reference. First we had, however, to preprocess the data, and visualize it by means of a 2-dimensional SOM and the fuzzy c-means clustering algorithm. Afterwards, we have examined various architectures of neural networks and explored their applicability to the prediction of the GDP of a country. The first type of neural networks we will use are MLPs. In this context, we will test various architectures of the networks and compare their relative performance in connection with the applied training algorithms.

One problem that may arise when using MLPs for time-series forecasting is the necessity to capture the long term dependencies that might be inherently present in the data. For the prediction task the MLP only looks at data from one time period. On the other hand, RNNs are networks that can handle long term dependencies by looking at a sequence of data and learning how it changes over time. To confirm this observation, we will use RNNs and compare their performance with the results obtained by the MLPs.

The thesis is divided into a theoretical and an experimental part. Within the framework of the theoretical part, we introduce basic notions used throughout the

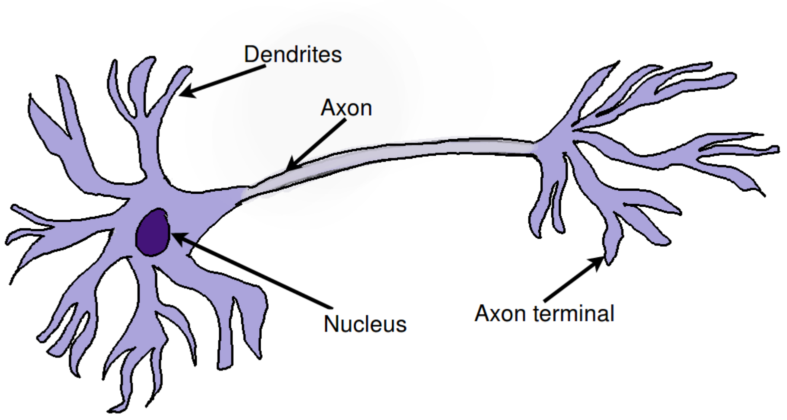
[1]The studied Spanish speaking countries are Argentina, Bolivia, Chile, Colombia, Cuba, Dominican Republic, Ecuador, El Salvador, Honduras, Mexico, Panama, Paraguay, Peru, Puerto Rico, Spain, Uruguay and Venezuela.

text in Chapter 1. In Chapter 2, we describe several well known algorithms for training multi-layered neural networks of the perceptron type. Recurrent neural networks (RNN) are introduced in Chapter 3, as well as a special type of a training algorithm for RNNs, the Back Propagation Through Time (BPTT). Then, in Chapter 4, we describe SOMs and the fuzzy c-means clustering algorithm. The experimental part starts with Chapter 5, where the models discussed in the theoretical part are applied to solve small and easy artificially generated example problems. The following chapter describes the real-world data used in our experiments, as well as its preprocessing. Then, data visualization for the time-series is explored with a SOM neural network and the fuzzy c-means algorithm. Afterwards, in the same chapter, we run the experiments for the GDP prediction with different neural network models and compare the obtained results. Then, we summarize the results obtained in the experiments and discuss further improvements that could be done to this work.

1. Artificial neural networks

1.1 Overview and motivation

An artificial neural network (ANN) is a biologically inspired computational model. The building block of ANN's is a formal neuron called the perceptron, which is a mathematical model of a biological neuron (Figure 1.1). An ANN is formed by a set of mathematical neurons which are mutually interconnected.



A graphical representation of a biological neuron and its main parts.

Figure 1.1: **Biological neuron**

1.2 Basic notions

The model of a perceptron is motivated by the biological neuron, which is a very specialized cell that forms the basis of the nervous system in biological organisms. Neurons are unique in that they transmit signals throughout the body. In its formal counterpart, both the signals going in and out of a perceptron are real numbers. We can think of a perceptron as function from $\mathbb{R}^n$ to $\mathbb{R}$, where n is the dimension of the input.

We will consider the perceptron to be the most simplified version of an artificial neural network. What is usually done, however, is to connect several of these units to form more complex (and often more useful) mathematical objects. The motivation for this construction arises again from a biological notion, the brain. We can understand this system as a function from $\mathbb{R}^n$ to $\mathbb{R}^m$, where n and m are the input and output dimensions, respectively.

These notions are formally defined bellow:

Definition 1.2.1. Perceptron

A perceptron is a triple $(t, \vec{w}, f)$, where:

- $t \in \mathbb{R}$, is the threshold,
- $\vec{w} \in \mathbb{R}^n$, is the weight vector, and

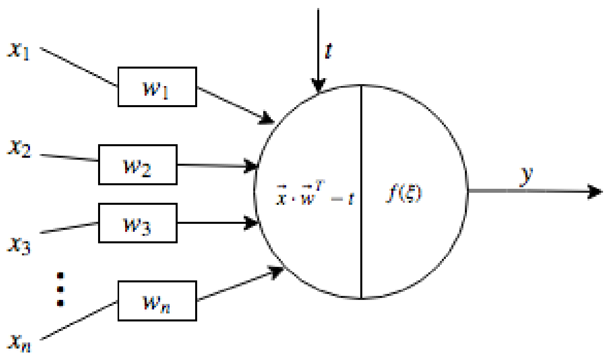
- $f : \mathbb{R} \to \mathbb{R}$, is the transfer function, also known as the activation function.

The value $\xi = \vec{x} \cdot \vec{w}^T - t$ is called the inner potential of the neuron.

Then, for a given input $\vec{x} \in \mathbb{R}^n$, the perceptron computes its output y according to:

$$y = f(\xi) \tag{1.1}$$

We can connect several perceptrons together, forming what is known as an artificial neural network.



In the diagram, the output y of the perceptron is computed from the input $\vec{x} = (x_1, x_2, \ldots, x_n)$, the weight vector $\vec{w} = (w_1, w_2, \ldots, w_n)$, and threshold t.

Figure 1.2: **Diagram of a perceptron**

Definition 1.2.2. Artificial neural network

An artificial neural network is a 6-tuple $M = (N, C, I, O, x, t)$, where:

- N is a finite non-empty set of neurons,
- $C \subset N \times N$ is a non-empty set of connections among the neurons,
- $I \subset N$ is a non-empty set of input neurons,
- $O \subset N$ is a non-empty set of output neurons,
- $w : C \to \mathbb{R}$ is a weight function, and
- $t : N \to \mathbb{R}$ is a threshold function.

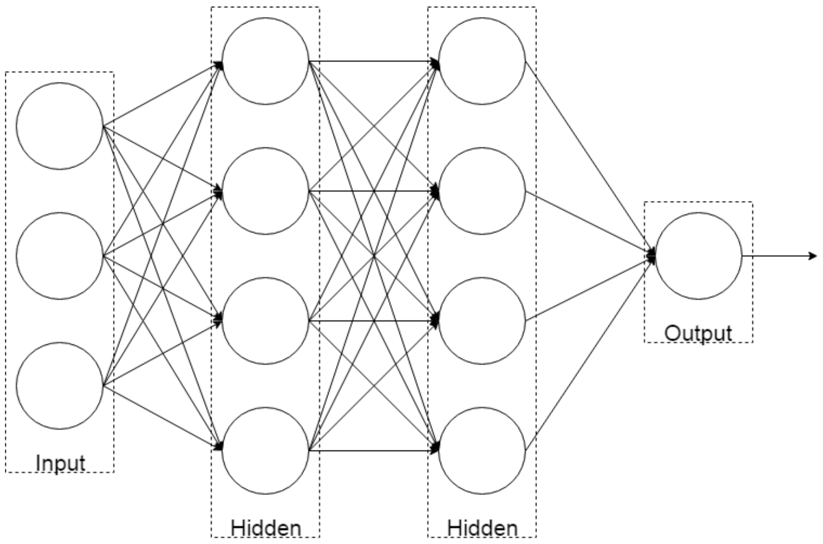
The pair (N, C) is called the inter-connection graph of M.

Different types of ANN can be defined depending on the types of inter-connections that are allowed in the model.

Definition 1.2.3. Multilayer perceptron

A multilayer perceptron is a neural network with a directed acyclic inter-connection graph. Its set of neurons consists of a sequence of $l + 2$ pairwise disjoint non-empty subsets called layers. There are three types of layers:

- The first layer is called the input layer. The neurons in this layer have no predecessors in the inter-connection graph, and their output value is equal to their input value.
- The last layer is called the output layer. The neurons in this layer have no successor in the inter-connection graph.
- All the remaining layers are called hidden layers.



In the diagram we observe a multilayer perceptron with two hidden layers which have 4 neurons each. The input and output dimensions of this MLP are 3 and 1, respectively.

Figure 1.3: **Multilayer perceptron diagram**

1.3 Transfer functions

The transfer function is a key component of the neurons. Different types of transfer functions can be used [4], and depending on the task and implementation at hand, some may perform better than others. Here we will describe a few commonly used transfer function, some of which will be used later in our experiments.

1.3.1 Sigmoidal

The sigmoid, or logistic transfer function shown in Figure 1.4, is a mathematical function having a characteristic S-shaped curve. Its range is $(0, 1)$, thus in some use cases where the desired output of the network is outside of this range, some additional computations, such as normalization, are required. The sigmoid transfer function is defined by the following formula:

$$f(\xi) = \frac{1}{1 + e^{-\lambda\xi}} \tag{1.2}$$

And its derivative is given by:

$$\frac{\partial f}{\partial \xi} = \lambda f(\xi)(1 - f(\xi)) \tag{1.3}$$

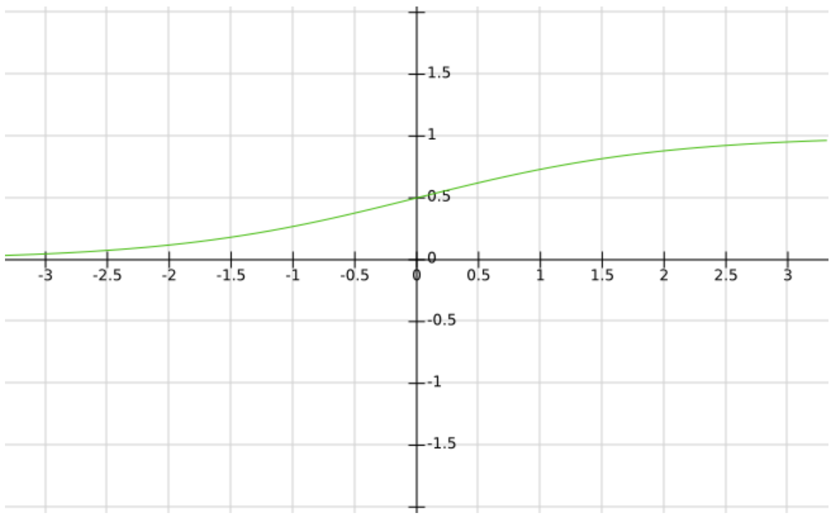


Figure 1.4: **Sigmoid transfer function**

1.3.2 Tanh

The hyperbolic tangent (tanh) function (Figure 1.5) also has a characteristic S-shaped curve, has a range of $[-1, 1]$, and it is defined by the following formula:

$$f(\xi) = tanh(\xi) = \frac{1 - e^{-2\xi}}{1 + e^{-2\xi}} \tag{1.4}$$

And its derivative is:

$$\frac{\partial f}{\partial \xi} = 1 - f^2(\xi) \tag{1.5}$$

1.3.3 ReLU

The rectified linear unit (ReLU) transfer function Figure 1.6 has become very popular within the last few years. One of its major benefits is the reduced likelihood of the gradient to vanish. The ReLU function is defined by:

$$f(\xi) = max(0, \xi) \tag{1.6}$$

Its derivative is:

$$\frac{\partial f}{\partial \xi} = \begin{cases} 1, \text{ for } \xi > 0 \\ 0, \text{ otherwise} \end{cases} \tag{1.7}$$

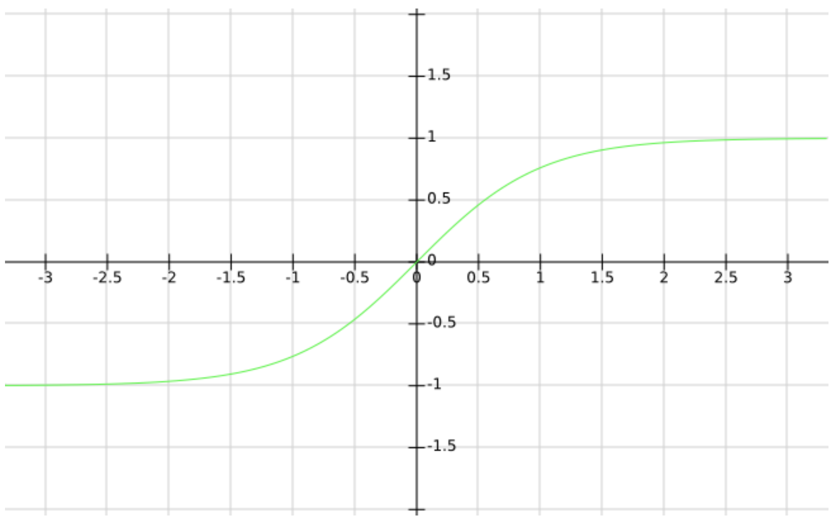


Figure 1.5: **Hyperbolic tangent transfer function**

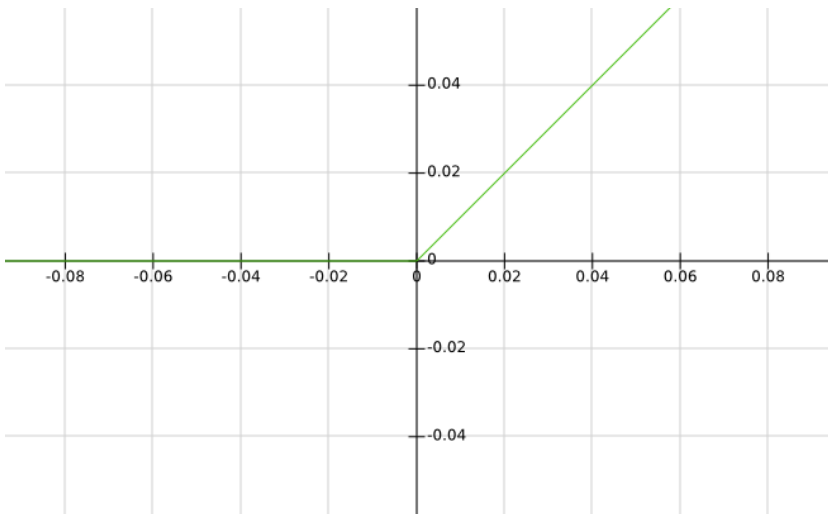


Figure 1.6: **ReLU transfer function**

1.3.4 Linear transfer function

The linear function (Figure 1.7) is sometimes used in the output layer instead of the sigmoid transfer function, because it is not bounded, and the network can thus output values outside of the range of the sigmoid function.

1.3.5 Softmax

The softmax function is a generalization of the logistic transfer function which transforms an n-dimensional vector $\vec{v} \in \mathbb{R}^n$ to an n-dimensional vector $\vec{\sigma}(\vec{v}) \in (0,1)^n$, and all the entries of $\vec{\sigma}(\vec{v})$ add up to 1. The function is given by Equation 1.8.

$$\vec{\sigma}(\vec{v})_i = \frac{e^{\vec{v}_i}}{\sum_{j=1}^{n} e^{\vec{v}_j}}, \text{ for } 1 \leq i \leq n \tag{1.8}$$

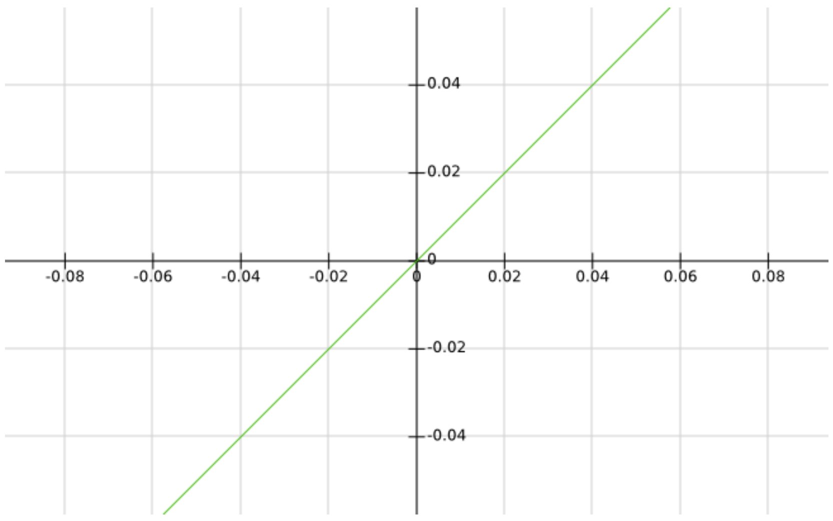


Figure 1.7: **Linear transfer function**

1.4 Notation

From now onwards, we will use a notation that allows us to include the neuron's threshold t into the weight vector $\vec{w}$. For this purpose, we define the extended input and weight vectors as $\vec{x}_E = (x_1, x_2, \ldots, x_n, -1)$, and $\vec{w}_E = (w_1, w_2, \ldots, w_n, t)$. where $\vec{x} = (x_1, x_2, \ldots, x_n)$, and $\vec{w} = (w_1, w_2, \ldots, w_n)$ are the original input and weight vectors and t is the threshold. Because $\vec{x}_E \cdot \vec{w}_E^T = \vec{x} \cdot \vec{w}^T - t = \xi$, in the remaining of the text we will refer to $\vec{x}_E$ and $\vec{w}_E$ simply as $\vec{x}$ and $\vec{w}$.

Since there are a lot of weights to take into account, we will use the following schema for indexing the weights in our network; w_k^{ij} denotes the weight connecting the k-th neuron in layer $i-1$ with the j-th neuron in layer i. Further, the weights might depend on time t, in this case we denote it by $w_k^{ij}(t)$.

1.5 Feed forward algorithm

In this section, we will show how to compute the output of a multilayer perceptron given a fixed input. Later on we will use this to train a neural network. An MLP consists of two or more layers of neurons where the signal flow is from left to right, that is, at the beginning the input vector is fed into the input layer, the output of the input layer is computed and is passed on as input to the next layer. This process continues until the last layer yields its output, which is the output of the network.

Assume we have an MLP with l layers. For a given input $\vec{x} \in \mathbb{R}^n$, the output of the input layer is $\vec{x}$. For the layer i, we assume that the output of the previous layer $\vec{y}^{i-1} \in \mathbb{R}^{m_{i-1}}$ has been already computed, the vector $\vec{y}^i = (y_1^i, \ldots, y_{m_i}^i)$ is then determined by:

$$y_j^i = f(\vec{y}^{i-1} \cdot \vec{w}^{ij}) \tag{1.9}$$

Where $\vec{w}^{ij}$ is the extended weight vector from the i-th layer and j-th neuron. The output of the MLP is then $\vec{y}^l$.

1.5.1 Algorithm

With this knowledge at hand, we can construct an algorithm for determining the output of an MLP for a given input. We assume that the MLP has l layers, and that the number of neurons per layer is stored in an array $L := [n_1, n_2, \ldots, n_l]$.

Algorithm 1: Feed forward

```
input : A multilayer perceptron $M$
        An input vector $\vec{x} \in \mathbb{R}^{n_1}$
output: A vector $\vec{y} \in \mathbb{R}^{n_l}$
// The input is also the output of the input layer
$\vec{y} := \vec{x}$;
$temp := []$;
for $i \leftarrow 2$ to $net.l$ do
    // Setting the vector to the correct size
    $temp := Vector(L_i)$;
    for $j \leftarrow 1$ to $L_i$ do
        $temp_j := f(\vec{y} \cdot \vec{w}^{ij})$;
    end
    // Move the current processed layer forward
    $\vec{y} := temp$;
end
return $\vec{y}$;
```

1.6 Supervised learning and error function

Supervised learning is the subfield of machine learning that aims at inferring the parameters of a function which maps inputs to appropriate outputs based on example input-output pairs. This set of examples that are used for the inference is called the training set. A supervised learning algorithm takes the training data as its input and produces a function, which can be used to map new inputs to their outputs.

Measuring the performance of a supervised learning algorithm is done by means of testing data, a set of input-output pairs, and an error function.

The mean squared error (MSE) function is the error function of choice for our experiments. It is defined by

$$E(W) = \frac{1}{2} \sum_{p=1}^{t} \sum_{j=1}^{m} (\vec{y}_j^p - \vec{o}_j^p)^2 \tag{1.10}$$

Where, W represents the weights and thresholds of the network,
$T = \{(\vec{x}^i, \vec{o}^i) | \vec{x}^i \in \mathbb{R}^n, \vec{o}^i \in \mathbb{R}^m, 1 \leq i \leq t\}$ is the testing data, and
$\vec{y}^p \in \mathbb{R}^m$ is the output of the network for the input $\vec{x}^p$.

The purpose of supervised learning is the minimization of the error function on testing sets.

1.7 Overfitting

Overfitting is a term used in statistics and machine learning that refers to a model that fits the training data set too well. It occurs when a model learns the details in the training data to the extent which negatively affects the performance of the model on new data. In other words, the model loses its ability to generalize.

To prevent this phenomenon, several techniques might be applied. In the following sections, we describe some methods that we used for our experiments.

1.7.1 Cross-validation

Cross-validation is a model evaluation technique for assessing how the results of a statistical analysis will generalize to an independent data set.

The holdout method is the simplest kind of cross validation. The data set is divided into two sets, called the training set and the testing set. The training algorithm fits the model to the training set only. Then the model is evaluated on the testing set. The evaluation of the model may heavily depend on which data end up in the training set and which end up in the testing set, therefore the evaluation may be significantly different depending on how the division of the data was performed.

K-fold cross validation is one of the options to improve the holdout method. The data set is divided into k disjoint subsets, and the holdout method is logically repeated k times. Each time, one of the k subsets is used as the testing set and the other k-1 subsets are used together as the training set. Then the average error over all k runs is computed. The advantage of this method is that the way in which we divide the data is less significant. Since some part of the data could be of great importance and it might not appear in the training set because of the division phase.

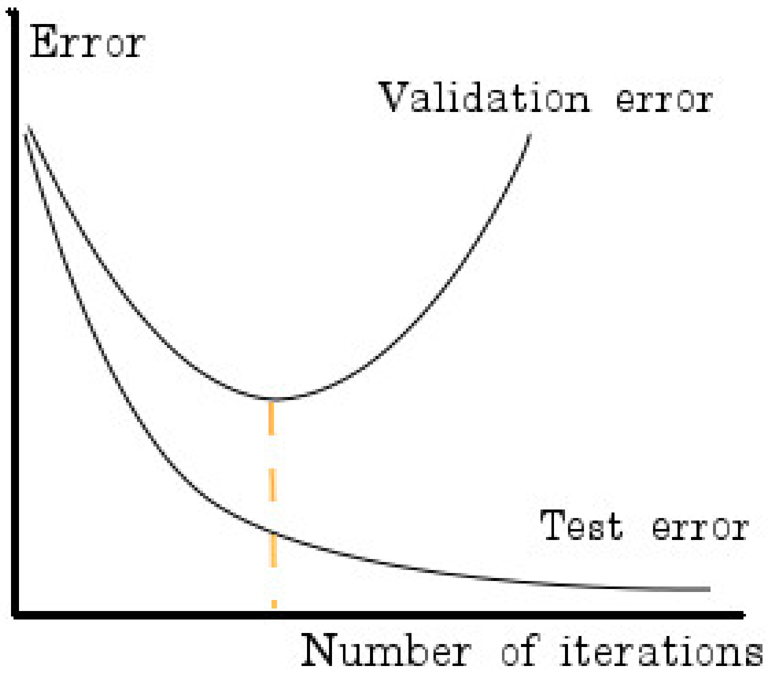
1.7.2 Confidence intervals

In statistics, a confidence interval is a type of interval estimate, computed from the statistics of observer data, that might contain the true value of an unknown population parameter. The interval has an associated confidence level that represents the proportion of possible confidence intervals that contain the true value of the unknown population parameter.

For our experiments, we used a confidence interval of 90% to estimate the mean error on the testing set.

1.7.3 Early stopping

This technique prevents overfitting by keeping track of the history of the performances of the model over the data. The early stopping technique divides the data set into two disjoint subsets, the training set and the validation set. At the end of every (or every N) iteration of the learning algorithm the performance of the model on the validation set is evaluated. If the current model outperforms the previous best model, it replaces the best model by the current one. The final model is the one with the best performance on the validation set.



Graphical representation of the early stopping condition.

Figure 1.8: **Early stopping**

2. Learning algorithms for MLP

2.1 Overview

In the previous section we described how to compute the output of the network for a given pattern, and how to compute the error value for a given finite set of input-output pairs. The next topic at hand is the minimization of this error with respect to the weights and thresholds on some training set.

In each of the optimization steps, we want to update the weights of the entire network according to:

$$w_k^{ij}(t+1) = w_k^{ij}(t) + \Delta_E w_k^{ij}(t) \tag{2.1}$$

The computation of $\Delta_E w_k^{ij}$ is where the algorithms explained in this section differ.

We will consider two classes of training algorithms, both of which have their advantages and disadvantages. First, we take a look at a subset of first order algorithms (Back Propagation [3] and Back Propagation with Momentum [20]), and then we consider some selected second order algorithms (Levenberg-Marquardt [6] and Scaled Conjugate Gradient [18]).

2.2 First order algorithms

2.2.1 Back propagation

First order algorithms take a step $\Delta_E w_k^{ij}$ (i.e. update the weights w_k^{ij} of the interconnections between neurons j and k) in the direction of the negative gradient of the error E with respect to the weights, that is:

$$\Delta_E w_k^{ij} = -\alpha \frac{\partial E}{\partial w_k^{ij}} \tag{2.2}$$

Where $\alpha \in \mathbb{R}$ is the learning rate, and it is usually set to values in the range $[0, 1]$. There are some algorithms (Silva & Almeida , Delta bar delta [17], super SAB [24], etc.) that have adaptive learning rates.

For the computation of $\Delta_E w_k^{ij}$, we distinguish two cases; one for the neurons of the output layer, and the other one for the neurons of the hidden layers.

First, we derive the formula for the neurons from the output layer.

$$\begin{aligned}
\Delta_E w_k^{ij} &= -\frac{\partial E}{\partial w_k^{ij}} \\
&= -\frac{\partial E}{\partial y_j^i}\frac{\partial y_j^i}{\partial \xi_j^i}\frac{\partial \xi_j^i}{\partial w_k^{ij}} \\
&= -\frac{\partial E}{\partial y_j^i}\frac{\partial y_j^i}{\partial \xi_j^i}\frac{\partial}{\partial w_k^{ij}}(\vec{y}^{(i-1)} \cdot (\vec{w}^{ij})^T) \\
&= -\frac{\partial E}{\partial y_j^i}\frac{\partial y_j^i}{\partial \xi_j^i} y_k^{(i-1)} \\
&= -\frac{\partial E}{\partial y_j^i} f'(\xi_j^i) y_k^{(i-1)} \\
&= -(y_j^i - o_j) f'(\xi_j^i) y_k^{(i-1)} \\
&= \delta_j^i y_k^{i-1}
\end{aligned} \tag{2.3}$$

Next, for the neurons in the hidden layer l we get:

$$\begin{aligned}
\Delta_E w_k^{lj} &= -\frac{\partial E}{\partial w_k^{lj}} \\
&= -\left(\sum_{\alpha=1}^{n_{l+1}} \frac{\partial E}{\partial \xi_\alpha^{(l+1)}}\frac{\partial \xi_\alpha^{(l+1)}}{y_j^l}\right)\frac{y_j^l}{\partial \xi_j^l} y_k^{(l-1)} \\
&= -\left(\sum_{\alpha=1}^{n_{l+1}} \frac{\partial E}{\partial \xi_\alpha^{(l+1)}}\frac{\partial}{y_j^l}(\vec{y}^l \cdot (\vec{w}^{(l+1)\alpha})^T)\right)\frac{y_j^l}{\partial \xi_j^l} y_k^{(l-1)} \\
&= -\left(\sum_{\alpha=1}^{n_{l+1}} \frac{\partial E}{\partial \xi_\alpha^{(l+1)}} w_j^{(l+1)\alpha}\right)\frac{y_j^l}{\partial \xi_j^l} y_k^{(l-1)} \\
&= \left(\sum_{\alpha=1}^{n_{l+1}} \delta_\alpha^{l+1} w_j^{(l+1)\alpha}\right) f'(\xi_j^l) y_k^{(l-1)} \\
&= \delta_j^l y_k^{(l-1)}
\end{aligned} \tag{2.4}$$

It is useful to compute the delta δ_i^l values for every neuron i in layer l, and from the formulae above we can compute them in the following manner:

Let neuron j be a neuron in layer l, then

$$\delta_j^l = \begin{cases} (o_j - y_j^l) f'(\xi_j^i), \text{ for output neuron j} \\ \left(\sum_{\alpha=1}^{n_{l+1}} \delta_\alpha^{l+1} w_j^{(l+1)\alpha}\right) f'(\xi_j^l), \text{ for hidden neuron j} \end{cases} \tag{2.5}$$

2.2.2 Algorithm

With this, we can construct an algorithm (Algorithm 2) that would update the weights of the MLP such that the error function on the training data set will decrease.

Algorithm 2: Back propagation single pass

```
input: A multilayer perceptron $M$
       An input vector $\vec{x} \in \mathbb{R}^{n_1}$
       The desired output vector $\vec{o}$ for the given input
$\vec{y^l} := feedForward(M, \vec{x})$;
// For every layer, starting from output layer
for $i \leftarrow l$ to 2 do
    // For every neuron in the layer
    for $j \leftarrow 1$ to $L_i$ do
        // output layer
        if $i == l$ then
            $\delta_j^i := (o_j - y_j^l) f'(\xi_j^i)$ ;
        // hidden layer
        else
            $\delta_j^i := \left(\sum_{k=1}^{L_{i+1}} \delta_k^{(i+1)} w_j^{(i+1),k}\right) f'(\xi_j^i)$ ;
        end
        for $m \leftarrow 1$ to $L_{i-1}$ do
            // Computing increments of weights
            $\Delta_E w_m^{ij} = \delta_j^i \cdot y_m^{i-1}$;
        end
    end
end
Adjust wieghts of the network according to $\Delta_E w$'s. It is assumed that the
  number of layers is $l$, and the number of neurons in the $i$-th layer is stored in
  $L_i$.
```

2.2.3 Back propagation with momentum

Adding a momentum term to the $\Delta_E w_k^{ij}$ value from the formulae above improves the efficiency of the BP algorithm in some problem instances. If only the gradient of the error is taken into account, wide oscillations of the search process might occur [19].

We then get the expression:

$$\Delta_E w_k^{ij}(t+1) = -\alpha \frac{\partial E}{\partial w_k^{ij}}(t) + \mu \Delta_E w_k^{ij}(t) \tag{2.6}$$

Where $\alpha \in \mathbb{R}$ is the learning rate, and $\mu \in \mathbb{R}$ is the momentum rate, usually set to a value from the range $[0, 1]$.

2.3 Second-order algorithms

In order to simplify the notation, in this section we will denote

$$\vec{w} = (\ldots, w_k^{ij}, w_{k+1}^{ij}, \ldots, w_{L_{i-1}}^{ij}, w_k^{i,(j+1)}, \ldots, w_{L_{i-1}}^{i,L_i}, \ldots) \tag{2.7}$$

For weight adjustment, second order algorithms consider more information about the shape of the error function than just the gradient. If we also consider

Algorithm 3: Back propagation

```
input: A multilayer perceptron $M$
       Training data

Initialize the weights of the network $M$ to small random numbers
while not satisfied with the error value do
    $[\vec{x}, \vec{o}]$:=some pair from the training set;
    $backPropSingle(M, \vec{x}, \vec{o})$
end
```

the curvature of the error an improved performance can be obtained [22]. In second order algorithms, a quadratic approximation of the error function is used. With a second order Taylor series approximation we get

$$E(\vec{w} + \vec{h}) \approx E(\vec{w}) + \nabla E(\vec{w})^T \vec{h} + \frac{1}{2}\vec{h}^T \nabla^2 E(\vec{w})\vec{h} \tag{2.8}$$

Where $\nabla^2 E(\vec{w})$ is the $n \times n$ Hessian matrix of second order partial derivatives:

$$\nabla^2 E(\vec{w}) = \begin{bmatrix} \frac{\partial^2 E(\vec{w})}{\partial w_1 \partial w_1} & \cdots & \frac{\partial^2 E(\vec{w})}{\partial w_1 \partial w_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial^2 E(\vec{w})}{\partial w_n \partial w_1} & \cdots & \frac{\partial^2 E(\vec{w})}{\partial w_n \partial w_n} \end{bmatrix} \tag{2.9}$$

If we differentiate with respect to $\vec{h}$ the expression given by the Taylor series approximation we obtain

$$\nabla E(\vec{w} + \vec{h})^T \approx \nabla E(\vec{w})^T + \vec{h}^T \nabla^2 E(\vec{w}) \tag{2.10}$$

Now, since we are looking for a minimum, we can demand the above expression to be zero, thus obtaining

$$\vec{h} = -(\nabla^2 E(\vec{w}))^{-1} \nabla E(\vec{w}), \tag{2.11}$$

This means that we can solve the minimization problem in one step, provided that the error function is quadratic and that the inverse of the Hessian matrix was previously computed [19].

Newton's methods make iterative use of the above equation, and compute the new weights $\vec{w}(t+1)$ according to

$$\vec{w}(t+1) = \vec{w}(t) - (\nabla^2 E(\vec{w}(t)))^{-1} \nabla E(\vec{w}(t)) \tag{2.12}$$

However, computing the Hessian matrix might be a computationally demanding task, thus there exist algorithms which make use of an approximation of this matrix.

Pseudo-Newton methods are a family of algorithms that rely on a simplification of the Hessian matrix. In this simplification, the non-diagonal elements of the matrix are set to zero, whereas the diagonal entries are computed as $\frac{\partial^2 E(\vec{w})}{\partial w_i^2}$.

2.3.1 Levenberg-Marquardt

Levenberg-Marquardt algorithm is a second order learning algorithm which combines both the gradient and Newton's methods. This algorithm makes an approximation of the Hessian matrix by

$$H \approx J^T J + \lambda I \tag{2.13}$$

Where J is the Jacobian matrix, whose entries are the first-order partial derivatives of the error function with respect to the weights, λ is always positive, called combination coefficient, and I is the identity matrix.

Determining the Jacobian matrix is computationally less expensive than the calculation of the Hessian matrix.

The weights adjustment rule is then given by

$$\vec{w}(t+1) = \vec{w}(t) - (J^T J + \lambda I)^{-1} \nabla E(\vec{w}(t)) \tag{2.14}$$

2.3.2 Scaled Conjugate Gradient

The Scaled Conjugate Gradient (SCG) algorithm [18] chooses the search direction and the step size for the weights also by taking into consideration second order approximations of the error function. The algorithm is based upon a class of optimization techniques well known in numerical analysis as the Conjugate Gradient Methods [5] [8].

In some particular experiments, Martin F. Møller [18] found the SCG to be at least an order of magnitude faster than BP.

2.4 Comparison of the algorithms

In this chapter we have compared different types of learning algorithms for neural networks. A natural question that arises is; which one to apply for a given task? The answer to this question might depend on the size of the network, and on space and time limitations.

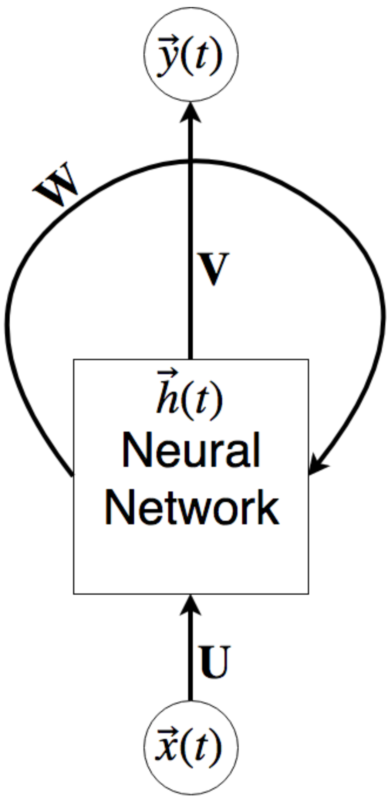
As [22] indicates, second order algorithms outperform first order algorithms in the task of forecasting share rates data. However, in the Levenberg-Marquardt algorithm, the Hessian matrix has to be stored, and this might be a limitation when we are dealing with big networks. When comparing the second order algorithms explained in this chapter, [1] showed that Levenberg-Marquardt algorithm had a better accuracy than SCG. But, the Scaled Conjugate Gradient algorithm performed better in terms of speed.

Overall, we can see that the choice of the algorithm highly depends on the task at hand and on the space and time limitations that we might have.

3. Recurrent neural networks

3.1 Overview

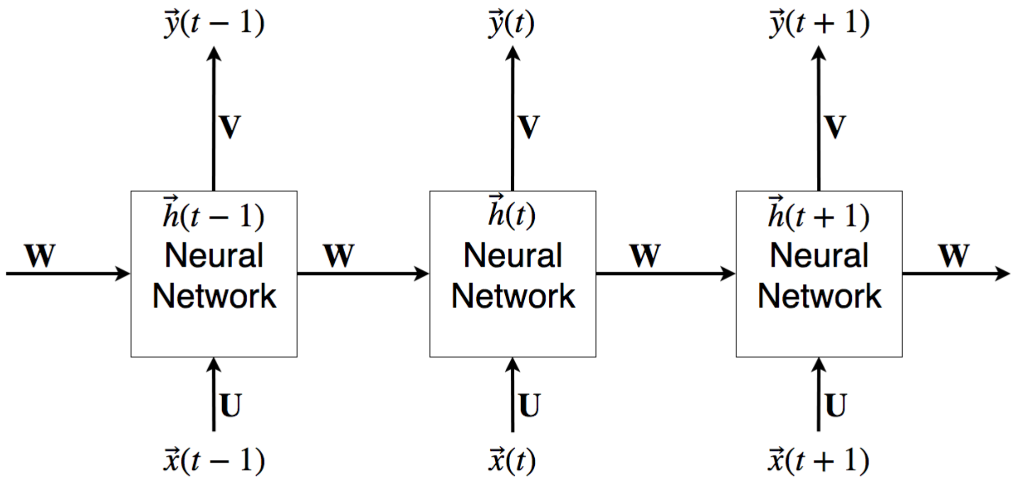
Recurrent neural networks (RNNs) [15], are a family of neural networks designed for the processing of sequential data. The interconnection graph of RNNs is allowed to have cycles, that carry the hidden state of the network at time t, denoted as $\vec{h}(t)$, to the next computation in time. We can see a graphical representation of an RNN in Figure 3.1. Sometimes these looping features of RNNs can be confusing, so it is easier to implement the models by means of unrolling (or unfolding) the network. By unfolding we simply mean that we write out the network for the entire sequence. We can see an illustration of this process in Figure 3.2. There are many architectures of RNNs such as Echo State networks [12], Bidirectional RNNs [16], Hopfield networks [11] and Long Short-Term Memory networks [10]. Since in this chapter we are dealing mostly with sequences of vectors, we will denote $\vec{x}(t)$ as the input vector $\vec{x}$ at time t, similarly for the output of the network $\vec{y}(t)$, where t goes from start time $t = 1$ to end time at $\tau \in \mathbb{N}$.



In the figure, **U**, **V** and **W** are the input-to-hidden, hidden-to-output and hidden-to-hidden weight matrices respectively, and $\vec{x}(t)$, $\vec{h}(t)$, $\vec{y}(t)$ denote the input, hidden state and output vectors at time t.

Figure 3.1: **Graphical representation of an RNN.**

The feed-back provided to the neurons by means of the cycles on the RNNs makes them able to memorize relevant information about the data over time. Standard MLPs can only map the input vectors to output vectors. RNNs on the



An unfolded version of the RNN shown in Figure 3.1 for time steps $t-1$, t, and $t+1$.

Figure 3.2: **Graphical representation of an unfolded RNN.**

other hand, can map also the entire input history of vectors to the output vectors. In order to gain similar information with MLPs, we would have to provide them with the entire sequence of inputs, which dramatically increases the number of parameters to train with. This feed-back feature makes RNNs more powerful when it comes to approximating sequences of data.

3.2 Forward pass

In this section, we will see how to compute the outputs $(\vec{y}(1), \vec{y}(2), \ldots, \vec{y}(t))$ of an RNN for a given input sequence $(\vec{x}(1), \vec{x}(2), \ldots, \vec{x}(t))$. We assume that the weight matrices $\mathbf{U}$, $\mathbf{V}$ and $\mathbf{W}$ are given. We also need to preset an initial hidden state of the network $\vec{h}(0)$ which will be assumed to be the zero vector $\vec{0}$.

Essentially, the output of an RNN at time t (i.e., $\vec{y}(t)$) is a function of the hidden state $\vec{h}(t)$ of the RNN at time t, which in turn is a function of the input $\vec{x}(t)$ at time t and the previous hidden state $\vec{h}(t-1)$. So the first step is to compute $\vec{h}(t)$, and with it, we compute $\vec{y}(t)$.

For the computation of $\vec{h}(t)$, we use Equation 3.1.

$$\vec{h}(t) = f(\mathbf{W}\vec{h}(t-1) + \mathbf{U}\vec{x}(t)), \text{ where } 1 \leq t \leq \tau \tag{3.1}$$

In this text we assume that f is the hyperbolic tangent, specified in Equation 1.4, and applied component-wise to each of the neurons.

After $\vec{h}(t)$ has been computed, we obtain the output of the network at time t with Equation 3.2.

$$\vec{y}(t) = softmax(\mathbf{V}\vec{h}(t)) \tag{3.2}$$

where the $softmax$ function is specified in Equation 1.8.

3.3 Back propagation through time

Training RNNs to approximate functions follows the same principles of gradient descent that we saw in Chapter 2. However, the training is done in the unfolded graph of the network, where the backward propagation pass moves from the right to the left in the graph. Then, the computed gradients at every time step are added together. The back-propagation algorithm applied to the unfolded graph is called back-propagation through time (BPTT).

Let $(\vec{x}(1), \vec{x}(2), \ldots, \vec{x}(\tau))$ be an input sequence, $(\vec{o}(1), \vec{o}(2), \ldots, \vec{o}(\tau))$ the desired output sequence for the inputs, and $(\vec{y}(1), \vec{y}(2), \ldots, \vec{y}(\tau))$ the outputs produced by the RNN. Then the error $E^{log}((\vec{o}(1), \vec{o}(2), \ldots, \vec{o}(\tau), (\vec{y}(1), \vec{y}(2), \ldots, \vec{y}(\tau)),$ E^{log} for short, is given as the negative log-likelihood specified by Equation 3.3.

$$E^{log} = -\frac{1}{\tau} \sum_{t \in \{1,\ldots,\tau\}} \vec{o}(t) \cdot \log \vec{y}(t) \tag{3.3}$$

For the sake of computational ease later on, we will define the error function E_t^{log} for time $1 \leq t \leq \tau$ as shown in Equation 3.4.

$$E_t^{log} = -\vec{o}(t) \cdot \log \vec{y}(t) \tag{3.4}$$

Note that the log functions in Equation 3.3 and Equation 3.4 are applied element-wise.

Now we have all that we need in order to compute the gradient of the error E_t^{log} with respect to all the parameters of our RNN (**U**, **V**, **W**). In the following sections we derive the partial derivatives of E_t^{log} with respect to all of the parameters. Since our error function 3.3 is simply a summation of the E_t^{log}'s. We can sum up the values that we have calculated to get the partial derivatives of E^{log}.

It is worth mentioning that, for the sake of computational ease, the error function that we used in the derivation of the formulae is different from the one used in the experiments. For the latter, we used the mean squared error (MSE) given by Equation 1.10.

3.3.1 V deltas

Let us first derive the equations for the matrix $\frac{\partial E_t^{log}}{\partial \mathbf{V}}$. Let $\vec{q}(t) = \mathbf{V}\vec{h}(t)$, where $\mathbf{V} \in \mathbb{R}^{m \times n}$ and $\vec{y}(t), \vec{h}(t) \in \mathbb{R}^n$.

$$\frac{\partial E_t^{log}}{\partial \mathbf{V}_{ij}} = \frac{\partial E_t^{log}}{\partial y_k(t)} \frac{\partial y_k(t)}{\partial q_l(t)} \frac{\partial q_l(t)}{\partial \mathbf{V}_{ij}}, \text{ for } 1 \leq l \leq m, 1 \leq k \leq n \tag{3.5}$$

From the definition of E_t^{log} in Equation 3.4, we get that

$$\frac{\partial E_t^{log}}{\partial y_k(t)} = -\frac{\partial o_k(t)}{\partial y_k(t)} \tag{3.6}$$

The function $\vec{y}(t)$ is the *softmax* function, thus

$$\frac{y_k(t)}{q_l(t)} = \begin{cases} -y_k(t) y_l(t), \text{ for } k \neq l \\ y_k(t)(1 - y_k(t)), \text{ for } k = l \end{cases} \tag{3.7}$$

Using Equation 3.6 and Equation 3.7, we obtain the following

$$\frac{\partial E_t^{log}}{\partial q_l(t)} = -o_l(t) + y_l(t) \sum_k o_k(t) \tag{3.8}$$

If we assume that the sum of the desired outputs $\vec{o}(t)$ is equal to 1, as it is the case with one-hot encoding[1] for example, we can further simplify Equation 3.8 by

$$\frac{\partial E_t^{log}}{\partial q_l(t)} = y_l(t) - o_l(t) \tag{3.9}$$

The only missing part now is $\frac{\partial q_l(t)}{\partial \mathbf{V}_{ij}}$, which is given by

$$\frac{\partial q_l(t)}{\partial \mathbf{V}_{ij}} = h_j(t) \tag{3.10}$$

Putting it all together we get that

$$\frac{\partial E_t^{log}}{\partial \mathbf{V}_{ij}} = (\vec{y}(t) - \vec{o}(t)) \otimes \vec{h}(t) \tag{3.11}$$

Where $\otimes$ denotes the outer product of two vectors.

3.3.2 W deltas

Before we start deriving the formulae for the derivatives of E_t^{log} with respect to $\mathbf{W}_{ij}$'s, it is important to notice that there is a direct dependence of $\vec{h}(t)$ on $\mathbf{W}$ as well as an indirect dependence of $\vec{h}(t)$ on $\mathbf{W}$ through $\vec{h}(t-1)$.

By applying the chain rule to $\frac{\partial E_t^{log}}{\partial \mathbf{W}_{ij}}$ we get

$$\frac{\partial E_t^{log}}{\partial \mathbf{W}_{ij}} = \frac{\partial E_t^{log}}{\partial y_k(t)} \frac{\partial y_k(t)}{\partial q_l(t)} \frac{\partial q_l(t)}{\partial h_m(t)} \frac{\partial h_m(t)}{\partial \mathbf{W}_{ij}} \tag{3.12}$$

Note that we have already computed the first two terms of Equation 3.12. The third term is simple and is given by

$$\frac{\partial q_l(t)}{\partial h_m(t)} = \mathbf{V}_{lm} \tag{3.13}$$

For the fourth term, however, we have to make use of the observation we made at the beginning about the indirect dependence of $\vec{h}(t)$ on $\mathbf{W}$ through $\vec{h}(t-1)$. If we apply this idea recursively we obtain the following relationship for $\frac{\partial h_m(t)}{\partial \mathbf{W}_{ij}}$:

$$\frac{\partial h_m(t)}{\partial \mathbf{W}_{ij}} + \frac{\partial h_m(t)}{\partial h_n(t-1)} \frac{\partial h_n(t-1)}{\partial \mathbf{W}_{ij}} + \cdots + \frac{\partial h_m(t)}{\partial h_n(0)} \frac{\partial h_n(0)}{\partial \mathbf{W}_{ij}} \tag{3.14}$$

Note that the first term in Equation 3.14 can be rewritten as $\frac{\partial h_m(t)}{\partial h_n(t)} \frac{\partial h_n(t)}{\partial \mathbf{W}_{ij}}$, thus we get

$$\frac{\partial h_m(t)}{\partial \mathbf{W}_{ij}} = \sum_{r=0}^{t} \frac{\partial h_m(t)}{\partial h_n(r)} \frac{\partial h_n(r)}{\partial \mathbf{W}_{ij}} \tag{3.15}$$

[1]One-hot encoding is a popular way of encoding categorical data onto $0-1$ vectors.

Bringing it all together, we obtain the following

$$\frac{\partial E_t^{log}}{\partial \mathbf{W}_{ij}} = (\vec{y}(t) - \vec{o}(t))\mathbf{V}_{lm} \sum_{r=0}^{t} \frac{\partial h_m(t)}{\partial h_n(r)} \frac{\partial h_n(r)}{\partial \mathbf{W}_{ij}} \tag{3.16}$$

3.3.3 U deltas

Taking the gradient of E_t^{log} with respect to **U** is very similar to doing it with respect to **W**. We have

$$\frac{\partial E_t^{log}}{\partial \mathbf{U}_{ij}} = \frac{\partial E_t^{log}}{\partial y_k(t)} \frac{\partial y_k(t)}{\partial q_l(t)} \frac{\partial q_l(t)}{\partial h_m(t)} \frac{\partial h_m(t)}{\partial \mathbf{U}_{ij}} \tag{3.17}$$

Note that we have already computed the first three terms of Equation 3.17. Following the same procedure as with **W**, we find that

$$\frac{\partial h_m(t)}{\partial \mathbf{U}_{ij}} = \sum_{r=0}^{t} \frac{\partial h_m(t)}{\partial h_n(r)} \frac{\partial h_n(r)}{\partial \mathbf{U}_{ij}}, \tag{3.18}$$

and thus we have

$$\frac{\partial E_t^{log}}{\partial \mathbf{U}_{ij}} = (\vec{y}(t) - \vec{o}(t))\mathbf{V}_{lm} \sum_{r=0}^{t} \frac{\partial h_m(t)}{\partial h_n(r)} \frac{\partial h_n(r)}{\partial \mathbf{U}_{ij}} \tag{3.19}$$

3.4 Long-term dependencies problem

Sometimes the current state may strongly depend on a preceding state far away in the sequence. This is known as long-term dependency, and we may encounter this very often. As an example, consider trying to predict the last word in the text ”I have lived in Spain all my live . . . I speak fluent *Spanish*”. There is a very strong dependency between ”Spain” and ”Spanish”, and the gap between the words might be far apart from each other.

In theory, RNNs are capable of handling long-term dependencies. With some easy problems, it might be possible to initialize the parameters of the RNN in such a way that the network will learn these dependencies. However, in practice, this is far from reality, where RNNs do not seem to be eable to learn them. This problem was explored in depth by Y. Bengio (1994) in [25].

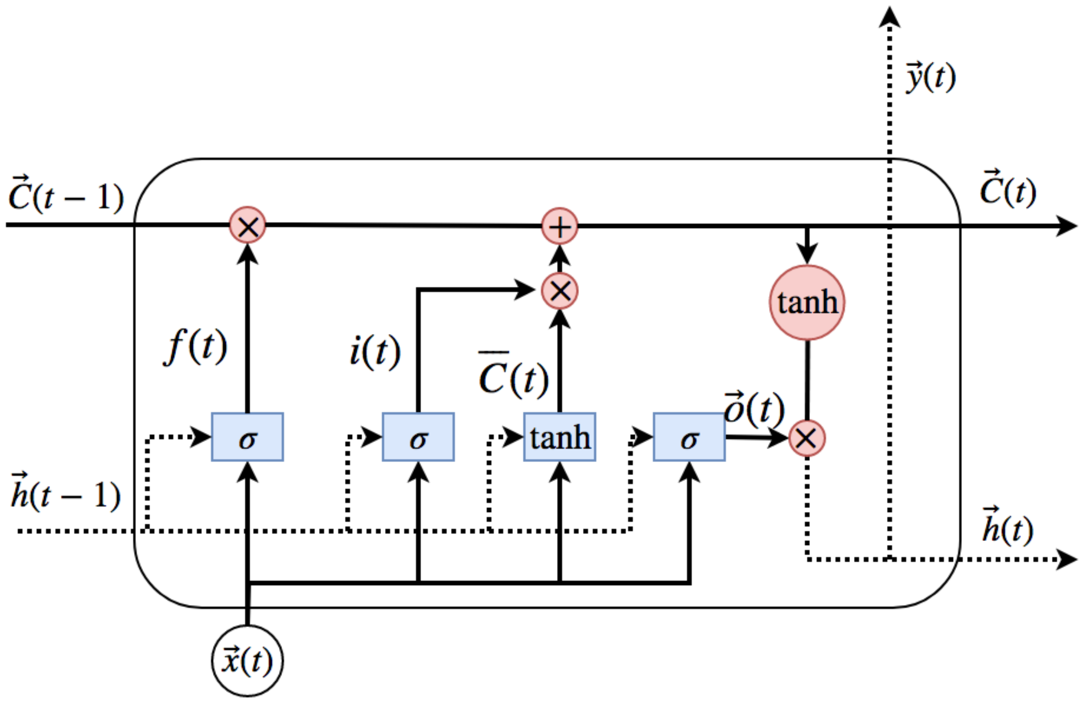
Many approaches have been proposed to solve the problem of long-term dependencies. We will describe the Long Short Term Memory (LSTM) network approach that is capable of solving this problem in the next section.

3.5 Long Short Term Memory

In this section, we describe the internal workings of Long Short Term Memories. It should be noted, however, that although this special type of networks are able to learn long-term dependencies, we did not use them in our experiments. The reason for this decision was that we considered that the available data was not big enough for the classic RNN to forget the time dependencies. Moreover, the size of the input sequences in our experiments with RNNs was set to low numbers (in the order of 5).

Long Short Term Memory networks, or LSTMs for short, are a special kind of RNNs, capable of learning long-term dependencies. They were first introduced by Hochreiter & Schmidhuber in [10]. We can see a graphical representation of an LSTM network for a given time t in Figure 3.3.

The key concept to LSTM is the cell state , denoted as $\vec{C}(t)$, which is a recurrently connected linear unit. We can think of these cell states as digital memories, where the LSTM can remove or add information to.



In this picture we can see the internal function of an LSTM cell at a given time t. The input for this LSTM cell is the vector $\vec{x}(t)$ from the input sequence, the previous hidden state $\vec{h}(t-1)$, and the previous cell state $\vec{C}(t-1)$. The output is the output vector $\vec{y}(t)$, the current hidden state $\vec{h}(t)$, and the current cell state $\vec{C}(t)$. The red circles with an operation symbol represent an element-wise operation. We can observe four different perceptron units inside the blue boxes interacting with one another.

Figure 3.3: **LSTM module at time t.**

The process of removing or adding information to the memory cell is regulated by structures called gates. Gates are a way to optionally let information through. They are composed of simple perceptron units (Definition 1.2.1) and an element-wise multiplication operation $\otimes$. The perceptrons outputs have values between zero and one, describing how much of each component should be let through. A value of zero means that nothing is passed, whereas a value of one means that everything is passed on.

3.5.1 LSTM Forward pass

In this section we explain how an LSTM cell computes its output $\vec{y}(t)$, hidden state $\vec{h}(t)$, and cell state $\vec{C}(t)$, given the input vector $\vec{x}(t)$, the previous hidden state $\vec{h}(t-1)$, and the previous cell state $\vec{C}(t-1)$.

The first step is applying a "filter" to the previous cell state $\vec{C}(t-1)$, which is carried out by sigmoid perceptron units that take as input $\vec{x}(t)$ and $\vec{h}(t-1)$, and output a vector $\vec{f}(t)$. This layer is known as the "forget gate layer", and it is computed by

$$\vec{f}(t) = \sigma(\mathbf{W}_f \cdot [\vec{h}(t-1), \vec{x}(t)]) \tag{3.20}$$

The next step is to decide what information is going to be stored in the cell state. This is done in two parts. First hyperbolic tangent perceptrons create a vector of new candidate values $\overline{C}(t)$ that could be added to the state, then these are "filtered" by sigmoid perceptrons $\vec{i}(t)$, called the "input gate layer", that decides which values will go through. Bellow we show the equation for these intermediate steps

$$\vec{i}(t) = \sigma(\mathbf{W}_i \cdot [\vec{h}(t-1), \vec{x}(t)]) \tag{3.21}$$

$$\overline{C}(t) = \tanh(\mathbf{W}_C \cdot [\vec{h}(t-1), \vec{x}(t)]) \tag{3.22}$$

Once these have been computed, we can now update the old state $\vec{C}(t-1)$ into the new state $\vec{C}(t)$. This is done by Equation 3.23

$$\vec{C}(t) = \vec{f}(t) \otimes \vec{C}(t-1) + \vec{i}(t) \otimes \overline{C}(t) \tag{3.23}$$

The final step is to compute the hidden state $\vec{h}(t)$, and from that, compute the output of the LSTM cell $\vec{y}(t)$. It is usually the case that $\vec{y}(t) = \vec{h}(t)$, thus it is sufficient to calculate $\vec{h}(t)$. For the computation of $\vec{h}(t)$ we transform the vector $\vec{C}(t)$ through an element-wise hyperbolic tangent function and then filter it out with the output sigmoid perceptron units $\vec{o}(t)$. The following equations are used

$$\vec{o}(t) = \sigma(\mathbf{W}_o \cdot [\vec{h}(t-1), \vec{x}(t)]) \tag{3.24}$$

$$\vec{h}(t) = \vec{o}(t) \otimes \tanh(\vec{C}(t)) \tag{3.25}$$

4. Clustering

4.1 Overview

In Machine Learning, clustering is an unsupervised learning method which consists on partitioning a data set D into subsets with similar characteristics called clusters [7].

Traditional approaches based on statistics, such as, hierarchical cluster analysis (HCA) or K-means are still extensively used in clustering tasks.

In recent years, due to their high performance in function approximation tasks, ANN's, more specifically Self Organising Maps (SOM), and fuzzy logic are being used as an alternative to the statistical methods. Combining both paradigms in a two-phase approach may be beneficial for reducing the computational cost as shown in [13], where SOM and K-means were combined for time series clustering.

4.2 Kohonen's neural network

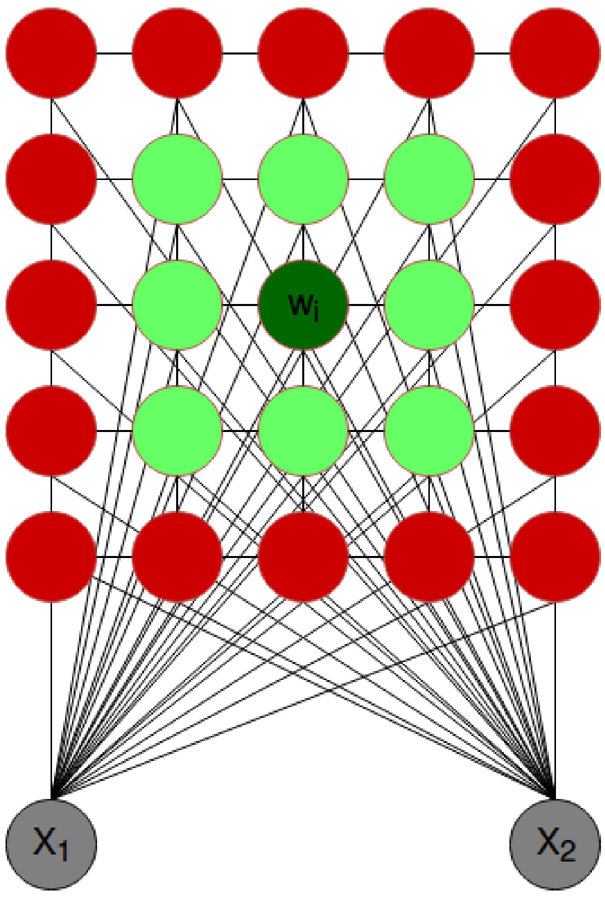
A self-organizing map (SOM), also known as Kohonen map, is a type of artificial neural network that uses unsupervised learning to produce a low-dimensional (typically 2-dimensional), discrete representation of high-dimensional input data. This representation is called a map. The process of reducing the dimensionality of the data is, in effect, a data compression technique known as vector quantisation. The architecture of the model is also biologically inspired and uses neurons to perform the data compression. In addition, SOMs tend to preserve the topological properties of the input space [14]. These properties make SOMs useful also for high-dimensional data visualization. [9]

Like most artificial neural networks, SOMs operate in two cycles: training and recall. Training builds the map using training data. And recall processes new inputs. Self-organizing maps differ from other artificial neural networks in that they use the so-called competitive learning as opposed to error-correction learning, like, e.g. MLPs or RNNs.

4.2.1 The model

In this section, we will deal only with the 2D version of Kohonen's maps, as it is the most widespread variant of the network. This network consists of nodes, or neurons, each of which has associated a weight vector $\vec{w} \in \mathbb{R}^n$, where n is the dimension of the input data. This vector $\vec{w}$ can be thought of as the position, in the input space, where this node is located. In addition, the neurons are organized to form a grid which has a fixed topology. On this grid, we can define the notion of a topological neighborhood for a given neuron, which can be then used by the training algorithms to update the weights. An example of a Kohonen map is shown in Figure 4.1.

We denote the weight vector of the i-th Kohonen neuron by $\vec{w}_i$. The neighborhood function $\phi(i, j, t)$ determines the strength of the connection between the i-th and j-th neuron at iteration t, and represents the distance measure on the grid. The value of ϕ depends on the relative distance of neurons i and j on the grid, and on the number of iterations done so far by the algorithm. It gets smaller with growing distance between the neurons and increasing time. As for distance measure in the input space, Euclidean distance is often used. Another coefficient used in the algorithm is the learning rate $\alpha(t)$ which also decreases with time.



An example of a 5×5 Kohonen map, with an input dimension of 2, and a grid with a rectangular topology. For neuron i, we have highlighted in green the neighbouring neurons for which ϕ would have a positive value at a given time. If this neuron would thus be selected as the closest one to a training sample, all highlighted neurons would update their weights too.

Figure 4.1: **Graphical representation of a Kohonen map**

4.2.2 The training algorithm for Kohonen networks

The training algorithm for Kohonen's maps (Algorithm 4) begins by initializing the weights of the neurons to small random numbers. Then, we iteratively present it with vectors from the training data set and update the weights of the neurons accordingly.

The input vector presented at time t is denoted by $\vec{x}^t$. The algorithm updates the weights of the neuron that is the closest to vector $\vec{x}^t$ together with the weights of its neighors. To determine the closest neuron, we compute the distance between every neuron and the input vector by means of the Formula 4.1, and take the neuron with the minimum distance, we call this neuron c.

$$d_j = ||\vec{w}_j - \vec{x}^t|| \tag{4.1}$$

$$c = \arg\min(d_j) \tag{4.2}$$

To determine the step by which we update the weights of the neurons at time t, we take into account the neighborhood function ϕ with respect to neuron c, the learning rate, and the difference between the weight vector and the input pattern. This is given by Equation 4.3.

$$\Delta\vec{w}_i = \alpha(t)\phi(i, c, t)[\vec{x}^t - \vec{w}_i] \tag{4.3}$$

With this at hand, we can now present a training algorithm for Kohonen's map.

Algorithm 4: Kohonen training

```
input: A set of training vectors $\mathbf{X}$, and a Kohonen map $\mathbf{W}$
// Initialize the time to 0
$t := 0$;
// Initialize the value of $\alpha$
$\alpha(t) := someInitialValue$;
// Initialize the weights to small random vectors
for $\vec{w}_i \in \mathbf{W}$ do
    $\vec{w}_i := randomVec(N)$;
end
while stopping condition not met do
    // Select a random vector from the input set
    $\vec{x}^t := radomSample(\mathbf{X})$;
    // Find the closest neuron to the vector $\vec{x}^t$
    $c := argmin_t(\mathbf{W}, \vec{x}^t)$;
    // Update the weights of every neuron in $\mathbf{W}$
    for $\vec{w}_i \in \mathbf{W}$ do
        $\vec{w}_i := \vec{w}_i + \alpha(t)\phi(i, c, t)[\vec{x}^t - \vec{w}_i]$;
    end
    // Increment the time
    $t := t + 1$;
end
```

The sequence $\alpha(t)$ should converge to 0 with increasing time, and $\alpha(t) \geq 0$, $\forall t$. Furthermore, it should not converge too rapidly, to give the network the time necessary to train.

Self-organization is the main property of Kohonen's maps. This property enables the network to adapt itself and to capture the distribution of the input data. For example, if the input data is uniformly distributed in the input space, then the neurons in the grid will arrange so that each of them covers roughly the same area.

4.3 Fuzzy C-means clustering

Conventional clustering methods, like K-means clustering, are referred to as crisp clustering, meaning that every object is assigned to one and only one class or cluster. In the case of fuzzy clustering, this constraint is relaxed, and the object may belong to all the clusters with a certain degree of membership [2]. This is especially helpful when the boundaries between the clusters are not well distinguishable. Fuzzy C-means clustering (FCM) is one of the most popular fuzzy clustering algorithms [21], which attempts to find the most characteristic point in each cluster, which is considered as the "centroid" of the cluster and then, the degree of membership for every object in the cluster. This is achieved by means of the minimization of an objective function. A commonly used objective function, and the one used for the experiments is:

$$J_m = \sum_{i=1}^{N} \sum_{j=1}^{C} (u_{ij})^m ||\vec{x}_i - \vec{c}_j||^2 \tag{4.4}$$

Where N is the total number of objects in the data set and C is the number of clusters, $X = \{\vec{x}_1, \vec{x}_2, \ldots, \vec{x}_N | \vec{x}_i \in \mathbb{R}^n, \forall i\}$ is the data set and $C = \{\vec{c}_1, \vec{c}_2, \ldots, \vec{c}_C | \vec{c}_i \in \mathbb{R}^n, \forall i\}$ are the centers of the clusters, $U = [u_{ij}]_{C \times N}$ is a fuzzy partition matrix composed of the degrees of membership of pattern $\vec{x}_i$ to each cluster j, $1 \leq j \leq C$. For a given data point x_i the sum of the membership values for all clusters must be 1. The fuzzy partition matrix exponent m controls the fuzzy overlap, with $m > 1$. Fuzzy overlap refers to how fuzzy the boundaries between clusters are.

The center of the clusters $\vec{c}_j$ and the membership degrees u_{ij} from (4.4) are given by the following equations:

$$\vec{c}_j = \frac{\sum_{i=1}^{N} (u_{ij})^m \vec{x}_i}{\sum_{i=1}^{N} (u_{ij})^m}, 1 \leq j \leq C \tag{4.5}$$

$$u_{ij} = \frac{\frac{1}{||\vec{x}_i - \vec{c}_j||^{\frac{2}{m-1}}}}{\sum_{k=1}^{C} (\frac{1}{||\vec{x}_i - \vec{c}_k||})^{\frac{2}{m-1}}} \tag{4.6}$$

4.3.1 Algorithm

With this knowledge at hand, we present Algorithm 5 for computing the centers of the clusters.

4.3.2 Number of clusters

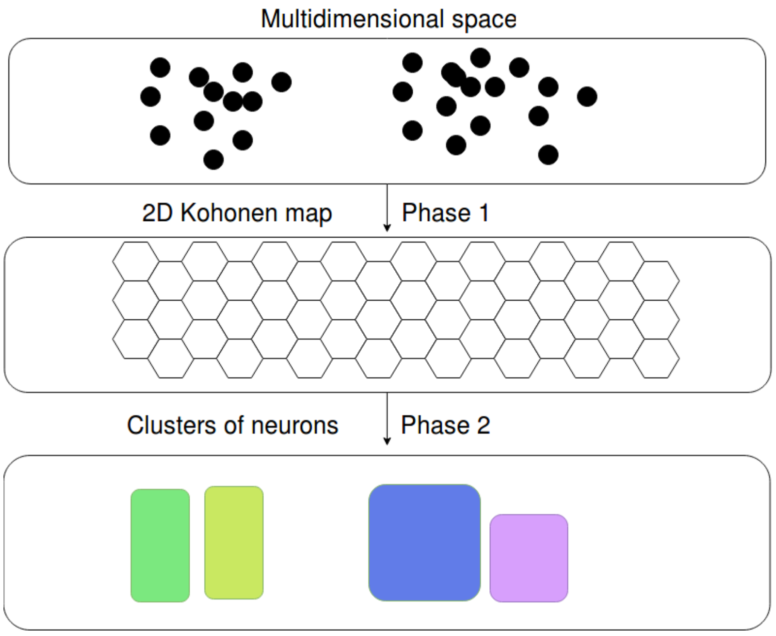
The FCM algorithm requires the number of clusters to be defined in advance. For some situations, this number might be well known. But, in other instances this number might be not that obvious. For the experiments, we used various numbers of clusters.

Algorithm 5: Fuzzy C-means clustering

```
input  : A predefined number of clusters $C$, the fuzzy partition matrix
         exponent $m$, and threshold $\epsilon$
output: A set of vectors $\vec{c}_i$, for $i = 1, 2, \ldots, C$, representing the centers of
         the clusters
Initialize the memberships $u_{ij}$ for vector $\vec{x}_i$ belonging to cluster $j$ such that
 $\sum_{j=1}^{C} u_{ij} = 1$;
while A certain number of iterations are performed or $J_m$ improves by
 more than $\epsilon$ do
    Compute the centers of the clusters $\vec{c}_j$ according to Formula 4.5;
    Use Equation 4.6 to update the values of $u_{ij}$;
end
```

4.4 A two-phase clustering approach

The number of neurons in a SOM may be large, and determining the areas of the map with a high density of neurons can be useful for a more rigorous classification of the input data. This is where a subsequent clustering of the neurons in the SOM layer becomes helpful, separating the neurons into clusters and removing some ambiguities that might have remained in the first stage of the classification.



In the illustration, the first phase creates neurons that are organized into a Kohonen map, and the second phase clusters these neurons.

Figure 4.2: **Two-phase clustering process.**

Figure 4.2 depicts the approach used in this section. First, Kohonen algorithm is run on the data set which creates a collection of neuron that preserve the input distribution. Then, the fuzzy C-means clustering algorithm is run on these neurons to generate the clusters.

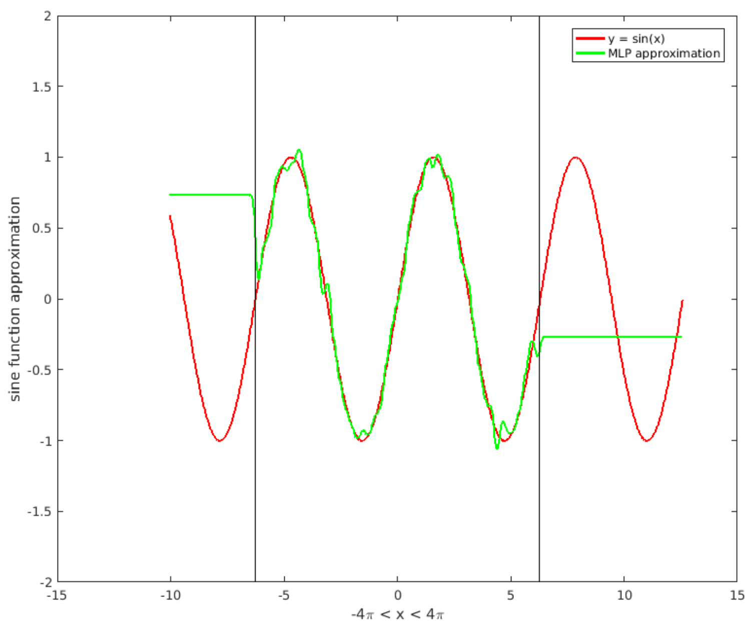
5. Case studies

In this chapter we apply the neural networks that we have discussed so far to relatively simpler tests. In the first case, we approximate a continuous function $f : \mathbb{R} \to \mathbb{R}$. Then we look at clustering data points in a two-dimensional space. The purpose of these case studies is to gain some insights into the workings of the neural networks, and to see how they could be used in some real world problems.

5.1 Function approximation

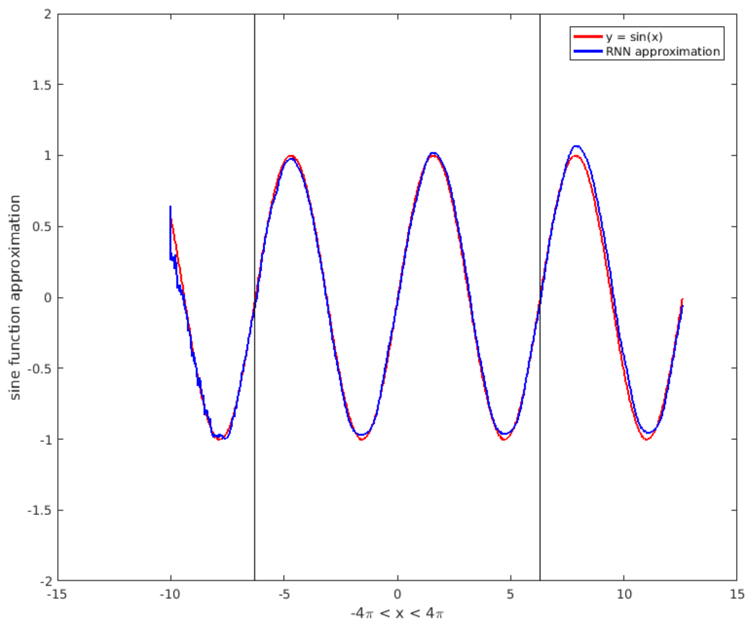
In this example we want to approximate the $\sin x$ function in the range $[-4\pi, 4\pi]$ by means of artificial neural networks. For this approximation, we first used an MLP network (Section 1.2.3) with two hidden layers of 50 and 10 neurons, respectively. We also used a RNN (Section 3) network with one hidden recurrent layer of 15 neurons, and an input sequence length of 100. In both approaches the back propagation (Section 2.2.1) training algorithm with 2000 iterations was used, and the learning rate α was set to 0.1.

The networks were trained using 500 equally separated data points from the interval $[-2\pi, 2\pi]$. And tested with 1000 equally separated data points from the interval $[-4\pi, 4\pi]$. We can observe the results in Figure 5.1 and Figure 5.2, where the approximations together with the actual $\sin x$ functions were drawn. In the graphs, we can observe two vertical lines that mark the training range.



In this figure we consider the MLP approximation drawn in green and the $\sin(x)$ function drawn in red. We can see the training range marked by two vertical lines.

Figure 5.1: **MLP approximation of** $\sin(x)$



In this figure we consider the RNN approximation drawn in blue and the $\sin(x)$ function drawn in red. We can see the training range marked by two vertical lines.

Figure 5.2: **RNN approximation of** $\sin x$

When comparing the two approximations in Figure 5.1 and Figure 5.2, one thing that we notice is the superior generalization that the RNN has over the MLP. We can see this fact when looking at the approximations outside of the bound training data regions (i.e. $[-4\pi, -2\pi)$ and $(2\pi, 4\pi]$). The RNN network clearly is able to approximate the function well even outside of the training data, whereas the MLP network flattens out in these areas.

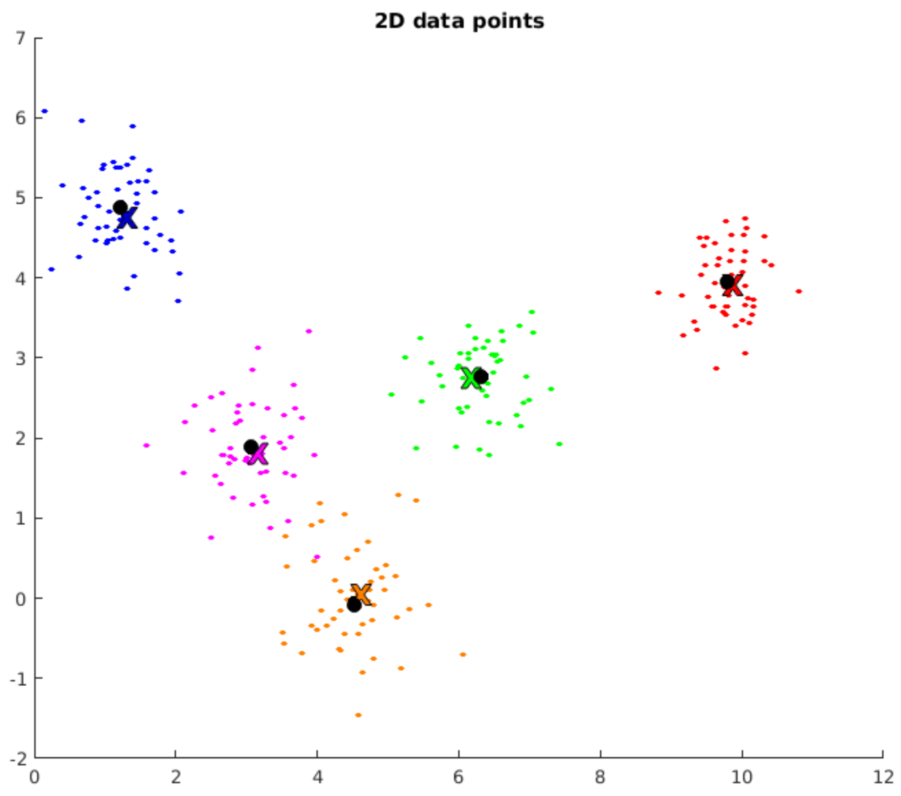
5.2 Clustering

In this example we use Kohonen's maps (Section 4.2) and fuzzy C-means clustering (Section 4.3) to cluster two-dimensional randomly generated data.

The data was generated using 5 seeds placed randomly in the square $[0, 10] \times [0, 10]$. Then, for each of these seeds, a batch of 50 points was generated using the Gaussian distribution with $\mu = 0$ and $\sigma = 1$. There are thus 250 points in total. We can see a visual representation of the data in Figure 5.3. In this figure we can also observe the points (drawn in black) returned by the fuzzy C-means clustering algorithm, and the colored crosses represent the center of the clusters. We see that this algorithm performed well in determining the different clusters and their centers.

The parameters used in the fuzzy C-means clustering algorithm were 5 clusters, 20 iterations and a fuzzy partition matrix exponent of 2.

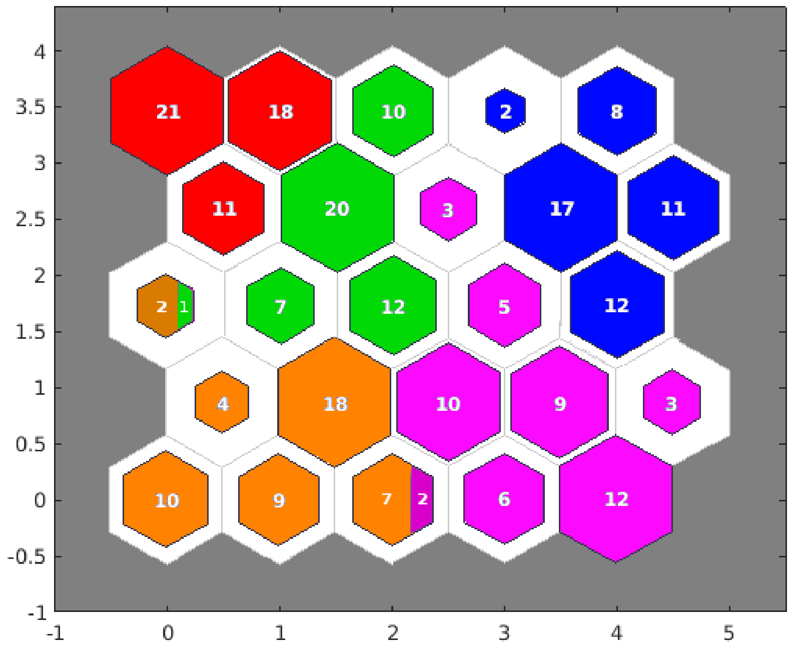
The Kohonen map used to cluster the same data was a 5×5 hexagonal grid, and the training was performed using 1000 iterations. In Figure 5.4 we see a graphical representation of the results obtained by the Kohonen map that we used.



In this figure, we see the five different clusters (red, green, blue, magenta and orange) that we generated in the two-dimensional plane. The colored crosses represent the centers of the clusters, and the black points are the two-dimensional vectors returned by the fuzzy C-means clustering algorithm.

Figure 5.3: **A graphical representation of the generated 2-D test data**

We can clearly observe different neurons being activated for data from distinct clusters, and the original distribution of the sample points being preserved by the SOM layer.



The points from the different clusters were given to the SOM network as inputs, and the activated neurons are color-coded accordingly.

Figure 5.4: **Neuron excitation for the different data points in the Kohonen's map.**

6. Experiments

6.1 Data

6.1.1 Introduction

The data for the experiments have been collected from the World Bank Open database [23]. The purpose of our experiments was to analyze and predict the socio-economic performance of Spanish speaking countries. Thus, the analyzed dataset contains annual national aggregates from 17 Spanish speaking countries [1] together with France and Germany. The timespan that we considered for the data was from 1980 until 2015.

The dataset used in this thesis consists of 18 different macroeconomic indicators in total, which are listed below. The chosen indicators reflect the state of the countries based on the following four criteria: economy, education, health and population.

- Economy:
  - Agricultural share on GDP (Gross Domestic Product).
  - Electricity produced from nuclear sources – percentage of total electricity production.
  - GDP per capita (in current US $).
- Education:
  - Children out of school – percentage of primary-school-age children who are not enrolled in primary or secondary school.
  - Pupil-teacher ratio primary – average number of pupils per teacher in primary school.
  - Pupil-teacher ratio secondary – average number of pupils per teacher in secondary school.
  - Pupil-teacher ratio tertiary – average number of pupils per teacher in tertiary school.
  - Percentage of female students in secondary education.
- Health:
  - Number of physicians per 1000 people.
  - Life expectancy at birth.
- Population:
  - Fertility rate, total (expected number of births per woman).

[1]The studied Spanish speaking countries are Argentina, Bolivia, Chile, Colombia, Cuba, Dominican Republic, Ecuador, El Salvador, Honduras, Mexico, Panama, Paraguay, Peru, Puerto Rico, Spain, Uruguay and Venezuela.

  - Fixed telephone subscriptions (per 100 people).
  - Mobile phone subscriptions (per 100 people).
  - Individuals using the Internet – percentage of total population.
  - Population, ages 0-14, percentage of total.
  - Population, ages 15-64, percentage of total.
  - Population, ages 65 and above, percentage of total.
  - Population density – people per square kilometer of country's total area.

Some statistical parameters of the data are shown in Table 6.1 and Table 6.2.

6.1.2 Data description

All the definitions of the indicators have been collected from the World Bank data. Together with the definitions, we have plotted the data used for the experiments.

- Agriculture, value added (% of GDP): Agriculture corresponds to International Standard Industrial Classification of All Economic Activities (ISIC) divisions 1-5 [5] and includes forestry, hunting and fishing, as well as cultivation of crops and livestock production. In developing countries a large share of agricultural output is either not exchanged (because it is consumed within the households) or not exchanged for money. Even though this large share of agricultural output is not sold, we can still observe in the data that in the less developed countries, among all taken for the experiments, the share agricultural share of GDP is substantially bigger. This fact might be due to the lack of secondary or tertiary economic sectors that represent an important part in economies of the more developed regions. Agricultural production often must be estimated indirectly, using a combination of methods involving estimates of inputs, yields, and area under cultivation.

- Electricity production from nuclear sources (% of total): Share of electricity produced by nuclear power plants on the total electricity production which is the total number of GWh generated by nuclear electricity and CHP (Combined heat and power) power plants. Electricity production shares may not sum up to 100 percent because other sources of generated electricity (such as geothermal, solar, and wind) are not shown.

[2]Some of the selected countries did not produce electricity from nuclear sources in the specified time interval.

[3]The first country, amongst the selected ones, to have mobile subscriptions was Germany in 1985.

[4]For most of the selected countries, Internet Access to the general public was not present until 1990.

[5]The International Standard Industrial Classification of All Economic Activities (ISIC) is the international reference classification of productive activities. Its main purpose is to provide a set of activity categories that can be utilized for the collection and reporting of statistics according to such activities.

Indicator	Min	Max	Mean	Std	Skewness
Agriculture (% of GDP)	0.48 (Puerto Rico 2007)	24.34 (Honduras 1994)	9.06	6.05	0.61
Nuclear electricity production (% of total)	0.00[2]	79.51 (France 2011)	6.93	17.39	3.07
GDP per capita (current US $)	601.66 (Honduras 1991)	47902.65 (Germany 2014)	7594.43	9531.44	2.17
Children out of school (% of primary school age)	0.00 (France 1981, Argentina 1997)	34.59 (Colombia 1986)	6.76	6.37	1.40
Pupil-teacher ratio (primary school)	8.90 (Cuba 2015)	46.35 (Dom. Republic 1985)	24.27	8.51	0.53
Pupil-teacher ratio (secondary school)	7.10 (Argentina 1988)	32.56 (Chile 2002)	17.76	5.94	0.54
Pupil-teacher ratio (tertiary school)	4.60 (Cuba 2014)	28.08 (Honduras 2008)	14.43	4.30	0.15
Secondary education pupils (% female)	44.74 (Mexico 1980)	56.05 (Uruguay 1998)	50.64	2.10	0.42
Physicians (per 1000 people)	0.28 (Paraguay 1994)	7.52 (Cuba 2014)	1.95	1.35	1.39
Life expectancy at birth (total years)	50.00 (Bolivia 1980)	83.38 (Spain 2015)	72.18	5.59	-1.07
Fertility rate, total (births per woman)	1.13 (Spain 1998)	6.31 (Honduras 1980)	2.79	1.05	0.77
Fixed telephone subscriptions (per 100 people)	0.80 (Honduras 1980)	65.35 (Germany 2005)	16.44	15.61	1.50
Mobile cellular subscriptions (per 100 people)	0.00 [3]	180.70 (Panama 2011)	32.46	45.66	1.20
Individuals using the Internet (% of population)	0.00 [4]	87.59 (Germany 2015)	13.59	21.23	1.67
Population ages 0-14 (% of total)	13.12 (Germany 2015)	47.43 (Honduras 1980)	30.48	8.48	-0.22
Population ages 15-64 (% of total)	49.30 (Honduras 1980)	70.15 (Cuba 2011)	61.57	4.72	-0.34
Population ages 65 and above (% of total)	3.07 (Dom. Republic 1980)	21.12 (Germany 2015)	7.95	4.43	1.00
Population density (people per square km of land area)	5.16 (Bolivia 1980)	431.44 (Puerto Rico 2004)	90.36	105.42	1.74

In this table, we can observe for every indicator the minimum, the maximum, the mean, the standard deviation, and the skewness.

Table 6.1: **Statistical data**

	GDP per capita	Agric.	Nuc. electr.	Child. Out of school	Pupil / teach prim.	Pupil / teach second.	Pupil / teach tertiary	Fem. pupils %	Life expect.	Phys. / (1000 people)	Fert. rate	Tlf. Subs.	Ind. Using Internet	Mob. subs.	Pop. Ages 0-14	Pop. Ages 15-64	Pop. Ages 65+	Pop. density
Pop. density	0.38	-0.36	0.11	0.15	-0.16	-0.04	0.10	-0.10	0.23	0.12	-0.34	0.35	0.13	0.08	-0.31	0.24	0.34	1.00
Pop. Ages 65+	0.84	-0.64	0.64	-0.48	-0.67	-0.48	-0.31	-0.26	0.70	0.76	-0.76	0.88	0.52	0.32	-0.92	0.72	1.00	
Pop. Ages 15-64	0.56	-0.72	0.35	-0.47	-0.67	-0.36	-0.34	-0.18	0.82	0.68	-0.95	0.62	0.47	0.42	-0.93	1.00		
Pop. Ages 0-14	-0.75	0.73	-0.53	0.51	0.73	0.45	0.35	0.24	-0.82	-0.77	0.93	-0.80	-0.53	-0.40	1.00			
Mobile subs.	0.47	-0.28	0.08	-0.29	-0.29	-0.10	0.05	-0.09	0.46	0.14	-0.39	0.38	0.86	1.00				
Ind. Using Internet	0.70	-0.40	0.23	-0.29	-0.39	-0.18	-0.06	-0.19	0.55	0.29	-0.45	0.54	1.00					
Telephone. Subs.	0.90	-0.66	0.77	-0.45	-0.51	-0.37	-0.17	-0.25	0.67	0.54	-0.68	1.00						
Fertility rate	-0.59	0.74	-0.39	0.44	0.64	0.37	0.31	0.11	-0.88	-0.67	1.00							
Physicians / (1000 people)	0.48	-0.52	0.40	-0.43	-0.63	-0.53	-0.46	-0.10	0.63	1.00								
Life expectancy	0.64	-0.71	0.41	-0.43	-0.59	-0.48	-0.31	-0.01	1.00									
Female pupils %	-0.30	0.20	-0.21	0.38	0.24	0.12	0.09	1.00										
Pupil/teacher tertiary	-0.17	0.26	0.04	0.16	0.30	0.35	1.00											
Pupil/teacher secondary	-0.38	0.30	-0.30	0.31	0.73	1.00												
Pupil/teacher primary	-0.51	0.54	-0.27	0.38	1.00													
Child. Out of school	-0.36	0.40	-0.37	1.00														
Nuclear elect. Prod.	0.68	-0.41	1.00															
Agriculture	-0.63	1.00																
GDP per capita	1.00																	

Table 6.2: **Correlation matrix**

In this table, we can see the correlation table for the different indicators used in our experiments. The most strongly correlated indicators appear underlined.

- GDP per capita (current US $): GDP per capita is gross domestic product divided by midyear population. GDP is the sum of gross value added by all resident producers in the economy plus any product taxes and minus any subsidies not included in the value of the product. Data are in current U.S. dollars. GDP per capita is one of the most widely used measures of an economy's output or production. It enables policymakers and central banks to judge whether the economy is contracting or expanding, whether it needs a boost or restraint, and if a threat such as a recession or inflation approaches in the near future. It is frequently cited when assessing standard of living. It is especially useful when comparing one country to another, because it shows relative performance of the countries.

- Children out of school (% of primary school age): Children out of school are considered as the percentage of primary-school-age children who are not enrolled in primary or secondary school. Children in the official primary age group that are in preprimary education should be considered out of school. The qualification of the population is vital for the development of a country. High levels of this indicator might be devastating to a country's social and economic growth.

- Pupil-teacher ratio: Pupil-teacher ratio is calculated by dividing the number of students at the specified level of education by the number of teachers at the same level of education.

  The comparability of pupil-teacher ratios across countries is affected by the definition of teachers and by differences in class size by grade and in the number of hours taught, as well as the different practices countries employ such as part-time teachers, school shifts, and multi-grade classes.

  The number of pupils per teacher is an important factor when it comes to education quality. A high pupil-teacher ratio can be related to underfunded schools or school systems. Thus, this might be an important aspect distinguishing developed countries.

- Secondary education, pupils (% female): Percentage of female enrollment is calculated by dividing the total number of female students in public and private schools at the secondary level by the total enrollment at the same level, and multiplying by 100.

  A low female participation in education might be related to sex discrimination, which is usually manifested in less developed countries.

- Physicians (per 1,000 people): Physicians include generalist and specialist medical practitioners.

  Health systems are increasingly being recognized as key to combating disease and improving the health status of populations. Data on health worker (physicians, nurses, midwives, and community health workers) density show the availability of medical personnel.

- Life expectancy at birth, total (years): Life expectancy at birth indicates the number of years a newborn infant would live if prevailing patterns of mortality at the time of its birth were to stay the same throughout its life.

Mortality rates for different age groups (infants, children, and adults) and overall mortality indicators (life expectancy at birth or survival to a given age) are important indicators of health status in a country.

- Fertility rate, total (expected number of births per woman): Total fertility rate represents the number of children that would be born to a woman if she were to live to the end of her childbearing years and bear children in accordance with age-specific fertility rates of the specified year.

  There is a concern about declining birth rates in both the developing and developed world. Fertility rates tend to be higher in poorly resourced countries but due to high maternal and perinatal mortality, there is a reduction in birth rates. In developing countries children are needed as a labor force and to provide care for their aging parents. Developed countries tend to have lower fertility rates due to lifestyle choices associated with economic affluence where mortality rates are low, birth control is easily accessible and children often can become an economic drain caused by housing and education costs, yet also other costs involved in bringing up children.

- Fixed telephone subscriptions (per 100 people): The indicator of fixed telephone subscriptions refers to the amount of active analogue fixed telephone lines, voice-over-IP (VoIP) subscriptions, fixed wireless local loop (WLL) subscriptions, ISDN voice-channel equivalents and fixed public payphones.

- Mobile cellular subscriptions (per 100 people): The indicator includes the number of postpaid subscriptions, and the number of active prepaid accounts (i.e. that have been used during the last three months). The indicator applies to all mobile cellular subscriptions that offer voice communications. It excludes subscriptions via data cards or USB modems, subscriptions to public mobile data services, private trunked mobile radio, telepoint, radio paging and telemetry services.

- Individuals using the Internet (% of population): Internet users are individuals who have used the Internet (from any location) in the past 3 months. The Internet can be used via a computer, mobile phone, personal digital assistant, games machine, digital TV, etc.

  The digital and information revolution has changed the way the world learns, communicates, does business, and treats illnesses. New communication technologies offer vast opportunities for progress (improved health, better service delivery, distance education, etc.) in all countries. Countries with a low Internet availability or a low number of telephone subscriptions do not exploit all the economic advantages that they provide.

- Population ages 0-14 (% of total): Total population between the ages 0 to 14 as a percentage of the total population. Population is based on the de facto definition of population, which counts all residents regardless of legal status or citizenship.

- Population ages 15-64 (% of total): Total population between the ages 15 to 64 as a percentage of the total population. Population is based on the de facto definition of population, which counts all residents regardless of legal status or citizenship.

- Population ages 65 and above (% of total): Population ages 65 and above as a percentage of the total population. Population is based on the de facto definition of population, which counts all residents regardless of legal status or citizenship.

  Patterns of development in a country are partly determined by the age composition of its population. Different age groups have a different impact on both the environment and infrastructure needs. Therefore, the age structure of a population is important to analyze the usage of the resources and to formulate future policy and planning goals with regard to infrastructure and development. In many developing countries, the once rapidly growing population group of the under-15 population is shrinking. As a result, the previously high fertility rates, together with declining mortality rates, are now reflected in the larger share of the 65 and older population.

- Population density (people per sq. km of country's total area): Population density is midyear population divided by land area in square kilometers. Population is based on the de facto definition of population, which counts all residents regardless of legal status or citizenship, except for refugees not permanently settled in the country of asylum, who are generally considered part of the population of their country of origin. Land area is a country's total area, excluding area under inland water bodies, national claims to continental shelf, and exclusive economic zones. In most cases the definition of inland water bodies includes major rivers and lakes.

  The population density of an area can be one of the most important determining factors for business and marketing planning. It is not enough to know how many consumers live in a specific state or city. You have to know how many people live within a particular radius from the metropolitan regions. Thus, the population density might be an important factor determining economic growth.

From the graphs of the datasets, we can observe some common patterns. The clustering of the countries according to their development or geographic locations is the most frequent one. We can note that Spain, France and Germany tend to attain similar values in most of the indicators. Also, Chile and Argentina tend to cluster together, as is the case with Colombia and Venezuela, amongst others. We can as well notice that Honduras usually has the worst values for the indicators. This might be explained by the fact that it is, amongst all the selected countries, the one with the least Human Development Index.

6.1.3 Data preprocessing

Missing data points

The dataset consists of 12312 values in total, 1395 (11.33%) of which were missing. For every indicator and for every country, if a missing value was surrounded by two known data points, linear interpolation was used to fill in the missing slot. In the case the missing input was surrounded by only one known value, it attained that known value. The situation that the entire time period for a given indicator and country was missing did not occur.

Normalization

For the experiments, three different formulae were used to normalize the data:

$$\overline{x} = \frac{x - min}{max - min} \tag{6.1}$$

$$\overline{x} = \frac{x - \mu}{\sigma} \tag{6.2}$$

$$\overline{x} = \frac{1}{2}(\tanh(0.01(\frac{x - \mu}{\sigma})) + 1) \tag{6.3}$$

6.1 Min-max normalization retains the original distribution of the data and scales it into the interval $[0, 1]$.

6.2 Standardization (Z-score normalization) is computed using the arithmetic mean and standard deviation of the dataset. This method guarantees a mean of 0 and a standard deviation of 1 in the resulting dataset.

6.3 Tanh-estimator is computed using the arithmetic mean and standard deviation of the given data. This method scales the data into the interval $[0, 1]$.

The experiments showed that Z-score normalization gives better results in average.

6.1.4 Input & output

The goal of the experiments was to predict the GDP of a given country at a given point in time. In the case of the MLPs, the network predicts the GDP value at the given time by only taking into consideration the other indicators at that same time. Thus, the input for the system is a vector of 17 dimensions, and the output is a scalar value.

In the case of RNNs, the network makes the GDP prediction based not on the other indicators at that same time but rather at a finite consecutive sequence of the other indicators, which ends at the point in time that we want to predict.

In order to evenly separate the training and the testing sets, we used the 12-fold cross-validation method. In this way, the data was separated into 12 equal-size consecutive blocks. Then, one was used as the testing set and the rest were used for training. This was then repeated 12 times (once for every fold) and the mean error was taken as the error of the run.

For the purpose of estimating the true testing error of the tested models, we used the confidence intervals with a confidence level of 90% and a sample size n of 50.

6.2 Data visualization

The first part of our analysis of the dataset consisted on the visualization of the entire time series, and its clustering.

For visualization, self-organizing maps (SOM) discussed in Section 4.2 were used, which project multidimensional data onto a two-dimensional lattice. We can use SOM to visualize the development of a time series over a given period of time. The clustering was done using the fuzzy C-means clustering technique described in Section 4.3, which clusters the neurons returned by the SOM algorithm (Algorithm 4) into groups of neurons with similar characteristics.

In our experiments, we take time series of macroeconomic data from the dataset described in Section 6.1.2 to visualize the economic development of the considered countries. Then, we find the representatives of the main clusters in the two-dimensional grid. The obtained results give an insight into how individual countries developed over time, as well as, a graphical representation of the development. We can also observe some common behaviors of the different nations.

6.2.1 Set up

It is important to mention that, for the input of the SOM layer, some indicators that presented a very strong correlation with others were discarded [6]. This reduction of input dimension proved to have almost no impact on the results, and in addition, it decreased the training time. Moreover, only Spanish speaking countries were considered, that is, France and Germany were not examined. The purpose of this last consideration was to have a better understanding of the development of Spanish speaking communities.

For the SOM algorithm, a Kohonen layer of 20×20 neurons was used. These neuron have an hexagonal topology, that is, every node in the grid, except for the ones on the border, has 6 neighboring nodes. The training took 50000 iterations, and for each of them, a random sample was drawn from the training set.

After the training was complete, we projected the time-series data for each of the Spanish speaking countries onto the obtained Kohonen network. The evolution of the economy over time is shown by a green line connecting data points representing two consecutive years. Lastly, the initial (1980) and final (2015) data points were marked in white and black, respectively.

Next, we clustered the neurons on the Kohonen layer by means of the fuzzy C-means clustering algorithm, for which the following parameters values were used:

- $m = 2$
- $MaxIterations = 200$
- $\epsilon = 1e - 5$

Different number of clusters were consider. For which you can see the results in Figure 6.5.

[6]The discarded indicators are: Pupil-teacher ration primary, Pupil-teacher ratio secondary, and Mobile phones subscriptions.

6.2.2 Analysis of the results

In the obtained Kohonen map, shown in Figure 6.1, we can distinguish two main clusters. One is located in the lower right region of the map, and the other one occupies the left portion of the SOM layer. We can observe that, the size of the first cluster is substantially bigger than that of the second one. Moreover, all studied Latin American regions are located within this major cluster, whereas Spain, in a certain period of time, is part of the smaller batch of neurons. This observation helps us to understand that there is a distinction between the Spanish and the Latin American economies.

Apart from the observation mentioned earlier, we can also differentiate the countries based on the displacement of their time-series in the SOM layer. With this criterion, we can thus distinguish the following two types of countries:

1. Countries that present a very local displacement in the Kohonen map. This can be understood as a steady economic development, with no considerable socio-economic changes, e.g. Puerto Rico (Figure 6.4) or Cuba.

2. Regions which, at some point in time, underwent a notable shift in its location in the map. This major changes in the country's position in the lattice often relate to significant social, political, or economic changes. Some examples of such countries are Peru (Figure 6.3), Spain (Figure 6.2), or Argentina.

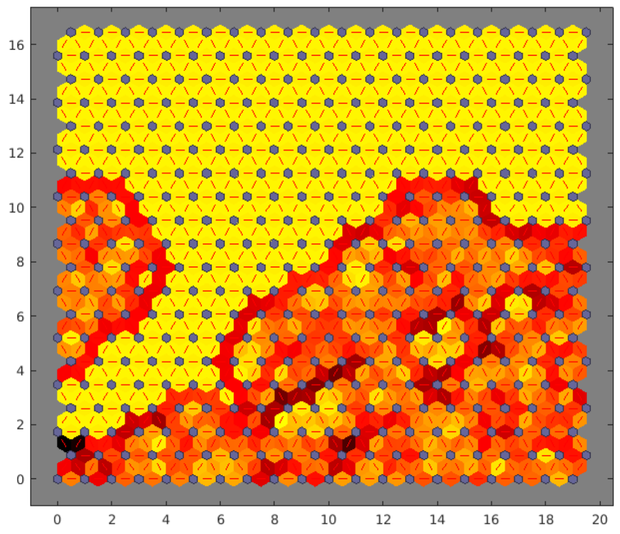
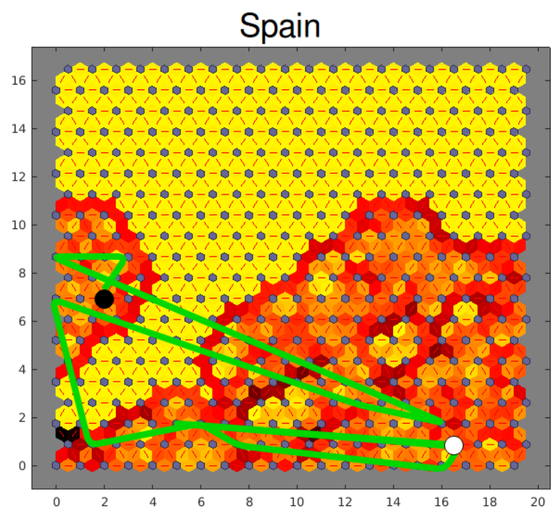


Figure 6.1: **Graphical representation of the SOM layer.**

Looking at the projection of the time-series for Spain (Figure 6.2), we can observe that it starts at the right portion of the bigger cluster and ends in the smaller cluster. The first big shift towards the left portion of the map occurs in 1982, which can be explained by its accession to the NATO[7], moreover, in

[7]The North Atlantic Treaty Organization (NATO) is an intergovernmental military alliance between 29 North American and European countries.



1980	81	82	83	84	85
16	16	5	16	16	16
1	1	2	1	0	0
1986	87	88	89	90	91
7	7	6	6	6	1
1	1	2	2	2	1
1992	93	94	95	96	97
1	0	0	0	0	12
1	8	8	8	8	3
1998	99	00	01	02	03
12	12	12	16	16	0
3	3	3	2	2	10
2004	05	06	07	08	09
0	0	0	2	2	2
10	10	10	10	10	10
2010	11	12	13	14	15
3	3	3	2	2	2
10	10	10	8	8	8

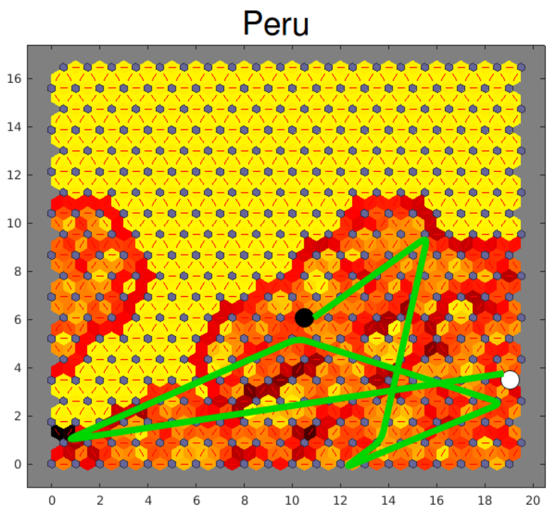
To the left the projection of the time-series of Spain onto the Kohonen layer obtained from the experiments. The table in the right represents the indices of the activated neurons in the layer for every year.

Figure 6.2: **Projection of time-series from Spain**

1982 the Spanish Socialist Worker's Party governed for the first time. The next considerable displacement occurs in 1986, year in which Spain and Portugal joined the EU. In 1991, the time-series moved towards the bottom left corner of the map. This movement can be related to the 1992 Summer Olympic Games celebrated in Barcelona and the socio-economic developments that it implies. This event might also explain the move of Spain to the second cluster, which occurred in 1993. In 1997, however, it moves back almost to the point in the lattice where it started. The reason for this major displacement could be the fact that in 1996 the presidential elections took place, which implied a significant change in the government. The time-series then undergoes only local changes until 2003, when it shifts back to the smaller cluster. This considerable shift can be explained by the adoption of Euro in 2002.

Similarly, if we look at the projection of the time-series for Peru (Figure 6.3), we can observe that, over the course of time, several considerable displacements occurred. The first big shift happened in 1981, and it could be explaied by the Paquisha War. Then, we can observe a significant movement that appeared in 1983, which could be related to the "El Niño" climatic phenomenon that took place between 1982 and 1983. In 1987 another considerable movement can be seen in Peru's time-series, which can be due to the nationalization of banks in that same year. In 1998 the "El Niño" phenomenon stroke the country again, which might be the reason for the displacement that occurred in the time-series. In 2009 we can observe another significant displacement in the map, which could be understood as a consequence of the global financial crisis of 2008. Finally, in 2011 the general elections might have been a possible reason for the considerable shift that occurred in 2012.

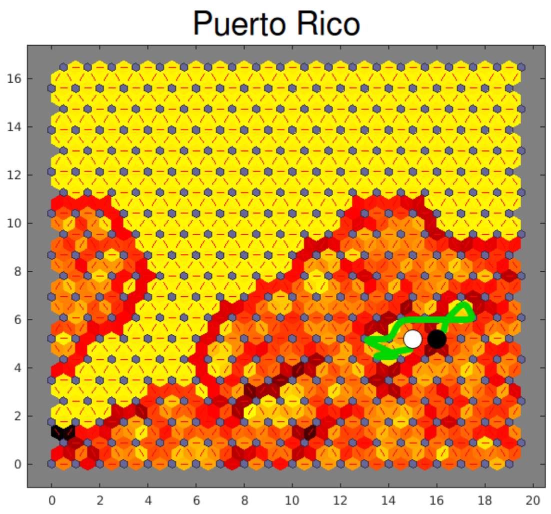
Now, we take a look at the clustering of the neurons in the SOM layer (Figure 6.5). For this, we ran the experiments multiple times with different number of clusters per run. The number of clusters varied from 10 to 15. We can observe that, in general, the cluster centers represent the two clusters mentioned earlier.



1980	81	82	83	84	85
19	0	0	10	10	10
4	1	1	6	6	6
1986	87	88	89	90	91
10	18	18	18	18	18
6	3	3	3	3	3
1992	93	94	95	96	97
18	18	18	18	18	18
3	3	3	3	3	3
1998	99	00	01	02	03
12	12	12	12	12	12
0	0	0	0	0	0
2004	05	06	07	08	09
13	13	13	13	13	15
1	1	1	1	1	11
2010	11	12	13	14	15
15	15	10	10	10	10
11	11	7	7	7	7

To the left the projection of the time-series of Peru onto the Kohonen layer obtained from the experiments. The table in the right represents the indices of the activated neurons in the layer for every year.

Figure 6.3: **Projection of time-series from Peru**



1980	81	82	83	84	85
15	15	15	13	13	13
6	6	6	5	5	5
1986	87	88	89	90	91
13	13	14	14	14	14
5	5	5	5	5	5
1992	93	94	95	96	97
14	14	13	13	13	13
5	5	6	6	6	6
1998	99	00	01	02	03
13	14	14	14	14	14
6	6	6	7	7	7
2004	05	06	07	08	09
17	17	17	17	17	17
7	7	7	7	8	8
2010	11	12	13	14	15
17	16	16	16	16	16
8	7	6	6	6	6

To the left the projection of the time-series of Puerto Rico onto the Kohonen layer obtained from the experiments. The table in the right represents the indices of the activated neurons in the layer for every year.

Figure 6.4: **Projection of time-series from Puerto Rico**

In addition, we can note that the allocation of the cluster centers is strongly related to the sizes of the clusters, that is, the bigger the cluster is, the more centers it will contain.

In conclusion, our experiments have shown the usefulness of Kohonen maps for the analysis of high-dimensional macroeconomic data. SOMs helped us group together countries with similar development. Moreover, major shifts in the positions of the countries on the Kohonen map were often accompanied by socio-political or economical reforms. Other reasons for the significant displacements may also include economic downturns, natural disasters or wars.

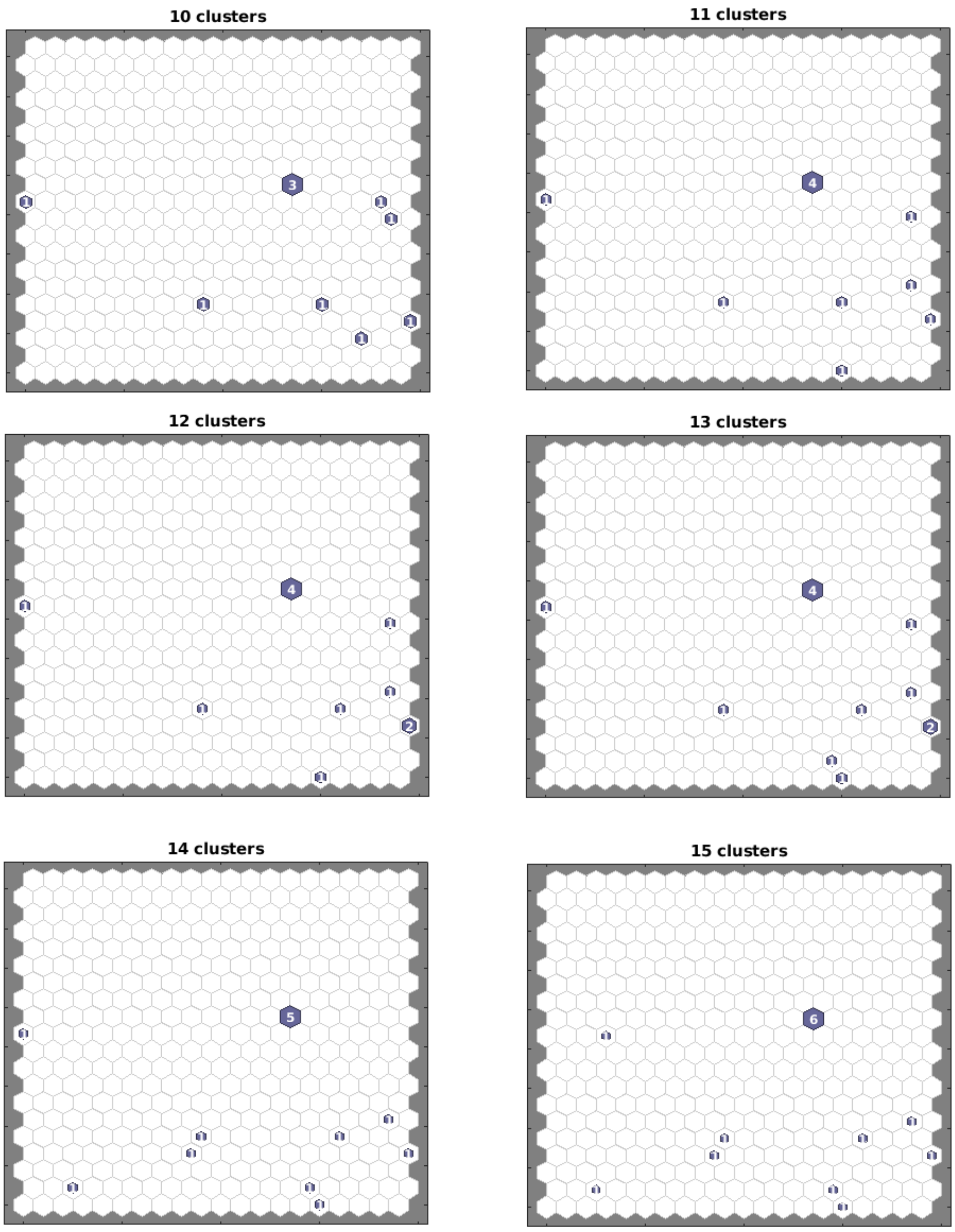


Figure 6.5: **Different runs of the fuzzy C-means clustering algorithm, each time with a different number of clusters.**

6.3 GDP Prediction

6.3.1 Criteria for picking the best model

There are many possible criteria for the selection of the best NN model out of many that might have undergone training. Over the time, many performance criteria have been suggested (e.g. Root Mean Squared Error, Mean Squared Error, Mean Absolute Error, Model Predictive Error, etc.). Other criteria might involve the required training time and memory costs, the number of neurons that the network has, and other characteristics of the architecture.

For our experiments, we measured several characteristics of the respective training process, such as, the MSE on the training, testing and validation sets, the average number of iterations to be performed, and time per iteration. We also computed confidence interval for the actual MSE on the testing set with a confidence level of 90%.

6.3.2 Training algorithms

Many training algorithms exist for neural networks. In these experiments we focus only on some first and second order methods supported by the Neural Network Toolbox from MATLAB. As representatives for the first order algorithms, we have chosen BP (Section 2.2.1) and BP with momentum (Section 2.2.3). For the second order methods, we have chosen the Levenberg-Marquardt (Section 2.3.1) and Scaled Conjugate Gradient (Section 2.3.2) algorithms. In the experiments for GDP forecasting we used MLP and RNN networks. For MLPs, we used all four algorithms mentioned above. And for RNNs we used the SCG algorithm to train the network.

6.3.3 MLP network

In the analysis, we tested various network architectures and training algorithms, as well as different transfer functions used in the hidden layers. One goal was to investigate which of the tested model had the best overall performance for the task at hand, and assess the difference in the respective training times and number of iterations. Thus, the parameters taken into account when looking for an optimum MLP performance were:

1. Different training algorithms.
2. The transfer functions used in the hidden layers.
3. The number of hidden layers in the network.
4. The number of neurons per layer.

As mentioned earlier, we have used four different training algorithms for the MLPs. Mainly, BP (Section 2.2.1), BP with momentum (Section 2.2.3), Levenberg-Marquardt (Section 2.3.1) and SCG (Section 2.3.2).

Three different transfer functions were taken into account in the experiments; the sigmoid function (Section 1.3.1), the hyperbolic tangent function (Section 1.3.2), and the ReLU function (Section 1.3.3).

Arch.	Mean train error	Mean valid. error	Mean test error	Epochs	Time (s)
17-10-1	0.054	0.073	0.238 ± 0.059	10.7	61.9
17-15-1	0.067	0.08	0.171 ± 0.023	10.86	45.61
17-30-1	0.057	0.079	0.162 ± 0.018	10.81	48.21
17-10-10-1	0.072	0.077	0.134 ± 0.017	12.26	52.11
17-15-15-1	0.065	0.077	0.157 ± 0.021	12.07	53.95
17-30-30-1	0.051	0.083	0.126 ± 0.017	11.53	61.8
17-30-10-1	0.064	0.073	0.136 ± 0.016	12.46	62.18
17-30-15-1	0.061	0.077	0.146 ± 0.020	12.17	61.46

Table 6.3: **MLP, Levenberg Marquardt with sigmoid transfer function**

For the architecture of the networks, we used 1 or 2 hidden layers with the number of hidden units ranging from 10 up to 30. In particular, the architectures used were; $[17-10-1]$, $[17-15-1]$, $[17-30-1]$, $[17-10-10-1]$, $[17-15-15-1]$, $[17-30-10-1]$, $[17-30-15-1]$ and $[17-30-30-1]$. Where $[x-y-z]$ means one input, one hidden and one output layers of x, y, and z neurons, respectively.

We can see the results of the MLP experiments in the tables (Table 6.3 to Table 6.14). Every table contains the outcomes of the experiments we have done for a given training algorithm and transfer function with all the architectures.

6.3.4 RNN network

For the experiments with the RNN networks, we tested the performance of different architectures, as well as various transfer functions used in the hidden layers. The training algorithm used was the SCG (Section 2.3.2). As with the MLP experiments, we measured the MSE on the training, testing and validation sets. And we examined the differences in training times and numbers of iterations required by the respective architectures. In all the cases, the sequence length equal to 5 was taken into account. In this way, 5 previous inputs need to be known to the network in order to predict the output.

Similarly to the MLP experiments, three different transfer functions were used; the sigmoid function (Section 1.3.1), the hyperbolic tangent function (Section 1.3.2), and the ReLU function (Section 1.3.3).

As it is the case with the MLPs, for the architecture of the networks, we used 1 or 2 hidden layers with the number of hidden units ranging from 10 up to 30. In particular, the architectures used were; $[17-10-1]$, $[17-15-1]$, $[17-30-1]$, $[17-10-10-1]$, $[17-15-15-1]$, $[17-30-10-1]$, $[17-30-15-1]$ and $[17-30-30-1]$. Where $[x-y-z]$ means one input, one hidden and one output layers of x, y, and z neurons, respectively.

We can see the results of the RNN experiments in the following tables (Table 6.15 to Table 6.17). Every table contains the outcomes of the experiments for a given training algorithm and transfer function with all the tested architectures.

Arch.	Mean train error	Mean valid. error	Mean test error	Epochs	Time (s)
17-10-1	0.081	0.084	0.181 ± 0.021	22.01	5.37
17-15-1	0.092	0.094	0.214 ± 0.021	21.99	6.42
17-30-1	0.1	0.118	0.283 ± 0.033	21	12.98
17-10-10-1	0.097	0.088	0.199 ± 0.024	22.74	7.74
17-15-15-1	0.087	0.101	0.226 ± 0.025	22.22	12.43
17-30-30-1	0.127	0.131	0.320 ± 0.037	21.25	133.87
17-30-10-1	0.082	0.086	0.223 ± 0.024	21.67	28.51
17-30-15-1	0.095	0.096	0.246 ± 0.024	21.47	48.8

Table 6.4: **MLP, Levenberg Marquardt with ReLU transfer function**

Arch.	Mean train error	Mean valid. error	Mean test error	Epochs	Time (s)
17-10-1	0.069	0.073	0.255 ± 0.054	22.27	5.71
17-15-1	0.099	0.09	0.255 ± 0.025	21.25	6.41
17-30-1	0.088	0.123	0.389 ± 0.042	20.68	13.47
17-10-10-1	0.071	0.093	0.251 ± 0.028	22.71	8.38
17-15-15-1	0.123	0.105	0.272 ± 0.035	21.78	12.69
17-30-30-1	0.141	0.165	0.387 ± 0.035	20.43	127.24
17-30-10-1	0.087	0.111	0.267 ± 0.037	21.27	28.58
17-30-15-1	0.099	0.121	0.258 ± 0.036	20.97	48.16

Table 6.5: **MLP, Levenberg Marquardt with tanh transfer function**

Arch.	Mean train error	Mean valid. error	Mean test error	Epochs	Time (s)
17-10-1	0.078	0.052	0.089 ± 0.005	85.01	23.5
17-15-1	0.074	0.054	0.095 ± 0.007	87.8	27.02
17-30-1	0.074	0.057	0.110 ± 0.015	93.94	35.23
17-10-10-1	0.092	0.051	0.094 ± 0.006	84.56	28.6
17-15-15-1	0.092	0.05	0.087 ± 0.004	85.21	33.13
17-30-30-1	0.096	0.052	0.103 ± 0.009	90.6	48.4
17-30-10-1	0.093	0.05	0.106 ± 0.008	82.86	34.75
17-30-15-1	0.091	0.05	0.104 ± 0.006	83.49	37.01

Table 6.6: **MLP, scg with sigmoid transfer function**

Arch.	Mean train error	Mean valid. error	Mean test error	Epochs	Time (s)
17-10-1	0.092	0.07	0.169 ± 0.024	101.04	28.48
17-15-1	0.081	0.08	0.205 ± 0.045	104.59	31.07
17-30-1	0.124	0.133	0.452 ± 0.113	114.1	39.1
17-10-10-1	0.101	0.092	0.188 ± 0.019	104.59	35.09
17-15-15-1	0.107	0.096	0.245 ± 0.037	109.49	41.45
17-30-30-1	0.149	0.199	0.549 ± 0.094	117.08	56.7
17-30-10-1	0.107	0.088	0.185 ± 0.015	106.27	42.13
17-30-15-1	0.128	0.1	0.334 ± 0.080	108.65	45.66

Table 6.7: **MLP, scg with ReLU transfer function**

Arch.	Mean train error	Mean valid. error	Mean test error	Epochs	Time (s)
17-10-1	0.068	0.059	0.136 ± 0.018	92.89	26.67
17-15-1	0.075	0.068	0.162 ± 0.021	100.3	33.08
17-30-1	0.096	0.098	0.288 ± 0.034	109.08	44.83
17-10-10-1	0.075	0.067	0.155 ± 0.016	95.99	35.04
17-15-15-1	0.09	0.07	0.176 ± 0.024	98.88	42.8
17-30-30-1	0.116	0.101	0.264 ± 0.031	108.61	67.31
17-30-10-1	0.054	0.068	0.150 ± 0.015	97.13	45.26
17-30-15-1	0.067	0.077	0.182 ± 0.021	101.11	50.95

Table 6.8: **MLP, scg with tanh transfer function**

Arch.	Mean train error	Mean valid. error	Mean test error	Epochs	Time (s)
17-10-1	0.599	0.265	0.415 ± 0.044	350.55	231.82
17-15-1	0.557	0.234	0.388 ± 0.037	358.15	252.71
17-30-1	0.396	0.216	0.425 ± 0.040	383.65	336.09
17-10-10-1	0.854	0.421	0.514 ± 0.029	399.89	353.59
17-15-15-1	0.898	0.354	0.459 ± 0.036	367.27	340.3
17-30-30-1	0.542	0.28	0.373 ± 0.025	355.59	416.4
17-30-10-1	0.769	0.366	0.494 ± 0.027	376.78	391.21
17-30-15-1	0.821	0.316	0.439 ± 0.022	357.92	371.66

Table 6.9: **MLP, BP with sigmoid transfer function**

Arch.	Mean train error	Mean valid. error	Mean test error	Epochs	Time (s)
17-10-1	0.356	0.238	0.582 ± 0.114	414.49	219.68
17-15-1	0.471	0.255	0.648 ± 0.094	416.29	237.29
17-30-1	0.724	0.465	1.256 ± 0.433	385.5	227.45
17-10-10-1	0.399	0.278	0.475 ± 0.060	387.26	228.48
17-15-15-1	0.407	0.251	0.476 ± 0.053	370.7	244.66
17-30-30-1	0.51	0.269	0.541 ± 0.054	374.5	265.89
17-30-10-1	0.562	0.266	0.394 ± 0.044	371.49	256.33
17-30-15-1	0.416	0.279	0.496 ± 0.081	370.96	274.51

Table 6.10: **MLP, BP with ReLU transfer function**

Arch.	Mean train error	Mean valid. error	Mean test error	Epochs	Time (s)
17-10-1	0.302	0.222	0.412 ± 0.046	417.3	325.77
17-15-1	0.358	0.236	0.571 ± 0.063	403.72	331.55
17-30-1	0.778	0.34	0.938 ± 0.129	387.01	345.51
17-10-10-1	0.245	0.196	0.391 ± 0.040	403.68	366.8
17-15-15-1	0.31	0.21	0.470 ± 0.063	398.98	406.8
17-30-30-1	0.242	0.16	0.364 ± 0.047	364.32	439.88
17-30-10-1	0.256	0.156	0.328 ± 0.044	394.79	432.85
17-30-15-1	0.296	0.183	0.413 ± 0.042	397.95	463.29

Table 6.11: **MLP, BP with tanh transfer function**

Arch.	Mean train error	Mean valid. error	Mean test error	Epochs	Time (s)
17-10-1	0.886	0.342	0.510 ± 0.039	748	929.61
17-15-1	0.839	0.315	0.594 ± 0.051	706.4	910.13
17-30-1	1.034	0.321	0.713 ± 0.077	723.91	1122.06
17-10-10-1	1.215	0.54	0.655 ± 0.043	672.25	946.6
17-15-15-1	1.268	0.486	0.613 ± 0.036	669.11	1063.02
17-30-30-1	1.201	0.336	0.536 ± 0.044	685.46	1440.01
17-30-10-1	1.124	0.462	0.539 ± 0.031	735.12	1362.4
17-30-15-1	1.161	0.414	0.557 ± 0.029	688.57	1288.25

Table 6.12: **MLP, BP momentum with sigmoid transfer function**

Arch.	Mean train error	Mean valid. error	Mean test error	Epochs	Time (s)
17-10-1	0.935	0.327	0.836 ± 0.120	794.27	994.19
17-15-1	0.938	0.318	0.947 ± 0.118	783.09	1005.72
17-30-1	1.218	0.441	1.402 ± 0.165	775.9	1050.18
17-10-10-1	1.04	0.309	0.761 ± 0.114	754.33	1008.31
17-15-15-1	0.99	0.31	0.767 ± 0.116	740.08	1056.91
17-30-30-1	0.894	0.313	0.893 ± 0.127	738.32	1280.99
17-30-10-1	0.654	0.319	0.519 ± 0.051	758.15	1115.24
17-30-15-1	0.81	0.264	0.614 ± 0.068	741.97	1192.72

Table 6.13: **MLP, BP momentum with ReLU transfer function**

Arch.	Mean train error	Mean valid. error	Mean test error	Epochs	Time (s)
17-10-1	0.66	0.303	0.689 ± 0.084	824.11	1089.8
17-15-1	0.83	0.354	0.752 ± 0.073	807.11	1177.25
17-30-1	1.427	0.436	1.219 ± 0.138	801.6	1364.16
17-10-10-1	0.585	0.272	0.573 ± 0.057	786.22	1242.94
17-15-15-1	0.554	0.235	0.599 ± 0.062	775.84	1354.46
17-30-30-1	0.702	0.257	0.764 ± 0.136	739.57	1651.39
17-30-10-1	0.567	0.216	0.454 ± 0.049	742.29	1394.61
17-30-15-1	0.584	0.218	0.535 ± 0.050	748.46	1492.88

Table 6.14: **MLP, BP momentum with tanh transfer function**

Arch.	Mean train error	Mean valid. error	Mean test error	Epochs	Time (s)
17-10-1	0.03	0.074	0.099 ± 0.014	347.98	381.28
17-15-1	0.026	0.066	0.074 ± 0.012	365.94	488.31
17-30-1	0.025	0.079	0.091 ± 0.015	357	640.28
17-10-10-1	0.03	0.074	0.090 ± 0.016	332.44	438.85
17-15-15-1	0.021	0.067	0.080 ± 0.015	360.98	620.16
17-30-30-1	0.022	0.096	0.085 ± 0.015	353.16	855.49
17-30-10-1	0.023	0.073	0.077 ± 0.012	337.8	659.82
17-30-15-1	0.022	0.072	0.094 ± 0.015	337.72	719.48

Table 6.15: **RNN, scg with sigmoid transfer function**

Arch.	**Mean train error**	**Mean valid. error**	**Mean test error**	**Epochs**	**Time (s)**
17-10-1	0.073	0.112	0.143 ± 0.027	453.94	621.85
17-15-1	0.069	0.133	0.127 ± 0.015	458.96	693.26
17-30-1	0.062	0.13	0.156 ± 0.024	479.98	956.55
17-10-10-1	0.087	0.136	0.161 ± 0.018	407.48	562.04
17-15-15-1	0.076	0.149	0.180 ± 0.028	435.24	764.54
17-30-30-1	0.077	0.346	0.384 ± 0.143	465.88	1184.03
17-30-10-1	0.081	0.141	0.167 ± 0.024	506.18	1226.07
17-30-15-1	0.113	0.287	0.260 ± 0.090	464.2	1026.9

Table 6.16: **RNN, scg with ReLU transfer function**

Arch.	**Mean train error**	**Mean valid. error**	**Mean test error**	**Epochs**	**Time (s)**
17-10-1	0.025	0.077	0.089 ± 0.013	331.1	401.29
17-15-1	0.016	0.085	0.085 ± 0.015	368.28	517.4
17-30-1	0.01	0.081	0.082 ± 0.013	401.48	789.27
17-10-10-1	0.017	0.075	0.073 ± 0.013	323.9	439.73
17-15-15-1	0.021	0.083	0.105 ± 0.022	335.84	555.45
17-30-30-1	0.019	0.091	0.099 ± 0.018	367.08	946.11
17-30-10-1	0.015	0.078	0.093 ± 0.019	349.04	704.22
17-30-15-1	0.024	0.078	0.109 ± 0.017	291.86	524.68

Table 6.17: **RNN, scg with tanh transfer function**

6.4 Analysis of the obtained results

From the results summarized in Tables 6.3 to 6.17, we can conclude that the sigmoid transfer function works best for our data. This option is closely followed by the hyperbolic tangent, and the highest MSEs, and thus the worst results, were obtained by the ReLU transfer function. The next thing that we can conclude from the tables is that, based on the achieved MSE values, the RNN models outperform almost all the MLP variants by an order of magnitude. However, the training time for the RNNs was considerably higher than most of those for the MLPs. As an example, the average training time for the RNNs was of 696.54 (s), whereas that for the MLPs with SCG was 38.89 (s)

Among the MLP models, the best results were those obtained by the SCG algorithm, followed by the Levenberg Marquardt. Thus, if our aim was to get low MSE, then it would be preferable to pick either the SCG or LM training algorithms. Nonetheless, for very big neural networks, the memory requirements for storing the Jacobian matrix becomes a disadvantage. Therefore, the Levenberg-Marquardt algorithm is not recommended when we have big neural networks and memory constraints.

Concerning the different architectures we have tested so far, there is not a big distinction among them. For some models, the MSE decreases with increasing number of neurons in the hidden layers. But, for some other models, the MSE reduces from an architecture of $[17-10-1]$ to $[17-15-1]$, and increases again from $[17-15-1]$ to $[17-30-1]$. As for the number of hidden layers we could say that it is usually better to have 2 hidden layers instead of 1, but this is not a general rule, since there are some instances of models, e.g, RNN with SCG and ReLU transfer function, where this rule does not hold.

Conclusion

Economic time-series analysis is, namely, of great importance in many aspects of business, such as resource allocation, state and local budgeting, and international policy making. This work has been focused on multidimensional time-series and their processing by means of MLPs and RNNs. For clustering and visualization of the data, we have applied self-organizing maps and the fuzzy c-means clustering algorithm.

The aim of this work was to explore the viability of various neural network models and their training algorithms for analyzing macro-economic data. The focus was set on two main tasks, clustering/visualization and time-series forecasting. For the former task, we employed self-organizing feature maps. For the latter one, we tested MLPs and RNNs. The data used to train and test the models is publicly available from the World Bank.

The data used in our experiments was obtained from the World Bank Data international organization. It includes 18 different macroeconomic indicators gathered for 17 Spanish speaking countries, along with France and Germany over the period from 1980 until 2015. We clustered and visualized the data using a 2-dimensional SOM. The visualization technique we have developed is shown to explain major trends in the evolution of the time-series data at hand - i.e., changes in the development of different countries. Countries with a stable economy are characterized by just local displacements on the SOM. Crises, on the other hand, tend to manifest themselves by violent moves all over the grid. We also studied the problem of GDP per capita prediction and investigated the performance of several MLP architectures together with different training algorithms for this task. Further, we applied different architectures of RNNs for the investigated task, and compared the achieved results.

For each of the tested models, we analyzed the influence of different parameter values on the model's performance, such as the number of hidden layers, the number of neurons per layer, or the transfer function used in the hidden layers. Based on the obtained results, we formulated statistically significant conjectures about the performance of the models on testing data. When comparing the performance of the MLPs and RNNs, we have observed a significantly better performance achieved by the recursive models.

In the clustering/visualization part, we have essentially shown a way how to detect various causes for economic changes, like financial crises, socio-political turmoils, natural disasters and wars, but also positive events like accession to economic unions or international sports events. We also provide possible reasons that explain the found trends. In time-series forecasting, we have managed to evaluate and compare different neural network models and training algorithms. The found results have confirmed the superiority of RNNs over MLPs in this particular task.

Despite of the overall positive results, here are several ways in which this work can be further extended. Determining the importance of the indicators in the forecasting task could help us to reduce the dimensionality of the data. Further, some parameters, such as the number of hidden layers or the number of neurons in the hidden layers remained fixed in one experiment. An automatic adjustment of these parameters might improve the network's generalization abilities. Data from other countries, could be considered for the tests, too.

The experiments were ran using MATLAB and its Neural Network Toolbox. However, computationally more efficient options for implementation could be considered instead (Caffe, Keras, Tensorflow, etc.).

Bibliography

[1] Batra, D. (2014). Comparison between levenberg-marquardt and scaled conjugate gradient training algorithms for image compression using mlp. *International Journal of Image Processing*, pages 412–422.

[2] Bezdek, J. C. (1981). *Pattern Recognition with Fuzzy Objective Function Algorithms*. Advanced Applications in Pattern Recognition. Springer-Verlag Berlin Heidelberg.

[3] Chauvin, Y. and Rumelhart, D. E. (1995). *Backpropagation: Theory, Architectures, and Applications*. L. Erlbaum Associates Inc., Hillsdale, NJ, USA.

[4] Duch, W. and Jankowski, N. (1999). Survey of neural transfer functions. *Neural Computing Surveys*, 2(1):163–213.

[5] Fletcher, R. (2013). *Practical Methods of Optimization*. John Wiley & Sons.

[6] Hagan, M. T. and Menhaj, M. B. (1994). Training feedforward networks with the marquardt algorithm. *IEEE Transactions on Neural Networks*, 5(6):989–993.

[7] Halgamuge, S. K. (2005). *Classification and Clustering for Knowledge Discovery*. Studies in Computational Intelligence. Springer-Verlag Berlin Heidelberg.

[8] Hestenes, M. (1980). *Conjugate Direction Methods in Optimization*, volume 12. Springer-Verlag New York.

[9] Himberg, J. (2000). A som based cluster visualization and its application for false coloring. *Proceedings of the IEEE-INNS-ENNS International Joint Conference on Neural Networks, IJCNN 2000*, 3:3–6.

[10] Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural Computation*, 9(8):1735–1780.

[11] Hopfield, J. J. (1982). Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the national academy of sciences*, 79(8):2554–2558.

[12] Jaeger, H. (2007). Echo state network. *Scholarpedia*, 2(9):2330. revision #186227.

[13] Khadir M. Tarek, Khdairia Sofiane, B. F. (2010). *Kohonen Maps Combined to K-means in a Two Level Strategy for Time Series ClusteringApplication to Meteorological and Electricity Load data*. Numerical Analysis and Scientific Computing. Springer-Verlag Berlin Heidelberg.

[14] Kohonen, T. (2001). *Self-Organizing Maps*. Springer Series in Information Sciences. Springer-Verlag Berlin Heidelberg.

[15] Larry Medsker, L. C. J. (1999). *Recurrent Neural Networks: Design and Applications*. International Series on Computational Intelligence. CRC Press.

[16] M. Schuster, K. P. (1997). Bidirectional recurrent neural networks. *IEEE Transactions on Signal Processing*, 45(11):2673–2681.

[17] Minai, A. A. and Williams, R. D. (1990). Back-propagation heuristics: a study of the extended delta-bar-delta algorithm. 1(1):595–600.

[18] Møller, M. F. (1993). A scaled conjugate gradient algorithm for fast supervised learning. *Neural Networks*, 6(4):525 – 533.

[19] Rojas, R. (1996). *Neural Networks a Systematic Introduction.* Springer-Verlag Berlin Heidelberg.

[20] Silva, F. M. and Almeida, L. B. (1990). *Acceleration techniques for the backpropagation algorithm*, volume 412. Springer Berlin Heidelberg, Berlin, Heidelberg.

[21] S.M. Szilágyi, D. Iclănzan, L. S. Z. B. (2009). Intensity inhomogeneity correction and segmentation of magnetic resonance images using a multi-stage fuzzy clustering approach. *International Journal on non-standard computing and artificial intelligence*, 19(5):513–528.

[22] Syed Muhammad Aqil Burney, Tahseen Ahmed Jilani, C. A. (2005). A comparison of first and second order training algorithms for artificial neural networks. *International Journal of Computational Intelligence*, 1(3):176–182.

[23] The World Bank (2015). World development indicators. data retrieved from World Development Indicators, `https://data.worldbank.org/indicator`.

[24] Tollenaere, T. (1990). Supersab: Fast adaptive back propagation with good scaling properties. *Neural Networks*, 3(5):561 – 573.

[25] Y. Bengio, P. Simard, P. F. (1994). Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, 5(2):157–166.

List of Figures

1.1 **Biological neuron** 5
1.2 **Diagram of a perceptron** 6
1.3 **Multilayer perceptron diagram** 7
1.4 **Sigmoid transfer function** 8
1.5 **Hyperbolic tangent transfer function** 9
1.6 **ReLU transfer function** 9
1.7 **Linear transfer function** 10
1.8 **Early stopping** 13

3.1 **Graphical representation of an RNN.** 19
3.2 **Graphical representation of an unfolded RNN.** 20
3.3 **LSTM module at time t.** 24

4.1 **Graphical representation of a Kohonen map** 27
4.2 **Two-phase clustering process.** 30

5.1 **MLP approximation of $\sin(x)$** 31
5.2 **RNN approximation of $\sin x$** 32
5.3 **A graphical representation of the generated 2-D test data** 33
5.4 **Neuron excitation for the different data points in the Kohonen's map.** 34

6.1 **Graphical representation of the SOM layer.** 44
6.2 **Projection of time-series from Spain** 45
6.3 **Projection of time-series from Peru** 46
6.4 **Projection of time-series from Puerto Rico** 46
6.5 **Different runs of the fuzzy C-means clustering algorithm, each time with a different number of clusters.** 47

A. Implementation and documentation

In this appendix we describe the contents of the appended files and how to run the experiments using our code.

A.1 Project overview

The following diagram describes the directory structure and the locations of the files and scripts in the attachment:

```
/
├── thesis.pdf
├── src
│   ├── GDPPrediction.m
│   ├── inputData.xlsx
│   ├── meanSquaredError.m
│   ├── normalisation_minmax.m
│   ├── normalisation_std.m
│   ├── normalisation_tanh.m
│   ├── outputData.xlsx
│   ├── Predictions.m
│   ├── RNNPredictions.m
│   ├── SOM01.m
│   ├── testCluster.m
│   └── testSin.m
└── results
    ├── netSom.mat
    ├── MLPtraingdlogsig10.csv
    ├── MLPtraingdlogsig15.csv
    └── ...
```

The data is divided into two folders, results and src. The results folder contains the experiment results - the tables obtained by the GDP prediction experiments and the trained SOM network used for the visualization. The src folder contains the MATLAB scripts, which are used for the visualization and GDP prediction, and the data used in all our experiments.

A.2 Requirements and installation

The scripts were developed and run with MATLAB version 2018a. Moreover, the Neural Network Toolbox is required to successfully execute the files. The scripts generate csv files with the entries for the tables in the GDP prediction task.

A.3 Running the data visualization experiments

In order to run the visualization experiments, we first need to add the folder containing the *.m scripts to the MATLAB search path . This can be done by searching for the src folder with the ”Browse for folder” tool and setting it as the current folder. Next, there are two options:

1. Loading the trained SOM network.

2. Training a new SOM.

For the first option, we only need to run the command

```
load netSom.mat
```

in the MATLAB shell, which will load the trained SOM (*net*) and all variables required for the visualization of the time-series. Once these are loaded, we can run the following command in the MATLAB console to visualize the Kohonen map and the distances between the neurons, as in Figure 6.1.

```
plotsomnd(net)
```

One of the variables that are created when we load the ”netSom.mat” file is the input_per_country matrix. As its name suggests, this matrix contains the data for the different countries separated. We can access the data for each individual country through an index in the matrix. The data for the i-th country can be accessed through input_per_country(:,:,i). With this, we can plot the projection of any of the countries onto the trained SOM layer. As an example, we show the code used for projecting the time-series for Spain onto the lattice.

```
plotsomhits(net,input_per_country(:,:,1))
```

If we would like to project the data for the i-th country in the j-th year, we can run the following command in the MATLAB console:

```
plotsomhits(net,input_per_country(:,j,i))
```

Note that $i \in [1, 19]$ and $j \in [1, 36]$. We can observe that MATLAB does not connect the time-series with a line in the plots, as shown in Figure 6.2, or Figure 6.3. These images were produced independently.

In order to get the coordinates of the activated neurons for the projection of the data, we can run the following code:

```
years = [1980:2015];
Out = sim(net, input_per_country(:,:,i));
Neurons = vec2ind(Out)-1;
NeuronsRow = fix(Neurons/20);
NeuronsCol = mod(Neurons,20);
Coords = [years;NeuronsCol;NeuronsRow];
```

where $i \in [1, 19]$ is the country that we want to project, and Coords contains the coordinates of the activated neurons and the corresponding year.

If we would like to train a new Kohonen map, we need to run the ”SOM01.m” file (by clicking the ”run” button in the MATLAB GUI). There are several parameters that we can change within it. First, we can set the normalization method to be used for transforming the input data. The normalization by standard deviation is the function used by default, it is set in lines

```
input = normalisation_std(xlsread('inputData.xlsx'));
output = normalisation_std(xlsread('outputData.xlsx'));
```

but functions ”normalisation_minmax” or ”normalisation_tanh” are defined as well, and can be used instead.

The number of rows and columns of the lattice are stored in the variables *dimension*1 and *dimension*2, respectively. These are set to 20 and 20 by default, but they can be changed to any positive natural number. Further, the number of training iterations is stored in *net.trainParam.epochs*, and is set by default to 50000, but it can be set to any positive integer. After the training is complete, we can create all the graphs and tables mentioned above with the exact same code.

Clustering the activated neurons by means of the fuzzy c-means algorithm (Section 4.3) and plotting the result is done by running the following piece of code

```
[centers,U] = fcm(input_final', c, [m epochs e true]);
plotsomhits(net,centers');
```

where c is the number of clusters that we would like to have, m is the exponent, *epochs* is the total number of iterations that we want the algorithm to execute, and e is the threshold for improvement. By default, these are set to $c \in [10, \ldots, 15]$, $m = 2.0$, $epochs = 200$, and $e = 1e - 5$ as explained in Section 6.2.1.

A.4 Running the GDP forecasting experiments

All the data obtained in Table 6.3 through Table 6.17 are the result of calling the ”GDPPrediction.m” and ”RNNPrediction.m” files with different arguments. The first thing these files do is to load and normalize the data. The default normalization technique is again set to normalization by standard deviation, but it can be set to ”normalisation_minmax” or ”normalisation_tanh” in lines

```
input = normalisation_std(xlsread(traininput));
output = normalisation_std(xlsread(trainoutput));
```

Then, they train and test n (50 by default) models with the same parameters. At the end, some statistical data about the trained models are written into a csv file in the working directory. The content of this file is one line with six values representing the mean training, validation, and testing errors, the average number of iterations, the average time (in seconds) taken by the algorithm, and the confidence interval. In each of the n iterations, the data set is divided into k (12 by default) equal-size parts, which are then used for performing the k-fold cross validation technique described in Section 1.7.1.

We can execute these files with the following commands

```
GDPPrediction('inputData.xlsx','outputData.xlsx',trainFnc,
    transferFnc,arch,maxFail,maxTime,maxEpochs,minError,alpha,n,k,
    confidence);
RNNPrediction('inputData.xlsx','outputData.xlsx',trainFnc,
    transferFnc,arch,maxFail,maxTime,maxEpochs,minError,alpha,n,k,
    confidence);
```

where all the parameters have to be set to concrete values. Examples of different runs of these scripts are shown in the ”Predictions.m” file.