TECHNISCHE UNIVERSITÄT MÜNCHEN
Fakultät für Informatik
Lehrstuhl VI Robotics and Embedded Systems

Modeling of Dynamical Systems with Complex-Valued Recurrent Neural Networks

Alexey S. Minin

Vollständiger Abdruck der von der Fakultät für Informatik der Technischen Universität München zur Erlangung des akademischen Grades eines Doktor der Naturwissenschaften (Dr. rer. nat.) genehmigten Dissertation.

Vorsitzender:	Univ.-Prof. Dr. Bernd Radig
Prufer der Dissertation:	1. Univ.-Prof. Dr. Alois Knoll
	2. Prof. Dr. Mark J. Embrechts, Rensselaer Polytechnic Institute, New York / USA.

Die Dissertation wurde am 01.02.2012 bei der Technischen Universität München eingereicht und durch die Fakultät für Informatik am 04.04.2012 angenommen.

Abstract

This thesis aims in collecting and generalizing facts about complex valued neural networks, extends feedforward neural networks to the complex valued case and applies complex valued neural networks to a set of real-world systems models. A brief review on real valued neural networks is presented to introduce notations and terminology, followed by a then a brief review of complex analysis. Complex valued system identification is extensively addressed in this thesis and novel. The difficulties of the complex valued minimization problem are explained and ways to overcome these problems are addressed. The generalization of real-valued feedforward neural networks to complex valued neural networks is introduced and this theory is then generalized to recurrent complex valued neural network in general. New interpretations of the error function are highlighted which lead to a novel recurrent architecture, which allows for continuous time modeling.

The developed methods are applied to two artificial chaotic time series datasets (i.e., the logistics map and the Lorenz system) to benchmark the novel approach introduced in this thesis. Complex recurrent neural networks are then applied to several real-world applications including (i) nonlinear transformer modeling, (ii) financial time series forecasting, (iii) the simulation of neuron synchronization in a real brain model, and (iv) a novel approach to solve the Binding problem. These applications clearly demonstrate the advantages of complex valued neural networks compared to the traditional real-valued approach. For example, the advantages are (i) possibility for continuous dynamics modeling, (ii) natural processing of complex-valued data, (iii) beter training procedures. Practical aspects for the implementation of complex-valued neural networks and some fine points to consider in benchmarking studies for regression problems are discussed.

Acknowledgements

Preparation of the current thesis was not only an interesting time, but also a hard work. A lot-of people supported me during the PhD studies and in thesis preparation, some of them I would like to mention, since their impact has been the most essential.
I would like to express my respect to the following people: Prof. Alois Knoll who was supporting me all the time, Dr. Hans-Georg Zimmermann for consultations and long evenings in neural network discussions, Dr. Martin Gitsels for his full support during the PhD studies from Siemens LLC side, Ms. Heidi Embrechts and Oxana Konareva for their outstanding support in thesis grammar checking, Prof. Mark J. Embrechts for his reasonable advises and improvements, Mr. Bernhard Lang for his instant support and advises, of course my parents and my brother (Mr. Sergey Minin, Ms. Elena Minina and Mr. Dmitriy Minin), which supported me all the time and to Mr. Alexander Peckarovskiy for his support with programming Benchitnow.de. Last nut not least are Dr.-Ing. Gerhard Schrott, Ms. Amy Bucherl and Gertrud Eberl which did a great job in consulting me with formalities and administration issues. Thank you all!

Contents

List of Figures

List of Tables

Glossary

ANN Artificial Neural Network

BDN Bounded Derivative Neural Network

BFGS BroydenFletcherGoldfarbShanno method

BP Back Propagation

BPTT Back Propagation Through Time

CG Conjugated Gradient

CRNN Closed system Recurrent Neural Network

CVBP Complex-Valued Back Propagation

CVCRNN Complex-Valued Closed system Recurrent Neural Network

CVFFNN Complex-Valued FeedForward Neural Network

CVGD Complex-Valued Gradient Descent

CVLR Complex-Valued Linear Regression

CVORNN Complex-Valued Open system Recurrent Neural Network

CVRNN Complex-Valued Recurrent Neural Network

DJIA Dow Jones Industrial Average

EA Evolutionary Algorithms

FFNN FeedForward Neural Network

GA Genetic Algorithms

GD Gradient Descent

GN Gauss Newton

HCNN Historical Consistent Neural Network

HCRNN Historical Consistent Real Valued Neural Network

HCVNN Historical Consistent Complex Valued Neural Network

MC Monotonicity Condition

MLP Multi Layer Perceptron Network

NN Neural Network

ORNN Open system Recurrent Neural Network

PC Personal Computer

RNN Recurrent Neural Network

RSA Random Search Algorithm

RVBP Real-Valued Back Propagation

RVFFNN Real-Valued FeedForward Neural Network

RVGD Real-Valued Gradient Descent

SOM Self Organizing Map

SQP Sequential Quadratic Programming

1

Introduction

Neural Networks form a modern discipline applicable to a variety of sciences, like physics (see [65]), medicine, economics (see [57], [35], [26], [61]), robotics (see [50], [24]) etc. There are thoutshands of applications for Neural Networks (NN)(e.g., see [64], [65], [47]) and therefore there is a lot of attention devoted to this topic. Real-Valued NN (RVNN) have been studied extensively. There are numerous papers on architectures (see [15], [37], [36]), types of training (see [15], [17]), internal semantics of neural networks (see [7], [5], [29]), benchmarking procedures to estimate the quality of a network (see [45]) etc. All these papers have been dealing with real-valued numbers and up to the moment it has been sufficient.
Thesis introduction is divided into two subsections, namely "Review on Real-Valued Neural Networks" and "Advantages of Complex-Valued Neural Networks". When the study on complex-valued neural networks have been started I thought that when one starts dealing with complex numbers everything related to real-valued networks crashes. Since this is about 90 percent of the complete knowledge on NN it was rather a disappointing prospect. After some time it turned out, that all I knew about the real-valued networks can be still be applied to complex-valued analogs. This thesis documents the transition from real-valued neural networks to complex-valued neural networks to show, that the majority of the knowledge in this domain can be extended and generalized for complex-valued networks and even some new features and properties can appear. In addition the most sophisticated and interesting part of the thesis will be elaborated in the thesis about the extension of recurrent complex-valued neural networks to the continuous time modeling. At the end of each chapter I will discuss the advantages of the complex-valued networks in comparison to the real-valued ones.
The structure of the thesis is the following: first I make a review to discuss some special chapters of complex-valued analysis, that it is easier for the reader to follow the ideas, then I will make a smooth transition from the real-valued system identification to the complex-valued one, after this I will discuss the problems

of the non linearities in complex-valued planes and how to overcome these problems, then I will discuss the complex-valued backpropagation algorithm. After the theoretical aspects of the thesis are finished I will talk about feedforward architectures, recurrent architectures and their applications for the real world systems. Then I will discuss the implementation specifics and some informatics topics related to the implementation of the complex-valued neural networks.

1.1 Review on Real-Valued Neural Networks

There were always some sets of applications where NN were obliged to work, but there was no success in these areas despite tremendous efforts done by the researchers. Examples of such applications can be found in economics [11], finance [13], some types of control applications [52]. Many problems in NN arise out of the paradigms which serve as the foundation for the complete machine learning area. Some of these foundations are based on mathematics; others are empirically based. Several constraints arise out of the paradigms of neuro-computing. For example, in case related to equidistant records of some systems behavior, only given timestamps forecasting is possible, which means that only discrete time modeling is possible. Another issue is that the transition (activation) functions of the neurons of the neural networks must be bounded; otherwise any computation will become unstable. Thus, only a limited class of functions can be used for NN modeling. In the past few decades, many scientists have paid much attention to the "whitening of the black box". Listed below are many references related to the expert knowledge induction: Lang [12], Sill [41][42], Tarca [64], Minin [5, 7]. The evidence of such "whitening" can be given with the monotonicity example. Prior knowledge about the monotonicity, smoothness and upper or lower bounds of inputs and/or targets can increase the robustness of neural networks training procedures. Here the monotonicity will be adressed explicitly. Such simple rules like "if A increases, then B will increase as well" can be transferred to the monotonicity constraint $dB/dA > 0$. Such behavior is often known, so that one can extend the classical training approach, namely the minimization of cost functions. During the "whitening" of the "black box" several cross-disciplinary methods emerged. Last decades the dozens of architectures appeared as well as numerous methods for network training. One can find the list of the training methods below. Training of neural networks is still one of the main issues. Dozens of different training algorithms (i.e. optimization schemes) exist. But the mechanism for error back propagation remains. Several authors consider training algorithms apart from the inner structure of the network ("black box" optimization). There are dozens of training methods in existance, based from gradient descent optimization and to the Genetic Algorithm based optimization.

The use of genetic algorithms (GA) for NN training is a promising development; on the other hand the training of the human brains utilizes another algorithm since GA requires much time for training. Sometimes, usage of the sequential quadratic programming (SQP) algorithm is needed due to the constrained optimization routine. The widely known methods for neural network training have their advantages and disadvantages which are briefly described in this thesis. The list of the training algorithm is provided below, along with their advantages and disadvantages:

- Deterministic Algorithms: used when the initial parameters of the neural network (weights) are close to the optimal values.
  - Advantages: Fast and robust convergence.
  - Advantages: Easy to explain and predict the behavior of the algorithm.
  - Disadvantages: Can be trapped in a local optimum.
  - Disadvantages: Very sensitive to the total number of variables.
- Stochastic Algorithms: used when the network parameters (weights) are far from being optimal and the initialization is not possible.
  - Advantages: Very robust for highly non linear data.
  - Advantages: Depend only on the relevant variables, do not depend on the total number of variables.
  - Disadvantages: Brute force approach, time consuming.

Examples of the stochastic algorithms as well as deterministic algorithms are given below:

- Stochastic algorithms
  - Branch and bound (slow for big problems, easy to program, very general).
  - Evolutionary algorithms (EA) (do not get trapped in local minimum, good for noisy functions, not stable, time consuming).
  - Simulated annealing (heavily dependent on starting point, difficult to establish the "temperature", good for big problems and nice physical meaning).
  - Tabu search (do not get trapped, good convergence, loops are possible, too many external parameters).
  - Random Search (do not get trapped, good convergence).

- Deterministic algorithms
  - Gradient descent (GD) (fast convergence, stack in local optima, possible fluctuations around the optima).
  - Conjugated Gradients (CG) methods (e.g. BFGS).
  - Gauss-Newton (GN) (slow convergence, no fluctuations around the optima).
  - Levenberg-Marquardt (mixture of GD and GN, fast convergence).

Many advances in stochastic training and training in general can be found in Zimmermann's papers [32, 33, 55, 60, 70, 71, 72, 73, 74, 75, 76, 78]. All of these algorithms have advantages and disadvantages, nevertheless one can see, that GD and EA are the most commonly used methods NN training. Another important issue related to the selection of the appropriate architecture for the given problem.

Architecture selection is a very important step when working with neural networks. There are two main types of NN architectures: FeedForward (FFNN) and Recurrent (RNN). The main difference between the two is that FFNN behaves like forward path filter, (the information propagates directly through the filter and produces the output), while the recurrent network has back connections which propagate the information from the output back to the input. Obviously the performance of each NN must be compared with a Linear Regression (further LR), since this is the most basic model one can construct. If linear models provides a good enough result, there is no need for more sophisticated models.
Last but not least is the selection of the activation function inside the nodes. It is clear that if the process one is trying to model is a Gaussian-like function, the activation function of the neural networks should be Gaussian. On the other hand, if the process function is unknown, the network should be able to approximate any arbitrary nonlinear function. The list of references used below shows, that major steps in this direction were done at the end of the 20^{th} century. Together with node pruning, early stopping almost all basis ideas were introdiced during the $1985 - 1998$ period. A lot of researchers are still developing the alternating ideas for the discussed above, which means that the NN society pays much attention to the architecture problem, but the problem remains unsolved. Here is the short list of such papers: Moody [40], Gomez [37], Barron [1],Haykin [58] and Fiszelew [2]. One should also admit, that there were a lot of attempts to automate the NN architecture selection with the different optimization algorithms: genetic algorithms, simulated annealing, taboo search, ant optimization etc (e.g., see Kondo [62]).

At some point, each neural network should be implemented at some real world application. However, real world applications come with many different practical constrains such as: amount of memory, CPU capacity, data type (float, double, integer) etc. Some of these constraints are discussed in the papers below. One of the major advances in this area was the creation of weightless machines compatible with computer hardware without any add-ons. One can find more information in the papers by Mayer [50], Erlhagen [24] and Alexander [36].
Moreover in order to perform NN training one needs linear algebra operations and CPU capacity for these operations (consider matrix operations).

Last but not least is the comparison of the results provided by different architectures, training methods etc. The philosophy and the semantics of NN have faded into the background, while the purely scientific approach for data approximation and data classification is gaining speed. Unfortunately, the neural network community had not established any unique benchmarks, rules and stamps for neural networks evaluation. Therefore dozens of architectures have appeared and many papers have stated that their neural network is capable of solving "the world's problems". On the other hand, some authors have specified the benchmarks, which can be used by the community (at least as an attempt for a comprehensive benchmarking). The most interesting fact is that most papers were written long time ago. Here is a list of papers, which attempt to approach the problem of benchmarking. One can find more information in the papers by Prechelt [45],Waugh [59] [20] and Ciftcioglu [53]. Unfortunately, the suggested datasets like "Proben 1" and some other collections were submitted long time ago and do not cover the existing state of the art concepts for result validity checking.

One can see from the list above, that it is impossible to compare the network with each other due to the wide range of data available in the internet. In general, one can argue that well known benchmarks exist, like ELENA or IRIS or WINE or BOSTON etc. These datasets are definately not new. The second thing, which should be discussed are benchmarking procedures. Even when the benchmark data are adequate, the benchmarking itself can be rather poor. Here one should refer to the Prechelt [45] due to the fact that his four criteria approach is not followed by most of the papers (Validity, Reproducibility, Comparability, and Volume). Last but not least the statistical performance estimations. After the forecast or classification has been comed out, one should estimate the quality of the neural network. Here the most popular measures are RMSE, R^2 (determination coefficient) and r (correlation coefficient). In general, there are other much more interesting measures one would want to know: presence of delays, border effects, how good the dynamics is coded into the weights etc.
Summarizing the discussion of the real-valued neural network one can say that this area is well studied and well established. Unfortunately, a big part of this

knowledge is no longer applicable when we change the algebra from real-valued variables to a complex-valued variables and this discussion will continue in the thesis.

1.2 Advantages of Complex-Valued Neural Networks

Complex-Valued Neural Network is a more complicated algorithm for complex-valued data processing. To make the neuron model work with vector information one has to implement complex-valued neuron, which must have all the properties of a real-valued neuron. Such proeprties are: boundness, differentiability of errors etc. The problem is that due to a lot of constraints this does not held in complex-valued case. Newertheless a complex-valued neural network construction is still possible. The remaining part of the thesis will show that, under certain conditions, one can use all the knowledge which exist for a real-valued theory. The thesis will demonstrate a transition from a real-valued neural networks to a complex-valued neural networks. Moreover, the thesis will present the evidence, that complex-valued neural networks work better in some case and allow to solve the problems which were not possible to solve with real-valued neural networks. Thesis also demonstrates that usage of complex-valued networks with the data, which is essentially complex is much more convenient and robust in terms of things like approximation quality, error decay, etc. During the thesis preparation author have published some papers about the complex-valued neural networks, which describe how elegant and beatiful the models are:

1. Yury S. Chistyakov, Elena V. Kholodova, Alexey S. Minin, Hans-Georg Zimmermann, Alois Knoll Modeling of electric power transformer using complex-valued neural networks at the 2011 IEEE International Conference on Smart Grid and Clean Energy Technologies, China, 2011, in press.

2. Zimmermann H.-G., Minin A., Kusherbaeva V., Historical consistent complex valued recurrent neural network, ICANN 2011, Part I, LNCS 6791, pp. 185-192, 2011.

3. Zimmermann H.-G., Minin A., Kusherbaeva V., Comparison of the complex valued and real valued neural networks trained with gradient descent and random search algorithms, Proc. of ESANN 2011, pp. 216-222, 2011.

4. Minin A., Knoll A. and H.-G. Zimmermann, A Novel Approach to Solving the Binding Problem Using Complex Valued Spiking Neural Networks, accepted to ESANN2012.

5. Minin A., Knoll A. and H.-G. Zimmermann, Complex Valued Recurrent Neural Network: From Architecture to Training, accepted to Journal of Signal and Information Processing, Scientific Research, 2011.

6. Minin A., Chistiakov Yu., Knoll A., Zimmermann H.-G., Complex Valued Open Recurrent Neural Network for Power Transformer Modeling, accepted to Journal on Applied Mathematics, WSEAS, 2011.

All these papers summarize the advantages and disadvantages of the complex-valued neural networks, one can see, that complex-valued networks open a new era for the machine learning algorithms. During the thesis time I have published 7 papers at the leading machine learning conferences and journals related to the machine learning and neural networks, where six out of them are devoted to the complex-valued neural networks and present the recent success in this area. First paper shows the transformer model and how good it can be modeled with CVNN. Second and third papers are theoretical and devoted to the CVNN construction, training and evaluation. Fifth paper is related to the novel approach for solving the Binding Problem with the use of the RCVNN. Sixth paper is devoted to the theoretical aspects of the RCVNN, e.g. architectural questions, training issues and error function behavior discussion. By having this papers one can efficiently utilize the RCVNN and exploit it for personal needs.

2

Mathematical Basis for Complex-Valued Neural Networks

This chapter is very important since the notations and theorems developed in it will be used throughout the thesis. Complex analysis, typically known as the theory of functions of complex variables, is the branch of mathematics which investigates functions of complex variables. It is useful in many branches of mathematics, physics, including quantum mechanics, hydrodynamics, thermodynamics and electrical engineering.

2.1 Brief Review of Complex Analysis

A complex function is a function in which the independent variable and the dependent variable are both complex numbers. More precisely, a complex function is a function, which values and range are subsets of the complex plane [68]. For any complex function, both the independent variable and the dependent variable can be split into real and imaginary parts according to:

$$z = x + iy = re^{i*\phi} \tag{2.1}$$

where r is an absolute value and ϕ is an angle. This is the so-called Euler notation. A general definition for a complex function can now be formulated as follows:

$$w = f(z) = u(x, y) + iv(x, y) = f\left(re^{i*\phi}\right) \tag{2.2}$$

where $x, y \in \Re$ and $u(x, y), v(x, y)$ are real valued functions [68].

2. MATHEMATICAL BASIS FOR COMPLEX-VALUED NEURAL NETWORKS

Definition. *Holomorphic functions are complex functions defined on an open subset of the complex plane which are differentiable.*

Definition. *In complex analysis, an entire function, also called an integral function, is a complex-valued function that is holomorphic over the whole complex plane.*

Complex differentiability has much stronger requirements than real differentiability. For example, holomorphic functions are infinitely differentiable; this is not true for real differentiable functions. Most elementary functions, trigonometric functions, exponential function, and all polynomial functions, are holomorphic.

Definition. *The derivative of the complex valued function can be defined in the following way:*

$$f'(z) = \lim_{\delta z \to 0} \frac{f(z + \delta z) - f(z)}{\delta z} \tag{2.3}$$

Let us now discuss the line integral. The integral around a closed path of a function which is holomorphic everywhere inside the area bounded by the closed path is always zero; this is the Cauchy integral theorem. The values of a holomorphic function inside a disk can be computed by a certain path integral on the disk's boundary (Cauchy's integral formula, see eq.2.4).

$$\oint_\gamma f(z)\, dz = 0 \tag{2.4}$$

Cauchy's integral formula (named after Augustine-Louis Cauchy) is a fundamental statement in complex analysis. It explains the fact that a holomorphic function defined on a disk is completely determined by its values on the boundary of the disk. It provides integral formulas for all derivatives of a holomorphic function. Cauchy's formula shows that, in complex planes, "differentiation is equivalent to integration": complex differentiation, like integration, behaves well under uniform limits - a result which is not achievable in real analysis.
Suppose U is an open subset of the complex plane C, $f : U \to C$ is a holomorphic function and the closed disk $D = [z : |z - z_0| \leq r]$ is completely contained in U. Let γ be the circle making the boundary of D. Then for every z_0 in the interior of D:

$$f(z_0) = \frac{1}{2\pi i} \oint_\gamma \frac{f(z)}{z - z_0}\, dz \tag{2.5}$$

where the contour integral is taken counter-clockwise.
The proof of this statement uses the Cauchy integral theorem and requires func-

tion f to be complex differentiable. In particular f is actually infinitely differentiable:

$$f^{(n)}(z_0) = \frac{n!}{2\pi i} \oint_\gamma \frac{f(z)}{(z - z_0)^{n+1}}\, dz. \tag{2.6}$$

This formula is sometimes referred to as Cauchy's differentiation formula. The circle γ can be replaced by any closed curve in U which has winding number one around z_0. After the small overview of the complex number derivatives one should proceed with the practical things, namely activation functions and derivatives for the nonlinear functions. The derivative definition is presented below:

$$\left.\frac{df\,(z)}{dz}\right|_{y=const} = \frac{\partial u\,(x,y)}{\partial x} + i\frac{\partial v\,(x,y)}{\partial x} \tag{2.7}$$

This is a correct definition of the complex valued derivative in case one is taking into account the Cauchy-Riemann equations given below:

$$\frac{\partial u}{\partial x} = \frac{\partial v}{\partial y} \text{ and } \frac{\partial u}{\partial y} = -\frac{\partial v}{\partial x} \tag{2.8}$$

In case that eq. **??** and eq.2.8 are applicable, one can say that this function is analytic and complex differentiable.

2.2 System Identification

System Identification is fundamental for the study of neural networks. Real-valued system identification has been widely adressed in the literature and is therefore only briefly discussed in the following subsection. Challenges for the identification of the complex valued systems will be adressed in the corresponding subsection.

2.2.1 Real-Valued System Identification

For real-valued systems, system identification is a well-known. Consider the following cost function:

$$E = \frac{1}{T}\sum_{t=1}^{T} \frac{1}{2}\left(y_t\left(wx_t\right) - y_t^d\right)^2 \to \min_w \tag{2.9}$$

where w are system parameters, x_t are system inputs and y_t^d is the desired system output, T is the number of input patterns and E is the RMS error. Then, in order to find the parameters w which delivering the system to its desired values, one should be able to perform a system parameters optimization. One can use a

gradient descent for this purpose. In order to be able to apply gradient descent, one should be able to calculate the derivatives for the cost function with respect to its parameters.

$$\frac{\partial E}{\partial w} = \frac{1}{T}\sum_{t=1}^{T}\left(y_t - y_t^d\right)\frac{\partial y}{\partial w} =: g \tag{2.10}$$

After the gradients are known, it is possible to establish a simple optimization rule, which can be derived from the following Taylor expansion.

$$E(w + \Delta w) = E(w) + g^T \Delta w + \tfrac{1}{2}\Delta w^T G \Delta w \tag{2.11}$$

A very simple rule in order to minimize the cost function now can be presented as:

$$\Delta w = -\eta \cdot g \tag{2.12}$$

Thus, for a small η, which is sometimes called "learning rate", one can easily find the optimal parameters w for the system.

2.2.2 Challenges for Complex-Valued System Identification

Complex valued system identification is hardly possible due to the following reasons. First of all, one must establish a cost function, which in the complex-valued case must be analytic in order to have existing derivatives. On the other hand, minimization in the complex planes cannot be defined due to the lack of relations such as "bigger" or "smaller". The question arises how to proceed with the system identification, when the system parameters are complex numbers and which cost function to use to make system identification possible? The answer is given by the Wirtinger calculus. On the other hand, authors often use the following cost function in several papers:

$$E = z \cdot \bar{z} = \frac{1}{T}\sum_{t=1}^{T}\left(y_t - y_t^d\right)\overline{\left(y_t - y_t^d\right)} \tag{2.13}$$

where $\overline{(\cdot)}$ means conjunction. This cost function operates with complex numbers and has the following property $f : \Im \to \Re$. Unfortunately this cost function is not analytic, which means that $\frac{\partial E}{\partial w}$ is not defined for the complete $\Im$. This fact is easy to prove. Let us consider $z = y_t - y_t^d$. Then:

$$\frac{\partial z \cdot \bar{z}}{\partial z} = \bar{z} + z\frac{\partial \bar{z}}{\partial z} \tag{2.14}$$

One can see that, namely, $\frac{\partial \bar{z}}{\partial z}$ is not defined in mathematical sense because:

$$\frac{\partial \bar{z}}{\partial z} = \lim_{h \to 0} \frac{\overline{(z+h)} - \bar{z}}{h} = \lim_{h \to 0} \frac{\overline{h}}{h} = \left\{ \begin{array}{cc} 1 & h \in \Re \\ -1 & h \in i\Re \end{array} \right\} \tag{2.15}$$

To overcome problems with the cost function derivative as well as problems with the transition function derivatives, one should use Wirtinger calculus discussed below.

2.2.3 Wirtinger Calculus

In order to have a defined derivative, the function must be analytic (the requirements for analyticity were discussed above). Then the complex function $f(z)$ must be decomposed into two functions dealing with the real and imaginary parts separately, namely $f(z) = u(x,y) + iv(x,y)$, here $z = x + iy$. As discussed above, the requirements for the analytic functions are given by the Cauchy-Riemann equations 2.29. Following the introduced notations, this would mean, that:

$$\left.\frac{df\,(z)}{dz}\right|_{y=const} = \frac{\partial u\,(x,y)}{\partial x} + i\frac{\partial v\,(x,y)}{\partial x} \tag{2.16}$$

if one wants to deal with the complex cost function f the following problem is to be faced: in the complex plane ordering relations like "bigger" or "smaller" are not valid. Therefore optimization of such function makes no sense. The only way to proceed is to construct the real-valued cost function which depends on several complex arguments. Instead of treating the $f(z)$ as a real function of the complex variable one can treat $f(z) = u(x,y)$ as a real function of two real variables.

Remark. $v(x,y)$ must be equal to zero. The only function in its class is $f(z,\bar{z}) = z \times \bar{z}$. The output of this function is a real number, therefore $v(x,y) = 0$. One can see that such a function is not analytic.Therefore there is no defined derivative.

The advantage of such a function is that for such functions ($f(z) = u(x,y)$) one can pose the minimization problem in the following manner:

$$f\,(z) \to opt \equiv u\,(x,y) \to opt \tag{2.17}$$

which means that:

$$\left\{ \begin{array}{l} \dfrac{\partial u\,(x,y)}{\partial x} = 0 \\ \dfrac{\partial u\,(x,y)}{\partial y} = 0 \end{array} \right. \tag{2.18}$$

2. MATHEMATICAL BASIS FOR COMPLEX-VALUED NEURAL NETWORKS

First of all, one should not consider the real-valued function as a non differentiable mapping from $\Im \to \Re$ but should rather consider it as a mapping from $\Re^2 \to \Re$. When one wants to minimize the real-valued function, one should consider the "normal" gradient of the function, thus causing the mapping to occur from $\Re^2 \to \Re$, even if the complex valued derivative does not exist. Wirtinger calculus can help in differentiation of the nonholomorphic functions and to find a way to minimize the real functions of the complex variables.

Definition. *According to Wirtinger, the partial derivatives of a complex function $f(z)$ of a complex variable z with respect to the z and $\bar{z}$ can be calculated in the following way:*

$$\begin{cases} \dfrac{\partial f}{\partial z} \triangleq \dfrac{1}{2}\left(\dfrac{\partial f}{\partial x} - i\dfrac{\partial f}{\partial y}\right), z = x + iy \\ \dfrac{\partial f}{\partial \overline{z}} \triangleq \dfrac{1}{2}\left(\dfrac{\partial f}{\partial x} + i\dfrac{\partial f}{\partial y}\right), \overline{z} = x - iy \end{cases} \tag{2.19}$$

The so-called Wirtinger calculus approach allows us to calculate the derivatives of non analytic functions. Lets discuss it in more details.
One can derive the following equations for the Wirtinger derivatives:

$$\begin{cases} \dfrac{df(z)}{dz} = \dfrac{1}{2}\left(\dfrac{\partial u(x,y)}{\partial x} + \dfrac{\partial v(x,y)}{\partial y} + i\left(\dfrac{\partial v(x,y)}{\partial x} - \dfrac{\partial u(x,y)}{\partial y}\right)\right) \\ \dfrac{df(z)}{d\overline{z}} = \dfrac{1}{2}\left(\dfrac{\partial u(x,y)}{\partial x} - \dfrac{\partial v(x,y)}{\partial y} + i\left(\dfrac{\partial v(x,y)}{\partial x} + \dfrac{\partial u(x,y)}{\partial y}\right)\right) \end{cases} \tag{2.20}$$

Taking the above conditions into account, we can calculate the derivative, which indicates the optimal direction for the parameters. For the Cauchy-Riemann equations, one can rewrite eq.2.20 in the following manner:

$$\begin{cases} \dfrac{\partial f}{\partial z} = \dfrac{\partial u(x,y)}{\partial x} + i\dfrac{\partial v(x,y)}{\partial x} \\ \dfrac{\partial f}{\partial \overline{z}} = 0 \end{cases} \tag{2.21}$$

For nonanalytic functions, $\frac{\partial f}{\partial \bar{z}} \neq 0$, and thus the optimization can be carried out in both directions, namely z or $\bar{z}$.
The necessary and sufficient condition for the function $f(z)$ to have the stationary point with respect to its real parameters is the equality of the $\bar{\Re}$-Derivative (see [43]) to zero and vice versa. Following Johnson, (see [19]) one should consider two useful theorems:

Theorem 1. If the function $f(z, \bar{z})$ is real-valued and analytic with respect to z

and $\bar{z}$, all stationary points can be found by setting the derivative with respect to either z or $\bar{z}$ to zero.

Theorem 2. Let $f(z, \bar{z})$ be a real-valued function of the vector-valued complex variable z where the dependence on the variable and its conjugate is explicit. By treating z and $\bar{z}$ as independent variables, the quantity pointing in the direction of the maximum rate of change of $f(z, \bar{z})$ is $\nabla_{\bar{z}}(f(z))$.

Proof. To prove this theorem one should consider δf:

$$\delta f = \sum_i \left(\frac{\partial f}{\partial z_i} \delta z_i + \frac{\partial f}{\partial \bar{z}_i} \delta \bar{z}_i \right) = \nabla_z(f)^T \delta z + \nabla_{\bar{z}}(f)^T \delta \bar{z} \tag{2.22}$$

This quantity is concisely expressed as $\delta f = 2Re\left(\nabla_{\bar{z}}(f)\delta z\right)$. Due to the existence of the Schwarz inequality [18] ($\left|\sum_{i=1}^{n} x_i \bar{y}_i\right|^2 \leq \sum_{j=1}^{n} |x_j|^2 \sum_{k=1}^{n} |y_k|^2$.), the maximum value of δf occurs when δz and $\nabla_{\bar{z}}$ have the same direction. Therefore the direction corresponding to the largest change in the quantity of $f(z, \bar{z})$ is the direction of $\bar{z}$. To implement the steepest descent method, one should use the gradient with respect to the conjugated z. To find the minima for the solution of the equation $\nabla_{\bar{z}}(f(z)) = 0$, the **Hessian** matrix of the second order partial derivatives must be positive definite ($\nabla_z(\nabla_{\bar{z}}(f(z))) > 0$). Here the discussion on complex analysis terminates and the discussion on complex-valued system identification starts.

2.2.4 Complex-Valued System Identification

Having carried out Wirtinger calculus, one can easily identify the complex valued system. The cost function for the complex arguments is defined in the following manner:

$$E = \frac{1}{T} \sum_{t=1}^{T} \frac{1}{2} \left(y_t(w, x_t) - y_t^d \right) \overline{\left(y_t(w, x_t) - y_t^d \right)} \rightarrow \min_w \tag{2.23}$$

where y_t is a model output, y_t^d is a desired model output, T is the number of data points and w is model parameter. Such cost function is doing the mapping from C to R (therefore the function is not analytic) and thus the Wirtinger calculus has to be applied to calculate the derivatives. According to Wirtinger we can make the following assumption. Let:

$$\frac{\partial E}{\partial w} = 0 \tag{2.24}$$

Then using the Theorem 2, one can calculate the following derivative:

$$\frac{\partial E}{\partial \overline{w}} = \frac{1}{T}\sum_{t=1}^{T}\left(y_t - y_t^d\right)\frac{\partial \overline{y}}{\partial \overline{w}} =: \overline{g} \tag{2.25}$$

One can now proceed the same way with the Taylor expansion. Keep in mind,that dealing with the Taylor expansion requires the calculation of real numbers calculations. Thus the optimization rule for the Δw which provides conjugates for the derivatives of the error has to be guaranteed. See the following equation:

$$E(w + \Delta w) = E(w) + g^T \Delta w + \tfrac{1}{2}\Delta w^T G \Delta \tag{2.26}$$

Using the previous equation, we can easily define the training rule:

$$\Delta w = -\eta \cdot \bar{g} \tag{2.27}$$

Thus the Taylor expansion can be rewritten as follows:

$$E(w + \Delta w) = E(w) - \eta g^T \bar{g} + \tfrac{\eta^2}{2} g^T G \bar{g} \leqslant E(w) \tag{2.28}$$

This optimization rule can now be used for complex system identification.

2.3 Analytical Functions and Their Derivatives

The selection of the activation function for the artificial neurons is an important issue in artificial neural networks. Activation functions play an important role in neural network theory. The main conditions with real-valued functions are that the function must be continuous and differentiable, an ideal case if it is bounded (in order to be robust against outliers in the input data). Examples of the bounded, differentiable and continuous functions are the *logistic*, *sin*, *cos* or *tanh* functions. Fig.2.1 shows the example of such real valued functions and their derivatives. In the complex-valued case, everything is different. Let's start with the requirements of differentiability. One of the strongest requirements for the complex-valued functions are the so called Cauchy-Riemann equations:

$$\frac{\partial u}{\partial x} = \frac{\partial v}{\partial y} \quad \frac{\partial v}{\partial x} = -\frac{\partial u}{\partial y} \tag{2.29}$$

if these requirements are correct, we are dealing with a function which is analytic meaning that $u(x, y)$ and $v(x, y)$ are harmonic. To summarize the properties of the complex derivative we can write the following:

- $f'(z)$ exists and is continuous

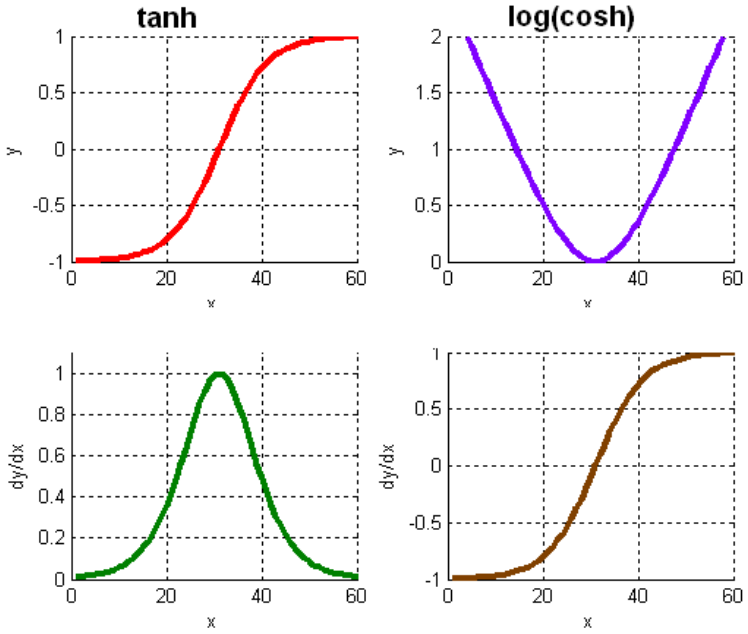

Figure 2.1: Different types of activation functions and their derivatives.

- $f(z)$ is holomorphic
- $f(z)$ should satisfy the Cauchy-Riemann conditions
- $f(z)$ should be representable as a converging power series expansion.

If the function $f(z) = u(x, y) + iv(x, y)$ then $u(x, y)$ and $v(x, y)$ should satisfy the Laplace equation:

$$\frac{\partial^2 u(x,y)}{\partial x^2} + \frac{\partial^2 u(x,y)}{\partial y^2} = 0 \quad and \quad \frac{\partial^2 v(x,y)}{\partial x^2} + \frac{\partial^2 v(x,y)}{\partial y^2} = 0 \tag{2.30}$$

From the Laplace equation one can see that both real valued functions are to be harmonic (sin, cos etc). Some examples of analytic funtions are: z^n, e^z, $ln(z)$, $sin(z)$, $cos(z)$. Some examples of nonanalytic functions are: $\bar{z}, Re(z), Im(z), |z|^2$ etc.

Discussion on the nonlinearity and boundedness of complex functions. After having discussed the requirements of differentiability, we will discuss the requirements for the bounded functions. The requirements for the bounded function are set be the Liouville theorem below.

2. MATHEMATICAL BASIS FOR COMPLEX-VALUED NEURAL NETWORKS

Liouville theorem. *For every holomorphic function f, there exists a positive number M such that $|f(z)| \leq M$ for all z in C is constant.*

Proof. The theorem follows from the fact that holomorphic functions are analytic. Since f is entire, it can be represented by its Taylor series around zero. $f(z) = \sum_{k=0}^{\infty} a_k z^k$. where (by Cauchy's integral formula 2.5): $a_k = \frac{f^{(k)}(0)}{k!} = \frac{1}{2\pi i} \oint_{C_r} \frac{f(\zeta)}{\zeta^{k+1}} d\zeta$ and C_r is the circle around 0 of radius $r > 0$. We can estimate directly:
$|a_k| \leq \frac{1}{2\pi} \oint_{C_r} \frac{|f(\zeta)|}{|\zeta|^{k+1}} |d\zeta| \leq \frac{1}{2\pi} \oint_{C_r} \frac{M}{r^{k+1}} |d\zeta| = \frac{M}{2\pi r^{k+1}} \oint_{C_r} |d\zeta| = \frac{M}{2\pi r^{k+1}} 2\pi r = \frac{M}{r^k}$,
where in the second inequality we have invoked the assumption that $|f(z) \leq M$ for all z and the fact that $|z| = r$ on the circle C_r. But the choice of r in the above is an arbitrary positive number. Therefore, letting r tend to infinity gives $a_k = 0$ for all $k \geq 1$. Thus, $f(z) = a_0$ proving the theorem.

The much stronger requirements for the functions are given by the Piccard's theorem below.

"Little" Picard Theorem. *If a complex function $f(z)$ is entire and non-constant, then the set of values that $f(z)$ assumes is either the whole complex plane or the plane minus a single point.*

Definition. *A real-valued or complex-valued function f defined on some topological space X is reffered to as locally bounded if for any x_0 in X there exists a neighborhood A of x_0 such that $f(A)$ is a bounded set, that is, for some number $M > 0$ one has $|f(x)| \leq M$ for all x in A.*

Obviously, if a function is bounded, it is also locally bounded. This definition can be extended to the case where f takes its values in some metric space. Then the inequality above needs to be replaced with $d\left(f(x), a\right) \leq M$ for all x in A, where d is the distance function in the metric space, and a is some point in the metric space. The choice of a does not affect the definition. Choosing a different a will at most increase the constant M for which this inequality is true.
The remarkable behavior of holomorphic functions near essential singularities is described by the "Great" Picard's Theorem.

Discussion of the singularities of the complex functions.
Other problems of the complex functions are related to the singularities. The example one can see at the figure 2.2 for the *tanh* function. Let us introduce some definitions related to singularities and its types.

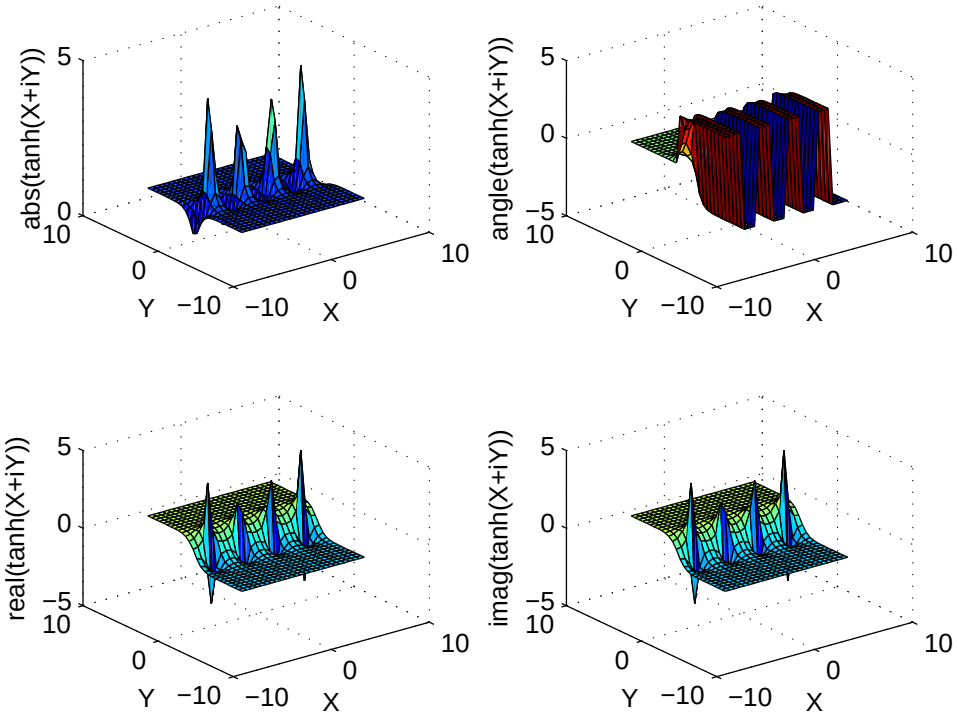

Figure 2.2: The complex-valued case of the hyperbolic tangent activation function.

Definition. *If $f(z)$ has a single point singularity at z_0, the singularity is named as removable if $\lim_{z \to z_0} f(z)$ exists.*

Definition. *If $\lim_{z \to z_0} |f(z)| \to \infty$ and $f(z)$ is analytic in the neighborhood of z_0, then the singularity is non removable and called isolated. Example is tanh function. See fig 2.2.*

Definition. *If a singularity is neither removable nor isolated, then it is known as an (isolated) essential singularity.*

Having defined singular points of the functions can be rather dangerous due to the unlimited values the network output can take and thus destroy the computations. Therefore we should not allow the functions to have singular points at the solution range. Concluding the discussion about possible values of the complex

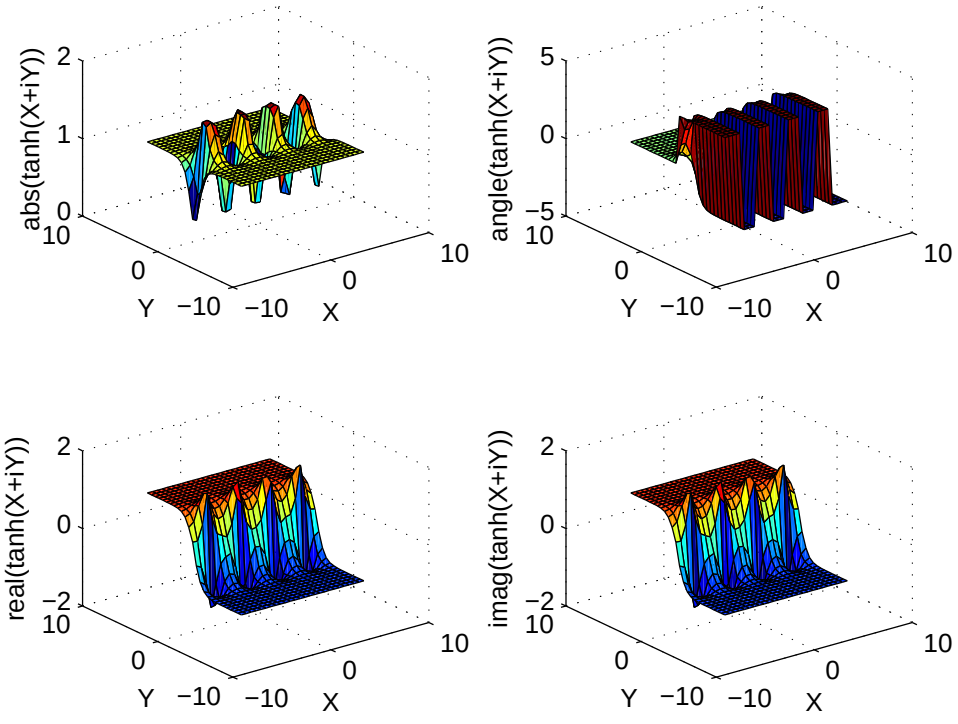

Figure 2.3: An example of the engineered function

function one can say, that any holomorphic function (and in this thesis we are interested in functions which do have derivative) will be unbounded or vise versa,

if the function is bounded then it is not holomorphic. But what to do if we still need the bounded and nonlinear function? The answer is to try to go through the Liouville theorem. At some point one can say, that the Liouville theorem is the deadlock for the neural computations. Hopefully the Liouville theorem is valid only for the entire complex plane. Therefore if one constrains the possible solution range, the Liouville theorem will not apply.

Proposition. *In a bounded domain of the complex plane $\Im$, a fully complex nonlinear activation function $f(z)$ needs to be analytic and bounded.*

Therefore, if one considers the solution to be at some bounded region of the complex plane, we might find the function, which is differentiable and bounded. Let us present an example of such functions. One can see from the figure 2.2 that singularities may occur periodically, making the function unbounded in some region. Since the optimization algorithms used in the current thesis do not allow constraints setting, it cannot be guaranteed that the parameters will not cause the unlimited growth during the optimization.

Discussion of the engineered complex functions.
To overcome the problems discussed above, one can construct the so-called engineered function (will be discussed in depth later). The main idea is that in order to get rid of the "bad" properties of a function, one has to replace the singularity of the function with some other function. Then one should "sew" the functions at the border. For more information see eq. 2.31 and fig. 2.3.

$$\begin{cases} \begin{bmatrix} f_1(z) = \tanh(z)\,,\ z \in C \backslash \left[-\frac{\pi}{2} - \varepsilon; -\frac{\pi}{2} + \varepsilon\right] \cup [-\varepsilon; \varepsilon] \\ f_2(z) = \tanh(\mathrm{Re}(z))\, e^{i\phi(z)},\ z \in \left[-\frac{\pi}{2} - \varepsilon; -\frac{\pi}{2} + \varepsilon\right] \\ f_3(z) = \log\cosh(z)\,,\ z \in [-\varepsilon; \varepsilon] \end{bmatrix} \\ f'_1(z) = f'_2(z)\,;\ z \in \left[-\frac{\pi}{2} - \varepsilon; -\frac{\pi}{2} + \varepsilon\right] \\ f'_1(z) = f'_3(z)\,;\ z \in [-\varepsilon; \varepsilon] \end{cases} \tag{2.31}$$

Other examples of the engineered functions are:

- $f(z) = u(Re(z)) + iv(Im(z))$, where u and v are some real valued functions.
- $f(z) = f(abs(z)) \exp^{i\phi(z)}$ where $abs(z)$ is the absolute part of the complex variable z and the $\phi(z)$ is the phase part of the complex-valued z.

2.4 Discussion on the Error Function

One should pay more attention to the error function [27]. As already discussed, we have to deal with nonanalytic error function with core of $z \times \bar{z}$ due to optimization reasons. Other combinations are also possible, for example $log(z) \times \overline{log(z)}$. One can show, that this is not an "easy" error function, and that it has very unique properties. Let us describe these properties in more detail. $E = z \times \bar{z}$, where Y - network output (observation), $\hat{Y}$ - network target (expectation). Now $z = Y - \hat{Y}$. Thus $E(z, \bar{z}) = (Y - \hat{Y}) \times (Y - \hat{Y})$. Let us now switch to the Euler notation to make the calculations easier. $E = r^2 + \hat{r}^2 - r\hat{r}\left(e^{i(\phi-\hat{\phi})} + e^{i(\hat{\phi}-\phi)}\right)$. This is the quadratic equation. The discriminator is $D = \sqrt{b^2 - 4ac}$, where $b = e^{i(\phi-\hat{\phi})} + e^{i(\hat{\phi}-\phi)}$ and $a = c = 1$. In order to make sure, that our equation has only one root (roots $x_{1,2} = \frac{-b \pm \sqrt{D}}{2a}$)(one minima) one should make discriminator $D = 0$. This happens only if $\phi = \hat{\phi}$. In case, the following equation to minimize $(r - \hat{r})^2$ which obviously only occurs if $r = \hat{r}$ must be used. But the necessary condition is the equality of the phases of the expectation and the observation. Thus upon optimizing the radius of the complex number, the phase will be optimized.

$$\begin{cases} (r - \widehat{r})^2 \to 0 \\ \phi = \widehat{\phi} \end{cases} \tag{2.32}$$

This property is very important for reasons which will become apparent later. To have a better impression about the ideas discussed above, consider an example below.
One can consider the following problem from a different perspective. This error function has very unique and desirable properties. Let us describe these properties more in detail. We rewrite 2.23 into Euler notation:

$$E = r^2 + \hat{r}^2 - r\hat{r}\underbrace{\left(e^{i(\phi-\hat{\phi})} + e^{i(\hat{\phi}-\phi)}\right)}_{2\cos(\phi)} \tag{2.33}$$

The discriminant of 2.33 is negative and only can be equal to zero that the equation has 1 root: $D = \sqrt{4\hat{r}^2\cos^2(\Delta\phi) - 4\hat{r}^2} = 2\hat{r}\sqrt{-\sin^2(\Delta\phi)} = 0$.Therefore if $\Delta\phi = 0$ then $r = \hat{r}$.We can also rewrite (4) in the following way: $E(y, \hat{y}) = a^2 + b^2 + \hat{a}^2 + \hat{b}^2 - 2a\hat{a} - 2b\hat{b} = (a - \hat{a})^2 + \left(b - \hat{b}\right)^2$. One can see that such error function minimizes both real and imaginary parts of the complex number. This is a very important result which can be explained in the follwoing way. The phase has to be synchronised for all neurons during the training. In case there is no phase equality, there is no training, since the error function does not have the minima at zero. It means that the error for radius can converge, but the phase

inequality means there is no global minima for the error function. The global minima occurs only with phase equality. This explains the success in the real world applications, where the effect of phase synchronization plays an important role.

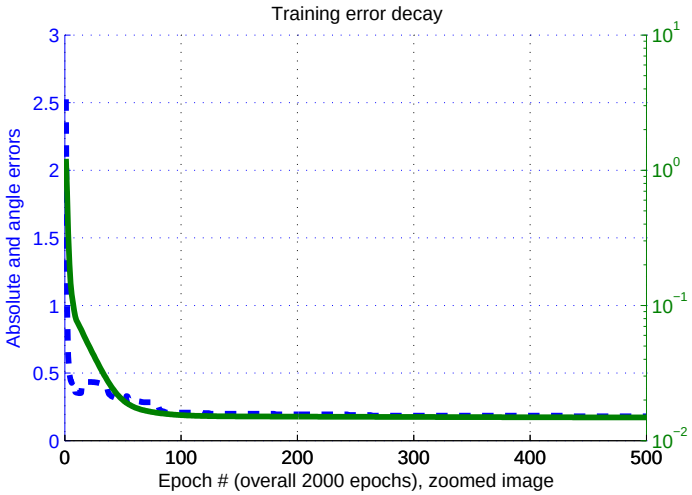

Figure 2.4: Error function behavior of the absolute error (solid line) and angle error (dotted line). Both errors decay to zero.

In Fig.2.4 one can see that absolute error, which consists of $z \times \hat{z}$ and the angle error, $E_{angle} = \left|\arctan\left(\mathrm{Re}\,(y)/\mathrm{Im}\,(y)\right) - \arctan\left(\mathrm{Re}\,(\widehat{y})/\mathrm{Im}\,(\widehat{y})\right)\right|$.

2.5 Complex-Valued Linear Regression

The real-valued case of the linear regression was widely discussed in the benchmarking section 4, where the definitions for the linear regression and basic notations which will be used in the discussion of the complex-valued linear regression have been introduced.
Complex-Valued Linear Regression ($CVLR$) is the simplest case of data approximation. Consider a linear model with n inputs. Let X be the data which is to be approximated and let w and b be linear regression parameters. Y is the linear regression output. Then the model can be presented as follows:

$$Y = \sum_{j=1}^{T} w_j X_j + b_j \tag{2.34}$$

The error which we can introduce, analogous to the mean squared error is the following expression:

$$E = \left(Y - Y^d\right)\overline{\left(Y - Y^d\right)} \tag{2.35}$$

To estimate the parameters of the linear regression one should calculate the derivatives of the error with respect to the parameters of the linear regression: Training rule from the gradient descent can be derived as follows (following Theorem 2 the optimization is to be conducted in the direction of $\bar{w}$):

$$w_{new} = w_{old} - \eta \frac{dE}{\overline{dw}}$$

The task now is to define the $\frac{dE}{\overline{dw}}$. This term is to be calculated using the Wirtinger calculus discussed above.

$$\frac{dE}{d\overline{w}} = \frac{dE}{dY}\frac{dY}{d\overline{w}} + \frac{dE}{d\overline{Y}}\frac{d\overline{Y}}{d\overline{w}}$$

Following Wirtinger

$$\frac{dY}{d\overline{w}} = 0 \text{ and } \frac{dE}{d\overline{Y}} = Y,\ \frac{d\overline{Y}}{d\overline{w}} = \overline{X}$$

then the training rule for the linear regression can be written as follows:

$$w_{new} = w_{old} - \eta Y \overline{X} \tag{2.36}$$

Using this simple rule one can train the linear regression with relatively small η.

3

Neural Networks

3.1 Real-Valued Feedforward Neural Network

3.1.1 General Introduction to Neural Networks

Neural Networks are a well-known approach for the data approximation and data classification. There are variety of architectures and training algorithms discussed in the literature. Unfortunately there is still no unique architecture which will prove to be suitable for all problems. Artificial Neural Networks are a bio inspired concepts, which is related to neuroscience. In order to make this algorithm understandable the real picture of the brain has been significantly simplified, keeping the main data processing elements, like axons and dendrites as well as neuron cell itself. One well-known extension of the traditional neural network is so-called spiking neural network which has more analogous to a real world neuron, nevertheless even this model is far from being realistic. Even the simple model of the neuron, which will be discussed further, can demonstrate an interesting behavior. The cell, which we are going to model, is the nucleus, it has dendrites which bring the information to the cell, nucleus, which weights up the incoming information with locally stored weights, compares this sum against its bias and amplifies this information sending it to other neurons with axon connection. Networks modeled with such approach can be used to map complex relationships between inputs and outputs or to find patterns in data. Artificial neuron visualization can find in the figure 3.1 below.

$$y_t = f\left(\sum_{i=1}^{n}(X_{i,t}w_i) + w_0\right),\ t = 1...number\,of\,patterns \tag{3.1}$$

Where X is a matrix of inputs, w_i are synaptic weights (nodes), f is an activation function and y_t is an output. Using such nodes one can construct different

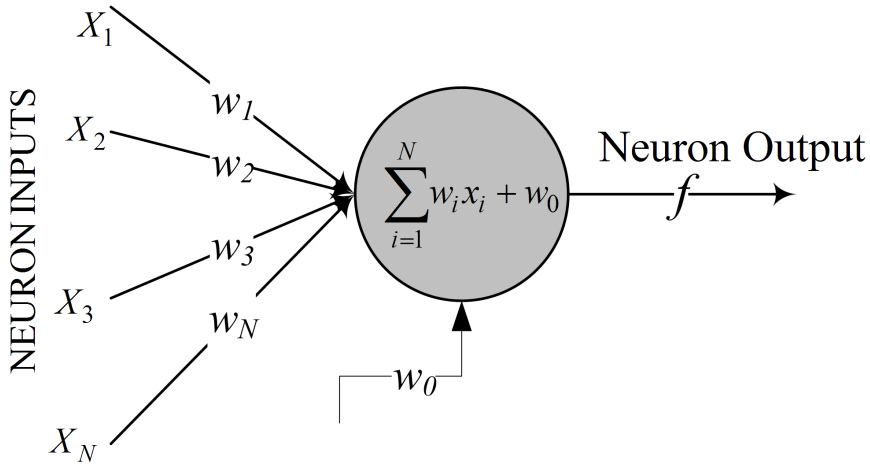

Figure 3.1: Graphical representation of a Mc.Cullough-Pitts artificial neuron (first described in 1942). Here x are inputs, w are weights, w_0 is bias and f is activation function. It sums its inputs with weights substracts the bias and applies the activation function to the product producing the activated output.

architectures to solve different problems. In general, it is possible to divide all architectures into two classes: Feedforward NNs propagates information from input to the output and Recurrent NNs which propagates information in a closed loop.

In many industrial areas such as medicine, finance and various control applications, one has to predict the next state of the system in order to prevent breakdowns, production losses, money losses, etc. Many of these problems are very difficult to model or even impossible to model a priori. The only possible solution is to measure data with the sensors and try to adapt a process model based on these measurements. Typically, original data have data gaps, outliers, noise, etc. In order to make better forecasts, one should preprocess the data. For example, one can use a filter to accomplish this (e.g. an adaptive filter). The problem arising is that the value to be forecasted with the use of this filtered data will also be filtered in some sense. Therefore the forecasted value cannot be used in terms of the initial problem. Moreover, in many cases the "operator" (the person who operates some complicated machines) does not care about exact value of the forecast, but only about whether the forecasted value belonged to the "good" condition or to the "bad" condition of the system. In order to overcome this difficulty, one should use one side classification to classify whether predicted value belongs to the "good" or to the "bad" condition of the system under investigation. A lot of papers in this area have been already published: [8],[46],[51],[6],[13]. Before starting with NN the simplest case - linear regression has to be considered. One should always follow the Occam's razor rule - which is to use the simplest

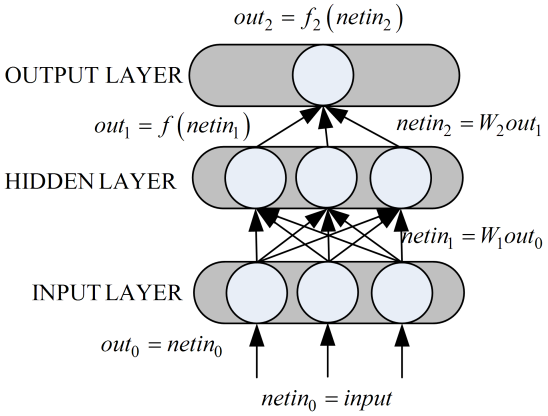

Figure 3.2: A feedforward multi-layer perceptron (MLP) neural network. Circles represent neurons. Arrows show information flow from inputs to output.

model out of the whole set of models. The simplest model one can find, is linear regression. In statistics, linear regression is a regression method that models the relationship between a dependent variable Y independent variables $X_{i=1..p}$ and a random term ε. The model can be written as:

$$Y = \beta_0 + \beta_1 X_1 + ... + \beta_P X_P + \varepsilon \tag{3.2}$$

where β_0 is the intercept ("constant" parameter - bias), the β_i are the respective parameters of independent variables, and p is the number of parameters to be estimated in the linear regression. Linear regression can be contrasted with non-linear regression. Methods for parameters estimation for the linear regression are well-known and we will not discuss them.

3.1.2 Real-Valued Feedforward Path

Feedforward Neural Network (further FFNN) consist of neurons structured in layers (see [38],[55]). The neurons in each layer are not connected to one another. However the neurons between the layers are fully connected (neurons might not be fully connected, if sparse matrices are used). Each neuron from the i^{th} layer is connected to all neurons in the neighboring neurons in the $(i-1)^{th}$ layer and $(i+1)^{th}$ layer. The information flow in such networks is straightforward; therefore it is called feedforward. It goes from the input layer to the output layer through the hidden layers. A neural network architecture is presented at the fig. 3.2. The architecture in the picture is the so-called multi-layer perceptron (MLP). Each neuron in this architecture is working according to the 3.1. A fully connected multi-layer perceptron network (MLP, see Fig. 3.2) with I inputs, a first hidden

layer with H nodes, a second hidden layer with L nodes and in this case a single output is defined by:

$$y(\mathbf{x}) = w_b + \underbrace{\sum_{l=1}^{L} w_l \tanh\left(w_{b,l} + \underbrace{\sum_{h=1}^{H} w_{lh} \underbrace{\tanh(w_{b,h} + \sum_{i=1}^{I} w_{hi} x_i)}_{\theta_2}}_{y_{MLP3}}\right)}_{\theta_1}. \tag{3.3}$$

This architecture (or architectures similar to this one) are well-known universal approximators. Moreover one can extend the network by embedding the expert knowledge into the network weights. In the current subsection "black box" whitening will be showed. This will not be done for the complex-valued case, since the procedure is similar. In order to do the knowledge incorporation one should apply special rules to NN training, namely to the derivatives of the NN. The values of the weights do not tell anything about the process which they model. Therefore for a long time NNs have been considered as a "black box" modeling approach. One of the ways to overcome this difficulty is to induce the monotonic behavior for the input-output (I/O) relations. Consider the example of the monotonic dependency. Imagine one has to model the Ideal Gas law ($PV = RT$), here P is pressure, V is volume, T is temperature and R is the ideal gas constant. In case P increases in a fixed volume ($V = const$), temperature T has to increase as well. Imagine P and V are available and T behavior has to be modeled. In this case one can induce the training constraints in order to make the network obey the monotonicity rules. The MLP can insure a monotonically increasing (decreasing) behavior with respect to the inputs $x_j \in \mathbf{x}$, if set the derivative of the output with respect to the input as positive or negative (depending on the relationships). Finally, the constraints will be translated to the weights (see eq.3.4).

$$\frac{\partial y}{\partial x_j} = \sum_{l=1}^{L} w_l \cdot \underbrace{(1 - \theta_1^2)}_{>0} \cdot \sum_{h=1}^{H} w_{lh} \cdot \underbrace{(1 - \theta_2^2)}_{>0} \cdot w_{hj} \geq 0 \tag{3.4}$$

An MLP is known as a universal approximator [12]. However, the constraints defined above reduce the degrees of freedom for the weights so that the capability to approximate arbitrary, partially monotonic functions has to be shown again. Detailed proof was presented in [12]. The four-layer feedforward network (see eq. 3.3) is an extension of a three-layer standard MLP. Since the three-layer topology is already sufficient for an universal approximator, the extension by a monotonic second hidden layer to a four-layer network respectively additional

calculations by hyperbolic tangents and the multiplications with positive weights w_l do not affect this property. Limitations in the sign of the weights w_{lh} can be eliminated by appropriate weights w_{hi} since $\tanh(x) = -\tanh(-x)$. To sum up, a four-layer feed-forward network under the weights constraints continues to be a universal approximator.

3.1.3 Real-Valued Backward Path

After the feedforward path is performed, the network output can be obtained. After the network output is obtained, the error between the network output and the desired output of the network can be calculated. The target is the desired (observable) value to which the network should map its inputs. Ideally, network maps its inputs to the desired targets (observables should meet the expectations, meaning that expectation versus observation has to be the same). In practice, the network output is never perfect to its target values; there is always an error in the approximation of the target values. In order to reduce and minimize the error training for an acceptable set of weights is necessary. To do the training one should calculate the backward path which allows calculating the impact of each neuron in the overall error. Using the ladder algorithm described below, the derivatives of the error with respect to the weights ($\frac{dE}{dw}$) can be obtained. To calculate the backward path one should consider the error as an input for the network and use it from the back to the beginning of the network. Thus the impact of each neuron in the overall error can be computed.

3.1.4 Real-Valued Neural Network Training

Once a network has been structured for a particular application, it is ready to be trained. To start this process the initial weights are chosen randomly (uniformly distributed). Then the training, or learning, begins. There are two approaches to training - supervised and unsupervised. Supervised training involves a mechanism of providing the network with the desired output either by manual "grading" the network's performance or by providing the desired outputs with the inputs. In supervised training, both the inputs and the outputs are provided. Then the network processes the inputs and compares its resulting outputs against the desired ones. Later the errors are propagated back through the system, causing the system to adjust the weights which control the network. This process occurs over and over as the weights are tweaked continuously. The set of data which enables the training, is called the "training set". During the training of a network, the same set of data is processed many times as the connection weights are continuously refined. A single presentation of the training patterns in to a neural network is called an epoch of training. Unsupervised training is when the

network has to make sense of the inputs without outside help. In unsupervised training, the network is provided with inputs but not with desired outputs. Then the system must decide itself what features it will use to group the input data. This is often referred to a self-organization or an adaption. In the present work authors will focus on supervised learning. In case of supervised training, the main idea of training is to minimize the error between target output and real output $E(\mathbf{w}) = E\{\mathbf{x}^\alpha,\ \mathbf{y}^\alpha,\ \mathbf{y}(\mathbf{x}^\alpha, \mathbf{w})\}$, where $\{\mathbf{x}^\alpha, \mathbf{y}^\alpha\}$ is a set of patterns,$\{\mathbf{y}(\mathbf{x}^\alpha, \mathbf{w})\}$ is an actual output of the network. Following [4], the most general case of neural network optimization is iteration procedure of weight selection (so called "learning" with tutor $\{\mathbf{x}^\alpha, \mathbf{y}^\alpha\}$). When the functional for the error is selected, then the problem is to minimize this functional. It is possible to use the next way of weight selection:

$$w_{ij}^{\tau+1} = w_{ij}^{\tau} - \eta^{\tau} \frac{\partial E}{\partial w_{ij}} \tag{3.5}$$

where η^τ is the rate of learning for step τ. It is possible to show that reducing the step, according to the simple law $\eta^\tau \propto 1/\tau$, the procedure described above will lead to finding the local minimum. Historically, the problem was in effective computing the $\frac{\partial E}{\partial w_{ij}}$. The error of the network can be computed at the output, so we have the link only to the output layer. The question arises here how to compute the weights changes in hidden layers? There was a problem how to propagate the error through the network in backward direction. Such algorithm was found and it is called now as backpropagation algorithm (see [58]). The idea of this algorithm is based on the differentiation rule for the composition of functions. Following the work [4], $x_j^{[n]}$ denotes the inputs of n^{th} layer of nodes. Neurons of this layer compute the following linear combinations:

$$a_i^{[n]} = \sum\nolimits_j w_{ij}^{[n]} x_j^{[n]} \tag{3.6}$$

Then one should propagate this to the next layer through the non linear activation function. Nonlinearity of the activation function is very important since superposition of linear function is still a linear function:

$$x_i^{[n+1]} = f\left(a_i^{[n]}\right) \tag{3.7}$$

To create the training algorithm, one has to know the derivative of the error with respect to the nodes weights:

$$\frac{\partial E}{\partial w_{ij}^{[n]}} = \frac{\partial E}{\partial a_i^{[n]}} \frac{\partial a_i^{[n]}}{\partial w_{ij}^{[n]}} \equiv \delta_i^{[n]} x_j^{[n]} \tag{3.8}$$

Therefore the impact of each node to the overall error can be computed locally, by multiplying the discrepancy $\delta_i^{[n]}$ with the value of concrete input. The inputs

of each layer are computed sequentially while the feed forward propagation:

$$x_i^{[n+1]} = f\left(\sum\nolimits_j w_{ij}^{[n]} x_j^{[n]}\right) \tag{3.9}$$

The discrepancy is calculated while backward propagation of the error:

$$\delta_i^{[n]} = f'\left(a_i^{[n]}\right) \sum\nolimits_k w_{ki}^{[n+1]} \delta_k^{[n+1]} \tag{3.10}$$

Using the chain rule for the derivative it is possible to obtain the next equation:

$$\frac{\partial E}{\partial a_i^{[n]}} = \sum\nolimits_k \frac{\partial E}{\partial a_k^{[n+1]}} \frac{\partial a_k^{[n+1]}}{\partial x_i^{[n+1]}} \frac{\partial x_i^{[n+1]}}{\partial a_i^{[n]}} \tag{3.11}$$

hence the larger the activation of the concrete node in the following layer, the larger the resulting error. Using this algorithm it is possible to train any feedforward architecture in a supervised mode. Let W be the overall number of nodes. Then the complexity of the algorithm is $O(W)$, where W is overall number of nodes. The straightforward algorithm for the computation of the derivative $\frac{\partial E}{\partial w_{ij}^{[n]}} = \frac{E\left(w_{ij}^{[n]}+\varepsilon\right)-E\left(w_{ij}^{[n]}\right)}{\varepsilon}$ the complexity here is $O(W^2)$.

3.1.5 Real-Valued Ladder Algorithm

The idea of the ladder algorithm, proposed by Zimmermann [55] is that there is a very good connection between the architectural representation of the NN and the algorithmic part of the forward and backward paths. Using this algorithm one can construct nearly any architecture and implement elegantly the back propagation for a particular architecture. Moreover, the ladder algorithm shows that backpropagation is a local algorithm (see Fig.3.3). At the figure one can see the main idea of it. The notations are the following: $netin_i$ is i^{th} layer input (vector containing the inputs of the neurons in the layer), $netout_i$ is i^{th} layer output, dev_i is the error which is produced by the i^{th} layer, δ_i is the derivative of the error produced by the i^{th} layer. At the end of the algorithm the task is to calculate the derivative of the error with respect to the network weights. One can see that using the ladder algorithm one can easily compute these things multiplying the $i-1$ layer output and the derivative of the i^{th} layer. This is the best feature of the algorithm. The back propagation calculations are local and do not depend on the network architecture. The overall procedure can be described in the following way. The input $netin_0$ goes through the input layer and becomes the output function out_0. This output is then transported to the first hidden layer and for this purpose it should be multiplied by the matrix of weights W_1. After multiplication, the new input $netin_1 = W_1 \times out_0$ comes to the first hidden layer one where it

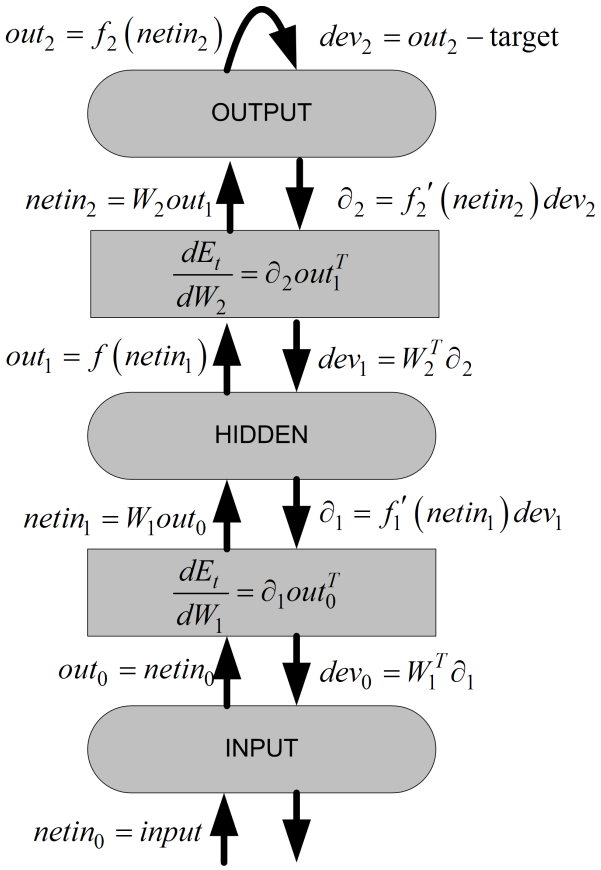

Figure 3.3: Ladder algorithm for a three-layer perceptron (see [55]).

is transformed with the nonlinearity f. After the transformation the first hidden output produces its output out_1 and so on. Then the last hidden output should be compared with the desired output. Here we calculate the error and start the backward path. The derivative of the MSE error is the difference between the last hidden output and the desired output, we mark it as dev_2. Doing the backward path according to the discussions in subsection 3.1.4 one should calculate δ. For the last hidden one it will be the derivative of the nonlinearity multiplied with the derivative of the output dev_2. Thus δ_2 can be obtained. Now one has to explain why the algorith is associated with the word - ladder. One can obtain the derivatives of the error with respect to the hidden weights absolutely for "free", having only those variables which were calculated before. Taking and multiplying δ_2 with the out_1 the derivatives can be obtained for "free". For "free" means that one will calculate these members to proceed with the error backpropagation anyway. Then by taking some members of the calculations discussed above, the derivatives can be calculated automatically. This procedure is to be done several times and it does not depend on the architecture of the neural network (see. fig. 3.3).

3.2 Complex-Valued Feedforward Neural Network

3.2.1 General Discussions

Complex-Valued Neural Network (CVNN) (see Haykin [30] and Kreutz [43]) is one of the recent new topics in the machine learning. One of the advantages of the complex-valued neural networks is their intrinsic capability to deal with the complex-valued input-output relations instead of just real ones. This property allows to broaden the range of applications for the neural networks overall. Unfortunately, the price for the complexity of the complex-valued neural networks (CVNN), is sometimes too high. One should clearly understand wether there is a need to go into more complicated complex-valued networks or can be satisfied with the real-valued networks (RVNN) (see [1, 40, 58]). This section briefly goes through the feedforward case of the real-valued neural network, then it discusses the differences between complex-valued and real-valued neural networks and at the end it considers the set of real world examples in order to answer the question: which networks are more appropriate for the considered examples. Moreover, this section tris to break through the problems of the complex version of the backpropagation algorithm using the Random Search Algorithm (RSA) and then it compares the complex-valued backpropagation (CVBP) with the real-valued backpropagation and the RSA for the definite examples. The out-

come of the current research is the combined RSA-CVBP algorithm for training the complex-valued neural networks which seems to be very promising for solving the considered examples.

3.2.2 Complex-Valued Feedforward and Backward Path

The current chapter aims to show the bridge between the real-valued and complex-valued neural networks. RVNN consists of neurons, which can be described with the following equation:

$$y_i = \tanh\left(\sum_{j=1}^{J} w_{ij}X_j + w_j\right), [X_j, w_{ij}, w_j] \in \mathbb{R}, \tanh : \mathbb{R} \to \mathbb{R} \tag{3.12}$$

Where Y_i is the output of the i^{th} neuron, X_j - is the j^{th} element of the input vector, T is the number of elements in the input vector, w_{ij} is the connection weights matrix, w_j is the vector of bias parameters and function $tanh$ is the activation function. All the weights, bias parameters and inputs are real numbers. The activation function maps its arguments into the real numbers as well. The limitation on the activation function is that it should be differentiable at every point and should be bounded (boundedness is not mandatory, but preferable). One can construct different architectures using the neurons mentioned above. Structuring neurons in layers and connecting the layers in a way that each neuron in the layer does not receive the information from the layer neighboring neurons within the layer, but receives the information from all previous layers neurons, one can end up with the feedforward architecture, displayed in Fig. 3.4. In the Complex-Valued Neural Network the description remains the same. Now the inputs are complex numbers and the weights are complex numbers as well as the bias parameters and outputs. Linear algebra operations in the eq.3.12 are well defined in complex-valued case. The only difference between the RVNN and CVNN can be found by inspecting eq.3.13.

$$y_i = \tanh\left(\sum_{j=1}^{J} w_{ij}X_j + w_j\right), [X_j, w_{ij}, w_j] \in \mathbb{C}, \tanh : \mathbb{C} \to \mathbb{C} \tag{3.13}$$

The first problems, arise in the complex representation of the activation functions. Using the Liouville theorem, one can show that if a function is bounded and differentiable at the complete complex space, it is also constant. Thus any differentiable and bounded funciton is a constant. For example, the $tanh$ function has periodically occuring singularity points. At these points the function goes to infinity resulting an invalid computations. In addition, there is the problem of the error function, but the solution has been described in the section related to

Wirtinger calculus. Many papers discuss the tricks how to overcome the nondifferentiability of the complex functions. One of the most common way to do that is to apply engineering functions as presented in the eq. 3.16 below.

Conventional signs:

l – layer number
$netin^l$, out^l – input and output vectors accordingly for the current layer l
$W^l[nh, nh]$ – matrix of weights, $B^l[nh, 1]$ – bias vector at the l-th layer
f – activation function at a current layer
s_l – state of the network at a layer l.
Error function:

$$E = \sum_{k=1}^{T} (y_k - y_k^d)\overline{(y_k - y_k^d)} \to \min_{W,B},$$

where y – network output, y^d – desired output. The structure of the network is described in the picture 3.4. Let us specify the architecture of the neural network

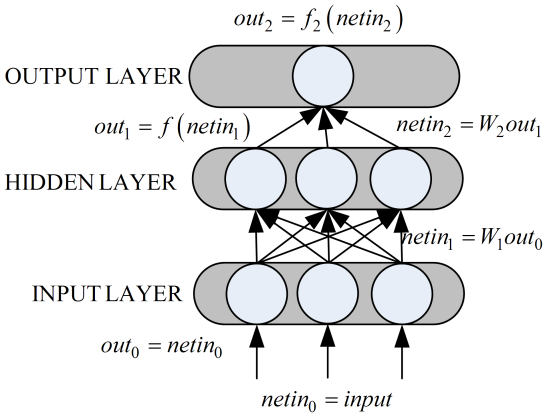

Figure 3.4: An example of the three-layer network.

using few arrays. One array *Architecture* contains the needed information about the architecture of the neural network, another array $Nonlinearities$ contains an information about the transition function at each layer.

$$\begin{aligned} Architecture = [&\text{number of inputs} \\ &\text{number of neurons at hidden layers} \\ &\text{number of outputs}] \\ Nonlinearities = [&\text{activation functions at each layer}] \end{aligned} \tag{3.14}$$

with the activation functions:

$$\begin{aligned} 1: &\ f(z) = tanh(z_r) + i \cdot tanh(z_{im}) \\ 2: &\ f(z) = tanh(z) \\ 3: &\ f(z) = sin(z) \\ 4: &\ f(z) = tanh(r)e^{i\varphi} \end{aligned} \tag{3.15}$$

Denote $netin^l$, out^l are input and output vectors accordingly for the current layer l, dev^l, d^l are input and output accordingly for the backpropagation algorithm. W^l is matrix of weights and b^l is bias vector at the l-th layer, f is activation function at a current layer. One should take into account that $tanh$ is preferable. This is because $tanh$ has periodical singularities which are known and can be avoided by a special control in the training loop. In the rest part the function behaves in a way we need and there is no need to change it. Moreover, it does not have any singularities close to zero and while training the weights are normally in the region between zero and one for real and imaginary parts.

The feedforward path can be described using the ladder algorithm and the following notations:

$$\begin{aligned} l &- \text{layer number} \\ s_l: \ netin^{s_l} &= W out^{s_{l-1}} + b^l \\ out^{s_l} &= f(netin^{s_l}) \end{aligned} \tag{3.16}$$

The backward path can be described with the ladder algorithm and following notations:

$$\begin{aligned} s_l: \ dev^{s_l} &= \overline{W} d^{s_{l+1}} \\ d^{s_l} &= \overline{f'(netin^{s_l})} dev^{s_l} \\ \frac{\partial E}{\partial w} &= d^{s_l} \overline{out^{s_{l-1}}} \\ \frac{\partial E}{\partial b} &= d^{s_l} \end{aligned} \tag{3.17}$$

The feedforward and backward paths can be efficiently computed using the ladder algorithm described above. The very good point is that the ladder algorithm does not change with the complexity of the inputs and outputs. Moreover it does not depend on the architecture; therefore it can also be translated to the recurrent neural networks which will be discussed later. Moreover, one can show that ladder algorithm remains the same for complex numbers (in comparison to the real-valued ladder algorithm), implying a nice correspondence between the locality of algorithms, equations and architectures.

3.2.3 Complex-Valued Ladder Algorithm

Consider the Complex Valued Neural Network (CVNN) which deals with complex inputs, weights, bias parameters and outputs. The ladder algorithm (proposed

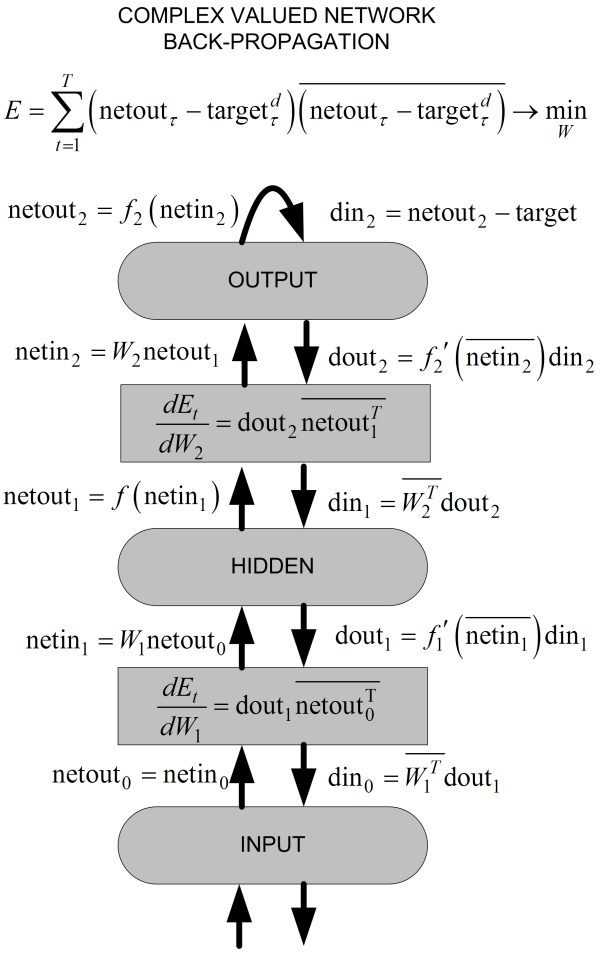

Figure 3.5: Complex Valued Backpropagation. The figure depicts the locality of the CVBP algorithm and independence of the BP from the network architecture. Notations: *netin* - layer inputs, *netout* - layer outputs, *din* -layer derivative input, *dout* -layer derivative output, W_i are network weights, arrows show the information flow, $(\bar{\cdot})$ - means complex conjunction.

by Zimmermann in [55]) notations written as follows:

- $netin^l$, out^l – input and output vectors according to the current layer l
- dev^l, d^l – input and output according to the backpropagaton algorithm
- W^l– matrix of weights and b^l – bias vector at the l-th layer

- f – activation function at a current layer

One can see that the ladder algorithm for the complex-valued network is absolutely the same as it was for the real-valued case despite some changes which came from the Wirtinger calculus, namely conjunctions. As it was noted in the section related to the real value networks the ladder algorithm provides a unique correspondence between the architecture, equations and locality of the algorithms. This can be easily seen at the figure (3.5).

3.3 Real-Valued Recurrent Neural Network

Recurrent Neural Networks (see [15, 17, 33, 39, 58]) have been an important topic in machine learning community for the past years. One of the natural extensions of recurrent neural networks is a complex-valued recurrent neural network, since modeling of complex-valued dynamics is needed in many application areas. First real-valued case of the general recurrent neural network is to be discussed. Then complex-valued case will be discussed. After the complex-valued case the most interesting case, namely Historical Consistent Neural Network and its real-valued and complex-valued cases respectively will be discussed.
However, first the systems to be modelled will be described. The system is described by its internal state s, its input signals u and its outputs y. The main property of this system is that it has the recurrent connection of the output to its input. This connection allows recurrent processing of information and results in a dynamical system. Therefore in case one needs to model dynamical system, recurrent architecture is to be used. The dynamical system can be described by the eq. 3.18 below:

$$\begin{cases} s_t = f\left(s_{t-1}, u_t\right) \\ y_t = g\left(s_t\right) \end{cases} \tag{3.18}$$

where u is a model input, s is an internal state of the model and y is the model output, f and g are some transition functions (recurrent connection is modelled by the states). One of the most appropriate architectures is so-called Elman architecture [39]. The Elman architecture considered in the current section has the following structure. The recurrent architecture is mapped into the feedforward network with three layers: an input layer, a hidden layer and an output layer (see Fig.3.6 below). This type is different from traditional ones is that the input layer has a recurrent connection with the hidden (denoted by the dotted lines and marked as "weight U" in the Fig.3.6). Therefore, at each time step the output values of the hidden units are copied to the input ones, which store them and use them for the next time step. This process allows the network to store some information from the past in the way to detect temporal properties of the

patterns better. In the original experiments presented by Jeff Elman, so-called

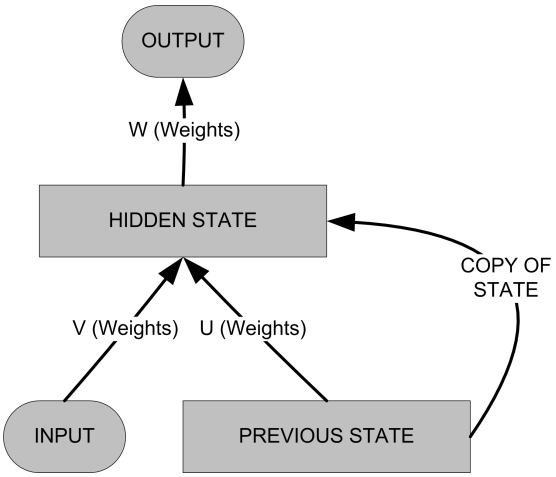

Figure 3.6: An Elman recurrent neural network. Connections between layers are represented by lines."Copy(delayed)" stands to show that network makes the copy of the hidden layer and presents it with some weights on the next iteration of the training (see [39]).

truncated backpropagation was used. This basically means that $y_j(t-1)$ was simply regarded as an additional input. Any error at the state layer (marked as "Previous state" in the Fig.3.6), $\delta_j(t)$ was used to modify weights from this additional input slot (see Fig.3.6). Errors can be back propagated even further. This is so-called backpropagation through time (BPTT; see Fig.3.7 and Rumelhart [21]) and is a simple extension of the traditional one, which can be easily done with the ladder algorithm. The only difference is that now one should use so-called shared weights paradigm (see Zimmermann [33]). The main idea is that the weights between the layers remain the same, thus they take part in all equations which arise during the forward or backward paths (The basic principle of BPTT is that of "unfolding" due to the feedforward architecture. All recurrent weights can be duplicated spatially for an arbitrary number of time steps τ. Each node which sends activation along a recurrent connection has τ number of copies as well (see Fig.3.7). Errors are back propagated according to the next simple rule $\delta_{pj}(t-1) = \sum_{h}^{m} \delta_{ph}(t) u_{hj} f'(y_{pj}(t-1))$ where h is the index of the recieving neuron and j is sender index. Such processing allows to calculate the error at time moment t for output neurons. Below are definitions for permanent notations:
l – layer number,
$netin^l$, out^l – input and output vectors accordingly for the current layer l
f – activation function at a current layer
u_l – state of the network at an input layer

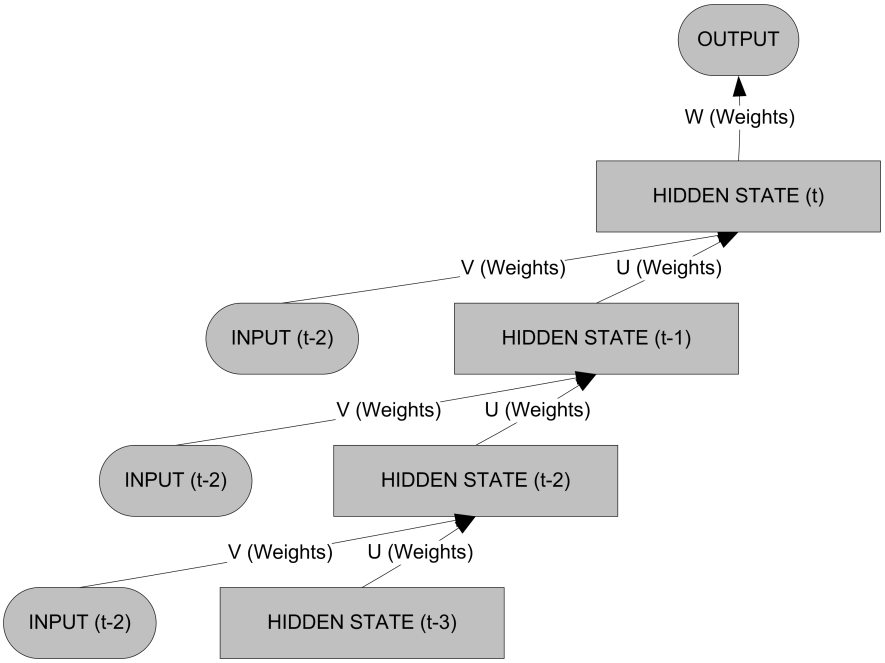

Figure 3.7: Backpropagation through time. Connections between layers are represented by lines.

s_l – state of the network at a hidden layer
y_l – state of the network at the output layer
The weights matrices are of the following dimensionality: $A\,[nh \times nh]$, $B\,[nh \times ni]$ and $C\,[no \times nh]$.

$$s_t = f(As_{t-1} + Bu_t);$$
$$y_y = Cs_t;$$
$$E = \sum_{t=1}^{T} (y_t - y_t^d)^2 \rightarrow \min_{A,B,C},$$

where y – output vector, y^d – target vector. The structure of the network is described in the Fig. 3.8. The core point of the complete discussion is so-called "Shared Weights" concept proposed by Zimmermann [33], states that all three weight matrices: A, B and C remain constant between layers, thus causing the recurrent procedure. The architecture itself is just unfolding in time, while the matrices remain constant among the layers. Matrix A compresses the data from the dimensionality of the input layer to the dimensionality of the state layer, matrix C makes the decompression from the dimensionality of the state layer to the dimensionality of the output layer. Matrix B makes the transition from one state to another, thus causing the complete story to be recurrent. In order to

make the architecture definition faster, let me introduce the *architecture* array. This array describes the architecture in a structured way.

$$\begin{aligned} NetworkArcitecture = [&1 - \text{number of states} \\ &2 - \text{number of hidden neurons in the state} \\ &3 - \text{number of forecasting states} \\ &4 - \text{number of inputs} \\ &5 - \text{number of outputs} \\ &6 - \text{activation functions}] \end{aligned}$$

where activation functions are presented as numbers:

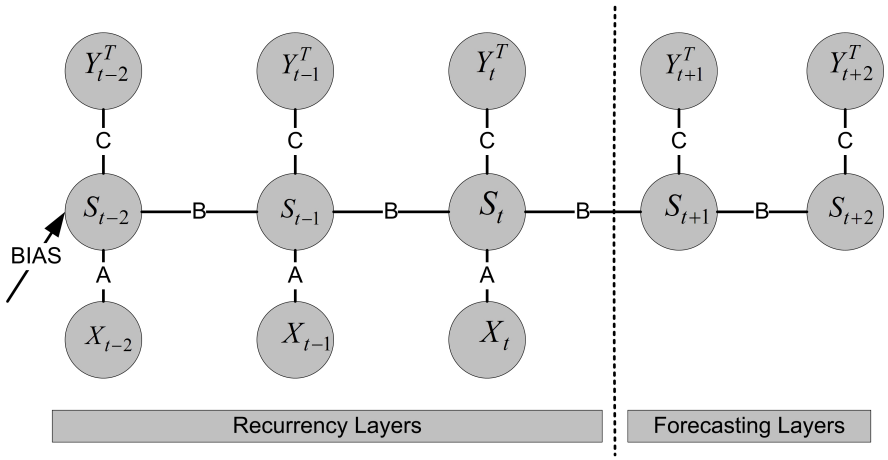

Figure 3.8: Open recurrent system. For more information see Zimmermann [33].

$$\begin{aligned} 1: &\ f(z) = tanh(z_r) + i \cdot tanh(z_{im}), \\ 2: &\ f(z) = tanh(z), \\ 3: &\ f(z) = sin(z), \\ 4: &\ f(z) = tanh(r)e^{i\varphi} \end{aligned}$$

Denote:

- $netin^{u_t}$, out^{u_t} – input and output vectors accordingly for the current state u and layer t
- dev^{u_t}, d^{u_t} – input and output accordingly for the backpropagaton algorithm
- A, B, C– shared matrices of weights
- f – activation function

Forward path - is done to obtain the network outputs which have given inputs. This is one way procedure, which can be described with ladder algorithm given above. Using the ladder algorithm, it is possible to obtain the following set of equations, describing the forward path:

$$\begin{aligned}
u_{t-1} : netin^{u_{t-1}} &= x_{t-1}, \\
out^{u_{t-1}} &= netin^{u_{t-1}}, \\
s_{t-1} : netin^{s_{t-1}} &= Aout^{s_{t-2}} + Bout^{u_{t-1}}, \\
out^{s_{t-1}} &= f(netin^{s_{t-1}}), \\
y_{t-1} : netin^{y_{t-1}} &= Cout^{s_{t-1}}, \\
out^{y_{t-1}} &= netin^{y_{t-1}},
\end{aligned}$$

$$\begin{aligned}
u_t : netin^{u_t} &= x_t, \\
out^{u_t} &= netin^{u_t}, \\
s_t : netin^{s_t} &= Aout^{s_{t-1}} + Bout^{u_t}, \\
out^{s_t} &= f(netin^{s_t}), \\
y_{t-1} : netin^{y_t} &= Cout^{s_t}, \\
out^{y_t} &= netin^{y_t},
\end{aligned}$$

$$\begin{aligned}
s_{t+1} : netin^{s_{t+1}} &= Aout^{s_t}, \\
out^{s_{t+1}} &= f(netin^{s_{t+1}}), \\
y_{t+1} : netin^{y_{t+1}} &= Cout^{s_{t+1}}, \\
out^{y_{t+1}} &= netin^{y_{t+1}}.
\end{aligned}$$

The backward path is to be used to find out each weight responsibility in the overall error. The impact of each weight has to be calculated. Since the shared

weights concept therefore the last statement has to be updated. Error back-propagation is to be calculated in a usual way, as all matrixes of weights would be different. Then after the backward compuatations are done the impacts for each shared weights matrix are to be avaraged and applied to the shared weight matrix. One can see these ideas are implemented in the equations below:

$$\begin{aligned}
y_t : dev^{y_t} &= dE^{y_t} = y_t - y_t^d, \\
y_{t+1} : dev^{y_{t+1}} &= dE^{y_{t+1}} = y_{t+1} - y_{t+1}^d, \\
d^{y_{t+1}} &= dev^{y_{t+1}}, \\
s_{t+1} : dev^{s_{t+1}} &= C^t d^{y_{t+1}}, \\
d^{s_{t+1}} &= f'(netin^{s_{t+1}})^d ev^{s_{t+1}},
\end{aligned}$$

$$\begin{aligned}
d^{y_t} &= dev^{y_t}, \\
s_t : dev^{s_t} &= C^t d^{y_t} + A^t d^{s_{t+1}}, \\
d^{s_t} &= f'(netin^{s_t})^d ev^{s_t}, \\
&\text{for all } t \\
\frac{\partial E}{\partial A} &= \frac{1}{T} \sum_t d^{s_t} out^{s_{t-1}}, \\
\frac{\partial E}{\partial B} &= \frac{1}{T} \sum_t d^{s_t} out^{s_t}, \\
\frac{\partial E}{\partial C} &= \frac{1}{T} \sum_t d^{y_t} out^{s_t}.
\end{aligned}$$

Once the backward path has been computed, the weights can be updated according to the simple gradient descent rules:

$$\begin{aligned}
A_{new} &= A_{old} - \eta \frac{\partial E}{\partial A} \\
B_{new} &= B_{old} - \eta \frac{\partial E}{\partial B} \\
C_{new} &= C_{old} - \eta \frac{\partial E}{\partial C}
\end{aligned} \tag{3.19}$$

Using this set of equations,the open recurrent architecture [57] can be trained. For the results of this architecture the author refers to the “Implementation” in Chapter 4.

3.4 Complex-Valued Recurrent Neural Network

Complex-valued neural network can be extended to complex-valued recurrent neural networks, with few modifications. Complex-valued recurrent neural networks are very difficult to train. Current thesis discusses the unfolding in time, then the networks utilizing this feature, last but not least the Historical Consistent Recurrent Neural Network and its extension to the complex-valued case will be introduced. It was called "Historical Consistent" due to the historical consistent data flow it uses for training to model the time series dynamics. The current chapter gives some insights into the complex-valued backpropagation and its application to the complex-valued recurrent neural network training. It shows that training error exponentially converges to a minimum. Finallythe results for some real world examples will be presented.
First of all the complex-valued ladder algorithm is to be used to compute the forward and backward paths. Then inputs and desired outputs are represented as complex numbers. This type of network can do the unfolding in time and use any inputs to produce any outputs. To proceed with the equations and explanations on this network some conventional signs has to be introduced:
l – layer number
$netin^l$, out^l – input and output vectors accordingly for the current layer l
f – activation function at a current layer
u_l – state of the network at an input layer
s_l – state of the network at a hidden layer
y_l – state of the network at the output layer
The weights matrices are of the following dimensionality: $A\,[nh \times nh]$, $B\,[nh \times ni]$, $C\,[no \times nh]$.

$$
\begin{aligned}
s_t &= f(As_{t-1} + Bu_t);\\
y_y &= Cs_t;\\
E &= \sum_{t=1}^{T} (y_t - y_t^d)\overline{(y_t - y_t^d)} \rightarrow \min_{A,B,C},
\end{aligned}
$$

where y – output vector, y^d – target vector. The architecture has been displayed before at the Fig. 3.8. Let us specify the array, which describes the architecture

the same way it was done for the real-valued network:

$$\begin{aligned} architecture = [&1 - \text{number of states} \\ &2 - \text{number of hidden neurons} \\ &3 - \text{number of forecasting states} \\ &4 - \text{number of inputs} \\ &5 - \text{number of outputs} \\ &6 - \text{activation functions}] \end{aligned}$$

where activation functions are the following:

$$\begin{aligned} 1: &\ f(z) = tanh(z_r) + i \cdot tanh(z_{im}), \\ 2: &\ f(z) = tanh(z), \\ 3: &\ f(z) = sin(z), \\ 4: &\ f(z) = tanh(r)e^{i\varphi}. \end{aligned}$$

Denote:

- $netin^{u_t}$, out^{u_t} – input and output vectors accordingly for the current state u and layer t
- dev^{u_t}, d^{u_t} – input and output accordingly for the back propagation algorithm
- A, B, C - matrices of weights
- f - activation function

To calculate the forward path of the complex-valued recurrent neural network one should use the ladder algorithm. The equations below were calculated by using this algorithm. Fortunately, there is no difference in calculation between the real-valued and the complex-valued cases.

$$\begin{aligned} u_{t-1}: netin^{u_{t-1}} &= x_{t-1}, \\ out^{u_{t-1}} &= netin^{u_{t-1}}, \\ s_{t-1}: netin^{s_{t-1}} &= Aout^{s_{t-2}} + Bout^{u_{t-1}}, \\ out^{s_{t-1}} &= f(netin^{s_{t-1}}), \\ y_{t-1}: netin^{y_{t-1}} &= Cout^{s_{t-1}}, \\ out^{y_{t-1}} &= netin^{y_{t-1}}, \end{aligned}$$

$$
\begin{aligned}
u_t : netin^{u_t} &= x_t, \\
out^{u_t} &= netin^{u_t}, \\
s_t : netin^{s_t} &= Aout^{s_{t-1}} + Bout^{u_t}, \\
out^{s_t} &= f(netin^{s_t}), \\
y_{t-1} : netin^{y_t} &= Cout^{s_t}, \\
out^{y_t} &= netin^{y_t},
\end{aligned}
$$

$$
\begin{aligned}
s_{t+1} : netin^{s_{t+1}} &= Aout^{s_t}, \\
out^{s_{t+1}} &= f(netin^{s_{t+1}}), \\
y_{t+1} : netin^{y_{t+1}} &= Cout^{s_{t+1}}, \\
out^{y_{t+1}} &= netin^{y_{t+1}}.
\end{aligned}
$$

The backward path is needed to calculate the responsibility of each weight in the overall error. The shared weights concept suggested by Zimmermann [33] was used in this case as well. The overall structure of equations remains the same as that of the real-valued example. The only difference is that now the functions are considered in much more wide class of functions, which are having the properties discussed in the "Brief Review of the Complex Analysis" chapter 2. Some conjunctions appear due to the Wirtinger calculus used to compute the derivatives:

$$
\begin{aligned}
y_t : dev^{y_t} &= dE^{y_t} = y_t - y_t^d, \\
y_{t+1} : dev^{y_{t+1}} &= dE^{y_{t+1}} = y_{t+1} - y_{t+1}^d, \\
d^{y_{t+1}} &= dev^{y_{t+1}},
\end{aligned}
$$

$$
\begin{aligned}
s_{t+1} : dev^{s_{t+1}} &= \bar{C} d^{y_{t+1}}, \\
d^{s_{t+1}} &= f'(netin^{s_{t+1}})^* dev^{s_{t+1}}, \\
d^{y_t} &= dev^{y_t},
\end{aligned}
$$

$$
\begin{aligned}
s_t : dev^{s_t} &= \bar{C} d^{y_t} + \bar{A} d^{s_{t+1}}, \\
d^{s_t} &= f'(netin^{s_t})^* dev^{s_t}, \\
&\text{for all } t
\end{aligned}
$$

$$\frac{\partial E}{\partial A} = \frac{1}{T}\sum_{t} d^{s_t}\overline{out^{s_{t-1}}},$$
$$\frac{\partial E}{\partial B} = \frac{1}{T}\sum_{t} d^{s_t}\overline{out^{s_t}},$$
$$\frac{\partial E}{\partial C} = \frac{1}{T}\sum_{t} d^{y_t}\overline{out^{s_t}}.$$

The results of the Complex-Valued Recurrent Neural Network (CVRNN) are discussed and presented in the "Implementation" in Chapter 4. One should admit that this network can forecast only by its forecasting layers. The forecast can be obtained by iterative application of the matrix B to the output of the state. Then, after the needed forecasting horizon has been obtained, one should use the matrix C to convert the state output into the network output. Normally, the network is to be trained with only one forecasting unit. Note, that this unit does not have any inputs, as they are not available. At some point the network struggles due to the absence of inputs since it was trained with inputs. This property decreases the quality of the forecast. This was the reason to look for another architecture which does not have such "nasty" property. This network is so-called Historical Consistent Neural Network (HCNN).

3.5 Historical Consistent Complex-Valued Recurrent Neural Networks

3.5.1 Complex-Valued Historical Consistent Neural Network

Historical Consistent Neural Networks (further HCNN) were first described by Zimmermann [77]. This architecture is very interesting due to the stability of training and simplicity of the construction. It allows for a unique correspondence between the dynamical equations, neural network architecture and the locality of the learning algorithms. This architecture models the behavior of the dynamical system which can further be described by eq. 3.18. Modeling of such systems is of interest for many application areas. But what if there is a need in complex-valued neural network inputs or in case if continuous time modeling is needed? Even with real-valued dynamics modeling there are many unsolved problems with the things with regard to stability of training, learning itself and stopping criteria. The following section stands to show the transition from the real-valued HCNN (RVHCNN) to the complex-valued HCNN (CVHCNN). Some insights for the complex-valued backpropagation and its application to the CVHCNN will

be considered. Continuous time modeling and the so-called time teacher forcing technique will be presented.
The core idea of this type of network is that there are no inputs, only outputs. For the first sight it can look strange, but the thing is that there is no difference between the inputs and the outputs for this model. Consider the example. Imagine, there are just data vectors which contain recorded values of some system which works according to the Ideal Gas law $PV = RT$, where P is pressure, V is volume and T is temperature (R is the ideal gas constant). Let us consider such system S. The system S can be totally explained with 2 variables, namely P and V. The normal setup for such system would be to use the pressure and the volume as an input and the temperature as an output (or vice versa). For such problem we could use the FFNN or the RNN discussed above. On the other hand, there is no need to define explicitly what the system inputs and what the system outputs are, since for the system S the temperature is not an output, this is just one of the systems parameters. An even harder example would be in case one is trying to model the whole world. It can be rather difficult to devide the data into the inputs and outputs. This is very essential for the autonomous systems. There is nothing coming from outside into the system. Therefore, for the system S, it would be natural to use P, V, T as the outputs which express the condition of the system. The network should use these variables to model the system itself. Then if the system is modeled, one can use some extraction mechanism to extract the values of some parameters, like temperature or pressure. That is how HCNN is working. It models the system with its realizations and then uses the extraction mechanism to take the needed variables values out of the system. First of all we should start from the forward path of the network and its general representation. The Historical Consistent Neural Network can do the modeling of the systems which correspond to the eq.3.18. This architecture is functioning in the following

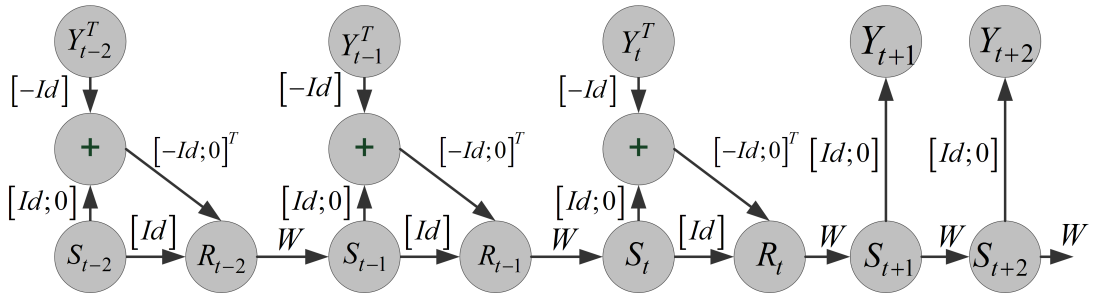

Figure 3.9: HCNN architecture. Notations: Y^T is the target, R_t is the teacher forcing block, S_i is the state vector, W is the shared weights matrix,$[Id]$ is the identity matrix, $(\cdot)^T$ stands for the matrix transpose (see [77] for more details).

manner (see fig. 3.9): some bias signal is coming to the left most state of the

network (often it is a random noise distributed normally). Here the bias sigmal will be discussed in more detail for clarification purposes.
HCVNN is a closed dynamic architecture, whith no inputs, only a limited amount of outputs according to its architecture. Therefore there is no input information flow. However something should start the feedforward process, since in case there is no information flow to the left hand side. For this purpose a bias signal can be introduced, which compensates absence of the rest of the HCVNN. Ideally, HCVNN should be of an unlimited length covering the history of the time series. Since such approach is not possible we normally have to reduce the HCVNN to some finite number of states. Following Zimmermann [77], one can use the adaptive noise to simulate abscense of information flow from the left. Noise cleaning is such an approach. Noise plays an important role in the model uncertainty. This uncertainty exists in the beginning of the network training. After some epochs of training, the model will have less uncertainty and thus the level of noise will be reduced. One can see the figure, which explains the basic on the noise cleaning, see Fig. 3.10. One can represent the cleaning with the manifold, shown at the Fig.3.11.

Once the cleaning noise issues have been resolved one can proceed with the model description.
As has been noticed, HCVNN network does not have inputs (in the usual sense of this word, states transfer the information from one state to the next one, doing unfolding in time), which means it is autonomous. To train the network one should use the so called Teacher Forcing (Zimmermann [77]) for the time moments from $t-n$ to t (see fig.3.9), where n is the number of network layers (recurrence layers). The following system of equations is to be used to describe the teacher forcing training:

$$\begin{aligned} \tau \leq t: &\quad s_{\tau+1} = f(W(s_\tau - [Id;0]\,(y_\tau - y_\tau^d))), \\ \tau > t: &\quad s_{\tau+1} = f(W s_\tau), \\ \forall t: &\quad y_\tau = [Id,0] s_\tau, \end{aligned} \tag{3.20}$$

where $[Id,0] = \left[\underbrace{\overbrace{1,1,...,1}^{N_s},0,0,...,0}_{N_o}\right]$, N_o is number of network outputs, N_s is number of network states (number of hidden neurons), f is non linear transition function, W is the weights matrix. Note, that matrix W and function f are every time (at each state of the model) the same matrix and function y_τ is the network output, y_τ^d is the network desired output. To have the forecast one should apply matrix W and then transition function f to the state vector, to obtain the network outputs the matrix $[Id;0]$ is to be applied to the obtained state vector. Iteratively applying a matrix W and a function f for the needed horizon the

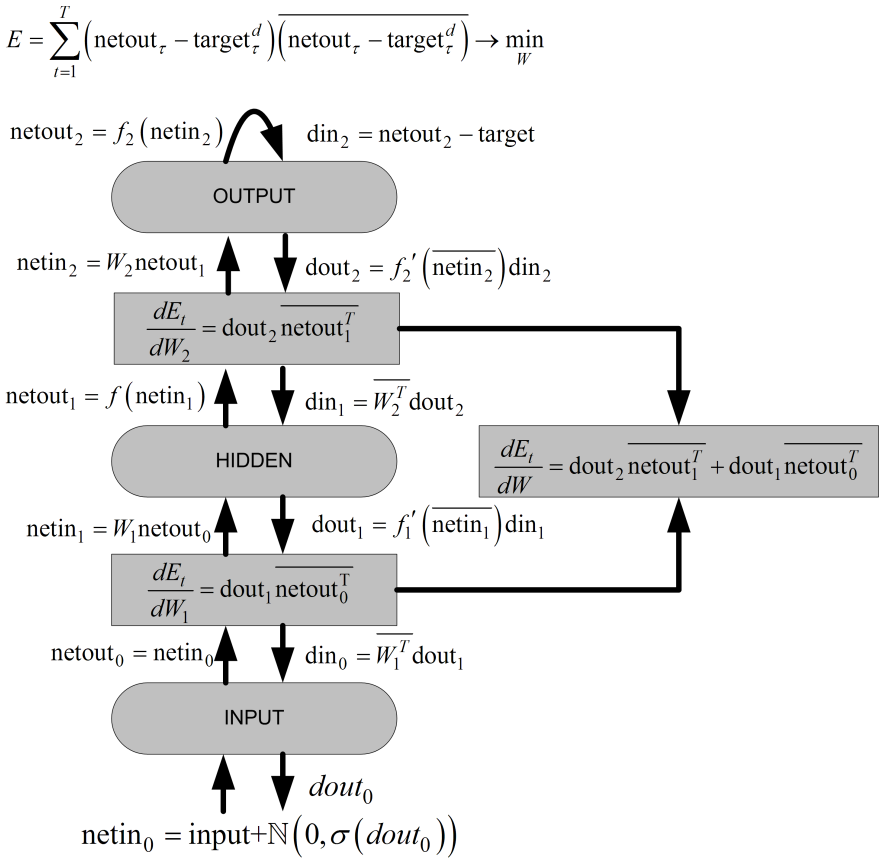

Figure 3.10: Representation of the backpropagation with cleaning. $N(0, \sigma)$ means adaptive uniform noise, where σ is dispersion of the $netin_0$.

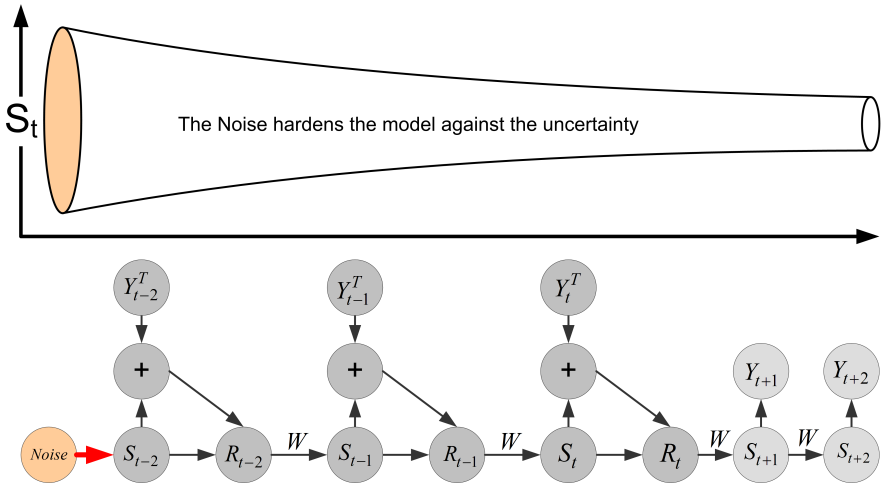

Figure 3.11: Representation of the cleaning idea. The level of noise reduces from the left to the right part of the HCVNN. This hardens the model against the uncertainty in training data.

needed forecast can be obtained.

Definition. *N steps forecast means that network uses its own 1 step forecasts to predict for N steps ahead (iterative forecasting).*

Note that with such an approach the error is accumulated relatively fast (depending on the problem).

Using the teacher forcing training and the CVBP algorithm the CVHCNN can be trained. The stability of CVHCNN during the training is to be underlined separately. The explanation to this fact is like following. Due to the teacher forcing,the network avoids uncontrolled behavior of the information flow. This uncontrolled behavior can be rather dangerous for the stability of computations. Remember, that such effects can be caused by the unlimited functions or function singularities.

3.5.2 Complex-Valued Recurrent Network Forward Path

The forward path can be easily computed using the Ladder Algorithm which is unique in a sense, that it can be applied to nearly any architecture we can imagine. Therefore the HCNN is not exclusion. The equations which describe the forward path of the HCNN are presented below.

Time moment $[t-1,t]$:

$$\begin{aligned} s_{t-1} : netin^{s_{t-1}} &= Ws_0, \\ out^{s_{t-1}} &= f(netin^{s_{t-1}}), \\ y_{t-1} : netin^{y_{t-1}} &= [Id,0]out^{s_{t-1}}, \\ out^{y_{t-1}} &= netin^{y_{t-1}}, \end{aligned}$$

$$\begin{aligned} r_{t-1} : netin^{r_{t-1}} &= out^{s_{t-1}} + [-Id,0](out^{y_{t-1}} - y^d_{t-1}), \\ out^{r_{t-1}} &= netin^{r_{t-1}}, \\ s_t : netin^{s_t} &= Wout^{r_{t-1}}, \\ out^{s_t} &= f(netin^{s_t}), \end{aligned}$$

Time moment $[t,t+1]$:

$$\begin{aligned} y_t : netin^{y_t} &= [Id,0]out^{s_t}, \\ out^{y_t} &= netin^{y_t}, \\ r_t : netin^{r_t} &= out^{s_t} + [-Id,0](out^{y_t} - y^d_t), \\ out^{r_t} &= netin^{r_t}, \\ s_{t+1} : netin^{s_{t+1}} &= Wout^{r_t}, \\ out^{s_{t+1}} &= f(netin^{s_{t+1}}), \end{aligned}$$

Time moment $[t+1,t+2]$:

$$\begin{aligned} y_{t+1} : netin^{y_{t+1}} &= [Id,0]out^{s_{t+1}}, \\ out^{y_{t+1}} &= netin^{y_{t+1}}, \\ s_{t+2} : netin^{s_{t+2}} &= Wout^{s_{t+1}}, \\ out^{s_{t+2}} &= f(netin^{s_{t+2}}), \\ y_{t+2} : netin^{y_{t+2}} &= [Id,0]out^{s_{t+2}}, \\ out^{y_{t+2}} &= netin^{y_{t+2}}. \end{aligned}$$

After the feedforward path is calculated we can proceed with the backward path to spread the error to the matrix W. One should take into account that he can easily produce the forecast by applying the matrix W iteratively to the state output and obtain the forecast. After one reached the needed forecast horizon he should just apply the identity matrix and take the needed variables out of the system (see fig.3.9).

3.5.3 Complex-Valued Recurrent Network Backward Path

The backward path aims to spread the overall approximation error to the network weights, to calculate the responsibility of each neuron in the overall error. For this purpose one should use the backward path. The efficient computation of the backward path can be done through the ladder algorithm discussed in detail above. The only difference from the real-valued case of the network is some conjunctions which arise out of the Wirtinger calculus. All the rest, the sequence and the logic remain the same. Time moment $[t+2]$:

$$\begin{aligned} y_{t+2} : dev^{y_{t+1}} &= dE^{y_{t+2}} = y_{t+2} - y^d_{t+2}, \\ d^{y_{t+2}} &= dev^{y_{t+2}}, \\ s_{t+2} : dev^{s_{t+2}} &= [Id, 0] d^{y_{t+2}}, \end{aligned}$$

Time moment $[t, t+1]$:

$$\begin{aligned} d^{s_{t+2}} &= \overline{f'(netin^{s_{t+2}})} dev^{s_{t+2}}, \\ y_{t+1} : dev^{y_{t+1}} &= dE^{y_{t+1}} = y_{t+1} - y^d_{t+1}, \\ d^{y_{t+1}} &= dev^{y_{t+1}}, \\ s_{t+1} : dev^{s_{t+1}} &= [Id, 0] d^{y_{t+1}} + \overline{W} d^{s_{t+2}}, \\ d^{s_{t+1}} &= \overline{f'(netin^{s_{t+1}})} dev^{s_{t+1}}, \\ r_t : dev^{r_t} &= \overline{W} d^{s_{t+1}}, \\ d^{r_t} &= dev^{r_t}, \end{aligned}$$

Time moment $[t-1, t]$:

$$\begin{aligned} y_t : dev^{y_t} &= dE^{y_t} + [-Id, 0] d^{r_t}, \\ d^{y_t} &= dev^{y_t}, \\ s_t : dev^{s_t} &= [Id, 0] d^{y_t} + d^{r_t}, \\ d^{s_t} &= \overline{f'(netin^{s_t})} dev^{s_t}, \\ r_{t-1} : dev^{r_{t-1}} &= \overline{W} d^{s_t}, \\ d^{r_{t-1}} &= dev^{r_{t-1}}, \\ y_{t-1} : dev^{y_{t-1}} &= dE^{y_{t-1}} + [-Id, 0] d^{r_{t-1}}, \\ d^{y_{t-1}} &= dev^{y_{t-1}}, \\ &\text{for all } t \\ \frac{\partial E}{\partial W} &= \frac{1}{T} \sum_t d^{s_t} \overline{out^{s_{t-1}}}. \end{aligned}$$

After the backward path is done one should think about training of the network. Here some problems arise, due to the network initialization, bias signal, rate of gradient descent, etc. These issues will be widely discussed in the next subsection.

3.5.4 Complex-Valued Recurrent Neural Network Training with Complex-Valued Random Search

Random Search Algorithm for Complex-Valued Case

Due to the problems with the nonanalytical functions and the derivatives of these functions (the error functions) the training typically begins with the global search algorithm, which does not require any gradient information. After the region for the local minimum has been found, the gradient descent algorithm is applied to converge to this minimum.
According to Kusherbaeva [66] and Sushkov [69], the adaptive random search method is a global stochastic optimization technique which does not require any priori information about its optimization problems. Let the global optimization problem be formulated as follows: $\Phi(x) \to \min_{x \in X}$ where $X = [0,1]^n \subset R^n$ is a non-empty set of feasible solutions, $x = (x_1, \ldots, x_n) \in X$, and $\Phi : R^n \to R$ is a continuous objective function. Let $I_i \subset X$ be a perspective interval for variable x_i, $i \in 1 : n$; a Cartesian product of sets I_i, $i \in 1 : n$ a perspective domain with center point x_i^0, $i \in 1 : n$. Then, in general, the algorithm of random search can be described in the following way. The process of random search is divided into N_s steps (also known as epochs). On every step of the vector x^j, $j \in 1 : N_s$ is randomly selected and the value of the objective function $\Phi^j = \Phi(x^j)$ is calculated. By using the following equation: $\Phi^j_{min} = \min\{\Phi^j, \Phi^j_{min}\}$ a minimal value of the objective function in the step j, $j \in 1 : N_s$ is calculated. A wide set of experiments have been conducted in order to show adaptive random search algorithm's effectiveness and to make recommendations for choosing heuristic parameter values [66] and [69]. In the case of the $CVNN$ the algorithm works independently with real and imaginary parts of the complex number, thus, expanding the dimensions of the optimization space twice. The objective function in this case can be described as follows:

$$E(w) = \mathrm{Re}\left(y_t - y_t^d\right)^2 + i\,\mathrm{Im}\left(y_t - y_t^d\right)^2 \tag{3.21}$$

Combination of the Complex-Valued Backpropagation and the Random Search Algorithm

In the previous chapters it became apparent that calculation of gradients presents

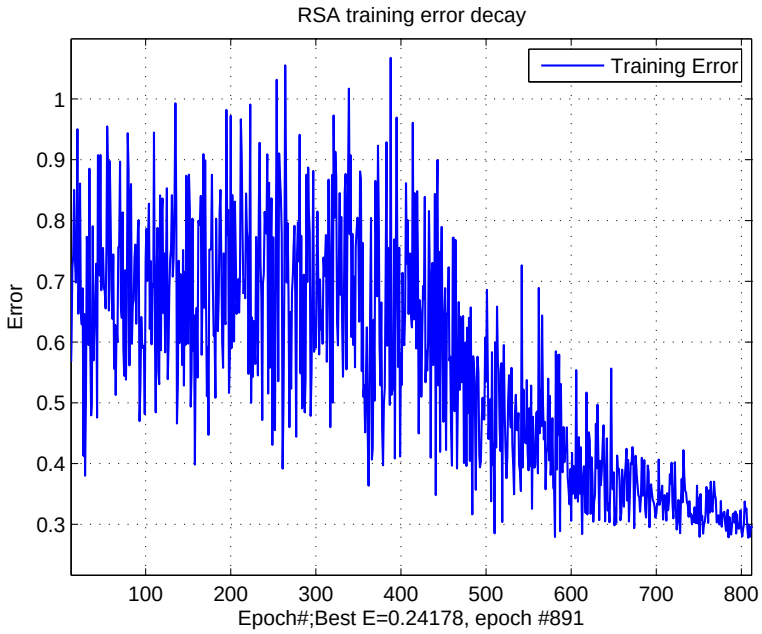

Figure 3.12: Random Search Algorithm training set error decay.

a problem in the complex-valued case. In order to simplify this problem the non gradient global optimization method (here Random Search Algorithm) is used as an initialization method in order to find the local minima region. In order to converge to the minima itself, the gradient descent method is applied with very small learning rate in order to reach the minima. Therefore several hundred epochs of RSA and then the CVGD were applied. This combination proved to be better than separate usage of both algorithms for the considered problem. (the concidered problem will be described in details in the Results chapter). The typical behaviour of the training error can be seen in fig. 3.12.

3.5.5 Time Teacher Forcing with Complex-Valued Historically Consistent Neural Network

There is also a possibility to enforce the time of the network outputs by doing a special architecture, note that this is only possible with the complex-valued case of the HCNN. The trick is the following. Let us assume we have trained the CVHCNN. Now we have to make the forecast into the future direction. Thus we have to apply matrix W iteratively to the states of the HCNN. But one can see that the error is accumulating through the iterative application of the matrix W since at the production side of the CVHCNN there is no more teacher forcing.

One can use so-caled time teacher forcing block. The idea is that one does not

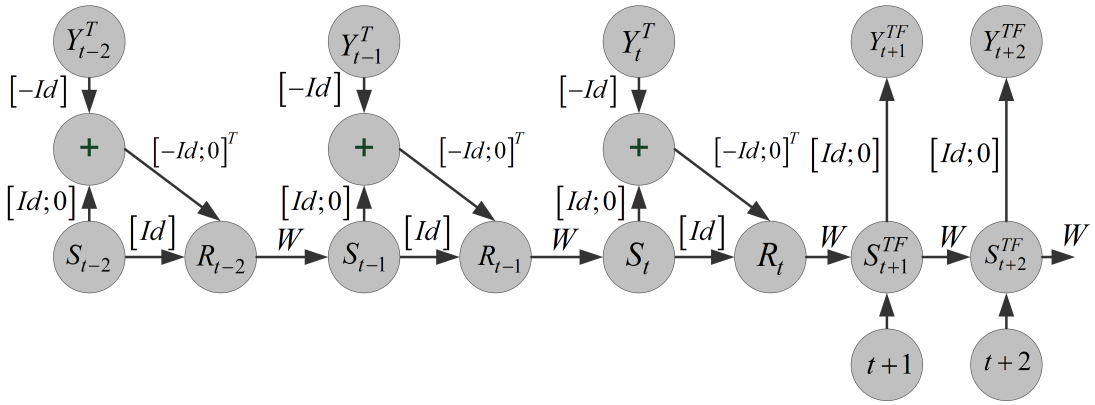

Figure 3.13: Time Forcing historical consistent neural network

know the values of the data he is trying to predict, but he knows the time moments for the prediction. Thus one can take out the time of the network output, substitute it with the correct time and replace the uncorrected state values with the correct ones (in case time has been putted into the exponent degree).
One can see at the Fig.3.13 that in the forecasting part of the network the states are marked with TF which means Teacher Forcing. The idea is like the following: each state knows to which time moment it should be related, therefore it replaces the recieved time moment with the correct one, leaving the absolute part unchanged. This artificial change in the states makes it possible to reduce the error while iterative application of matrix W and function $tanh$.

3.5.6 Shared Weights Matrix for the CVHCNN

One should say a few words about the weights matrix W which is to be used in case of CVHCNN. In all experiments a sparse weights matrix has been used. From the biology it is became known that humans brains have very sparse connections, see Fig.3.14 and Hooney [34]. The figure has been taken from: *www.scholarpedia.org/article/Brain-connectivity*. By doing the sparse matrices a faster calculations (due to many zeros in the matrix W) can be done. Moreover the information channels of neurons will not be overloaded with the useless information (which means the computations will not go to singularity points of the transition functions due to a large amount of uncontrolled sums while calculating the feedforward path of the $CVHCNN$). One can see the sparse matrix of weights with about 5% of live (non zero) weights at Fig.3.15.

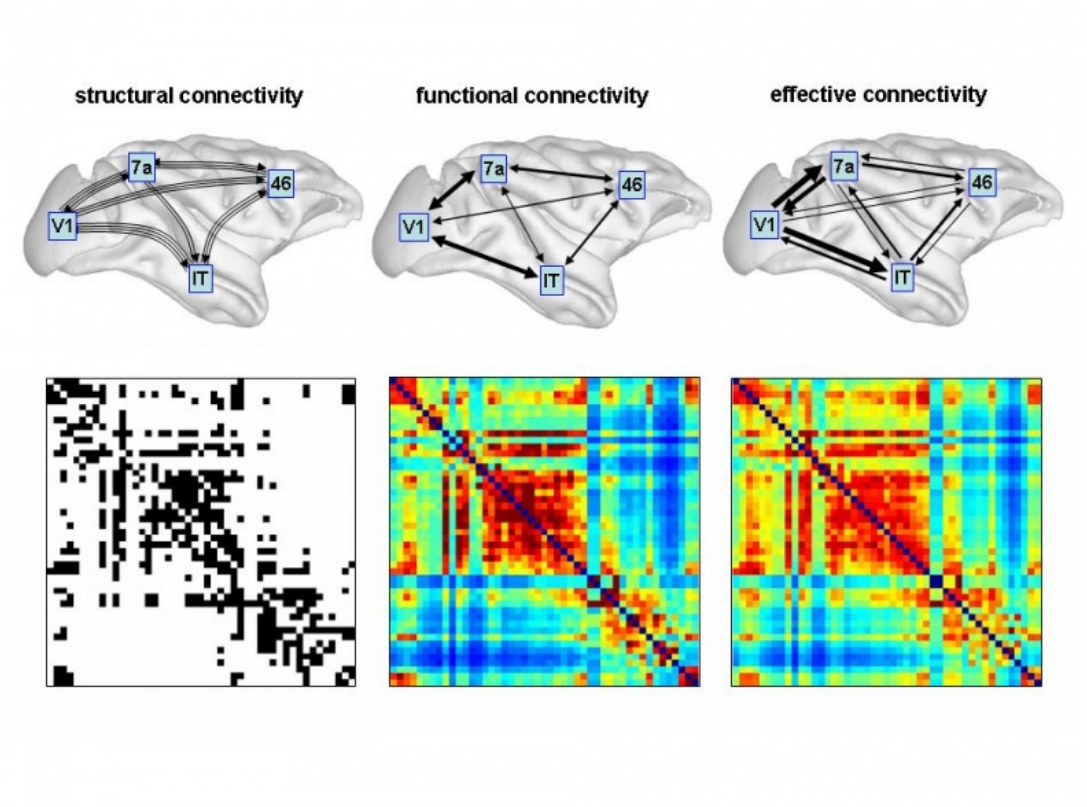

Figure 3.14: Sparsity of human brain. Matrixes on the left-hand side show binary structural connections, symmetric mutual information (middle) and non-symmetric transfer entropy (right). Data was obtained from a large-scale simulation of cortical dynamics.

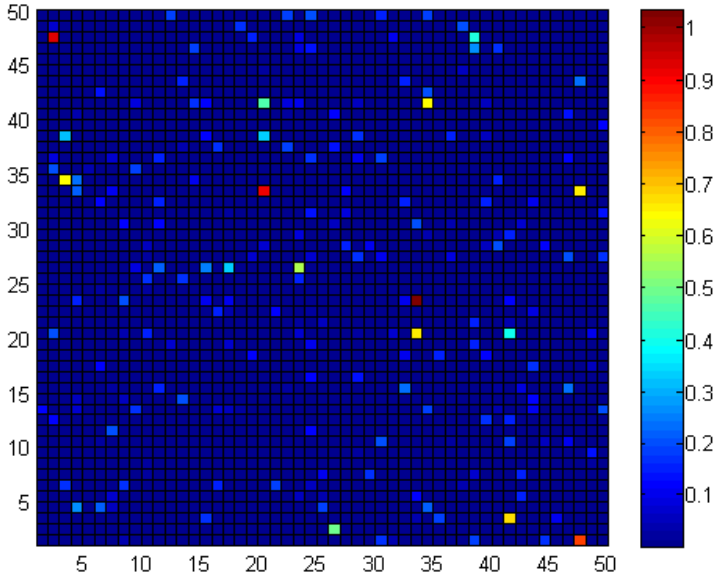

Figure 3.15: HCNN matrix of weights. The color represents the absolute part of the weight value.

3.5.7 Continuous Dynamics Modeling with HCVNN

This subsection describes the unique properties of the HCVNN. This subsection was missing in previous chapters because the other architectures have well-documented properties. Much information on MLP networks or Elman (Unfolding in time) recurrent architectures can be found in the literature. The disadvantage of these networks is that the data used for NN training must be strictly discrete with equal time stamps. If the data is recorded with unequal time stamps (a common situation in the industry) the data must be somehow approximated to make the time stamps equidistant. Only then the NN can be trained and used. This is so-called discrete time modeling. But what if continuous dynamics must be modelled? At the moment there is no answer to this question. In this subsection an attempt to solve the problems of continuous dynamic modeling using neural networks will be carried out.

Continuous Dynamics System

An example of the continuous system will first be presented. To make it simple, real-valued sin function, a continuous and infinitely differentiable function will be presented. One must first assume the network to be trained with t time stamps. But what happens if the forecast for this function with time stamps $t/2$ is required? This is impossible using "traditional" neural networks. One should note

that in complex numbers the number can be presented in Euler notation, namely as $A\exp^{i\phi}$. It would be logical to raise the time stamp to the exponent degree in the following manner: $sin(t)\exp^{t}$. Then the output of the CVNN contains the absolute value $sin(t)$ and angular value t. Coding the system values using the Euler notations can be rather useful since one can then obtain not only the absolute value of the forecast, but also the time moment to which this number is related. Forcing the time moment into the network output allows the network to more easily produce an output. This "time forcing" is partial teacher forcing which can be used for the part of the network, which does not participate in the neural network training. The interesting thing is that one can continue forcing the network at the test set simply because he knows the time moment in the future to which his forecast is to be related. The result for *tanh* function is presented.

Experimental Results for Time Forcing in $HCVNN$

First the network was trained using the training set. The training set consisted of the sin function, which had four periods for 100 equidistant points. The resulting

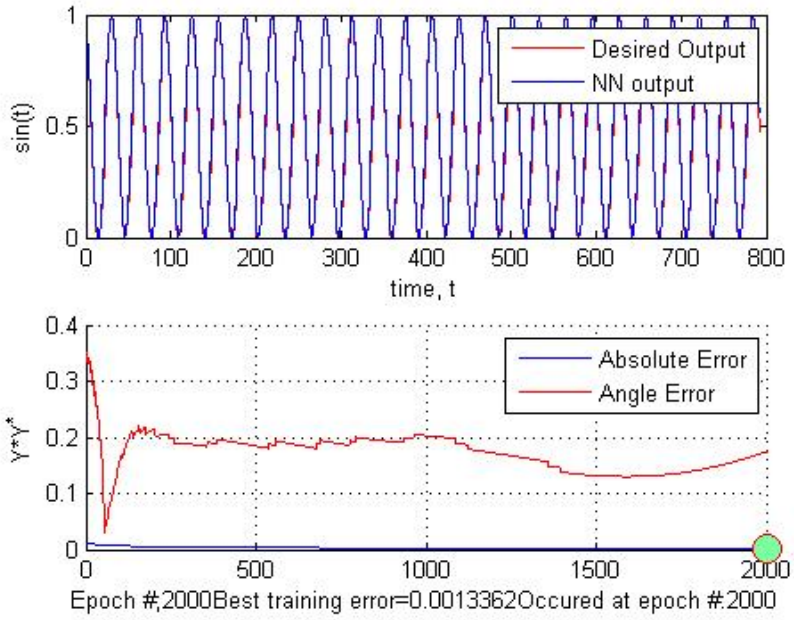

Figure 3.16: Training Set. This set was used to train the network. Lower picture shows how good the network was trained (absolute value). The phase of the error should come to zero, but this is not obligatory.

training error quickly converged to zero. However from the test set results it became apparent that the phase of the output was uncorrelated with the desired phase (see fig. 3.18). In order to compensate for the lack of correlation between

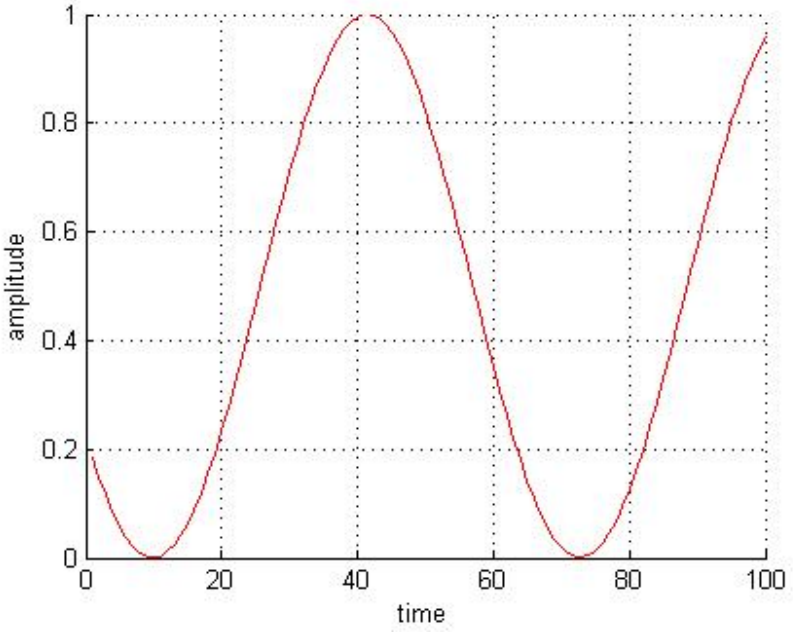

Figure 3.17: Test set.

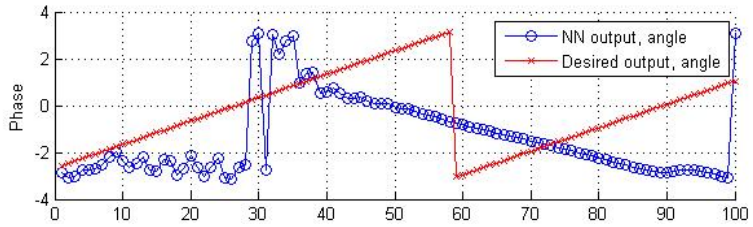

Figure 3.18: Effect of the uncorrelated phase for the test set.

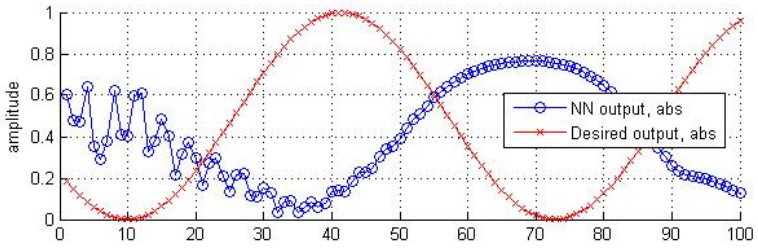

Figure 3.19: The absolute value of the network output for the test set

phases, time forcing is to be applied to the HCVNN outputs for the recurrent part of the network. Therefore the procedure is the following: one has to force the outputs at the recurrent part of the HCVNN with the correct time moment. The results can be seen in the figures below (fig.3.20) The figure for the phase at

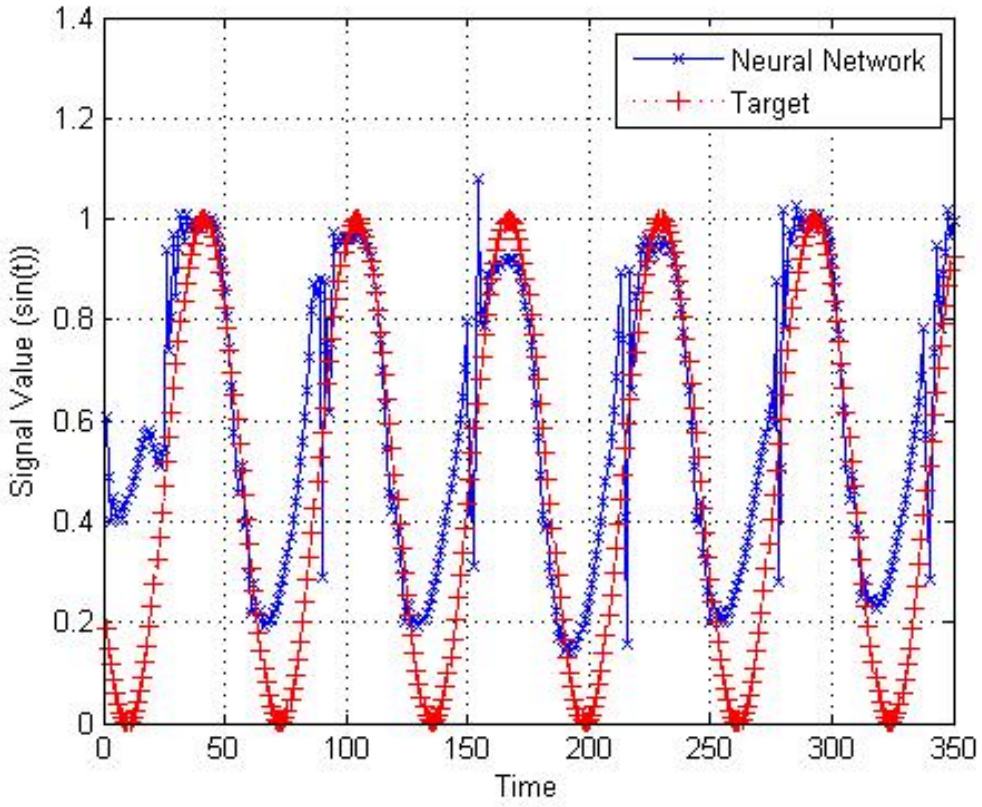

Figure 3.20: The prediction corrected with time teacher forcing.

the production set, since it is equal to the desired phase. As can be seen from the fig. 3.20, the situation is ideal. More than 300 prediction steps can be made on the same network, note that the network was trained with different time stamps. Some more interesting properties of the CVHNN arise in the topic of brain synchronization, which can be found in the corresponding "Implementation and Application" chapter. In the following chapter the well-known binding problem will be addressed. Moreover it will be shown that brain memory can be modeled with the described architecture. It will be shown that it is easier to solve the binding problem when using the complex numbers than when using real ones.

4

Benchmarking

Benchmarking is important to validate new methods and algorithms and its value cannot be underestimated. The aim of this chapter is to simplify the benchmarking procedure and apply this process to benchmarking of complex-valued neural networks. All results presented in this thesis should be considered using the tools given in this section because one cannot state the progress in the area of neural networks without accurate benchmarking. One can always find a dataset (with some prior knowledge about the process) that any particular method is "'better"' than all other methods. In general, this discussion is applicable to the majority of the papers. First, real-valued statistics will be discussed in order to verify the notations, then a few existing artificial and real world datasets will be described and two methods of benchmarking for the complex-valued models will be presented. At the end of the section, an Internet based database with embedded properties for the benchmarking of complex-valued regression models will be suggested.
The importance of the philosophy and semantics of neural networks seems to be walking to the background, while the pure scientific approach for data approximation and data classification is gaining more attention. Unfortunately, the neural network community has not established a standard for benchmarking in general, and obviouslysuch a standard is lacking for complex-valued neural networks in particular. Consequently dozens of architectures have appeared and many papers have stated that their neural network is capable of solving "all the problems of the world". On the other hand, other authors have tried to find benchmarks, which could be used by the community to at least attempt a comparison of results. The current level of understanding of complex models is quite poor. This thesis attempts to suggest some ways to improve this situation. The real-valued benchmarking case will be discussed before the complex-valued one.
The literature that adresses benchmarking problems is presented further: Prechelt [45], Waugh [59], Wolpert [20] and Ciftcioglu [53]. The list of the datasets was taken from the following website [14].

Some well known benchmark datasets include the following ones:

- Neural Networks Databases - Benchmarks
- Database of Machine Learning Group
- Universal Problem Solvers, Inc., Machine Learning Data Sets
- ELENA Database . The databases are splitted into two parts: artificial ('Gaussian', 'Clouds' and 'Concentric') and real ('Satimage', 'Texture', 'Iris' and 'Phoneme').
- The CMU Learning Benchmark Archive
- Data at UCLA Stat. Dept.
- StatLib at CMU
- NIST databases

One can see from the list above, that it is impossible to objectively compare networks due to the wide range of data available on the Internet. In general, it is possible to argue that there are some well known benchmark e.g.e Elena, Iris, Wine or Boston datasets. However, these datasets are not new, do not reflect modern problems, and do not meet industrial needs.
The second thing, which should be discussed, are benchmarking procedures. Even when the benchmark data is good enough, the benchmarking itself can be rather poor. Here, one can refer to Prechelt [45]. He has established a four criteria approach which is not always obeyed (**Validity**, **Reproducibility**, **Comparability** and **Volume**). Last but not least, the performance of the statistical estimations is discussed. After the forecast or the classification is done, one should estimate the performance of networks. Here, the most popular performance measures are $RMSE$ - root mean squared error, R^2 - determination coefficient, the correlation r and some other statistical values. In general, it is possible to show, that there are many much more interesting things that can be determined: the presence of a delay, border effects, how good the dynamics coded into the weights of the algorithms are and so on. All these things will be discussed in depth in the following subsection.

4.1 Statistics Review

> "... it is only the manipulation of uncertainty that interests us. We are not concerned with matter that is uncertain. Thus we do

> not study the mechanism of rain; only whether it will rain." -Dennis Lindley, "The Philosophy of Statistics", The Statistician (2000).

According to Dodge [54], statistics is the science of effectivly using numerical data relating to groups of individuals or experiments. It deals with all aspects of this, including the collection, analysis and interpretation of such data, as well as planning of the collection of data, in terms of the design of surveys and experiments. Statistics is a very important science when one is talking about the benchmarking. Each benchmark is based on statistical measures like MSE or R^2, which describe the quality of the constructed models. Every one is fighting for a higher R^2 (we will discuss this coefficient later) and a smaller RMS. However the R^2, correlation and RMS are not sufficient in all cases. When one deals with only a subset of the measures, he or she is using some assumptions which start working in the background of the research. Sometimes these assumptions are incorrect, while people use the measures which serve different assumptions about the data. Assume the dataset D ,which contains p patterns and v variables, so that $D = \begin{pmatrix} D_{11} & \dots & D_{1v} \\ D_{21} & \dots & D_{2v} \\ \vdots & \ddots & \vdots \\ D_{p1} & \dots & D_{pv} \end{pmatrix}$, where $D_{i,j} \in \Re, i = [1..p], j = [1..v]$. Then the dataset should be divided into the training set D_{TR}, the validation set D_V and the test set D_T.
The goal is to build a model (Neural Network) using training data, that validates the model parameters using the validation data, and then to test the model using the test data.

Definition 1. *The training set is a subset of D, on which the model is trained. This subset is not used for validation of the model and not used for testing of the model.*

Definition 2. *The validation set is a subset of D, on which the model is validated. This subset is not used for training the model and not used for the testing the model. This subset should be used for the tuning of model parameters.*

Definition 3. *The test set is a subset of D, on which the model is tested. This subset is not used for training the model and not used for model validation. This subset should only be used for model tests. No parameter tuning and no model training is possible with this subset.*

Remark. Sometimes the validation set and the test set are united. In the present work, the validation set and the test set are split.

Therefore the general rule, for dividing the D into 3 subsets is:

$$\begin{aligned} D_{TR} \cap D_T \cap D_V &= \oslash \\ D_{TR} \cup D_T \cup D_V &= D. \end{aligned} \tag{4.1}$$

Equations 4.1 make subsets absolutely independent from each other. However these datasets can intersect, depending on the problem. This is an important step in the way towards model verification and statistical data analysis.
Remark. Please note that the current subsection considers only the supervised learning of neural networks in which target data are available. After the model is trained with the D_{TR} , the parameters of NN are to be fixed with the D_V. Only after the validation is done and the network parameters are fixed one can perform the statistical tests for D_T.

Problem formulation. Compare two vectors of length t, where t is the length of the D_T and then conclude whether the approximation made by the model data is correct or not. Vector $T = [1..t]$ is a vector of target values, to be later used for comparison. The model M produces the output of length t according to eq.4.2:

$$P_1^t = M\left(\omega, D_T\right) \tag{4.2}$$

where ω contains model parameters.

The problem is: Define the explanatory variables based on experimental data (observation data) . Considering the random components of the data as a random variable influence, obtain estimations for the data distribution parameters. Denoting the dependent variable as y, and its explanatory part by $X = (x_1, x_2, ..., x_k), k \leq v$ can be written as $f(X)$. The random component will be denoted as ϵ. Then the model can be defined as:

$$y = f\left(X\right) + \epsilon \tag{4.3}$$

As an explained part of the $f(X)$ it is obvious to select its average (expected) value with given X. Then one can write:

$$E_x(y) = f(X) \tag{4.4}$$

where $E_x(y)$ is the mathematical expectation. One can rewrite eq. 4.3 in the following manner:

$$y = E_x(y) + \epsilon \tag{4.5}$$

Eq. 4.3 is known as the theoretical equation for the regression, $f(X)$ is theoretical function of the regression and eq. 4.5 is the equation for the regression model. One should note that one of the main conditions for this equations is that $E_x(\epsilon) = 0$. This condition for noise is sometimes crucial. One of the most crucial points of this research is to select the observations y and their explanatory variables $x_j, j = 1..k$. Denote such observations in the following manner: $x_{i1}, x_{i2}, ..., x_{xi}, y_i$, where $i = 1, ..., n$ is the number of observation and k is number of explanatory variables.
Typically, there are two types of observation selection:

- Cross-sectional data. Independent samples of data, received at specific point in time (no cause and effect relation).
- Time series data. Data where not only the samples are important but also the time that these observations were taken.

After the set of explanatory variables have been defined and the data has been received, the next task is to find the $f(X)$. This function cannot be derived explicitly, therefore an estimation to the function must be built. The standard procedure for function estimation consists of two steps:

1. One should select the type (the family group) of the function $f(X)$.
2. Using the mathematical statistics estimate the parameters of this function.

Unfortunately, there is no formal procedure on how to properly select the function in step 1. One of the most oftenly used function families is linear function. The usage of the linear function has many statistical advantages. Below is the simplest case:

$$\hat{y} = \hat{f}(X, \vec{B}) \tag{4.6}$$

where $\hat{y}$ is the estimation of y with a given X. Here $\vec{B}$ is the parameters vector for $\hat{f}$ which is an approximation of f. For the construction of $\hat{f}(x)$, one should use the least-squares method. According to this method the parameters for $\hat{f}(x)$ should be selected such in a way that:

$$\sum_{i=1}^{n} \left(y_i - \hat{f}(x_i) \right)^2 \underbrace{\longrightarrow}_{par} min \tag{4.7}$$

where *par* are the function parameters. The parameters should deliver the minimum for the sum. The parameters, which can be found with this method are

called parameter estimations. The error of the approximation can be defined as $e_i = y_i - \hat{y}_i$. The error explicitly shows the quality of the approximation. The analytical method can be interpreted with the following example: if y represents the costs of production, x is the volume of the needed resource, then the dependency should look like: $y = \alpha + \beta x + \epsilon$, where α are the fixed costs, and βx are the variable costs.

Main assumptions of regression analysis.
The main issue of regression analysis is the estimation of ϵ. In the classical model of regression analysis, the following conditions are assumed:

1. ϵ_i should be random
2. The mathematical raise expectation is equal to zero. $E(\epsilon) = 0$
3. ϵ_i and ϵ_j are uncorrelated: $E(\epsilon_i)E(\epsilon_j) = 0, i \neq j$
4. The dispersion of the noise ϵ_i is constant for each i:$D(\epsilon_i) = \sigma^2$
5. Noise is not correlated with any explanatory variables.

These assumptions are so-called first group of assumptions. They provide indispensable conditions. So-called sufficient conditions are given by the second group of assumptions:

1. The joint distribution of random values $\epsilon_1, ..., \epsilon_n$ should be normal (Gaussian).

Statistical properties of estimations. Gauss-Markov theorem.
Theorem (Gauss-Markov). If the regression model $y = \alpha + \beta x + \epsilon$ satisfies the groups of conditions above-mentioned, then least squares estimations of model parameters (a and b) will have the minimum dispersion in the class of linear unbiased estimations.

Note that after constructing the sample regression, the observed variables y_i can be presented as $y_i = \hat{y}_i + e_i$, where $\hat{y}_i = a + bx_i$, in which a and b are parameters. The residuals e_i with respect to the noise ϵ_i are explicitly observed values. It is possible to say, that e_i is sample estimation of the noise ϵ_i. The statistics (sample residue dispersion) is defined with the residuals e_i:

$$S^2_{res} = \frac{\sum_{i=1}^{n} (y_i - \hat{y}_i)^2}{n-2} = \frac{\sum_{i=1}^{n} e_i}{n-2} \tag{4.8}$$

This is the unbiased estimation of σ^2 , the dispersion of the noise.

How to estimate the quality of the approximation.
The determination coefficient is one of the most effective estimations regression model accuracy and serves as a measure of the quality of the regression model. The equation $y_i = \hat{y}_i + e_i$, e_i however can not be explained with the selected regression. It is possible to show that the dispersion consists of $D(y) = D(\hat{y}) + D(e)$, where $D(y)$ is the explained part of the data and $D(\hat{(y)})$ is the unexplained part of the data. The second task is to influence the unrecorded variables. Therefore, it is possible to write the following equation:

$$\sum_{i=1}^{n} (y_i - \bar{y})^2 = \sum_{i=1}^{n} (\hat{y}_i - \bar{y})^2 + \sum_{i=1}^{n} (y_i - \hat{y})^2$$

$$R^2 = 1 - \frac{\sum_{i=1}^{n} (y_i - \hat{y})^2}{\sum_{i=1}^{n} (y_i - \bar{y})^2} \tag{4.9}$$

The properties of R^2 are clear from the equation. One can estimate the quality of the approximation along with the average error, which can be computed with the following equation:

$$\bar{A} = \frac{1}{n} \sum_{i=1}^{n} \frac{|y_i - \hat{y}_i|}{|y_i|} 100\% \tag{4.10}$$

If the error is in the range of 5-7%, the model can be considered to be a good one.
The next measure should be correlation coefficient:

$$r_{x,y} = \frac{n \sum_{i=1}^{n} x_i, y_i - \sum_{i=1}^{n} x_i \sum_{i=1}^{n} y_i}{\sqrt{n \sum_{i=1}^{n} x_i^2 - \left(\sum_{i=1}^{n} x_i\right)^2} \cdot \sqrt{n \sum_{i=1}^{n} y_i^2 - \left(\sum_{i=1}^{n} y_i\right)^2}} \tag{4.11}$$

here x and y are two observed variables.

How to apply these measures to complex numbers.
In order to apply these statistics to complex variables, the complex number $z = a + ib$ should be represented in the so-called Euler notations $z = \sqrt{a^2 + b^2} \exp^{arctan(\frac{b}{a})}$. Then the information in the absolute part of the complex number $\sqrt{a^2 + b^2}$, can be encoded into the phase or angle part, $arctan\left(\frac{b}{a}\right)$. In order to calculate the quality of the model and apply the statistics, R^2 (see eq. 4.9) must be divided into R^2 for the abolute and phase parts.$R^2_{absolute}$ corresponds to the absolute part and

to R^2_{phase} corresponds to the phase part. The average $R^2_{Generalized} = \frac{R^2_{absolute}+R^2_{phase}}{2}$ is then taken. In the second method the error can be calculated using the same method as it was discussed in the Chapter 2 "Brief review of complex analysis". Thus the approximation error is equal to $E(y,\hat{y}) = (y-\hat{y})\overline{(y-\hat{y})}$. Many different error functions can be introduced. Each of them have their advantages and disadvantages which are not in the scope of the current chapter. One can see Gangal [27] for more error functions.

Important remark. Very often in case of multiple regression, R^2 automatically increases with increase in the new explanatory variables. However this does not guarantee an increase in the quality of the regression. In order to overcome this problem, one should use the corrected determination coefficient:

$$\tilde{R}^2 = 1 - \frac{n-1}{n-k-1}\frac{\sum_{i=1}^{n}(y_i-\hat{y})^2}{\sum_{i=1}^{n}(y_i-\bar{y})^2} \tag{4.12}$$

If the added explanatory variable does not give the needed effect for the dependent variable $\tilde{R}^2$ could decrease .
While analyzing the multiple linear regression, it is sometimes required to calculate not only R^2 but also the so-called coefficient of the partial correlation between y and x_i and not the rest of x to know the "pure" influence of the variable. This coefficient can be defined according to the equation below:

$$r_{y,x_i|x_1\ldots x_{i-1}..x_k} = -\frac{A_{yi}}{\sqrt{A_{yy}A_{ii}}} \tag{4.13}$$

where A_{yi} is the algebraic adjunction for the element r_{yx_i}. If two variables are considered the above presented equation can be rewritten as:

$$r_{y,x_1|x_2} = \frac{r_{y,x_1} - r_{y,x_2}r_{x_1,x_2}}{\sqrt{(1-r^2_{y,x_2})(1-r^2_{x_1,x_2})}}, r_{y,x_2|x_1} = \frac{r_{y,x_2} - r_{y,x_1}r_{x_2x_1}}{\sqrt{(1-r^2_{y,x_1})(1-r^2_{x_2,x_1})}} \tag{4.14}$$

where $r_{y,x_1}, r_{y,x_2}, r_{x_2,x_1}$ are the coefficients of the traditional (pair) correlation. There is also a connection between the partial correlation and determination:

$$r_{y,x_1|x_2} = \frac{R^2 - r^2_{y,x_2}}{1-r^2_{y,x_2}} \tag{4.15}$$

Average The average can be the simple average or the weighted average. The simple average is:

$$\bar{x} = \frac{\sum_{i=1}^{n} x_i}{n} \tag{4.16}$$

where x_i is the i^{th} value of the series, and n is the volume of series. If the frequencies vary, the weighted average should be used:

$$\bar{x} = \frac{\sum_{i=1}^{k} m_i \cdot x_i}{\sum_{i=1}^{k} m_i} \tag{4.17}$$

Dispersion. This value characterizes the volatility of the series:

$$D(X) = \frac{\sum_{i=1}^{n} (x_i - \bar{x})^2}{n} = \frac{\sum_{i=1}^{k} (x_i - \bar{x})^2 m_i}{\sum_{i=1}^{k} m_i} \tag{4.18}$$

Then the Root Mean Squared Deviation (further RMS) can be defined in the following manner:

$$\sigma(X) = \sqrt{D(X)} \tag{4.19}$$

The variation coefficient can be defined as:

$$V(X) = \frac{\sigma(X)}{x} \cdot 100\% \tag{4.20}$$

Everything discussed above is valid for the complex regression as well, the sole difference being that all procedures should be now done twice (until a better procedure is developed), for the absolute part and the phase part of the complex number.

4.2 Benchmarking Methods

4.2.1 Incorrect Benchmarking

Below are some examples of incorrect benchmarking. The most typical errors occur during the data preparation stage. The most common error is incorrect data filtering. It is known that all filters have border effects. Therefore let us denote the filter as $F(\cdot)$. Now the natural experiment setting would be taken the training data D_{TR}, adjust and apply the filter to this data $D(D_{TR})$ with filter parameters F_P and then the train the neural network. After the neural network is trained, the filter can be applied to the validation set $F(D_V, F_P)$ and the neural network is applied to D_V. If the quality of the model is satisfactory, the model should be trained with the D_V, where the starting point for Neural Network training (the weights) is taken from the training on the training set D_{TR}. The last step is to apply the filter to the test set D_T and to apply the Neural Network to the filtered test data. This is the most common mistake of many research papers and occurs with time series forecasting very often. The thing is that filters have side effects and cannot be so easily applied. Therefore the comparison of the solutions is just not possible in case filters have been applied differently.

4.2.2 Correct Benchmarking

It is impossible to discuss new issues in training and neural networks understanding without discussing model evaluation.
Any model can be evaluated according to several criteria. Two are cited below:

- The "cleverness" of the method. Many methods to compare cleverness exist. One the most advanced is IQ test. However, these methods always result in relative values. For example, if only model A underwent an IQ test, one cannot claim, that A is more clever than B until B has undergone the same test. The main point is that B can be compared only with respect to A or some other maximum possible result. The result can be interpreted with a number, for example A is 40% more clever than B.

- The "beauty" of the model. This measure is absolutely relevant and can be considered only by a committee of experts. This estimation can be ranked with numbers. The results should not depend on the personal opinion of one expert. Therefore, the bigger the committee, the better the evaluation.

One can find many parameters for model comparison. Some limits on evaluations are:
- Any measure should be numeric or converted into a numeric.
- Any model can be evaluated only with respect to other models.
The question arises what to do if there is only one model to evaluate? In this paper, a scoring system, typically used by economists for credit rating analysis (the so-called Altman model [10]) is recommended. The idea of the Altman model is to take several parameters (in our study it can be R^2, r , RMS, σ etc) and to construct a linear regression, giving the final score to the model. The parameters of the linear regression should be tuned on a training set of different model results. The details about Z-benchmarking can be found in the subsection below.
The history of benchmarking starts with attempts to compare model quality. It is obvious, that one can only compare methods 'ceteris paribus' (common conditions). Obviously, if one must compare the training speed for neural networks, and takes the Gauss-Newton and Gradient-Descent algorithms and tries to compare these two with 1.4 GHZ and 2.2 GHZ CPU the results will be different even if the methods give the same quality and speed. Therefore only one parameter should be varied at a time during the course of the experiment (here training algorithms, as well as dataset, model type, environment, and anything else which can influence the result will be fixed). Listed below are 10 parameters which obviously influence the final quality of the model (here "+" denotes, that the parameter was discussed in the majority of publications, "-" denotes the opposite case): type of model(+), training algorithm (+), number of training epochs

(+), division of dataset during training, test and validation sets (+) number of parameters for tuning (number of hidden units)(+), dataset (+), the computational environment (+-), hardware and the operational system (+-), the accuracy of computations (+-),etc. In case one only fixes one parameter, the experiment can not be repeated. Thus the reproducibility of the result can be lost. This is one of the main concerns in the Prechelt [45]. The problem here arises that there is no such settlement of an experiment on any PC, that all parameters except one are fixed. Therefore, the results, achieved on different machines cannot be considered or proved. Therefore, only to evaluate model quality, relative measures should be taken into account. The measure should be known to all scientists. In the current work, the author will divide the parameters above mentioned into two classes: internal (further *int*) and external ones (further *ext*).
The internal parameters of the experiment (CPU power, RAM volume, motherboard frequency, RAM frequency, operational system) are all dependent parameters on the PC being used, and training algorithm, number of hidden, number of epochs, accuracy of computations are all related to the model construction and training. The external parameters are needed to evaluate the model. They are the training, test and validation sets, and statistical measures for the forecast quality (to be discussed later) such as R^2, r, RMS, σ.
Many researchers in neural networks use well-known datasets in their papers, and then simply cite the dataset used. Sometimes more advanced information is available, such as the training algorithm used for training. Therefore, in the majority of papers, one can find several statistical coefficients and references to publicly available datasets. These datasets are quite old and do not cover all the advantages of these models. Some advantages, which arose in the past few decades should clearly be taken into account. The main problem faced by the scientific community is that scientific methods are far scientific benchmarking techniques. This gap makes it impossible to verify more advanced results then and makes benchmarking impossible. The aim of the current subsection is to find an autonomous method for correct benchmarking, so that researchers can easily test their models and can compare them with other models.
The whole NN community should use the same formulas, statistical measures, and external parameters for the same datasets. People introducing new datasets should provide as much information as possible about the datasets, but also select statistical measures for these datasets, so that others can compare their models with a "perfect" models.
Such a system has been created using PHP scripts, a database, containing datasets with descriptions, the results of all scientists, and Altman model coefficient model evaluation. The general idea is to have an one world database, which can perform simple calculations in two different regimes: blind and open. For each dataset some coefficients might me more important, e.g. for regression modeling onw

owuld prefer higher R^2 then smaller RMS. For this purpose an Altman model can be suggested to weight factors (different statistical coefficients), see Altman [10]. The blind regime does not show the Altman coefficients. The open regime shows the model coefficients. Scientists can then see what parameters make their work worse than that of some one else. If the author of the dataset hides the Altman coefficients, it is impossible to tell how the model lacks up the other model. Each case has advantages and disadvantages to be discussed later. The underlying model for the Z-score Altman model is the linear regression $Z = \sum_{i=1}^{n} \alpha_i C_i$ where n is the number of statistical measures, C_i is the value of the statistical measure like R^2 and α_i are the Altman coefficients, which deliver the importance of each statistical coefficient to the final estimation. Therefore, in order to evaluate the maximum possible Z-score, one should deliver the ideal forecast, approximation and classification to the Altman model, calculate all needed statistical measures and then use the number which will give the maximum possible score for this dataset. The same should be done for the average of the training set. These two numbers will serve as guiding lights for model evaluation. Any model used for the training set should not be worse than average. A detailed description of the Altman model construction for approximation and classification problems will be given in the subsection - Statistical Benchmarking.

4.2.3 Databases for statistical regression benchmarking

To overcome the difficulties of complex-valued benchmarking, an attempt will be made to create a database, able to store the results and data for the given dataset, and evaluate the results.
Technically, the database was created using Microsoft Access Tool. It consists of two interconnected tables, capable of storing the benchmarking results and author information. Then there is a web page written with PHP technology which makes SQL calls via ODBC driver to the SQL database.
The user web page consists of a login and a registration page. After the login a "Bench-It Now" (further BIN) page appears. First part of the page shows the login information. This information will be used to create labels in the database. External users can only see the login name of the author, but not his or her affiliations and real names. The second part consists of several subsections:

1. The dataset selection pop up menu gives the link to the needed dataset. After the dataset is selected, the user can see the results of previous researchers. Then the user can download the description file, the training set with a target and the test set without a target (absolute and angle parts listed separately).

   The absolute and angle parts of the complex numbers must be gained "by

hand" due to the fact that the automatic processing of complex numbers was not possible with the PHP on the web site (in the future this obstacle could be improved through the use of a special data parser). In order to obtain complex input, the absolute part of the number must be multiplied with the exponent which degree is angle part of the number $((absolutepart) \times exp^{i*(anglepart)})$

2. If the user wishes to add his or her own dataset to the system, then he or she should upload the description file, the training set plus the target file, the test set without the target file and the test set target in a separate file. After these four files have been uploaded, the dataset will appear in the list of datasets and will be available for other users. Note, that other users can download only three files out of four (the description file, training set with test set file and test set without the target file).

The last part of the system to be discussed is benchmarking itself. After the active dataset has been selected from the list, the user can upload his or her result (test set model output) into the system. The system will automatically calculate `RMS`,R^2,r,$\tilde{R}^2$ for the phase and angle parts respectively, and then will give the `BIN-score` , which is a linear combination of the previous values. For each dataset all user results are ranked with respecct to their `BIN-score`. The `BIN-Score`.
is described below. First data is normalized with respect to minimal and maximal values. This can be easily done with the next formula: $X_N = \frac{X-Min(X)}{Min(X)-Max(X)}$, where X is not normalized value, X_N is normalized value, $Min(X)$ and $Max(X)$ are minimal and maximal values of X.
The score is calculated through the use of statistical numbers according to one rule: the better the approximation, the higher its statistical significance, the higher the score. First we calculate the BIN for only the absolute parts of the network output and target:

$$\texttt{BIN} = \frac{1}{-\tilde{R}^2\left(y,\hat{y}\right) + \texttt{RMS}(y,\hat{y})/2 + 2} 100\% \tag{4.21}$$

Then we do the same for the angle parts of the neural network output and target. After the `BIN-score` has been found for both parts of the neural network output and target, they should be summed using the 0.75 and 0.25 coefficients, respectively.

$$BIN = \frac{0.75 * BIN_{abs} + 0.25 * BIN_{ang}}{2} \tag{4.22}$$

The benchmarking data base was created using the SQL engines and PHP programming language. One can find the database at **www.benchitnow.de**. The

manual uploaded on the database explains how to prepare the data and use the system.

5

Implementation and Applications

The following section contains the numerical results obtained from the networks simulation in application to different datasets. First the CVHCNN against the CVFFNN in application to the Logistics map and Lorenz chaotic system will be considered. Then CVHCNN will be applied to the real world problems, e.g. Transformer modeling, Stock Market modeling, Neurons Synchronization modeling and Binding effects modeling. All these results are based upon the application of the CVAIT toolbox implemented in MATLAB using an object oriented approach.

5.1 MATLAB Object Oriented Implementation

In order to perfom the experiments a MATLAB toolbox which implements all theory discussed above has been implemented. The toolbox has been written using the Object Oriented Programming. Toolbox contains three classes of type Layer, Network and Weights. The hierarchy of objects is presented at the fig.5.1 below. The main description of the network is contained in the `define-architecture.m` file. This file can be briefly described here in the thesis:

```
%%Network architecture definition
function [network_layer, weight,lag]=cvffnn()
disp('Creating architecture from architecture file');
%% NN PARAMETERS (MANDATORY)
number_of_inputs=3;
number_of_outputs=3;
number_of_hidden1=20;
number_of_hidden2=10;
```

This block describes the variables which will be used in the rest of the file: i.e. the definition of variables like number of inputs, number of hidden and number

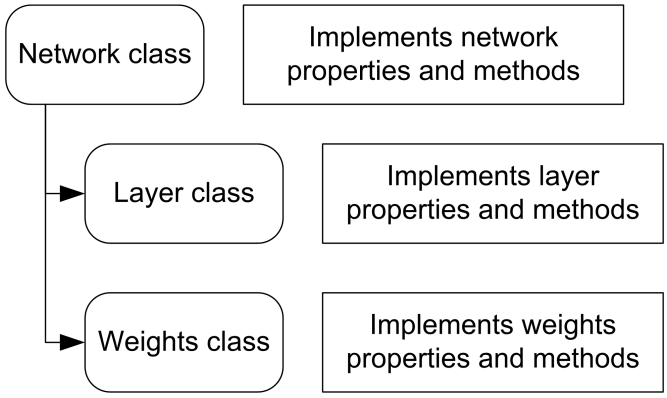

Figure 5.1: Structure of the implementation

of outputs. The next block describes the architecture: i.e. general description of the layer class. The philosophy of the description is the next. Each layer has ID, which must be unique. This ID recieves data through weights from the concrete senders and sends it via transition functions. Inputs have additional fields like data columns which whould be used as inputs and lag which tells about time sift of the data column if any. Output layer must have additional fields like error type, whether output layer should induce dE into the network during the backpropagation path and type of the output layer, which means what the layer dataset belongs to, namely training set, validation set or production set.

```
%% LAYERS PARAMETERS (MANDATORY) (you may use loops)
%IMPORTANT:: the structure of IDS MUST BE CONSISTENT 1 to n
%EXAMPLE FOR GENERAL LAYER (all possible parameters of the layer):
% network_layer{ID}=layer('layer_type','type',...
%         'recieves_from',ID,...
%         'connection_weigths',W_ID,...
%         'layer_size',n,...
%         'activation','sin',...
%         'io_data_columns',1:n,...
%         'io_lag',lag,...
%         'time_data_column',n+1,...
%         'injects_dE',true,...
%         'error_type','MSE',...
%         'output_type','TR');
lag=1;
```

The MATLAB code for the network definition for the training set is listed below.

```
%training set description
```

```
for k=1:6:100
    network_layer{k}=layer('layer_type','input',...
        'io_data_columns',1:3,...
        'io_lag',lag);
    lag=lag+1;
    network_layer{k+1}=layer('layer_type','bias');
    network_layer{k+2}=layer('layer_type','bias');
    network_layer{k+3}=layer('layer_type','hidden',...
        'recieves_from',[k;k+1],...
        'connection_weigths',[3;1],...
        'layer_size',number_of_hidden1,...
        'activation','tanh');
    network_layer{k+4}=layer('layer_type','hidden',...
        'recieves_from',[k+3;k+2],...
        'connection_weigths',[5;2],...
        'layer_size',number_of_hidden2,...
        'activation','tanh');

    network_layer{k+5}=layer('layer_type','output',...
        'recieves_from',k+4,...
        'connection_weigths',4,...
        'activation','tanhtanh',...
        'io_data_columns',4:6,...
        'io_lag',lag,...
        'injects_dE',true,...
        'error_type','MSE',...
        'output_type','TR');
end
k=k+6;
```

The MATLAB code for the network definition for the validation set is listed below.

```
%validation set description
for j=k:5:k+100

    network_layer{j}=layer('layer_type','bias');
    network_layer{j+1}=layer('layer_type','bias');

    network_layer{j+2}=layer('layer_type','hidden',...
        'recieves_from',[j-1,j],...
        'connection_weigths',[3;1],...
```

```
        'layer_size',number_of_hidden1,...
        'activation','tanh');
    network_layer{j+3}=layer('layer_type','hidden',...
        'recieves_from',[j+2,j+1],...
        'connection_weigths',[5;2],...
        'layer_size',number_of_hidden2,...
        'activation','tanh');

    network_layer{j+4}=layer('layer_type','output',...
        'recieves_from',j+3,...
        'connection_weigths',4,...
        'activation','tanhtanh',...
        'io_data_columns',4:6,...
        'io_lag',lag,...
        'injects_dE',false,...
        'error_type','MSE',...
        'output_type','V');
    lag=lag+1;
end
```

After the network is described one has to describe the weights. All weights must have unique ID. Each weight has to be sugned as "shared" or "nonshared" and the mask for training has to be defined. The mask contains ones and zeros for the weights which have to be modified during the training and not correspondingly. See the weight definition below:

```
%% SPECIAL WEIGHTS DESCRIPTION (MANDATORY),...
 IT IS POSSIBLE TO LOAD FROM THE DISK
    temp1=sprandn(number_of_hidden,...
    number_of_inputs,0.4)/10+...
        1i*sprandn(number_of_hidden,...
        number_of_inputs,0.4)/10;
    temp3=(sprandn(number_of_outputs,number_of_hidden,0.4))/10+...
        1i*(sprandn(number_of_outputs,number_of_hidden,0.4))/10;
    temp4=(randn(number_of_hidden,1))/10+...
        1i*(randn(number_of_hidden,1))/10;
    weight{1}=weights('values',temp1,'mask',...
    iszero(temp1),'shared',false);
    weight{3}=weights('values',temp3,'mask',...
    iszero(temp3),'shared',false);
    weight{4}=weights('values',temp4,'mask',...
    ones(size(temp4)),'shared',false);
```

```
%% END OF DEFINITION
```

By changing this code one can create any architecture. Currently a feedforward architecture is shown. A detailed explanation about all parameters and features is given in the Tutorial to the toolbox. The code shown to point out the complexity for the definition of the network architecture. The complete scheme of the system is presented at the Fig.5.2.

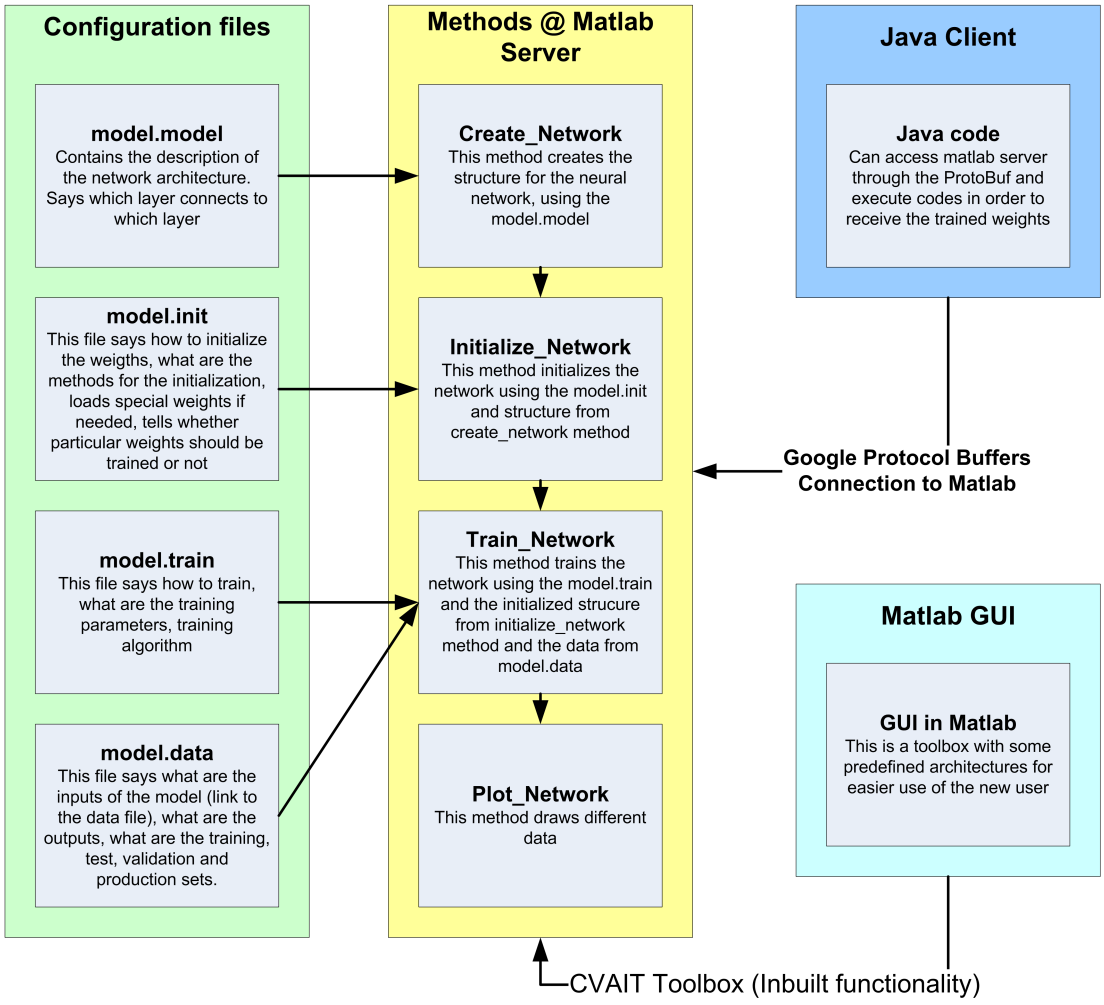

Figure 5.2: Complete system representation

5.2 Artificial Datasets Used to Test Models.

1. Logistic map

This function is a famous example of the chaotic data generation. This map

was very popular in 1976 paper by R. May, see [49]. Before this publication a discrete-time demographic model has been suggested by P.F. Verhulst (see [28]). The model itself can be presented with the following equation:

$$\begin{cases} x(t+1) = \lambda x(t)(1-x(t)) \\ \lambda > 0 \end{cases} \tag{5.1}$$

where λ is a positive constant sometimes known as the "biotic potential" gives the so-called logistic map. Complex values can be obtained by multiplying $x(t)$ by $e^{i \cdot sin(t)}$, where t – time. The development of the chaotic behavior of the logistic sequence as the parameter λ varies from approximately 3.5 to approximately 3.8 is characterized by a periodic phase interrupted by bursts of aperiodic behavior. For λ slightly less than 4 we are in the chaotic regime. Beyond $\lambda = 4$, the values eventually leave the interval $[0, 1]$ and diverge for almost all initial values. In the current thesis $\lambda = 3.9$ has been taken. In the pictures below 5.3-5.5 one can see the behavior of the mapping for different values of λ. Denote $r_x = abs(x)$.

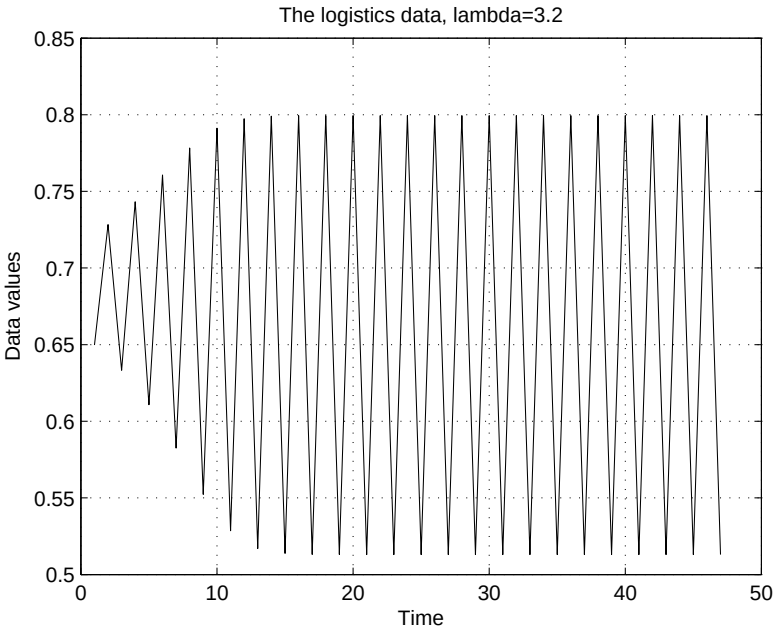

Figure 5.3: Logistic map sequence of r_x for $\lambda = 3.2$.

2. Lorenz Attractor

The Lorenz attractor (see Lorenz [23]) is an example of a nonlinear dynamic system corresponding to the long-term behavior of the Lorenz oscillator. The

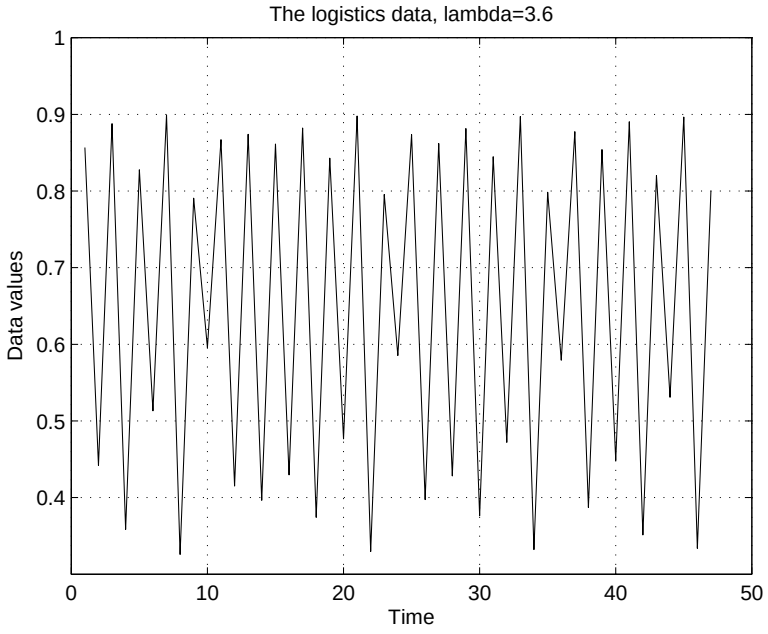

Figure 5.4: Logistic map sequence of r_x for $\lambda = 3.6$.

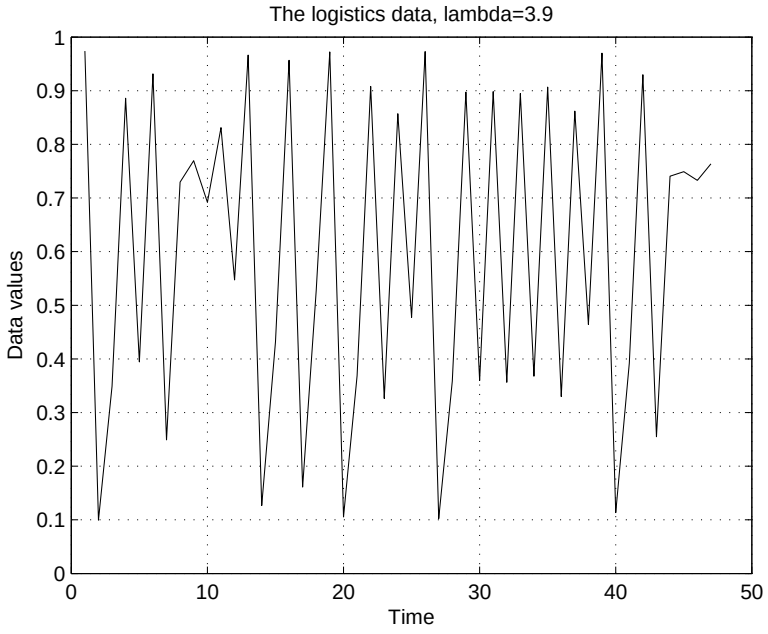

Figure 5.5: Logistic map sequence of r_x for $\lambda = 3.9$.

Lorenz oscillator is a three dimensional dynamical system:

$$\begin{cases} \dfrac{dx}{dt} = \sigma(y - x) \\ \dfrac{dy}{dt} = x(\rho - z) - y \\ \dfrac{dz}{dt} = xy - \beta z, \end{cases} \tag{5.2}$$

here x, y, z are variables, and the parameters were chosen to be:

$$\begin{cases} \beta = 8/3; \quad \rho = 28; \\ \sigma = 10; \quad h = 0.005, \end{cases}$$

where h – the step in Runge-Kutta 4th order scheme of differential equation system solving.

Complex values can be obtained by multiplying x, y, z by $e^{i \cdot sin(t)}$, where t – time.

In the Fig.5.6-5.8 we can see the Lorenz attractor for different values of step of the solver h. Denote $r_x = |x|, r_y = |y|, r_z = |z|$.

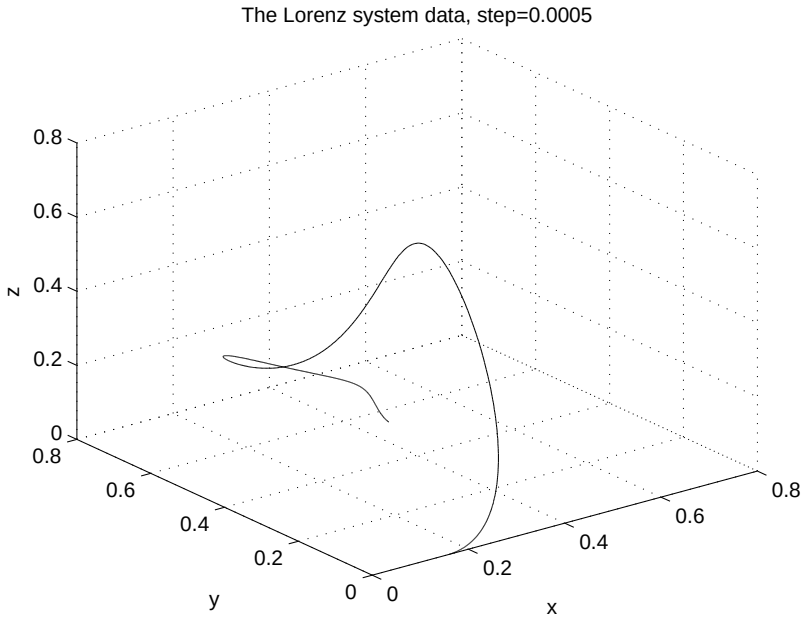

Figure 5.6: Lorenz attractor (r_x, r_y, r_z) for $h = 0.0005$.

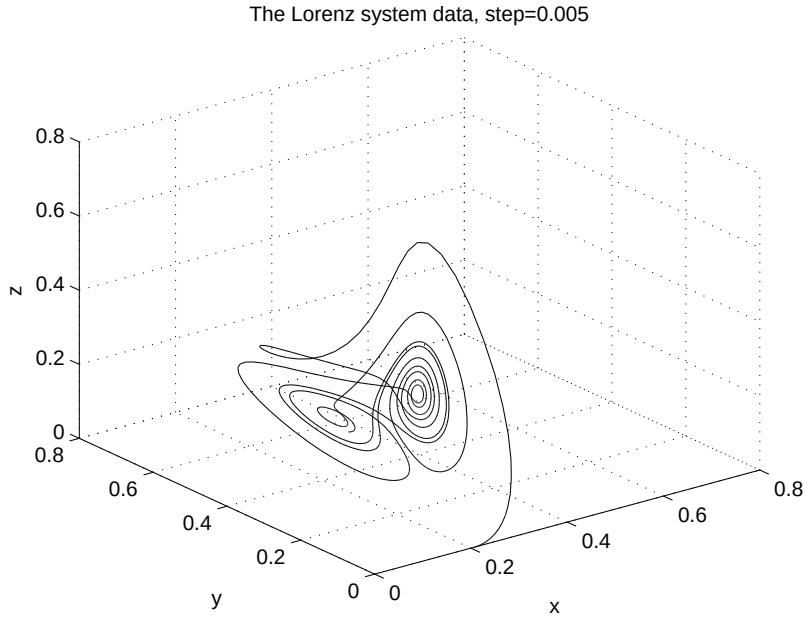

Figure 5.7: Lorenz attractor (r_x, r_y, r_z) for $h = 0.005$.

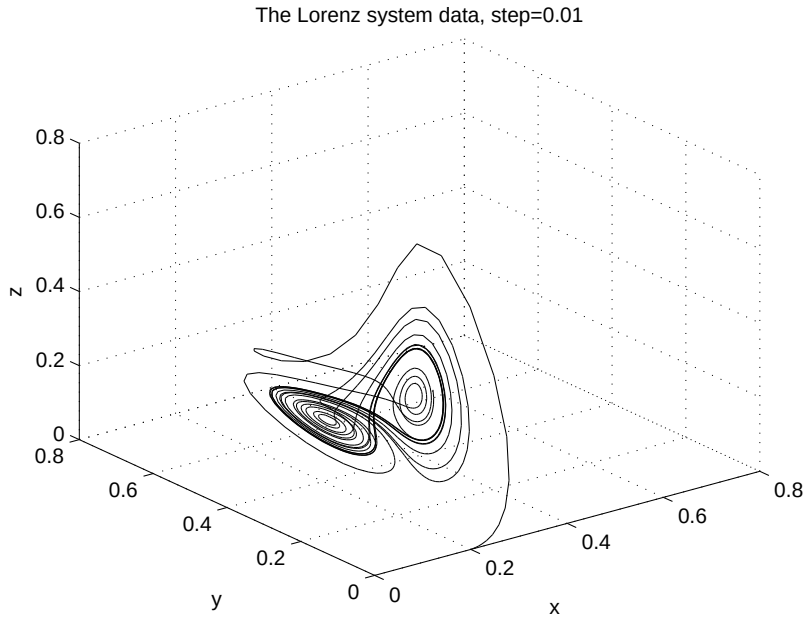

Figure 5.8: Lorenz attractor (r_x, r_y, r_z) for $h = 0.01$.

Further experiments are described for these two tasks. These two tasks are used to give the reader the impression regarding the performance of the complex-valued networks and to make the conclusion that CVNN are of the same performance as RVNN.

5.3 Modeling with Feedforward Neural Networks

In the following section the advantages and disadvantages of different approaches to the logistics map and Lorenz system modeling with the FFCVNN will be shown.
First the architecture have been used for the Logistics map modeling has to be presented. The first architecture is a well-known FFCVNN. It can be parametrized in the following way:

$$\begin{aligned} Architecture = &[\text{number of inputs},\\ &\text{number of neurons at hidden layers},\\ &\text{number of outputs}],\\ Nonlinearities = &[\text{activation functions at each layer}]. \end{aligned}$$

where activation functions:

$$\begin{aligned} 1: &\ f(z) = tanh(z_r) + i \cdot tanh(z_{im}),\\ 2: &\ f(z) = tanh(z),\\ 3: &\ f(z) = z,\\ 4: &\ f(z) = tanh(r)e^{i\varphi}, \end{aligned}$$

The described architecture is presented at the Fig.5.9 and its zoomed part at the Fig.5.10. One can see that at the validation set the outputs of the network as its inputs have been used. In order to see the structure of the network better, one can find a zoomed picture of it (see Fig.5.10). From this picture one can see that starting from the validation set the network uses its outputs as inputs.

Logistic map modeling results
Selected architecture can be expressed by the following set of parameters:

```
FFCVNN Parameters:
Number_epochs = 300;
Architecture = [3, 20, 3];
Nonlinearities=[linear, tanh, tanh];
Weights_Sparsity=0.4;
learning_rate = 0.05;
```

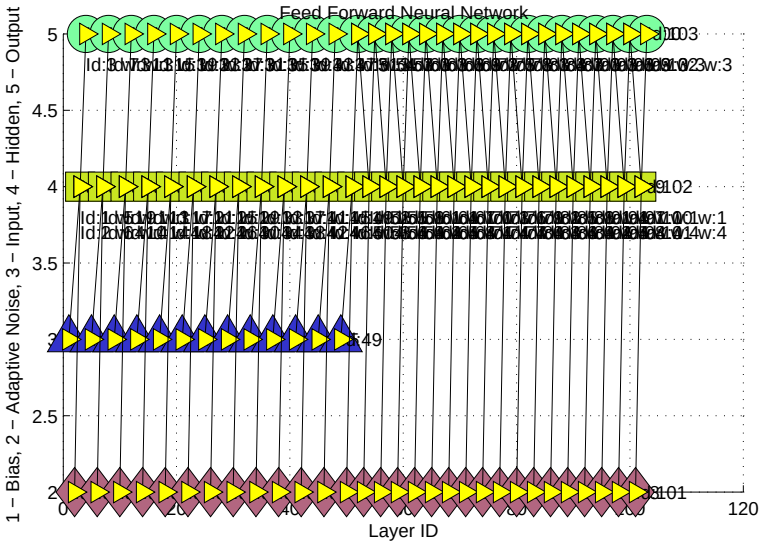

Figure 5.9: CVFFNN architecture

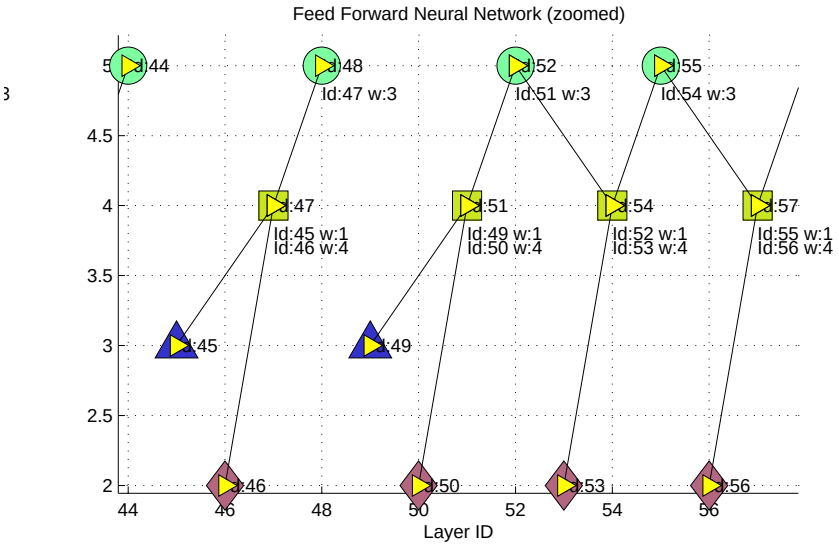

Figure 5.10: Zoomed region of the CVFFNN. From this picture one can see that for validation set the network uses its outputs as inputs.

Now the experiment starts. First the FFCVNN to the logistics map with $\lambda = 3.6$ has to be applied. In order to simplify the discussion a little bit, the Test set D_T in this task will not be used.Training set D_{TR} and Validation set D_V are in the focus of the current research. For the Logistics map $D_{TR} = 200$ patterns and $D_V = 100$ patterns. The network uses three previous values of the logistics map as inputs $t-2, t-1, t$ and produces three outputs $t-1, t, t+1$. These outputs are to be used as inputs for the validation set. Each experiment will be conducted with 4 statistical measures for the absolute part, angle part, real part and imaginary part of the complex-valued output. One can see the explanation of all 4 coefficents in the Benchmarking chapter. There is only one coefficient introduced here, which is a relative standard deviation: Relative Standart Deviation:

$$Std = \left(\sqrt{\frac{1}{N}\left(\hat{y}_i - \bar{\hat{y}}\right)^2} - \sqrt{\frac{1}{N}(y_i - \bar{y})^2} \right) \Bigg/ \sqrt{\frac{1}{N}(y_i - \bar{y})^2} \tag{5.3}$$

where N is the total number of patterns. Now, the results of numerical experiments follow.

First on should start with neural network training. The error decays in a stable way as it should do, see Fig. 5.11.

Therefore training has finished smoothly and one can proceed with the checks of the network for the training set and for the validation set. First the training set results are to considered and analyzed. One can see the results for the training set below. One can analyze the data by looking at the real-imaginary parts of the complex-valued numbers. To estimate the quality of the training we also can look at the absolute-phase parts of the complex numbers. As one can see the resutls at the training set are quite promising. Absolute part as well as angle parts fit perfectly and therefore we can proceed with the Validation set. The results for the validation set are presented below. Statistics on D_{TR} and D_V is presented in the Tables 5.1-5.2 below. The results at the validation set are quite good (according to the personal authors opinion). The forecast is stable and can be used for more than 10 points ahead. Obtained results are quite reasonable due to several reasons. One of them is that according to the Logistics map equation 5.1 new value of the system depends on the nonlinear combination of the previos value. Thus FFCVNN is an ideal tool to model such mapping. There is no need in more sophisticated recurrent models. Further one has to show what happens in case such network is applied to a recurrent problem, which is Lorenz problem.

One can see that statistics for the validation set is rather poor, but this happens because of high forecasting horizons. Therefore the results are to be inspected visually by the figures provided. In case one looks at the figures he

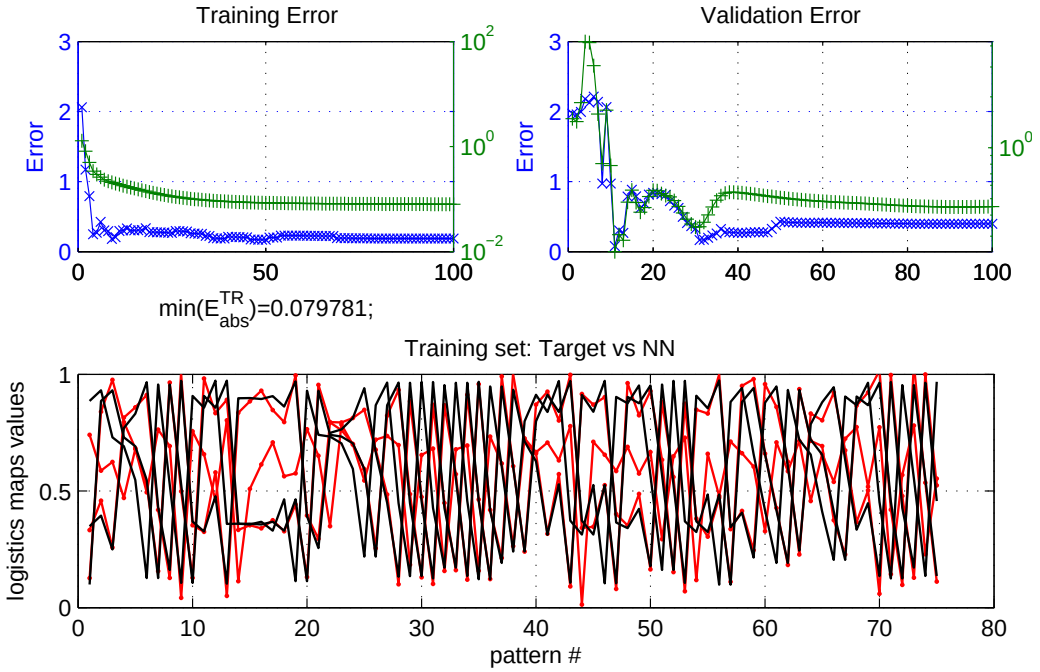

Figure 5.11: Training error decay for the logistics map modeling. One can see two curves at the error plots. These curves correspond to the absolute and angle errors respectively.

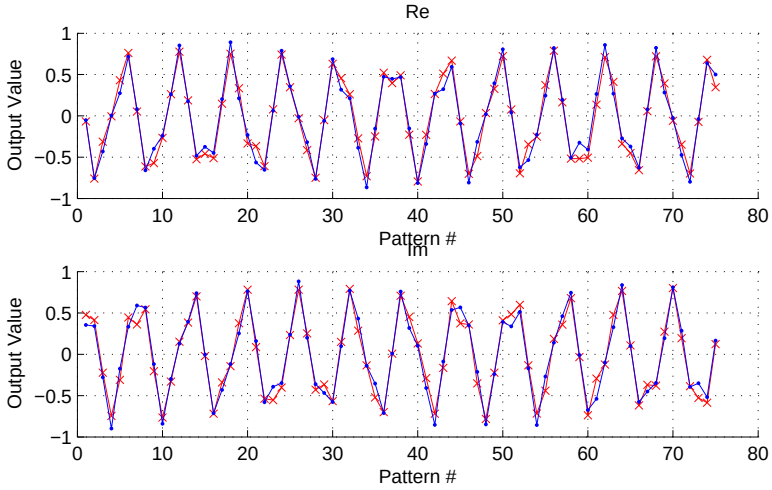

Figure 5.12: Training set results for the Real vs Imaginary parts of the observation (x marked line) against expectation (dotted line).

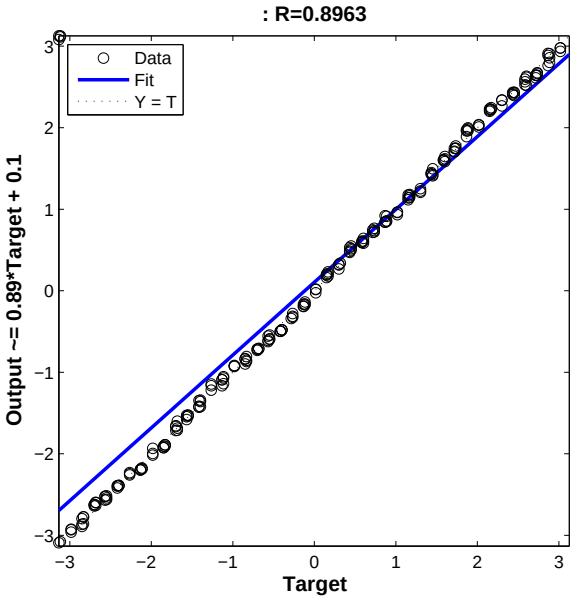

Figure 5.13: Regression plot for the absolute part of the expectation vs observation for the training set. Here target - expectation and output - observation. R - correlation coefficient.

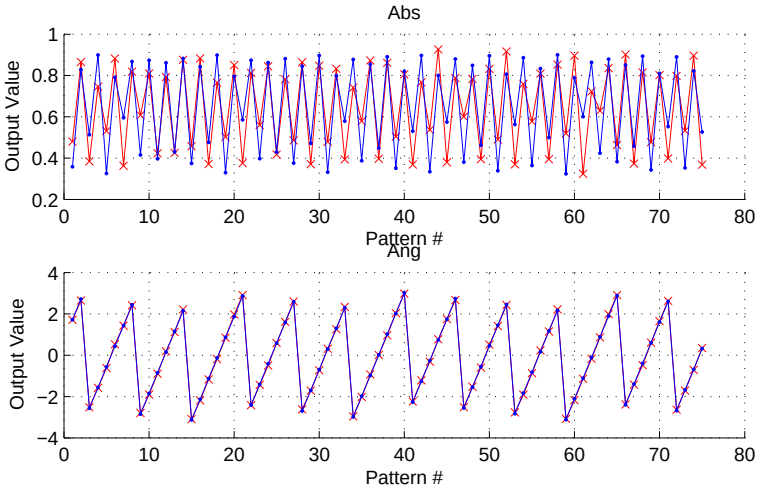

Figure 5.14: Training set results for the absolute part vs angle part of the expectation vs observation

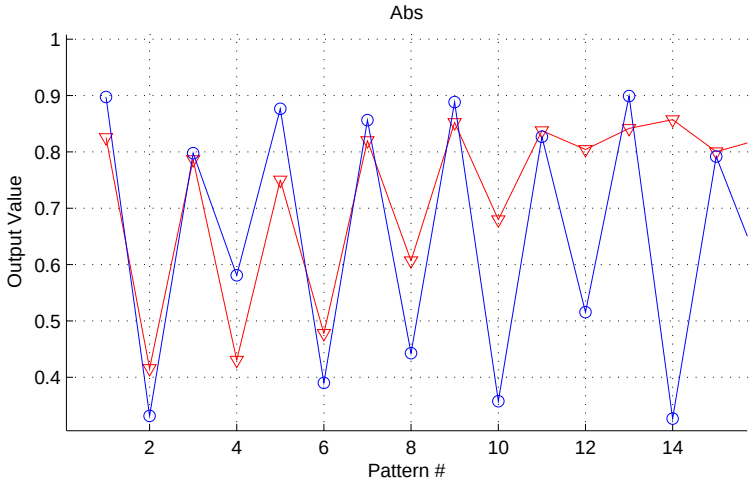

Figure 5.15: Plot for the validations set for the absolute parts of the expectations (line with circles) vs observations (line with triangles)

Table 5.1: Integrated table with the results concerning the Logistic map $\lambda = 3.6$

$h = 0.005$	Architecture type	FFCVNN	FFCVNN
Coefficient	Data Set	D_{TR}	D_V
R^2	Abs/Ang	0.66/0.99	$< 0/ < 0$
	Re/Im	0.99/0.99	$< 0/ < 0$
r	Abs/Ang	0.98/0.84	0.21/0.18
	Re/Im	0.98/0.98	0.12/0.08
Std	Abs/Ang	-0.09/0.00	-0.43/0.007
	Re/Im	-0.01/-0.01	0.24/0.25
RMS	Abs/Ang	0.01/0.002	0.09/5.48
	Re/Im	0.008/0.008	0.51/0.56

Table 5.2: Integrated table with the results concerning the Logistic map $\lambda = 3.9$

$h = 0.005$	Architecture type	FFCVNN	FFCVNN
Coefficient	Data Set	D_{TR}	D_V
R^2	Abs/Ang	0.23/0.38	-0.22/0.69
	Re/Im	0.82/0.84	0.73/0.68
r	Abs/Ang	0.51/0.71	0.24/0.84
	Re/Im	0.90/0.92	0.88/0.86
Std	Abs/Ang	-0.33/0.01	-0.48/-0.00
	Re/Im	-0.09/-0.07	0.12/0.10
RMS	Abs/Ang	0.06/2.0	0.09/0.94
	Re/Im	0.03/0.03	0.05/0.07

will see that this network can do a stable prediction for nearly 10 points ahead or even more. See Fig.5.16.

Lorenz system modeling results

Further the the Lorenz system is under the consideration, see Eq.5.2 (solved for the step $h = 0.01$). This is a chaotic system which strongly depends on its previous values. The experiment setup will try to imitate this dependence by giving the previous values of the Lorenz system to the neural network. Such approach follows the Takens theorem (in the follwoing the theorem is modified and uses the conclusions out of the theorem itself), which says that one can reconstruct a phase attractor of the chaotic system inside the weights of the neural network by using the historical values of the system. Further the neural network itself will be described. Selected architecture can be expressed with the following set of parameters:

```
FFCVNN Parameters:
Number_epochs = 300;
Architecture = [3, 20, 3];
Nonlinearities=[linear, tanh, tanh];
Weights_Sparsity=0.4;
learning_rate = 0.05;
```

The error decay for the particular problem is presented at the Fig. 5.18. One can see the results at the D_{TR} - training set at the Fig. 5.19 (real and imaginary parts of the network output). Now it becomes evident, that FFCVNN cannot solve this problem. In order to make sure it is the case one has to apply this network to the validation set D_V.

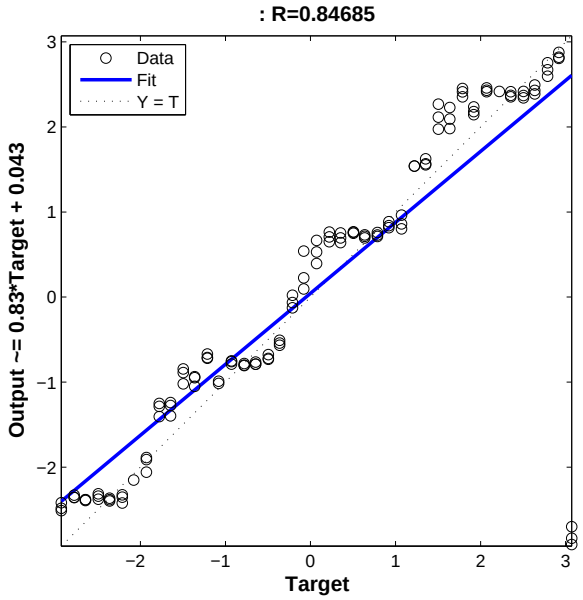

Figure 5.16: Regression plot for the validation set of the logistics map modeling with $\lambda = 3.9$.

As one can see at the Fig. 5.20 and Fig. 5.21 the network cannot predict the next values for the Lorenz system. It was not able to predict the Lorenz system even for several points. It means that a more sophisticated neural network of the recurrent type is to be used.The next section considers the mentioned case. One can see the table with the Lorenz modeling statistical results below. The tables are presented for different Lorenz of different solver step 0.005 and 0.01 respectively.

5.4 Modeling with Complex Valued Open Recurrent Neural Network

This architecture can be represented in the following way:

$$\begin{aligned} Architecture = &[\text{number of inputs},\\ &\text{number of states},\\ &\text{number of neurons at state layer},\\ &\text{number of outputs}],\\ Nonlinearitis = &[\text{input activation, state activation, output activation}]. \end{aligned}$$

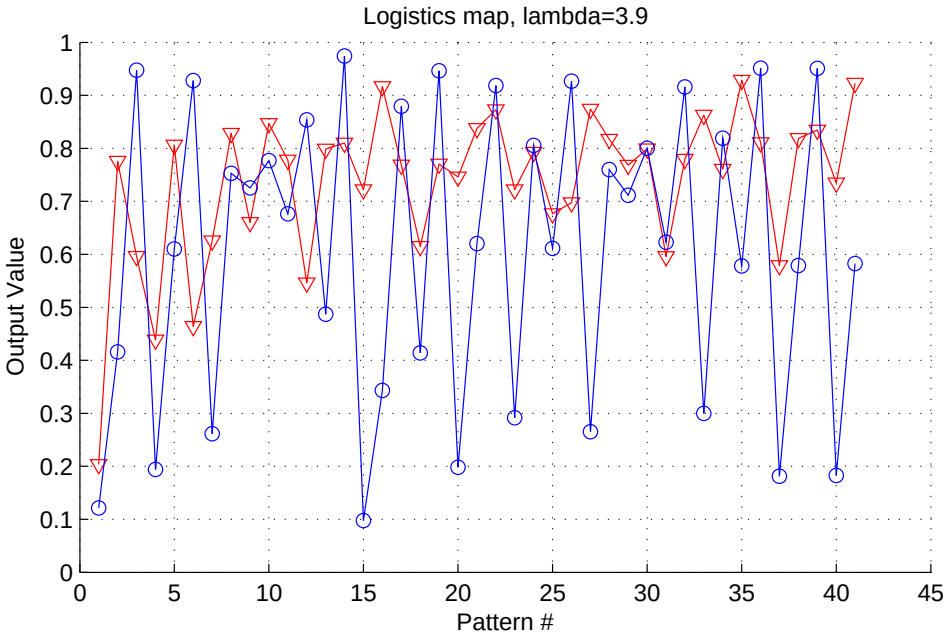

Figure 5.17: Absolute part of the expectation vs observation for the logistics map modeling with $\lambda = 3.9$. Line with circles was made by expectations, triangles line - are observations.

Table 5.3: Integrated table with the results concerning the Lorenz task$h = 0.005$

$h = 0.005$	Architecture type	FFCVNN	FFCVNN
Coefficient	Data Set	D_{TR}	D_V
R^2	Abs/Ang	0.79/0.73	$< 0/ < 0$
	Re/Im	0.75/0.68	$< 0/ < 0$
r	Abs/Ang	0.96/0.87	-0.02/0.02
	Re/Im	0.94/0.86	0/-0.02
Std	Abs/Ang	-0.40/0.05	0.85/0.77
	Re/Im	-0.42/0.05	0.32/1.15
RMS	Abs/Ang	0.002/0.02	0.01/1.17
	Re/Im	0.003/0.001	0.04/0.027

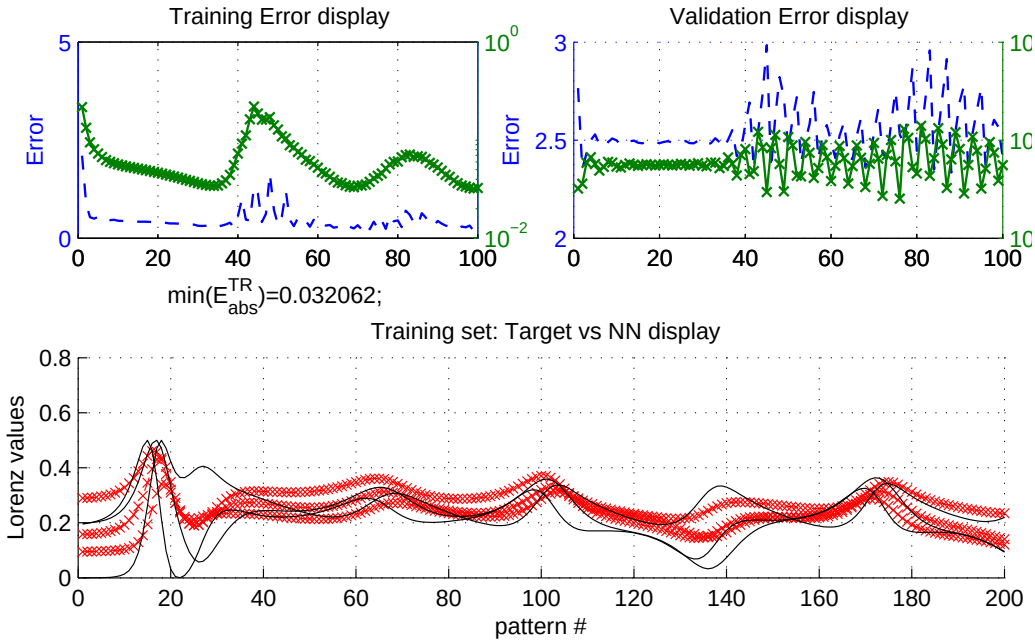

Figure 5.18: Error decay for the Lorenz problem modeling with the FFCVNN. One can see two lines in the error decay pictures. One (solid) corresponds to the absolue part of the error, another (dotted) corresponds to the angle of the error.

Table 5.4: Integrated table with the results concerning the Lorenz task$h = 0.01$

$h = 0.005$	Architecture type	FFCVNN	FFCVNN
Coefficient	Data Set	D_{TR}	D_V
R^2	Abs/Ang	$0.53/< 0$	$< 0/ < 0$
	Re/Im	$0.24/0.44$	$< 0/ < 0$
r	Abs/Ang	0.75/0.90	-0.05/-0.07
	Re/Im	0.96/0.91	0.38/-0.24
Std	Abs/Ang	-0.42/-0.32	-0.78/-0.99
	Re/Im	-0.29/-0.41	-0.91/-0.93
RMS	Abs/Ang	0.003/0.34	0.004/10
	Re/Im	0.01/0.003	0.006/0.07

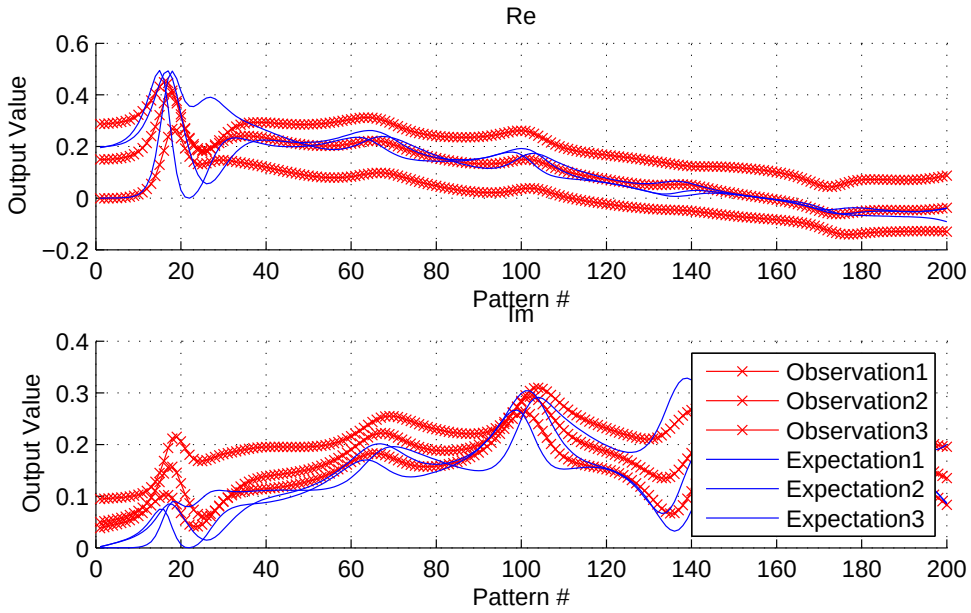

Figure 5.19: Real and Imaginary parts of the observations and expectations (x,y and z coordinates). Note that Lorenz system values have been put into the absolute part of the complex number. The phase part contained linear time.

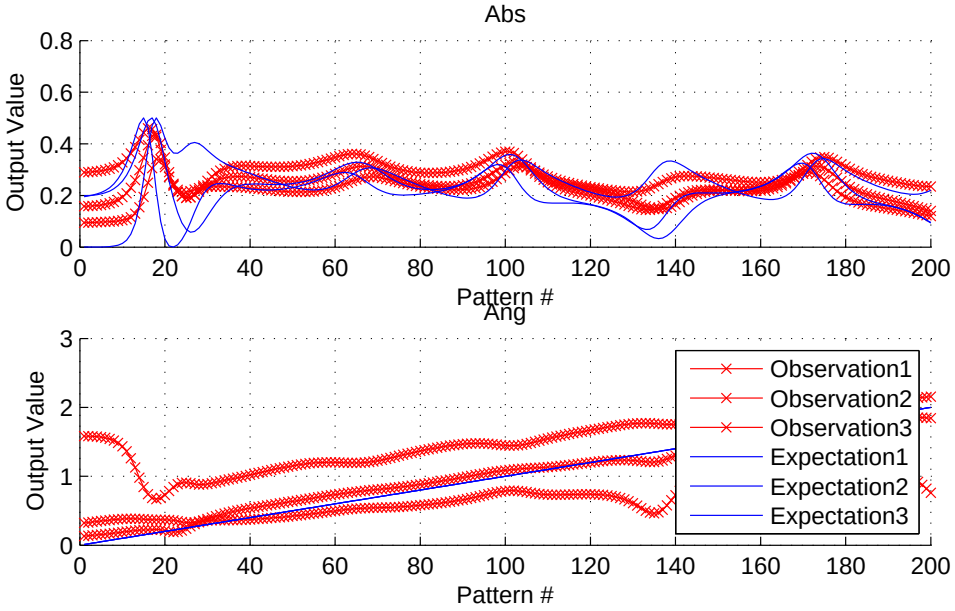

Figure 5.20: Absolute and phase parts of the expectations against observations for the training set.

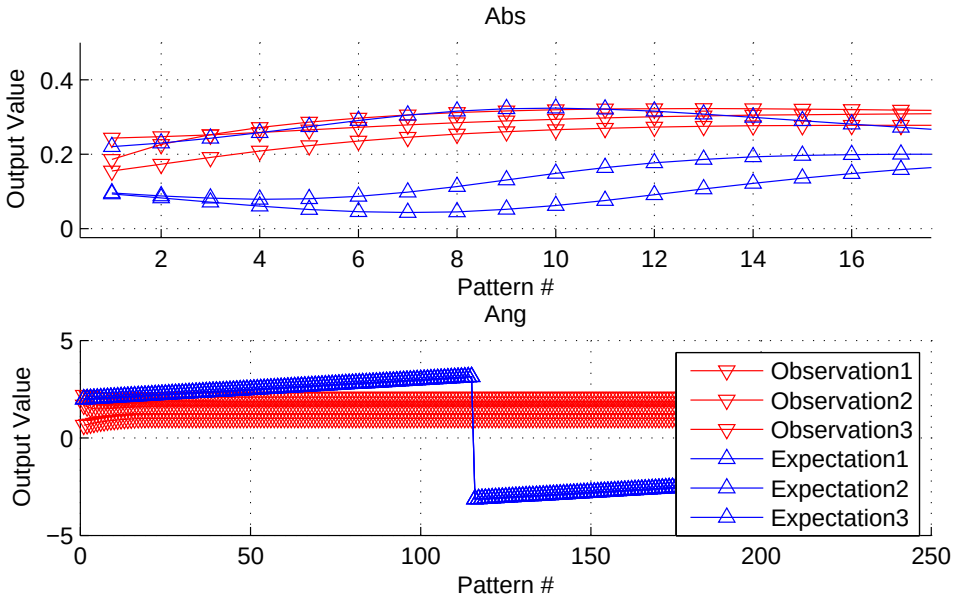

Figure 5.21: Absolute and phase parts of the expectation vs observation for the validation set.

where activation functions:

$$
\begin{aligned}
1: &\ f(z) = tanh(z_r) + i \cdot tanh(z_{im}), \\
2: &\ f(z) = tanh(z), \\
3: &\ f(z) = z, \\
4: &\ f(z) = tanh(r)e^{i\varphi},
\end{aligned}
$$

At the Fig.5.22. One can also see a zoomed region for the $CVHCNN$ at the Fig. 5.23. It is clear from the picture, that teacher forcing is not used for the validation set. Now the point where the data is to be discussed comes.

Logistic map.

Experiment 1.

In this experiment 10 recurrence states with 10 hidden neurons at each state have nbeen used. Activation function was selected to be *tanh* and *tanhtanh*. Learning rate is 0.05. Number of inputs is 3, number of outputs is 3. The set of parameters can be described in the following way:

```
Number_epochs = 500;
Architecture =[3,200,10,3];
Nonlinearity =[linear,tanh,tanhtanh];
```

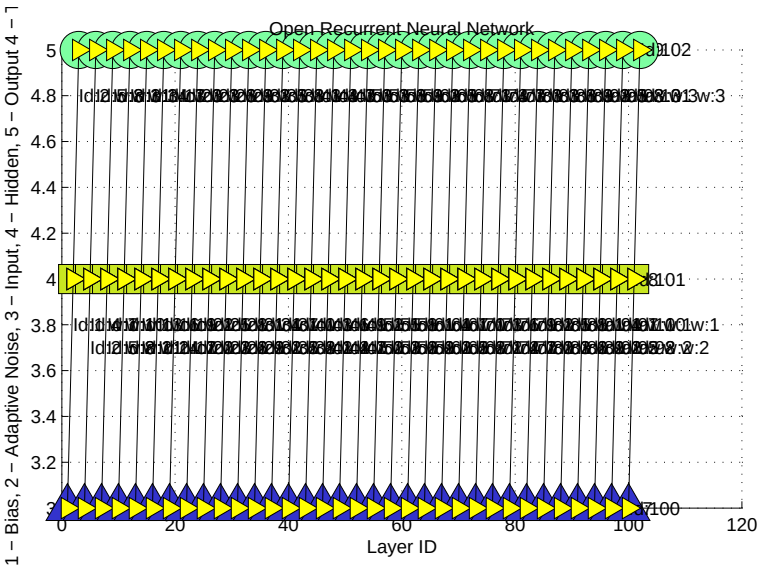

Figure 5.22: CVORNN network visualization

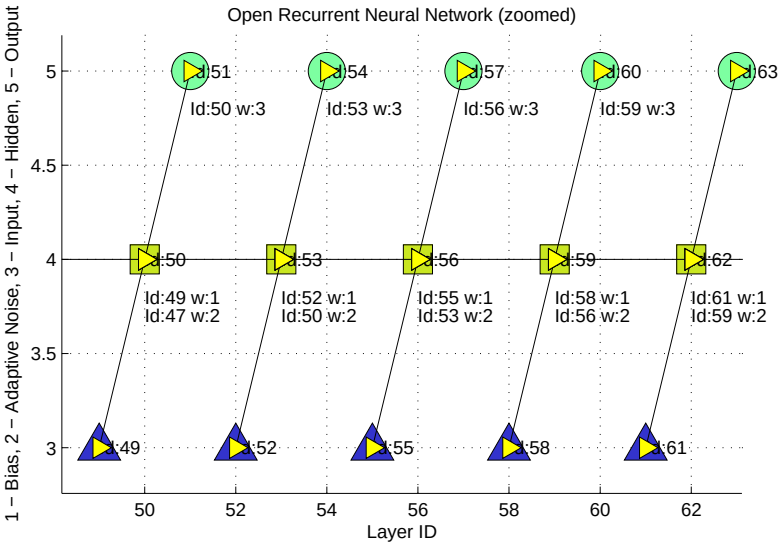

Figure 5.23: Zoomed part of the CVORNN network visualization

```
Weight_Sparsity=0.4;
learning_rate = 0.05;
```

Training set results.
After the training has been performed and all necessary checks were done one can see the picture for the error convergence and training error. The network was trained using the training set data D_{TR}. Since training has been successful

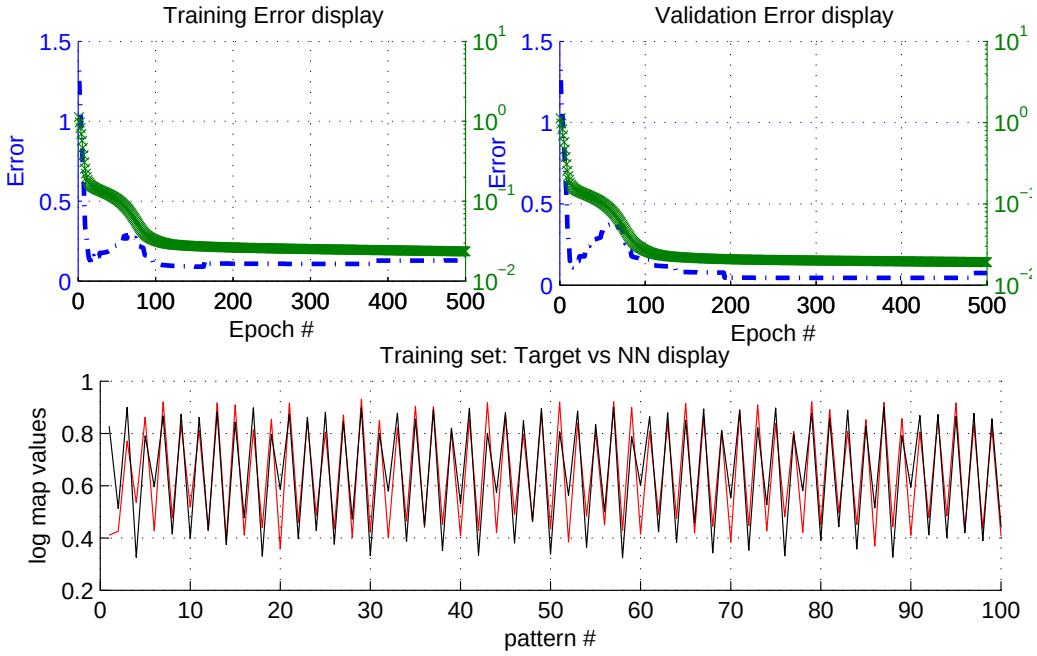

Figure 5.24: Training set error decay at the training and validation sets. Angle (dotted line) vs Absolute error (line marked with x). The approximation results for the absolute part of the expectation vs observation for the training set can be seen.

and the error converged to small values we can now perform model testing and forecast with the trained network for some values ahead. The results for the test set D_T are presented below.
Test set results. First the statistics for the test set has to be calculated. For the test set R^2 and RMS are acceptable (according to the authors private opinion). To see the calues check the tabular 5.5. After one has checked that the test set statistics is very good, he can now perform the production set and check whether the network can realy predict the next values of the logistics map. Here the exclusion is to be done and test set (D_T) will be performed.
Test set results. In order to create the test set the predicted values are to be

Table 5.5: Test set statistics for the RCVNN, logistics map.

RMS for the test set		R^2 for the test set	
absolute	angle	absolute	angle
0.00	0.00	0.99	0.99

used iteratively instead of the inputs (target values). Thus one can see whether the network can really predict. For this purpose one can use the following data - D_T. The results are presented at the Fig.5.25-5.26 and in the Table 5.6.

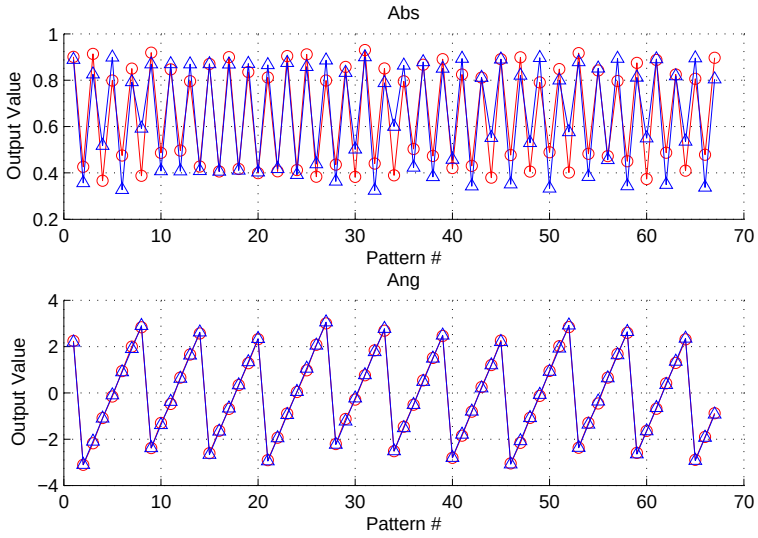

Figure 5.25: Real and Imaginary parts of the expectations (circles) vs observations (triangles).

Measures R^2, RMS are the following, see Table 5.6. One can see that the statistics is perfect for the complete test set, which means that the selected network is great in modeling the Logistics map.

Lorenz task

Experiment 1. In this experiment 200 recurrence states with 20 hidden neurons at each state have been used. Activation function was selected to be *tanh*. Learning rate is 0.05. Number of inputs is 3, number of outputs is 3. The set of parameters can be described in the following way:

```
Number_epochs = 500;
Architecture =[3,200,6,3];
```

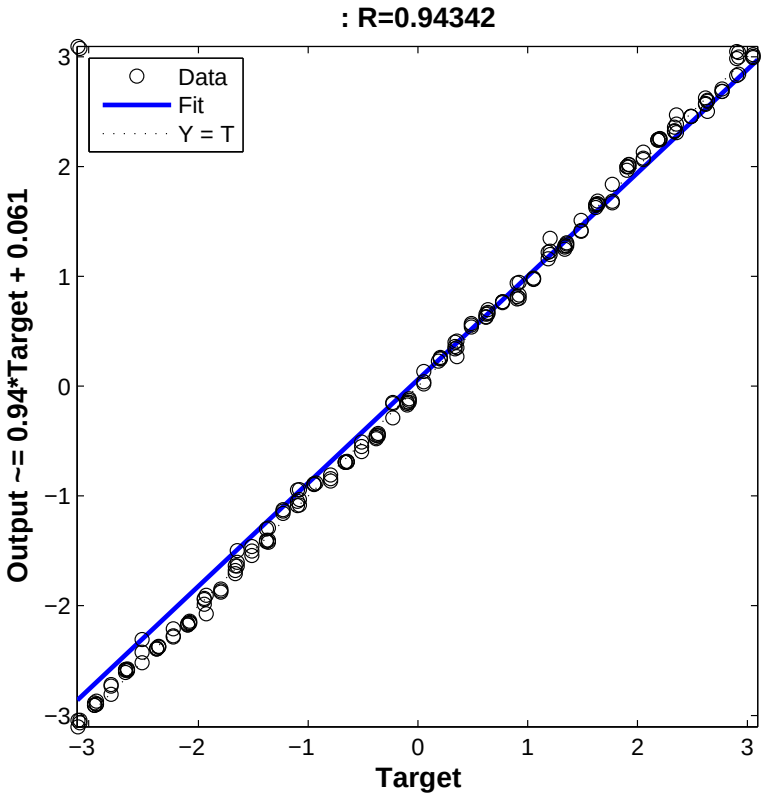

Figure 5.26: Absolute and phase parts of the expectations (circles) vs observations (triangles) for the test set.

Table 5.6: Statistical results of the RCVNN for the test set, logistics map.

RMS for the test set		R^2 for the test set	
absolute	angle	absolute	angle
0.00	0.00	0.99	0.99

```
Nonlinearity =[linear,tanh,tanhtanh];
Weight_Sparsity=0.3;
learning_rate = 0.05;
```

textbfTraining set results. The network has been trained for 500 epochs and the error convergence was smooth during the training. The final error is less than $10^{-}3$ which means we can perform the models tests for the test set D_T.
The error convergence is presented at the Fig. 5.27. The results of the validation

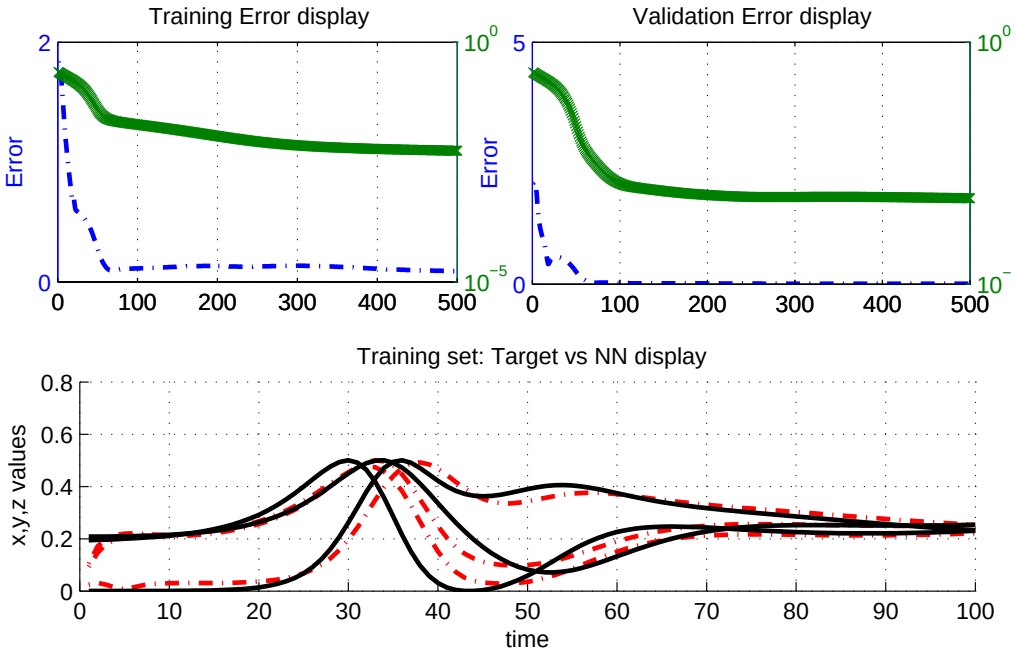

Figure 5.27: Training error decay - absolute (line marked with x) vs angle error (dotted line) for training and validation sets. Below one can see absolute part of the expectation vs obervation for the training set.

set are quite promising, all statistics is close to its best limits (see Table 5.7, which means that the network can explain up to the 90% of the data behavior. Thus one can start the real testing of the network which is test set.
Test set results. After the model applicability at the validation set have been cheked one has to produce the real test which is application of the RCVNN to the test set data D_P. The results are presented at the Fig. 5.28-5.29 and in the Table 5.8 The values of R^2 and RMS for the forecast are poor as the forecast horizon is too large. The negative R^2 values arise because of the smooth behavior of the system (desired outputs are close to their average values, which make R^2 negative despite network outputs). For other transition functions results are not

Table 5.7: Statistical results of the RCVNN for the validation set, Lorenz problem.

RMS for the validation set			R^2 for the validation set		
output #	absolute	angle	output #	absolute	angle
x 1	0.0002	0.0002	x1	0.9770	0.9992
y 2	0.0001	0.0001	y 2	0.9870	0.9998
z 3	0.0001	0.0001	z 3	0.9887	0.9998

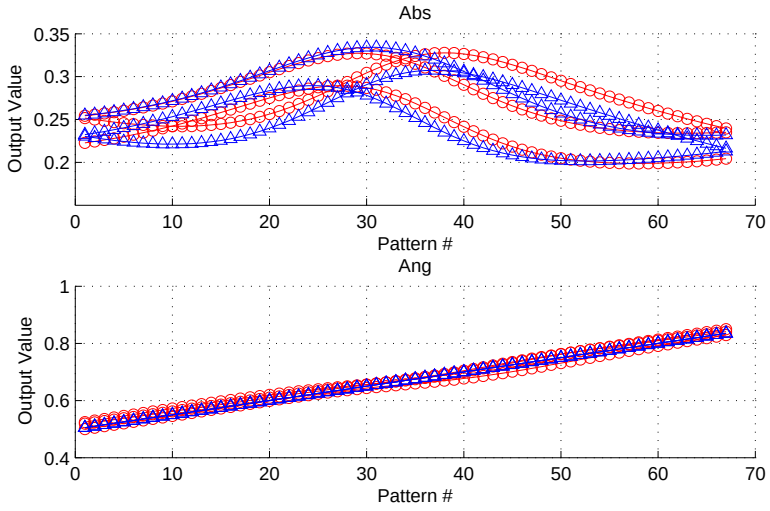

Figure 5.28: Predictions for Lorenz task for the CVORNN. One can see the absolute part of the expectation (lines with circles) against observation (lines with triangles) as well as phase part for the same series.

Table 5.8: Statistical results of the RCVNN for the test set for the Lorenz system.

RMS for the test set			R^2 for the test set		
output #	absolute	angle	output #	absolute	angle
output 1	0.0005	0.0.0004	output 1	0.98	0.99
output 2	0.0005	0.0012	output 2	0.99	0.99
output 3	0.0002	0.0026	output 3	0.98	0.99

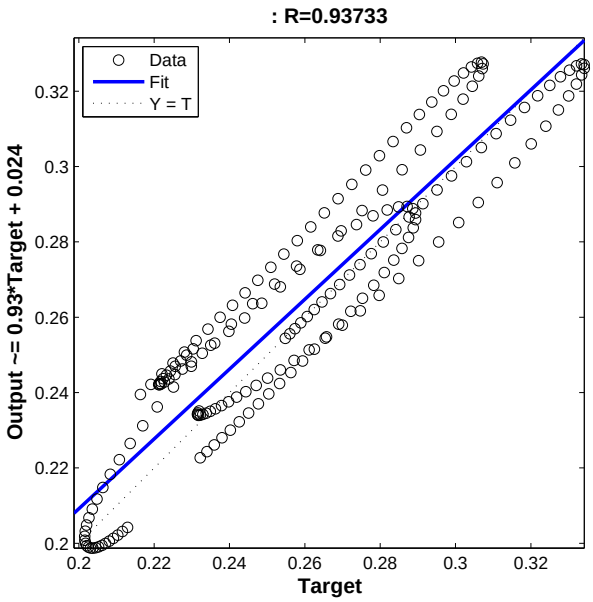

Figure 5.29: Regression plot for the absolute part of the expectation against observation.

very different. Therefore one can see that CVORNN is a universal network for the given problems. One should note that the noise has not been used to make the learning more complicated, therefore one can argue that the systems are to simple without the noise in them. The contrargument is that the task was to show the general applicability of the models for some datasets and how the networks behave. Further neural networks will be applied to a real world problems which obviously have all data artefacts.

5.5 Modeling with Historical Consistent Complex-Valued Neural Network

HCCVRNN architecture can be represented in the following way:

$$\begin{aligned} Architecture = &[\text{number of states}, \\ &\text{number of neurons at state layer}, \\ &\text{number of outputs}], \\ Nonlinearity = &[\text{activation function}]. \end{aligned}$$

where possible activation functions are:

$$
\begin{aligned}
1: &\ f(z) = tanh(z_r) + i \cdot tanh(z_{im}), \\
2: &\ f(z) = tanh(z), \\
3: &\ f(z) = z, \\
4: &\ f(z) = tanh(r)e^{i\varphi},
\end{aligned}
$$

At the Fig.5.30. One can also see a zoomed region for the CVHCNN at the Fig. 5.31. It is clear from the picture, that teacher forcing is not used for the validation set. Now comes the point where the data is to be considered.

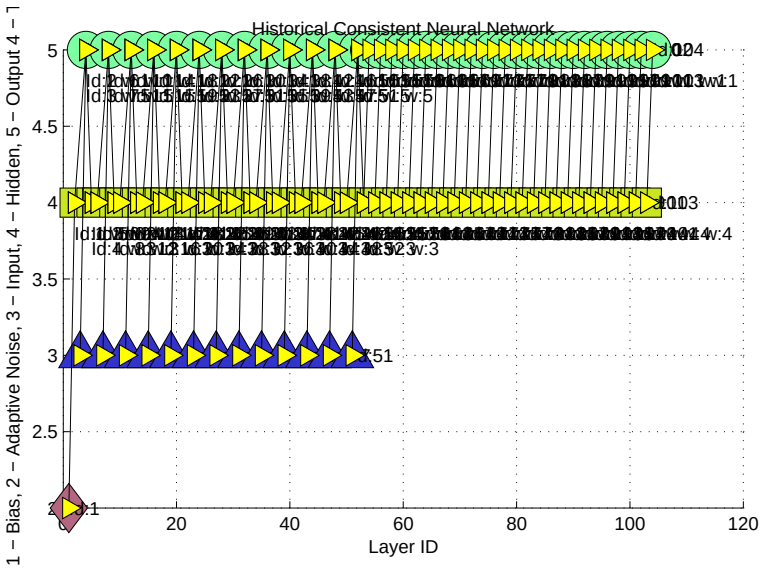

Figure 5.30: HCNN network visualization

Logistic map modeling results

Experiment 1. In this experiment the HCVNN is to be applied to the logistic map data with the following parameters:

```
Number_epochs = 1000;
Architecture =[200, 50,3];
Nonlinearity =[tanh];
Weight_Sparsity: 0.2;
learning_rate = 0.05;
```

Training set. One can see the results of the network at the training set D_{TR} at the Fig. 5.32. The error convergence is smooth and the final error is relatively

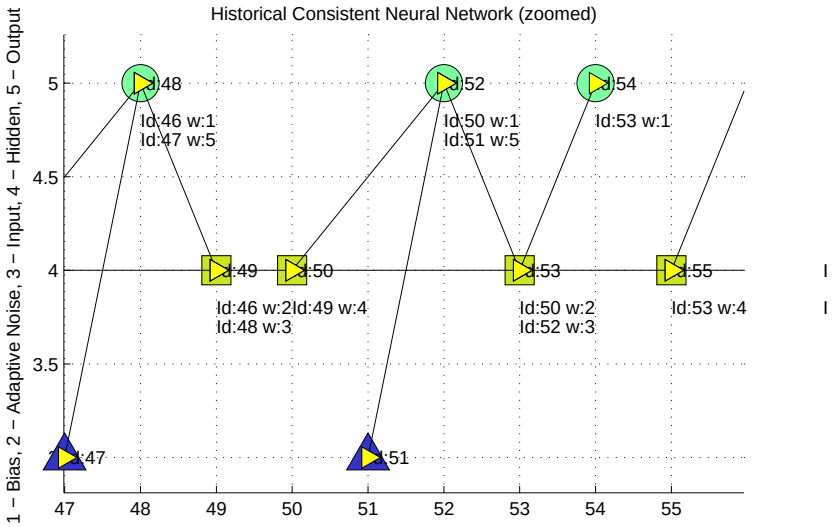

Figure 5.31: Zoomed part of the HCNN network visualization

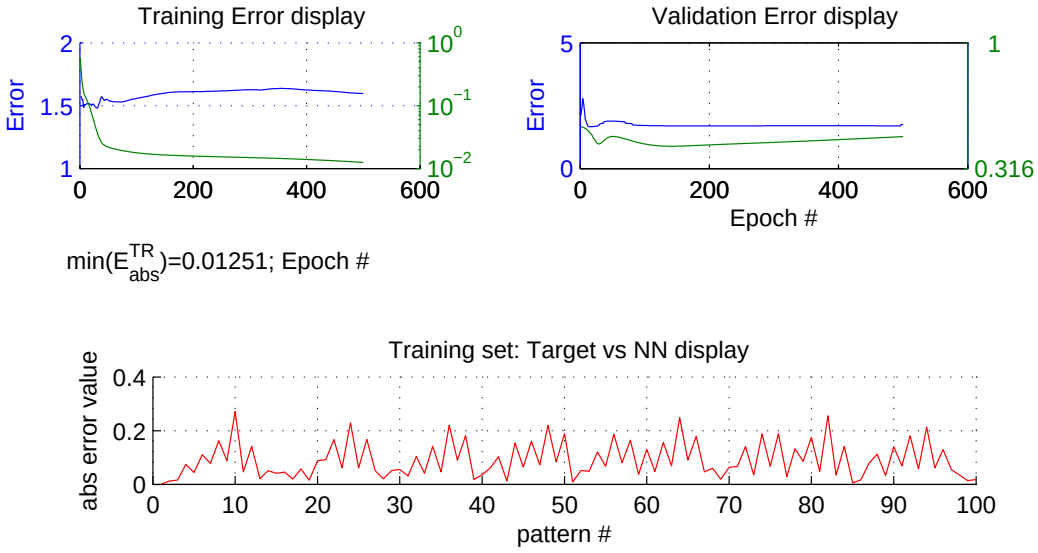

Figure 5.32: Training error decay for the logistics map example. One can see absolute and angle errors for both training and validation datasets. Below one can see the target error for HCNN.

small, see Fig. 5.32. Since the error converged during the training, one has to calculate the statistics at the D_V. Measures R^2, RMS are presented in the Table 5.9. After the statistics at the validation set D_V has been calculated, the

Table 5.9: Statistical results for the validation set, logistic map problem.

RMS for the validation set		R^2 for the validation set	
absolute	angle	absolute	angle
0.01	0.00	0.92	0.99

HCVNN has to be applied to the test set D_T. **Test set.** One can see the results of the network at the Fig. 5.33 and in the Table 5.10

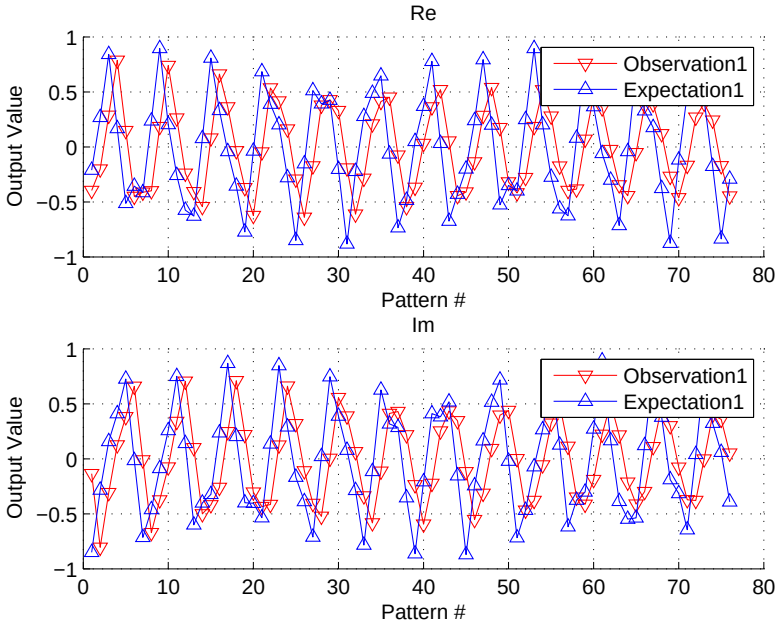

Figure 5.33: Real and Imaginary parts of expectations vs observations for the test set

Such poor results for the logistics map can be explained by some points. First of all, this network is perfect for modeling of the closed dynamical systems, which develop in an autonomous way, while for the logistics map it is driven by its previous values. There is no long memory in this system, therefore CVORN appeared to be better than HCVNN. In any case the prediction for one step was

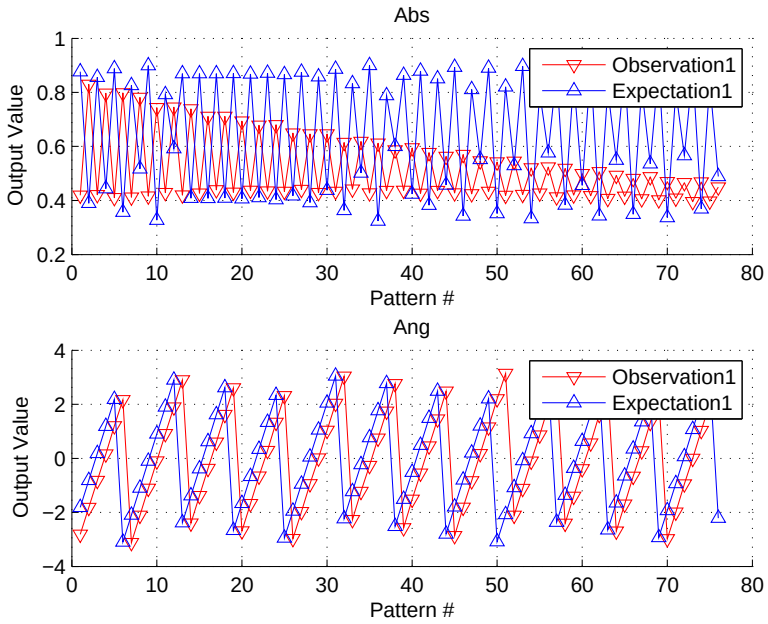

Figure 5.34: Absolute and angle parts of the expectations vs observations at the test set

Table 5.10: Statistical results for the test set for the logistic map.

RMS for the test set		R^2 for the test set	
absolute	angle	absolute	angle
0.12	5.6	< 0	< 0

still possible and was conducted with a very good statistics.
Now one has to apply a very interesting trick which is possible with this type of network. Since the prediction for nearly 100 steps ahead are available and predictions are always delaying for one step (this can bee seen from the phase part of the predictions). One can shift the obtained forecast in time by one step back. It is possible to do, since it is known in advance to which time the forecast has to be related, HCVNN gives the time of the forecast in the phase part of it. This is unique property of this neural network. One can see the updated results at the Fig.5.35 and in the Table 5.11 below.

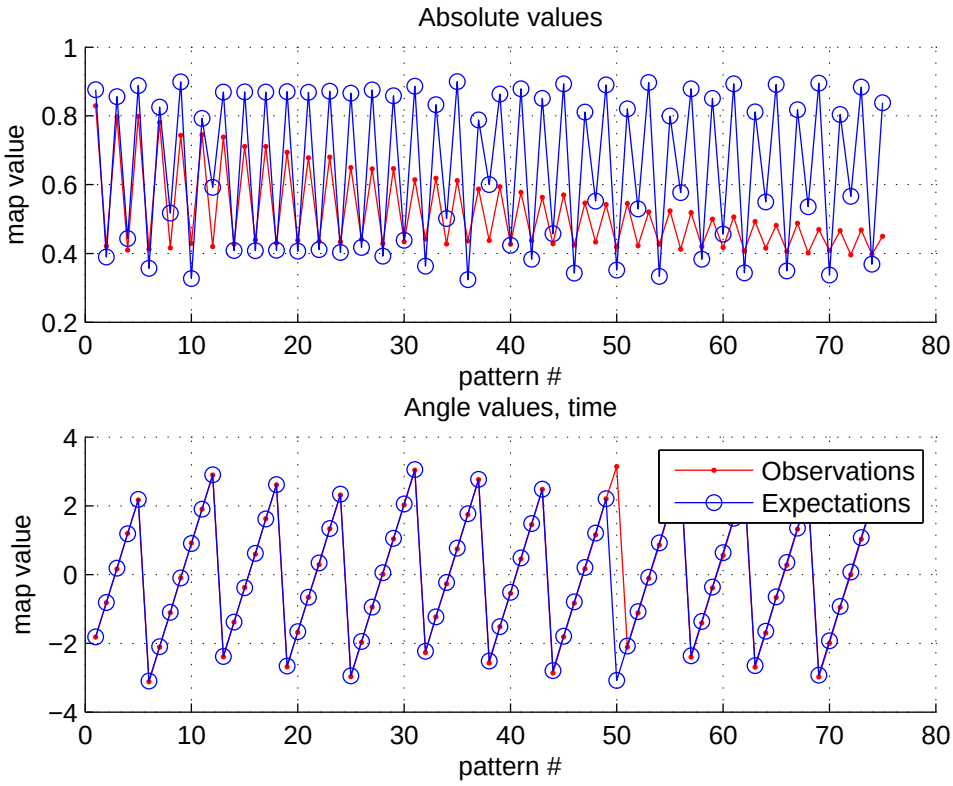

Figure 5.35: Absolute and angle parts of the shifted observations vs expectations, production set.

This approach is unqie in a way and has been called Time Teacher Forcing (TTF). One cannot do TTF with any other neural networks, only with CVHCNN. Time forcing is a very important feature of particulary this neural network. From this experiment one can see clearly, that network has learned the dynamics of the logistics map. This is a unique result of the thesis which is called historical consistent complex-valued time teacher forcing neural network (HCCVTTFNN)!

Table 5.11: Statistical results for the Production Set, Logistics Map problem.

RMS for the test set		R^2 for the test set	
absolute	angle	absolute	angle
0.03	0.00	0.4	1

Lorenz system modeling results
Now HCVNN has to be applied to a more interesting example, which is generated by a dynamical system, namely Lorenz system.
Experiment 1. The parameters of the network used for the Lorenz modeling are shown below:

```
Number_epochs = 1000;
Architecture =[200, 50,3];
Nonlinearity =[tanh];
Weight_Sparsity: 0.2;
learning_rate = 0.05;
```

Here the step in Runge-Kutta 4th order scheme is $h = 0.005$. As it was done in the previous section one has to start the neural network training using the D_{TR}. The results of the training are shown below. **Training set.** As one can see from the Fig. 5.36 the error decay is smooth and converges to a small number. Thus we can perform the Test Set calculations and see, how good the statistics is at D_T.

Validation set. The statistics for the validation set 1 step iterative predictions is calculated and presented in the Table 5.12. In this case the values of R^2 and RMS for the validation set are close to their good values and then test set calculations are to be performed, which were done below. **Test set.** The Test

Table 5.12: Validation set results for the HCVNN for the Lorenz system.

RMS for the test set			R^2 for the test set		
output #	absolute	angle	output #	absolute	angle
output 1	0.0005	0.0002	output 1	0.9773	0.9993
output 2	0.0011	0.0001	output 2	0.9022	0.9995
output 3	0.0008	0.0001	output 3	0.8802	0.9996

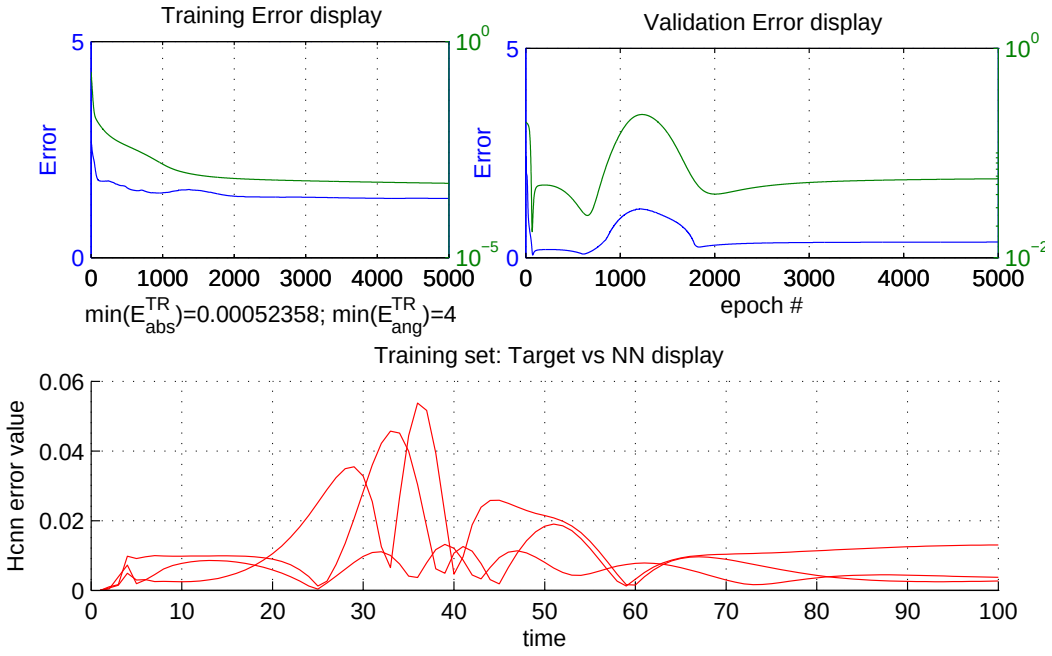

Figure 5.36: Training error decay. One can see absolute and phase values for training and validation errors. Below one can see the error of the HCVNN at the training set.

set D_T calculations were performed and the results can be seen at the Fig.5.37 and in the Table 5.13.

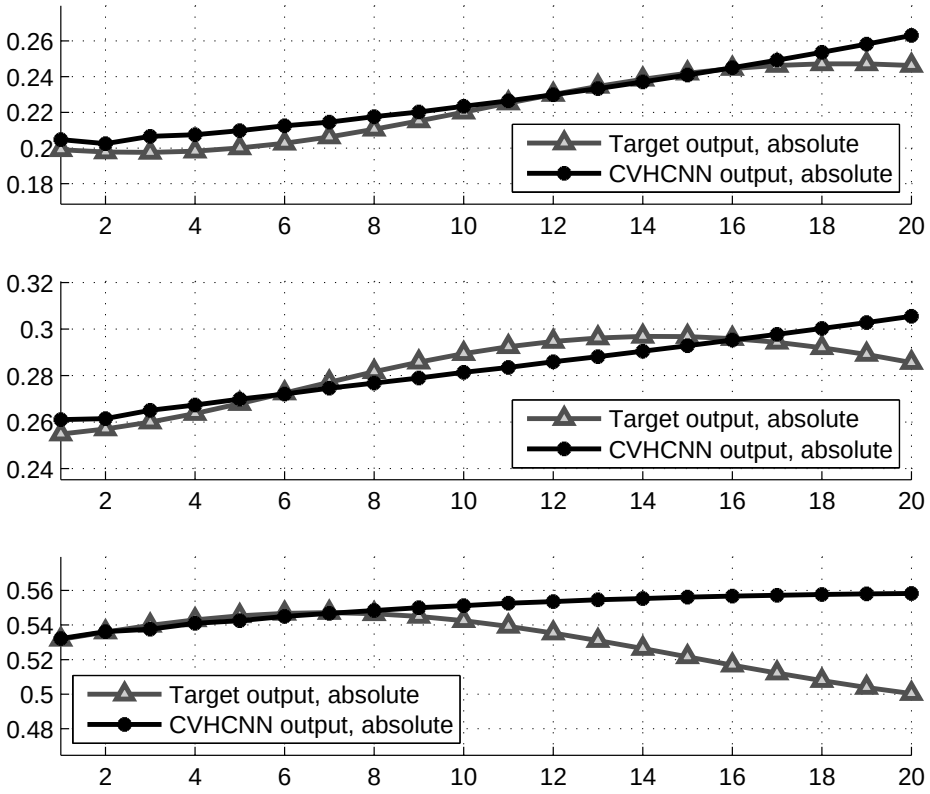

Figure 5.37: Forecast for the absolute part for 20 steps of the Lorenz system.

The values of R^2 and RMS for the forecast are quite good. The negative R^2 values arise because of the smooth behavior of the system (desired outputs are close to their average values, which make R^2 negative despite the network outputs).

Since the visual inspection of the test set results shows, that HCVNN can perform better, it has been decided to apply it once more, but with a different topology.

Experiment 2.
In this experiment the neural network will be trained for a significant anount of epochs. One should take into account, that CVHCNN requires a lot of training

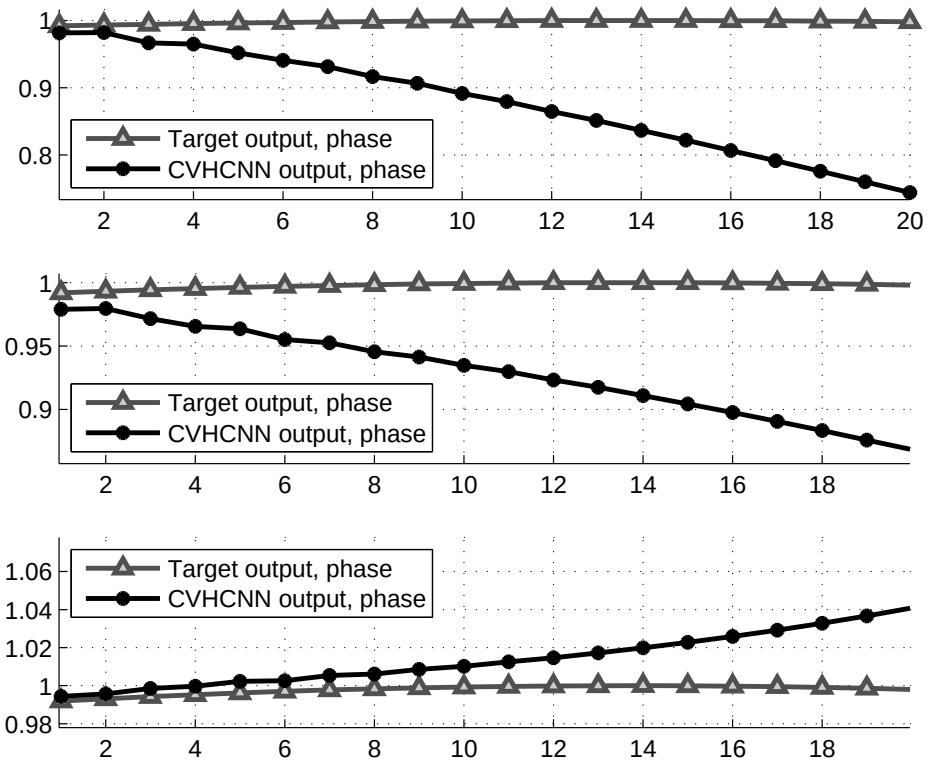

Figure 5.38: Forecast for the phase part for 20 steps of the Lorenz system.

Table 5.13: Production Set results for the HCVNN, Lorenz problem.

RMS for the test set			R^2 for the test set		
output #	absolute	angle	output #	absolute	angle
output 1	0.0001	0.0201	output 1	0.8615	< 0
output 2	0.0001	0.0059	output 2	0.7019	< 0
output 3	0.0007	0.0004	output 3	< 0	< 0

due to a teacher forcing training algorithm. The new neural network parameters for this experiment were selected in the next way:

```
Number_epochs = 10000;
Architecture =[1000, 50,3];
Nonlinearity =[tanh];
Weight_Sparsity: 0.05;
learning_rate = 0.03;
```

Each experiment (here experiment 2) was repeated ten times and the results were averaged except the best and the worst result.
Error function for all three outputs is decreasing exponentially and is close to 10^{-4}.
The test set results are presented for the absolute and angle parts of the CVHCNN output. One can see a set of training error decay stages at the Figures below.

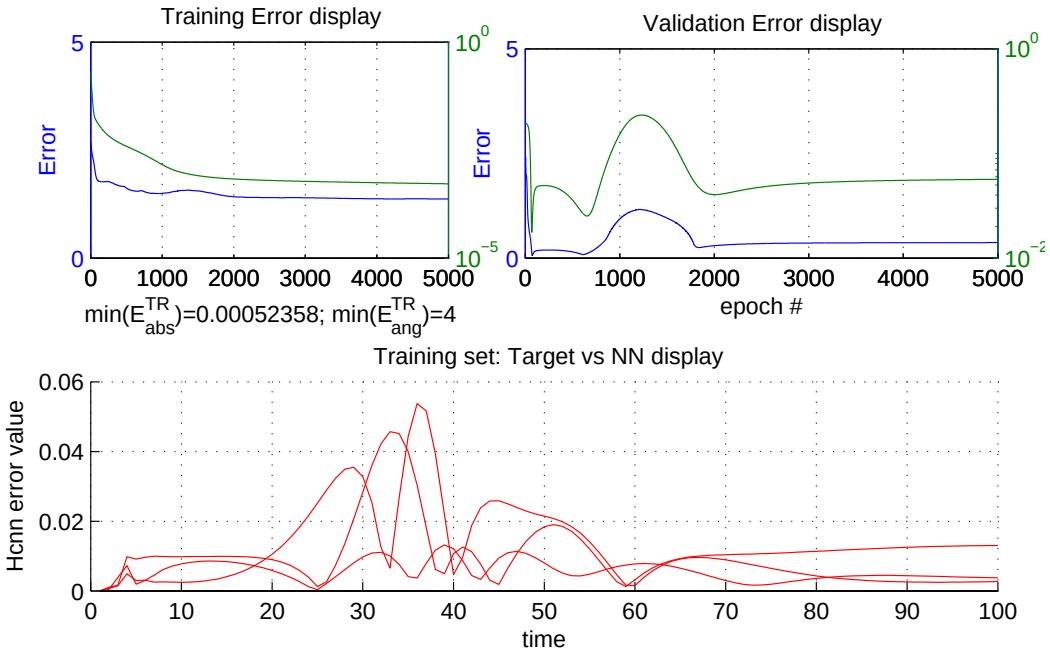

Figure 5.39: Training error for the first 1000 epochs. One can see absolute and phase values for training and validation errors. Below one can see the error of the HCVNN at the training set.

One can see that error is fluctuating very much. This occures due to the pattern by pattern training which is stochastic. One can see the stochasticity of the error.

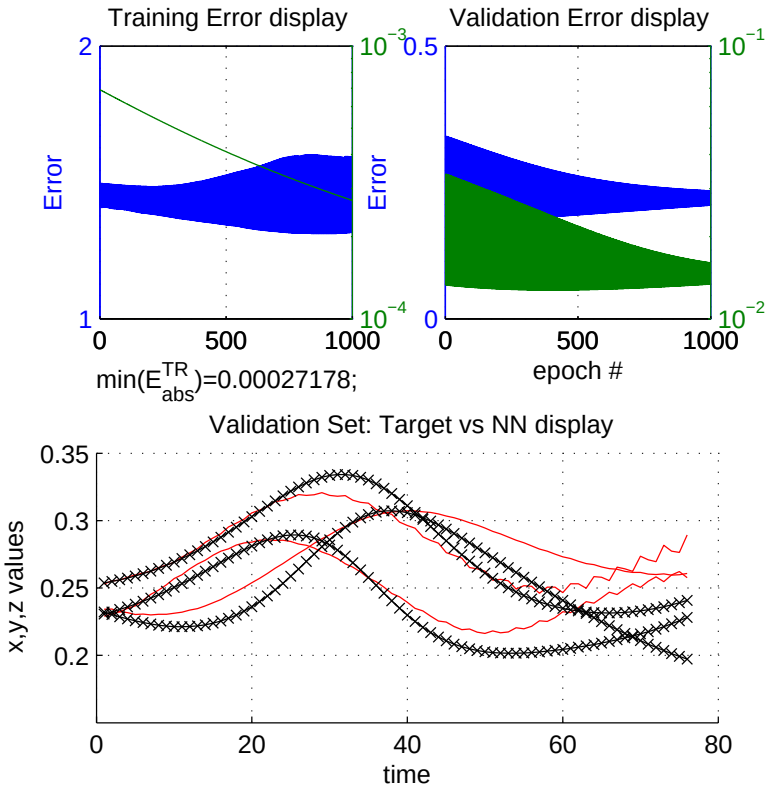

Figure 5.40: Training error for the second 1000-2000 epochs. One can see absolute and phase values for training and validation errors. Below one can see the error of the HCVNN at the training set.

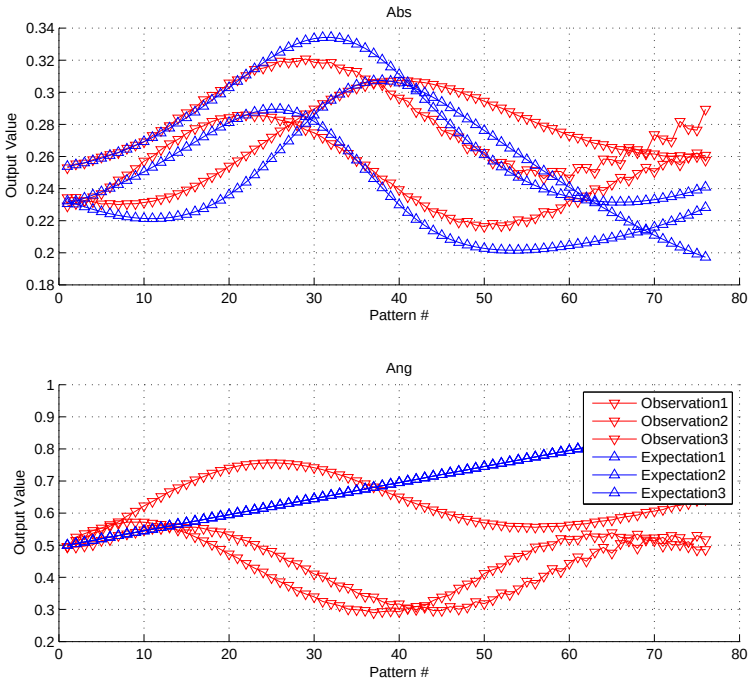

Figure 5.41: Absolute and phase parts of the expectations vs observations for the validation set.

Resutls for the valuidation set are presented below. One can see the error decay after 10000 epochs of training below. From the tables below one can see that the test set results are very promising: all RMS values are close to 0, R^2 values are close to 1. After the network has been validated and tested one can try to

Table 5.14: Test set results.Output 1 is x coordinate, Output 2 is y coordinate etc.

RMS for the test set			R^2 for the test set		
output #	absolute	angle	output #	absolute	angle
output 1	0.0005	0.0002	output 1	0.97	0.99
output 2	0.0011	0.0001	output 2	0.90	0.99
output 3	0.0008	0.0001	output 3	0.88	0.99

apply it for the completely different set. One can call this set as Test set, but in order not to mix the notations it will be called Forecasting set (this set is similar to the test set in its logic, but different in numbers). Forecast set results are presented for the absolute part (see fig. 5.42) and for the angle (phase) part (see fig. 5.43) of the CVHCNN output. Tables below (see 5.15) show the statistics for the 20 steps prediction. One should be very careful while treating the $R^2 < 0$. In case desired outputs do not change significantly, the denominator of the R^2 is very small, which makes the complete fraction very big. Subtracting this big value from 1 makes the R^2 negative. One should admit that the CVHCNN gives not only the forecast for the absolute part of the complex output, which contains Lorenz system values, but also predicts the *sin* of time, which stands at the angle (phase) part of the complex-valued output. Therefore by looking at the angle value one can say to which moment in time the prediction is relates. In case the prediction of the time (phase of the complex-valued output) starts behaving incorrectly (remember, that time is changing in a linear manner), one

Table 5.15: Forecast set results. Output 1 is x coordinate, Output 2 is y coordinate etc.

RMS for 20 steps forecast			R^2 for 20 steps forecast		
output #	absolute	angle	output #	absolute	angle
output 1	0.0000	0.0201	output 1	0.86	< 0
output 2	0.0000	0.0059	output 2	0.70	< 0
output 3	0.0007	0.0004	output 3	< 0	< 0

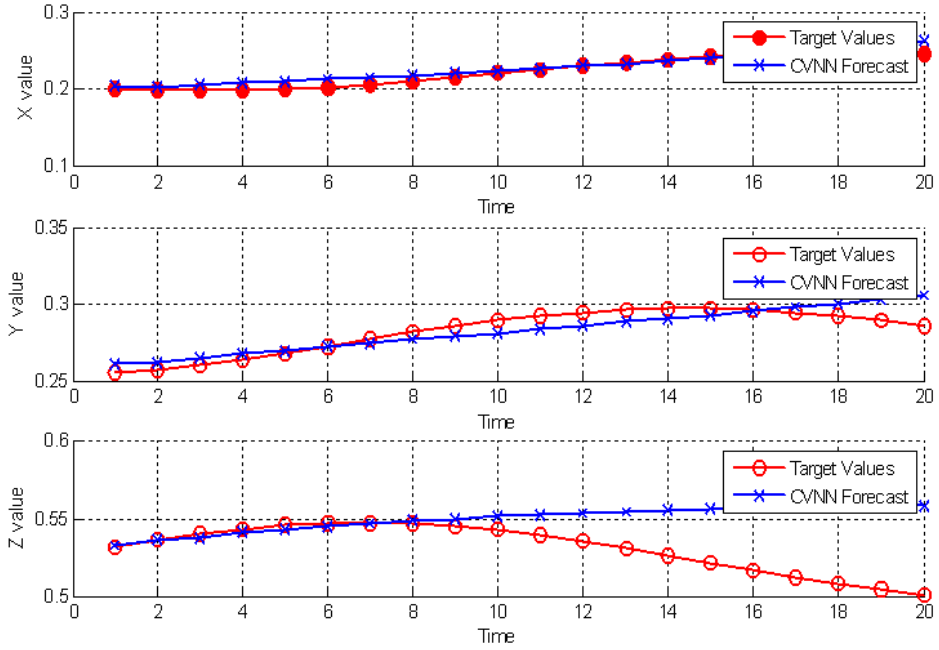

Figure 5.42: Forecast for 20 steps, absolute parts of the CVHCNN outputs.

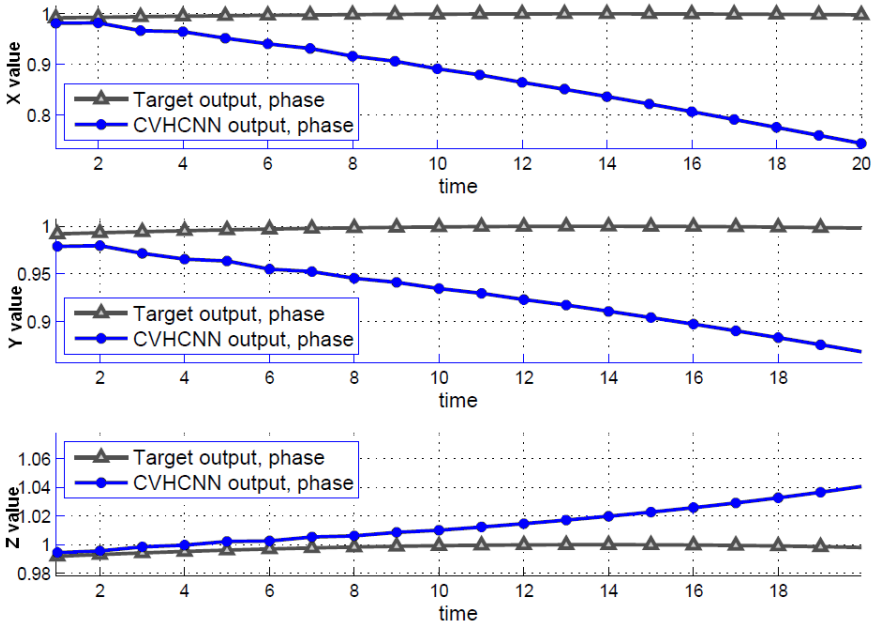

Figure 5.43: Forecast for 20 steps, angle parts of the CVHCNN outputs.

cannot trust the absolute part of the predictions. The last statement is not proven statistically or theoretically, therefore it should be checked for consistency. These results have been published in ICANN 2011 in the work by Zimmermann [31].

5.6 Transformer Modeling

Detailed modeling of electrical power network's elements is necessary for receiving accurate model data. At the same time, element's model complication may lead to the significant increase of calculation time, the memory overrun and other computation problems. Proposed modeling with Complex-Valued Neural Networks (CVNN) makes it possible for easy model equipment nonlinearities and uniqueness to keep model complexity on the appropriate level. As power grid element for CVNN-based modeling, power transformer has been chosen. Model was performed with the use of two methods - conventional analytical model and CVNN-based model. The modeling results of transformer simulation were used for CVNN training. Distinctive feature of introduced complex-valued neural network is its intrinsic capability to deal with the complex numbers instead of the real ones. This feature is quite useful in the frame of power grid elements modeling. This chapter shows promising results for further research in this direction.

Analytical Modeling of the Transformer

In a basic transformer one of the winding, named a primary winding, is energized by an external voltage source. The alternating current, flowing through a primary winding, creates a variable magnetic flux in the magnetic core. The variable magnetic flux in the magnetic core creates electromotive force (EMF) in all windings, including primary one. When current sinusoidal absolute value of EMF is equal to the first derivative of a magnetic flux, EMF induces current in the secondary winding. Ideal transformer without losses is shown in Fig.5.44 Equivalent circuit of generic transformer is shown in 5.45. Power losses are represented as resistances R_1 (primary) and R_2 (secondary), flux leakage - as reactances X_1 (primary) and X_2 (secondary). Iron losses caused by hysteresis and eddy currents in the core are proportional to the core flux and thus to the applied voltage. Therefore they can be represented by resistance R_m. To maintain the mutual flux in the core magnetizing current I_μ is required. Magnetizing current is in phase with the flux. Since the supply is sinusoidal, the core flux lags the induced EMF by 90 degrees can be modeled as a magnetizing reactance X_m in parallel with the resistance R_m. R_m together with X_m are called magnetizing branch of the model. In case of open-circuit, current I_0 represents the transformer's no load current as it was shown in the works of Voldek[9] and Galkin [56].

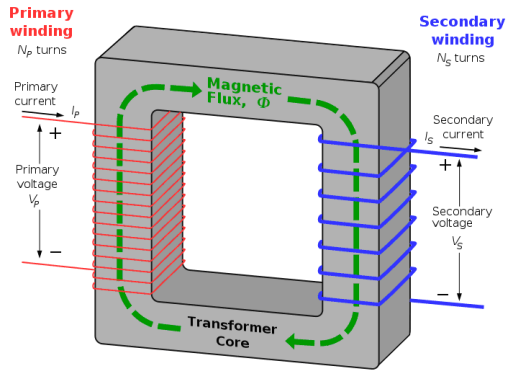

Figure 5.44: Ideal transformer (http://en.wikipedia.org/wiki/Transformer)

Analysis of circuit simplifies significantly if the circuit with magnetically con-

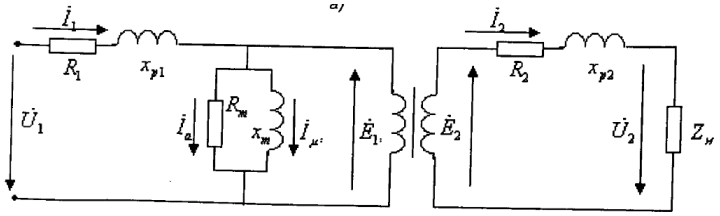

Figure 5.45: Transformer OMP-10

nected windings will be replaced by an equivalent circuit, elements of which are electrically connected with each other (see Fig.5.46). Here the number of turns in primary (N_1) and secondary (N_2) is equal, so the parameters of the transformer have to be changed in order to maintain all energy relations. The secondary winding is moved (or 'referred') to the primary side utilizing scaling factor:

$$N^2 = \left(\frac{N_1}{N_2}\right)^2 \tag{5.4}$$

Finally, transformer equations can be written as follows[9]:

$$\begin{cases} U_1 = E_1 + I_1(R_1 + jX_1) = E_1 + I_1 Z_1 \\ E_1 = R'_2 I'_2 + jX'_2 I'_2 + U'_2 \\ I_1 = I_0 + I'_2 \end{cases} \tag{5.5}$$

where $U_1, E_1, I_1, R_1, X_1, Z_1$ - primary winding voltage, EMF, current, resistance, reactance and impedance respectively. Secondary winding is described with sim-

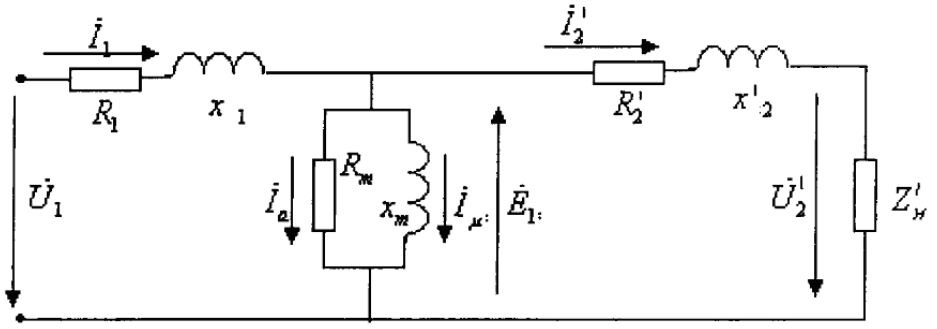

Figure 5.46: Equivalent circuit of a transformer referred to the primary winding

ilar values, but already referred to the first winding:

$$U_2' = U_2 N, I_2' = \frac{I_2}{N}, R_2' = R_2 N^2, X_2' = X_2 N^2 \tag{5.6}$$

Given transformer model is based on the real transformer data of Russian transformer OMP-10/10 (see [67]). All the parameters and more detailed descriptions of the model are available in the paper by Minin [16]
Analytical modeling has been carried out in MATLAB. Consequence of modeling is the following: with given source voltage U_1, primary winding current I_1 is calculated:

$$I_1 = \frac{U_1}{Z_1 + Z_m \cdot (Z'_{load} + Z'_2))/(Z_m + (Z'_{load} + Z'_2))} \tag{5.7}$$

Then E_1 is found. After that, magnetizing current is computed and according to the Kirchhoff's law current in the secondary winding (I_2) is obtained. Then, with use of main transformer eq. 5.5, U_2 is calculated. Results, obtained from transformer simulation are presented in eq. 5.47. As it can be seen, variation of temperature leads to adequate voltage and current response. The rise of temperature increases load impedance, which decreases primary current (I_1), secondary current (I_2) and secondary voltage (U_2) by-turn. One should note that the aim of the simulation was to generate the data which will show general possibility of CVNN to model such a device. Range of temperature changed within the simulated time period (0.4 s) is not physical, but it does not matter in the neural network training business.
CVORNN modeling. For the transformer modeling recurrent archietcture will be used. The network is recurrent through its weights, therefore shared weights concept is to be used. The weights A, B and C are shared weights. This means that A, B and C are always the same at each iteration of the network. The network is called open since it has external drives. One can see that such network

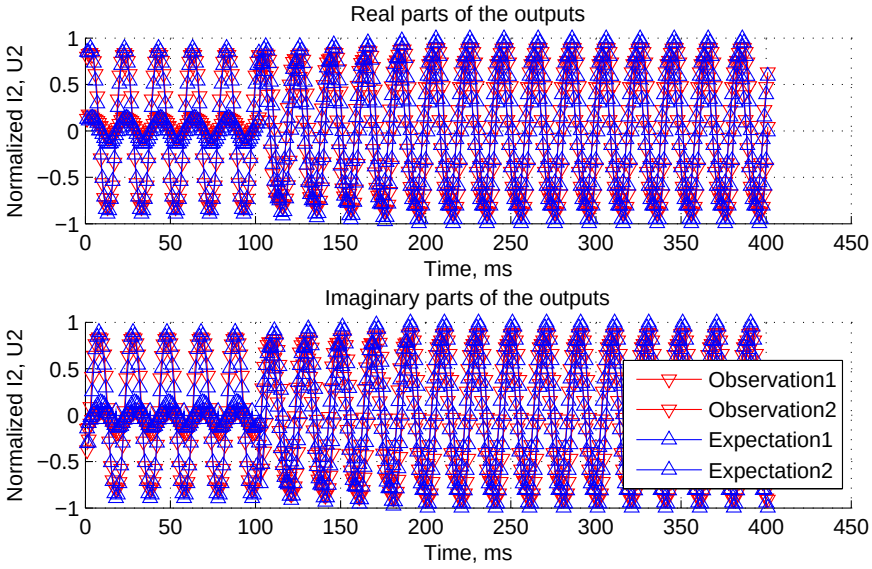

Figure 5.47: Results of the simulation. Due to introduced temperature dependencies changing temperature change of windings and load impedances occur. Finally, voltages and currents are affected.

is perfectly suited for the time series modeling. Time series, obtained from the modeling, are to be used at the input side of the neural network (I_1, U_1, T, Z, $OLTC$) and U_2 and I_2 are used at the output side. Then the training of the network with the complex-valued gradient descent minimization discussed above is to be initiated. To obtain the prediction of the U_2 and I_2 pne has to continue the feedforward path of the network by iterative application of matrix A, B and C to the states and inputs respectively. In case the system is autonomous (develops on its own) $B = 0$; such architecture is called Complex-Valued Closed Recurrent Neural Network (CVCRNN).

NN Training for the Given Problem

The inputs for the network are the following parameters: input current, input voltage, load and temperature, the outputs (desired values), but here we are interested in the output current (I_2) and the output voltage (U_2). According to the transformer model described above a set of 60000 patterns is generated. One can see that there are two peaks at the generated data, which correspond to the short circuits. For the training 2000 data points around the first peak (this is so-called Training set) have been taken. Then a set of 1000 data points around the second peak is to be used to see how well the network generalizes (validation set). For this experiment the network had 50 state neurons, 5 inputs and 2 outputs. Learning rate has been chosen to be $\eta = 0.05$. The amount of epochs

for training is equal to 2000. 1 epoch of training means the network has been applied in forward direction, derivative of the error has been calculated and back propagated. Then gradients have been calculated and weights matrixes A, B and C have been updated. 1000 epochs meaning that one has to do it 1000 times. At the Fig. 5.48 one can see the error decay while learning. One can see how good

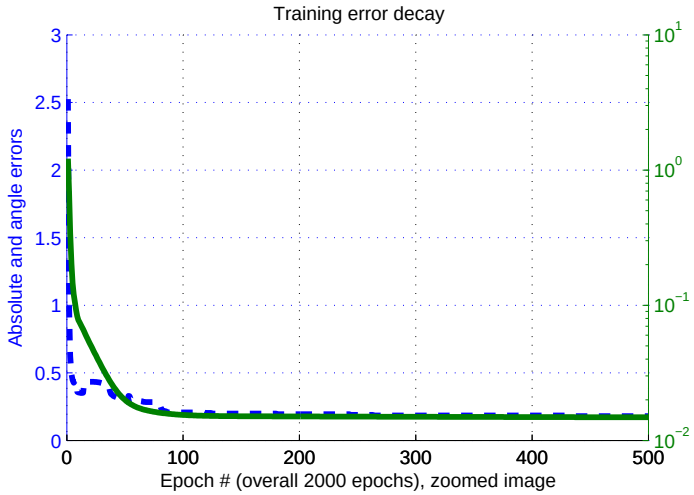

Figure 5.48: Error decay for the absolute part of the error function and for the angle part of the error function.

the network is learned at the training set at the Fig. 5.49

Then one has to apply the trained network to the validation set and see how good the network generalizes. At the Fig. 5.51 one can see the absolute part of the network output and at the Fig. 5.50 real and imaginary parts of the neural network outputs, since these outputs are more important for the particular problem. Here one can see that expectation values almost coincide with the observation ones and only differ slightly at the bump area; statistical coefficients are also close to their corresponding best values. One can see that stistics for the validation set D_V below 5.16.

Table 5.16: Statistical results for the transformer modeling at the Validation set

Real/Imaginary parts	RMSE	R	R^2
I_2	$6 \cdot 10^{-3}/8 \cdot 10^{-3}$	0.99/0.99	0.98/0.97
U_2	$2 \cdot 10^{-3}/2 \cdot 10^{-3}$	0.99/0.99	0.99/0.98

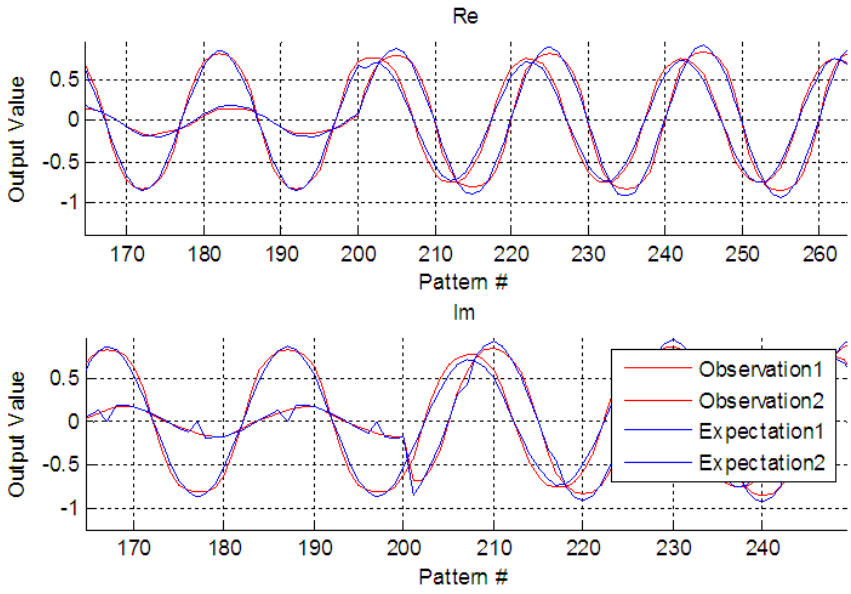

Figure 5.49: Results of the transformer modeling for the subset of data containing leaps. One can see the real part of the network outputs and the actual values of I_2, U_2 on the training set. Zoomed image, since otherwise one cannot see the short circuit moment. One can see that network reacted perfectly, but this is training set.

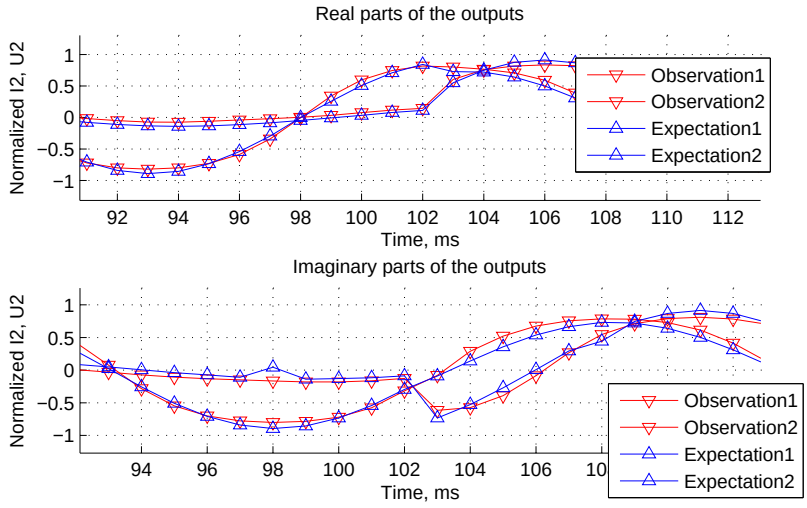

Figure 5.50: Results of the transformer modeling for the subset of data containing leaps. One can see the real (imaginary) part of the network outputs and the actual values of I2, U2 on the Validation set. Zoomed region to see the short circuit moment better. Real and Imaginary values for voltage and current.

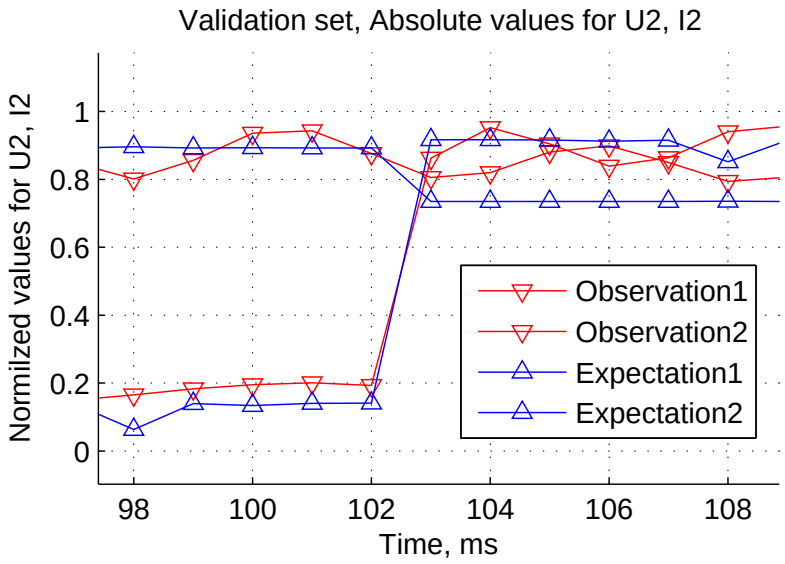

Figure 5.51: Results of the transformer modeling for the subset of data containing leaps. One can see the absolute (phase) part of the network outputs and the actual values of I2, U2 on the Validation set. Zoomed region to see the short circuit moment better.

Conclusion

From obtained results the following conclusions can be formulated:

- General possibility of CVORNN to model dynamics of advanced transformer model has been shown.
- Further tests with enhanced model have to be carried out in order to prove the preliminary simulation results. Injection of appropriate nonlinearities and adding noise in the analytical model for generating data will make the task more realistic.
- Tests with data from real devices have to be implemented. The attractive feature is that it is possible to model each grid device individually, just teaching the CVNN with measured data from the particular device.
- CVORNN could be applied to another power engineering equipment simulation.

5.7 Binding Problem

Brains are constantly faced with the problem of grouping together features of the objects they perceive so as to arrive at a coherent representation of these objects. Such features are shape, motion, color, depth and other aspects of perception. There is experimental evidence and a large body of theoretical work that support the hypothesis that brain solves this so-called binding problem by synchronizing the temporal firing patterns in neuronal assemblies with neurons that are sensitive to different features. According to this hypothesis, temporal correlations between neuronal impulses on the order of milliseconds would represent the fact that different object features have to be associated with one and the same perceived object.
In this subsection a new model for solving the binding problem will be suggested. This will be done by introducing complex-valued recurrent networks. These networks can represent sinusoidal oscillations and their phase, i.e., they can model the binding problem of neuronal assemblies by adjusting the relative phase of the oscillations of different feature detectors. As feature examples, we use color and shape, but the network would also function with any combination of other features.
The suggested network architecture performs image generalization but can also be used as the image memory. The information about object color is represented in the phase of the network weights, while the spatial distribution of neurons codes the object's shape. It is possible to show that the architecture can generalize object shapes and recognize object color with very low computational overhead.
The Binding Problem is a well-known problem for the object - color recognition. Imagine one sees a green car. Before that, he has seen only different colored cars. He has never seen the green car before, but how he knows that he still sees a car and that its color is green? The answer to this question was given in a set of papers [1, 2, 3] and this is well-known Binding Problem. The thing is that our brain has a basic idea about the car, which contains only the general shape information and this information generates the basic idea about the cars class. This is what ancient Greek philosopher Plato called the "basic idea" about the object. But how does the man recognize that he sees namely a green car? The answer is that the human has cortex neurons which can absorb only a certain wave length diapason of the light. Thus, different neurons are sensitive to different colors. When a man sees the green color, the neurons, which are responsible for storing the information about the cars class, start oscillating and providing the information about the cars class. On the other hand the neurons which are sensitive to the green color start oscillating when the light scattered by the car is coming to the human eyes. After some time the neurons, sensitive to the green color as well as neurons which are devoted to the cars class start oscillating with

the same phase, which means they are devoted to the same object. Thus the car obtains its green color in the "recognition imagination".
The architecture used in the following work is described at the figure 5.52 below: The only thing one should ensure is that the weights A, B, C are always the

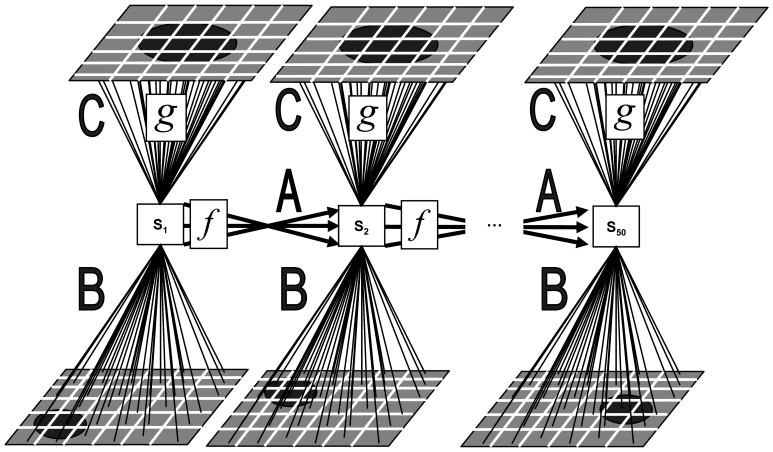

Figure 5.52: Complex-Valued Recurrent Neural Network used in the paper for the image recognition. A,B and C are weight matrixes. Lines show the connection between the layers. The figure is simplified version of the used architecture to show the connection. In the experiments the amount of states has been equal to 50.

same. For this purpose the shared weights concept proposed by H.-G. Zimmermann [33] discussed in the chapter related to the Recurrent Neural Networks has been used. This means that the matrixes A, B and C are always the same in all parts of algorithms while training and using the trained network. One should be very careful with the nonlinearities (functions) since due to the Liouville theorem (this issue has been widely discussed by Brandwood and in the current thesis) they are unbounded. In the present work the function has been used for the states interaction and for the state - output interaction. The input layer propagates the information with the linear function. Matrixes A, B and C are sparse matrixes with the amount of zeros about 70 percent. Sparsity is very important to have faster and more reliable computation. The network has been trained using the pattern of pattern complex-valued gradient descent training.

Complex-Valued Recurrent Neural Network Simulates Network of Oscillating Units

Since CVRNN deals with complex inputs it has been decided to code the images in a more "physical" way, which means each pixel of an image (images 50 by 50 pixels have been used) is now presented by two numbers which are the amplitude

(strength of the light wave) and the phase (color of the light wave) according to the following table: Current work used images of 4 colors, namely red, green,

Table 5.17: The table presents the color coding of the shapes

Color	Red	Green	Blue	Yellow
Coding	$0.3e^{i\pi/6}$	$0.3e^{i\pi/3}$	$0.3e^{i2\pi/3}$	$0.3e^{i5\pi/6}$

blue and yellow. In order to see how good the network can generalize test images of yellow color (see figure below) have been used. Current thesis section relies

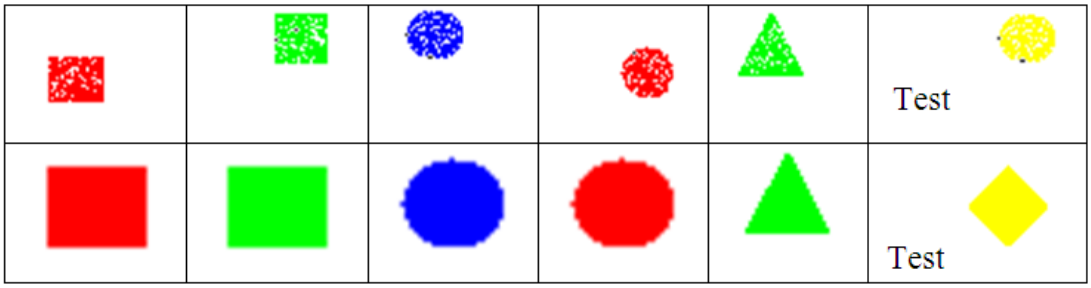

Figure 5.53: The training set for the CVRNN (upper row of the table). From left to right: red rectangle, green rectangle, blue circle, red circle, green triangle and yellow circle (this is a limited set of figures to be shown in the paper, training set contains 15 images of such type). All images were 20% corrupted by Gaussian noise (while dots at the input images). The background of all images is black. Target output for CVRNN is lower row of the figure. Last image is absent due to generalization reasons. The CVRNN should be able to recognize last yellow circle from the upper row.

upon the fact, that brain has some basis ideas about the objects following Plato (Greek Philosopher) in the book by Perlovsky [[44]]. That means one should first train the network to generalize some objects to the predefined classes (objects are in different colors (see Fig. 5.53) and then try to show the image which the RCVNN has not seen before and looks whether it can generalize the object if the color of the object changes.
Therefore the experiment consists of two parts. First part is to see how the color will be represented in the network at the training set in order to show that neurons of the output layer, which correspond to some image, will be synchronized in phase to represent the color and spatially to represent the shape. In the second part the network will be presented with some objects from the class it should know but with a changed color. The task is to see which neurons will be synchronized

and which phase the output neurons will have. Moreover for all experiments all outputs of the CVRNN will be tracked. This is done in order to see how the memory works in case of image recognition (note that RCVNN should have memory due to its structure). In the current work the images used for the training were static while in general to form the training set one should use the dynamic images with always the same target;e.g. one should create different images of triangle (under the different angles and sizes) with always same target triangle. Thus the network will be able to generalize any triangle to the triangles class and give the color by the synchronization of the neurons in the triangle class with the color of the input image.

Results on Image Recognition

The RCVNN has been trained for 10000 epochs using the extended training file similar to the Fig. 5.53. The size of each image was pixels. The learning has been conducted with the complex-valued back propagation and gradient descent method. The learning rate is $\eta = 0.02$. Final training error after the training step (see eq. 2) has been equal to 10^{-11}. The hidden dimensionality was equal to 50. Therefore the size of matrix A was 50×50, size of matrix B was 2500×50 and the size of matrix C has been 50×2500. All matrixes have been initialized by random weights and have been selected to be sparse analogous to the real world neurons (the sparsity of the real world network is about 20%). In the current work the sparsity of matrixes B and C have been 60%, the sparsity of the matrix A was 30%.

Neurons synchronization results

In the current subsection an attempt to simulate the synchronization of the neurons is going to be done. The image at the input is a corrupted image of some shape (e.g. rectangle) which is smaller than the target and stands at some random place of the input image. Output image is a generalized shape of the object (see Fig. 5.53) of the same color. The training set contains same objects of different colors to make the task more complicated. Overall the training set contained 20 different objects of different color located in the different places of an image. The experiment is to present the network the object which it has seen during the training with the color it has seen during the training but it has never seen both at the same time. In case the input signal can produce the output from the needed class and of the input color one can say that he has a binding. One can see the synchronization simulation results below at the Fig. 5.54. One can see from the Fig.5.54 that neurons synchronize not only in a phase, but also in the spatial distribution representing the color and the shape of the object. Here we

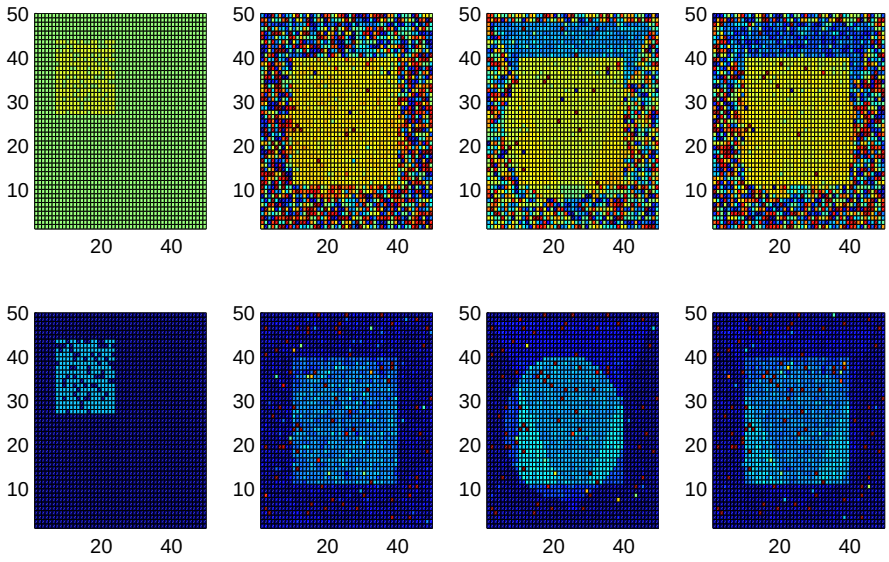

Figure 5.54: Example of the neurons synchronization for the pattern from the training set. Upper row left to right: angle of the input pattern, angle of the network output from the 5th state, phase of the network output from the last 50th state. Lower row, left to right: absolute part of the input pattern, absolute part of the output from the 6th state, absolute part of the output from the 50th state. Bar to the right hand side displays the color ranges for all pictures in a row.

can see the coupling of the phase and the spatial distribution. This means that the particular input which is from the training set can excite the correct target object (the shape of the target object) with the color of the input object. This allows us to think that CVNN offers a nice and elegant solution to the binding problem.

Capability of the network to recognize known object with unknown color

After it has been shown that neurons do really synchronize for the same color, moreover the spatial structure of the synchronized neurons represented the shapes one can proceed and try to force the network to generalize objects, which means that the present image has not been seen during the training. The task is to see whether neural network will be able to recognize the shape and the color. For the results see the Fig. 5.55 below. The network receives the yellow noised circle. One can see from the Fig. 5.55 that network have been still possible to recognize the shape but not the color (same experiments have been conducted for different shapes which are not in the training set) which answers the question whether the network can generalize, the answer is affirmative, but without the color in case it has not seen the color before (see Fig.5.55). Since there is no possibility to show all pictures in the paper we have to note, that if the image at the generalization set will have a new shape (not from the training set) and a new color the network will not recognize the object at all.

Capability of network to recognize unknown object with unknown color

Now one should consider what happens if an unknown object paired with the unknown color is presented to the network. Input object is yellow rhomb. As it can be seen from the Fig. 5.56 obtained from the simulation, the neural network has very much noisy output which consists of two shapes, namely a circle (with maximum possible phase) and a triangle with minimum possible phase. Absolute part of the output is nearly noise with some shadows of the circle and the triangle from the training set. The phase picture is more definite. The interpretation of two shapes appearance is that the network tries to remember the most topologically equal objects from the memory which exists due to the states interaction. The triangle and the circle appear due to the fact, that if to look to the structure of the data matrix containing the triangle or the circle, these structures are very similar to the rhombus (in the case very low resolution which is 50 by 50 have been used), therefore these both shapes appear as an output. In the case rhomb has been used for training the network can reproduce it with its color.

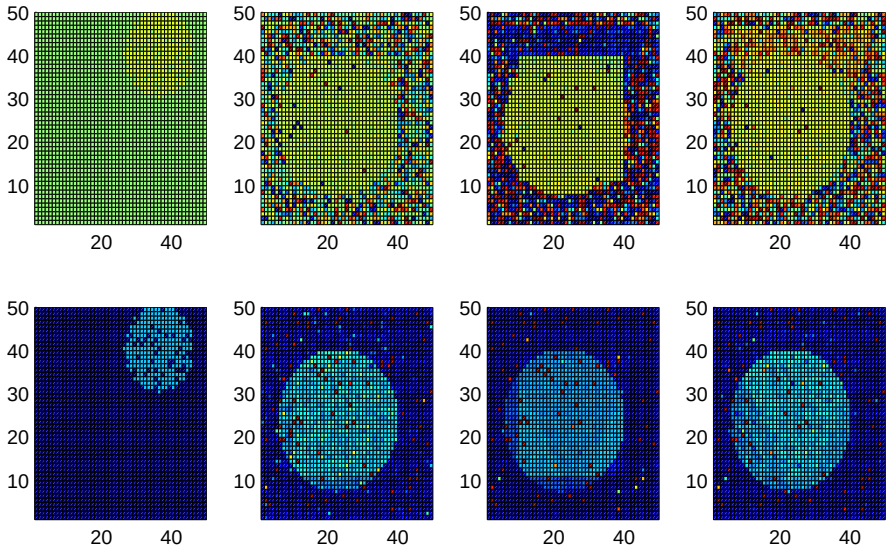

Figure 5.55: Example of the neurons synchronization for the pattern from the training set. Upper row left to right: input pattern - angle, network output from the 5th state - angle, network output from the last 10th state - angle. Lower row, left to right: absolute part of the input pattern, absolute part of the output from the 6th state, absolute part of the output from the 10th state. Bar to the right hand side displays the color ranges for all pictures in a row.

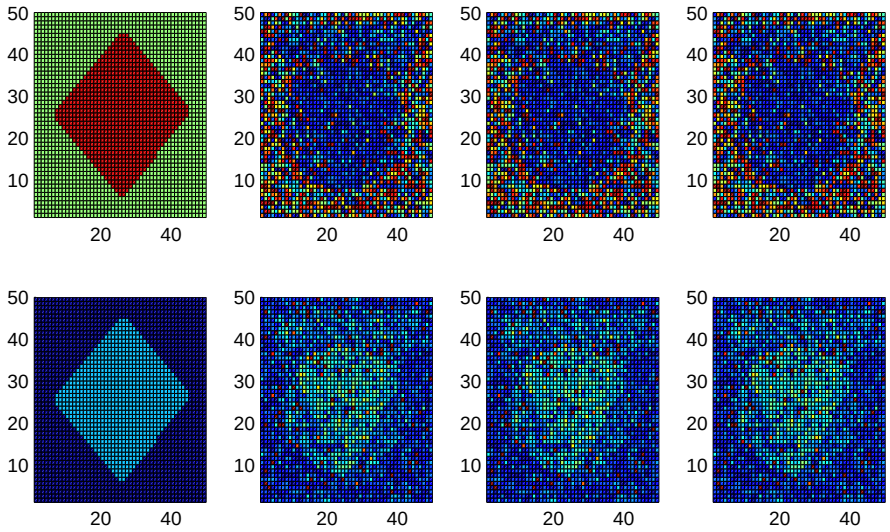

Figure 5.56: Example of the neurons synchronization for the pattern from the training set. Upper row left to right: input pattern - angle, network output from the 5th state - angle, network output from the last 10th state - angle. Lower row, left to right: absolute part of the input pattern, absolute part of the output from the 6th state, absolute part of the output from the 10th state. Bar to the right hand side displays the color ranges for all pictures in a row.

Summary and Outlook

It has been shown that CVORNN can simulate the synchronization of weight coupling the information about the shape of the object together with its phase. Moreover the memory, the generalization and the synchronization can be simulated by the interaction of the phases of complex values by using the CVORNN. The future work in this direction would be online training of the CVORNN in order to use the information about the object by looking on it from different sides (the way a little child does it in the beginning). This will allow creating the big data base of images. Then one will be able to show the generalization capabilities of the CVORNN for more classes of objects and the memory capacity of the CVORNN then could be estimated.

5.8 Advantages on Brain Synchronization Modeling, Biological example

The brain is constantly faced with the task of grouping together features of objects that it perceives. Such features, for example, are shape, motion, color and depth. There is experimental evidence that supports the hypothesis that the brain solves this so-called "binding" problem by synchronizing the temporal firing patterns in neuronal assemblies. Mathematically, this phenomenon can be modeled by a system of differential equations for the oscillating spiking neural network. In this paper we suggest a new model for solving the binding problem by introducing complex-valued recurrent networks. These networks can represent sinusoidal oscillations and their phase, i.e., they can model the binding problem of neuronal assemblies by adjusting the relative phase of the oscillations of different feature detectors. Specifically, it can be shown that these networks solve the binding for color and intensity if, for example, the information about color is represented by the phase and the intensity by the modulus of the neuronal oscillation. We also show that neurons are synchronized with respect to their phase - and not their amplitude (as is the case for the real-valued spiking neural network). The amplitude plays the important role of a spiking barrier. The section can be concluded by showing some useful properties of the complex-valued approach for modeling the oscillating spiking neural network and by showing how to model the memory of such a network.
Cortical oscillations (see, e.g., [25]) can be described by their frequency, amplitude and phase. In large-scale oscillations (meaning a large number of neurons taking part in the oscillations), amplitude changes caused by changes in synchronization within a neural committee (zone in the brain, where neurons are oscillating in a sin phase way), also called as local synchronization, have been considered as a

basis for cognitive features, such as perception or pattern recognition. Moreover, for the case of local synchronization, changes in the phase synchronization among oscillatory development of remote neural committees were found in the real world experiments. This may be considered as a neural process for information communication. The research into neural oscillations is a sub-area of "neurodynamics", the area of neuroscience that focuses on the dynamic character of neural activity in describing brain functions). In the suggested approach the differential equations to describe how neural activity evolves over time will be used. The aim of the research is at relating cognitive functions to specific dynamic patterns in the brain.

This part of the research is based on the works and results obtained by of A. Maye [3], Shillen [63] and Li [48] where the authors propose an oscillator network for modeling the work of cortical neurons. The idea of these papers is like follows. The work of two connected groups of excitatory (x) and inhibitory (y) neurons can be described by the dynamical system:

$$\begin{cases} \dot{x} = -\tau_x x - g_y(y) + L_0^{xx} g_x(x) + I_x + \eta_x \\ \dot{y} = -\tau_y y + g_x(x) - I_y + \eta_y \end{cases} \tag{5.8}$$

Here τ_α ($\alpha \in \{x, y\}$) are constants that can be chosen to match refractory times of biological neurons, L_0^{xx} describes self-excitation of the excitatory population and η_α white noise, that models fluctuations within the populations. With external input I_α above threshold, the solutions of (5.8) are limit-cycle oscillations.

The transfer function is typically modeled by sigmoid-shaped functions. From a metabolistic viewpoint it is not desirable for real neurons to reach saturating activity in every cycle. In this model a semi-linear transfer function,

$$g_\alpha(x) = \begin{cases} m_\alpha(x - \theta_\alpha), \text{ if } x > \theta_\alpha \\ 0, \text{else} \end{cases} \tag{5.9}$$

The external input $I_\alpha = I_0 + I_{lat}^\alpha$ consists of static input from the feature detectors I_0 and input from neighboring oscillators via lateral connections I_{lat}^α. According to the author, the excitatory coupling $I_{lat}^x = \sum_{s \in S} L_s^{xx} x_s$ synchronizes oscillators, whereas the inhibitory coupling $I_{lat}^y = \sum_{s \in S} L_s^{yy} y_s$ desynchronizes them. S is the local neighborhood of the oscillator under consideration and $L^{\alpha\beta}$ ($\alpha, \beta \in \{x, y\}$) positive weights. As it was found, connections originating from excitatory neurons have a synchronizing and those from inhibitory neurons a desynchronizing effect, so all connection types could have been used.

Parameters selection. In this subsection the input signal will be represented as a complex number characterizing the light; it has the following form: $inp = Ae^{i\phi}$, where A – intensity, ϕ – phase of the signal. The light divides into a number

of parts corresponding to the dimension of space of color representation (RGB, CMYK color models etc.). Two dimensional space representing the color (with corresponding ϕ_1, ϕ_2) was used. Thus, for every light k – external input to the system we have $\phi = (\phi_1^k, \phi_2^k)$. The intensity has to be taken equally for each dimension of color and is to be transfered between neurons as the parameter θ which is used as a threshold in the function g (eq.:5.9).
The work represents the system as a recurrent network with characteristics depending on time. While the input signal is represented as a complex number, the system is solved for only the phase part of the signals, that is, it works with real numbers.
The network is presented in Fig. 5.57. For each input color we have the same picture.

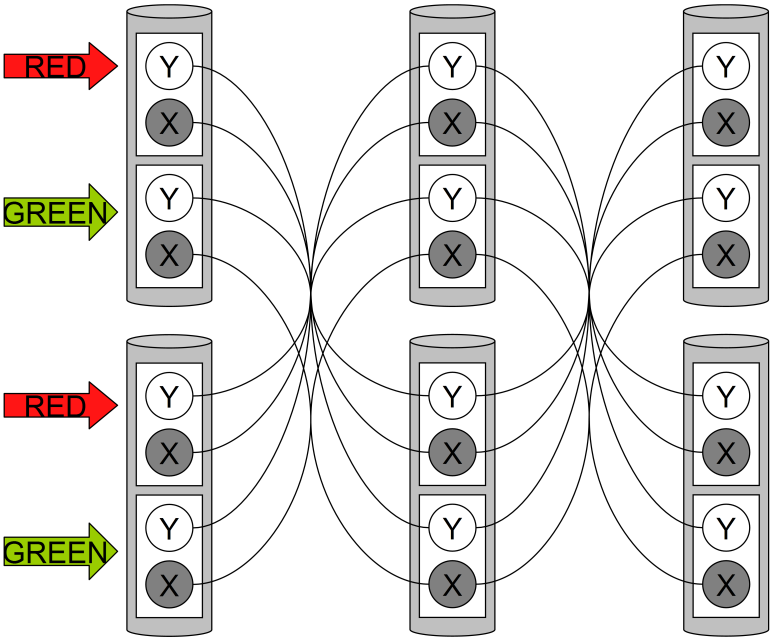

Figure 5.57: Recurrent network for oscillations modeling

Experiments. For the network structure in Fig. 5.57 the results are displayed later in the thesis. The selected parameters for the system or equations are presented below: $\tau_x = \tau_y = 1$, $m_x = 16$, $m_y = 8$, $L_0^{xx} = 0.1$, $\sum_{s\in S} L_s^{xx} = 1.1$, $\sum_{s\in S} L_s^{yy} = 0.3$, $\eta_x \sim 0.2N(0,1), \eta_y \sim 0.2N(0,1)$, ϕ_1, ϕ_2. Two lights come into the system: the first with $\phi^1 = \pi/2$, the second with $\phi^2 = \pi/3$. The lights divide into two parts with intensities $\theta_x = 2, \theta_y = 1$, $\phi_1^k = 3\phi^k/2$, $\phi_2^k = 3\phi^k/3$ for each light $k \in 1..2$ on input. As the output of the network we take $x(t)$ corresponding to the first coordinate of color representation. Denote *recurrency* – number of iterations. The output values correspond to the last iteration –

recurrency. $CCor$ – matrix of correlations between x-outputs of neurons, $CCor_{avg}$ – average matrix of correlations between x-outputs of neurons for all iterations. The weights of inputs from neighbors are randomly selected so that the conditions $\sum_{s\in S} L_s^{xx} = 1.1$, $\sum_{s\in S} L_s^{yy} = 0.3$ are satisfied. So we use 10 iterations to get average measures (correlation). If the weights are equal, the resulting outputs corresponding to the same input light are equal (see fig. 5.58). One can see that at the beginning of $x(t)$ the outputs are almost the same for the neighboring oscillators.

Observation 1. For different recurrency different results have been obtained.

Observation 2. With the recurrency increasing the influence of noise in the system on the solution x decreases. The results for different input parameters will be presented. The inputs parameters and corresponding correlation matrixes paired with pictures of oscilaltions at the last iteration will be shown. To describe different experiments the following notations are to be used:

```
Input=[Weights=...;
Noise=true;
Recurrency=...;
Colors={...};
Number of iterations = ...];
```

Where: "Weights=" have the following possibilities: Equal, Random, Different; "Noise=true" it means that there is presence of small noise in the system, "Recurrency" is amount of states used in the RCVNN model, "Colors" means inputs of the network can have Red, Green and both Red and Green, "Number of iterations" means the amount of times the system of Eq. 5.8 iterated. Now let me start with experiments.

Experiment 1. Experiment setup:

```
Input=[Weights=Equal;
Noise=true;
Recurrency=2;
Colors={Red, Green};
Number of iterations = 10];
```

results are presented at the Fig. 5.58. Matrix of correlations between outputs.

$$\mathbf{CCor} = \begin{pmatrix} 1.0000 & 0.7128 & 0.5065 & 0.5160 \\ 0.7128 & 1.0000 & 0.5390 & 0.5196 \\ 0.5065 & 0.5390 & 1.0000 & 0.9432 \\ 0.5160 & 0.5196 & 0.9432 & 1.0000 \end{pmatrix}$$

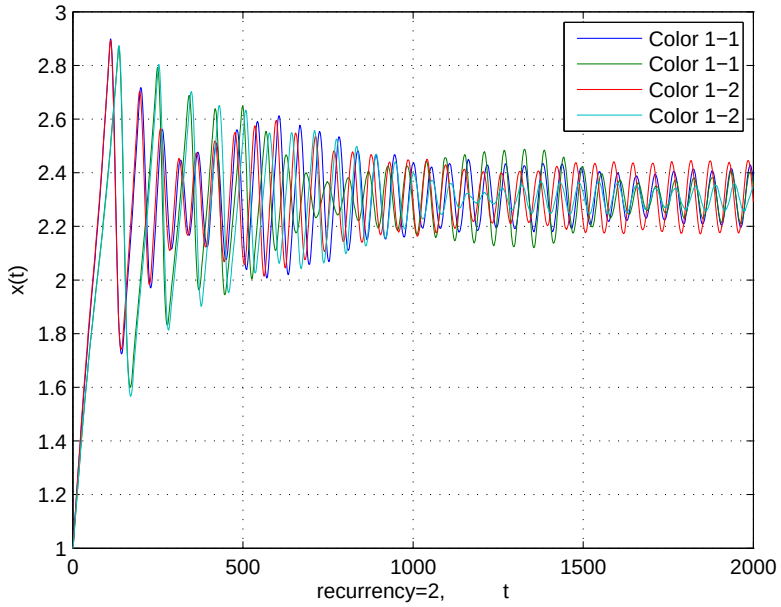

Figure 5.58: $x(t)$ for recurrency $= 2$ when weights from neighboring oscillators are equal.

Matrix of average correlations between outputs for equal weights of neighboring oscillators. The randomness arises from noise in the system 5.8.

$$\mathbf{CCor_{avg}} = \begin{pmatrix} 1.0000 & 0.8922 & 0.5109 & 0.5108 \\ 0.8922 & 1.0000 & 0.5269 & 0.5266 \\ 0.5109 & 0.5269 & 1.0000 & 0.9380 \\ 0.5108 & 0.5266 & 0.9380 & 1.0000 \end{pmatrix}$$

With recurrency increasing the amplitude of oscillation increases too (see Fig. 5.58-5.61). The outputs for neighboring oscillators coincide.
Experiment 2. Experiment setup:

```
Input=[Weights=Different;
Noise=true;
Recurrency=6;
Colors={Red};
Number of iterations = 10];
```

Results are presented at the Fig.5.59.

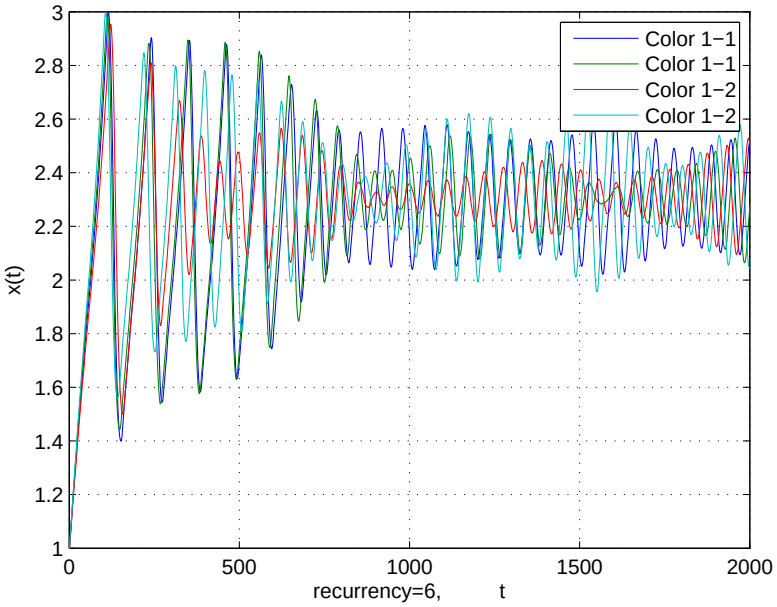

Figure 5.59: $x(t)$ for recurrency $= 6$, for the system without noise.

$$\mathbf{CCor} = \begin{pmatrix} 1.0000 & 1.0000 & 0.3563 & 0.3563 \\ 1.0000 & 1.0000 & 0.3563 & 0.3563 \\ 0.3563 & 0.3563 & 1.0000 & 1.0000 \\ 0.3563 & 0.3563 & 1.0000 & 1.0000 \end{pmatrix}$$

In the case when weights coming from neighboring oscillators are not equal the result is similar but discrepancies in outputs start earlier (see fig. 5.63 and corresponding matrix of correlations).
Experiment 3. Experiment setup:

```
Input=[Weights=Equal;
Noise=true;
Recurrency=10;
Colors={Red;Green};
Number of iterations = 10];
```

Results are presented at the Fig.5.60.

$$\mathbf{CCor} = \begin{pmatrix} 1.0000 & 1.00000.2748 & 0.2748 \\ 1.0000 & 1.00000.2748 & 0.2748 \\ 0.2748 & 0.27481.0000 & 1.0000 \\ 0.2748 & 0.27481.0000 & 1.0000 \end{pmatrix}$$

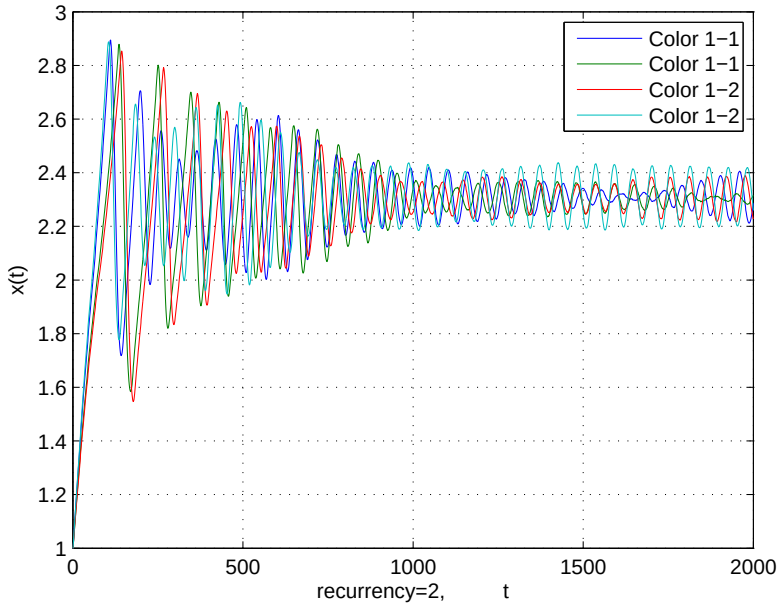

Figure 5.60: $x(t)$ for recurrency $= 10$, for the system without noise.

Experiment 4. Experiment setup:

```
Input=[Weights=Equal;
Noise=true;
Recurrency=6;
Colors={Red;Green};
Number of iterations = 20];
```

Results are presented at the Fig. 5.61.

$$\mathbf{CCor} = \begin{pmatrix} 1.0000 & 1.0000 & 0.2154 & 0.2154 \\ 1.0000 & 1.0000 & 0.2154 & 0.2154 \\ 0.2154 & 0.2154 & 1.0000 & 1.0000 \\ 0.2154 & 0.2154 & 1.0000 & 1.0000 \end{pmatrix}$$

If the same light comes into the system in which there are two pairs of neighboring oscillators then we have the following picture 5.62. As we see here the outputs coincide at the beginning which gives high correlations between them. Thus, the oscillators which are not neighbours but have the same input signal (light or color) synchronize in both amplitude and phase.

Experiment 5. Experiment setup:

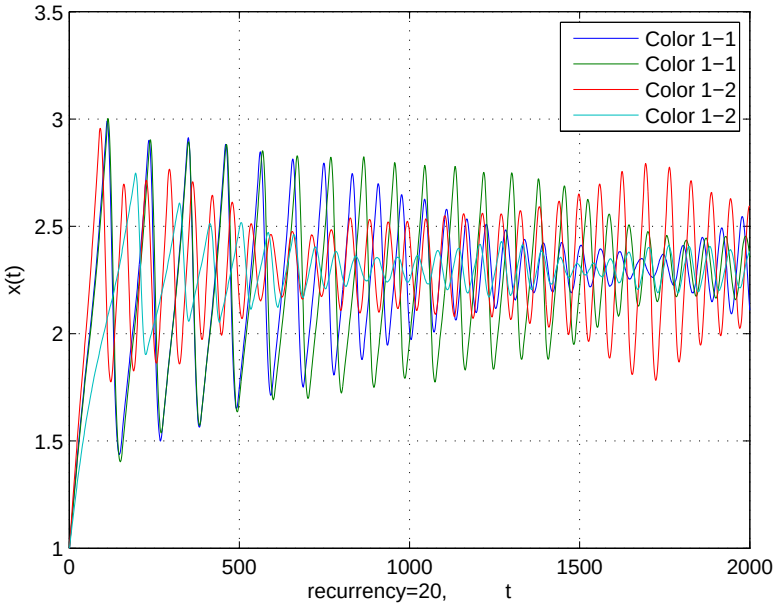

Figure 5.61: $x(t)$ for recurrency $= 20$, for the system without noise.

```
Input=[Weights=Equal;
Noise=true;
Recurrency=6;
Colors={Red};
Number of iterations = 10];
```

Results are presented at the Fig.5.62

$$\mathbf{CCor} = \begin{pmatrix} 1.0000 & 0.8450 & 0.8600 & 0.8411 \\ 0.8450 & 1.0000 & 0.8112 & 0.8442 \\ 0.8600 & 0.8112 & 1.0000 & 0.8189 \\ 0.8411 & 0.8442 & 0.8189 & 1.0000 \end{pmatrix}$$

Experiment 6. Experiment setup:

```
Input=[Weights=Equal;
Noise=true;
Recurrency=6;
Colors={Red};
Number of iterations = 10];
```

Results are presented at the Fig.5.63

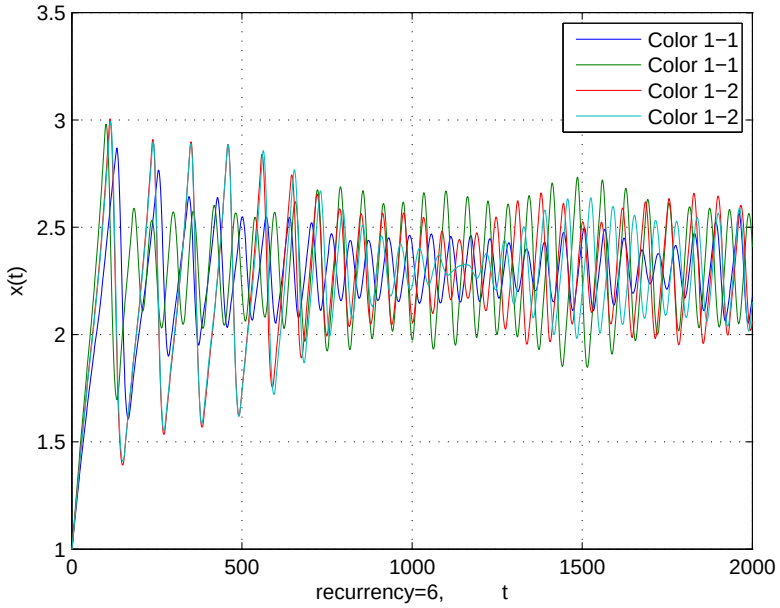

Figure 5.62: $x(t)$ for recurrency = 6, for the system with equal weights from neighboring oscillators.

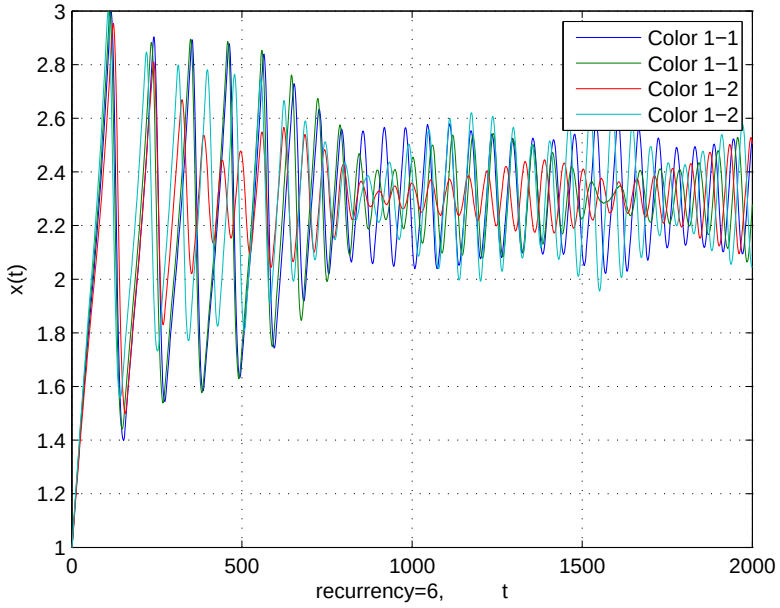

Figure 5.63: $x(t)$ for recurrency = 6, for the system with different weights from neighboring oscillators.

$$\mathbf{CCor} = \begin{pmatrix} 1.0000 & 0.4895 & 0.6845 & 0.7465 \\ 0.4895 & 1.0000 & 0.4283 & 0.4969 \\ 0.6845 & 0.4283 & 1.0000 & 0.8432 \\ 0.7465 & 0.4969 & 0.8432 & 1.0000 \end{pmatrix}$$

Experiment 7. In the case when weights of neighboring oscillators are randomly chosen, the results are the following. Experiment setup:

```
Input=[Weights=Random;
Noise=true;
Recurrency=2;
Colors={Red;Green};
Number of iterations = 10];
```

Results are presnted at the Fig.5.64. Matrix of correlations between outputs.

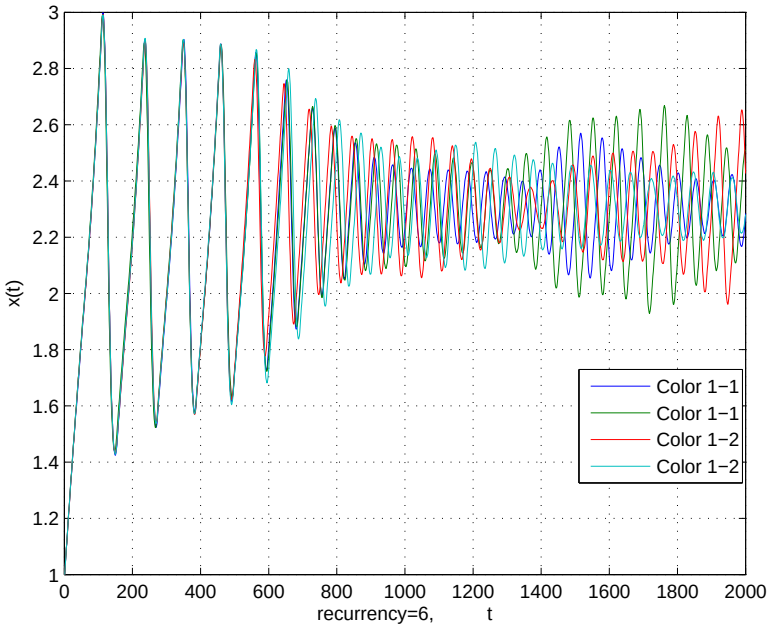

Figure 5.64: $x(t)$ for recurrency $= 2$.

$$\mathbf{CCor} = \begin{pmatrix} 1.0000 & 0.4479 & 0.4664 & 0.5103 \\ 0.4479 & 1.0000 & 0.6303 & 0.4965 \\ 0.4664 & 0.6303 & 1.0000 & 0.7153 \\ 0.5103 & 0.4965 & 0.7153 & 1.0000 \end{pmatrix}$$

Matrix of average correlations between outputs in case when weights of neighboring oscillators are randomly chosen.

$$\mathbf{CCor_{avg}} = \begin{pmatrix} 1.0000 & 0.4722 & 0.5169 & 0.5335 \\ 0.4722 & 1.0000 & 0.5786 & 0.5498 \\ 0.5169 & 0.5786 & 1.0000 & 0.7380 \\ 0.5335 & 0.5498 & 0.7380 & 1.0000 \end{pmatrix}$$

Experiment 8.

Experiment setup:

```
Input=[Weights=Random;
Noise=true;
Recurrency=6;
Colors={Red;Green};
Number of iterations = 10];
```

Results are presented at the Fig.5.65. Matrix of correlations between outputs.

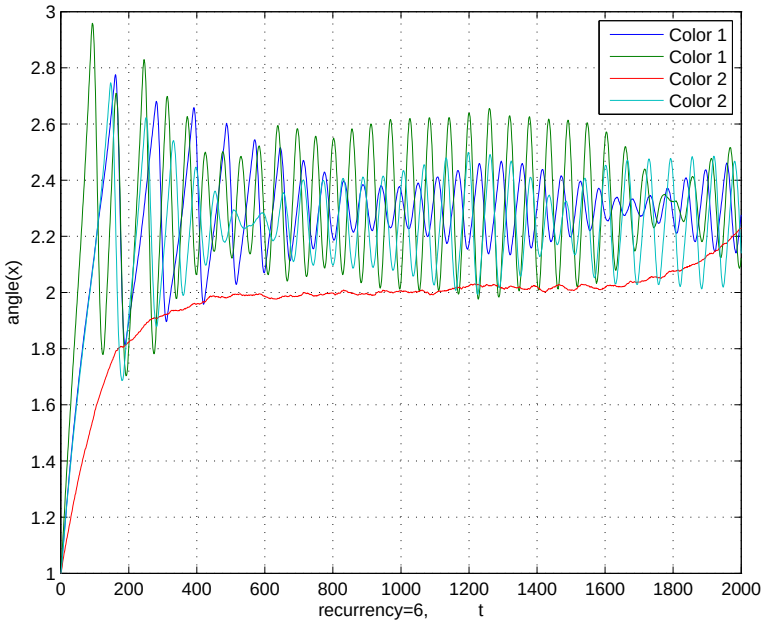

Figure 5.65: $x(t)$ for recurrency $= 6$.

$$\mathbf{CCor} = \begin{pmatrix} 1.0000 & 0.2705 & 0.6526 & 0.5768 \\ 0.2705 & 1.0000 & 0.3646 & 0.4840 \\ 0.6526 & 0.3646 & 1.0000 & 0.6154 \\ 0.5768 & 0.4840 & 0.6154 & 1.0000 \end{pmatrix}$$

Matrix of average correlations between outputs in case when weights of neighboring oscillators are randomly chosen.

$$\mathbf{CCor_{avg}} = \begin{pmatrix} 1.0000 & 0.4349 & 0.4284 & 0.4227 \\ 0.4349 & 1.0000 & 0.3963 & 0.4049 \\ 0.4284 & 0.3963 & 1.0000 & 0.6359 \\ 0.4227 & 0.4049 & 0.6359 & 1.0000 \end{pmatrix}$$

Experiment 9.

Experiment setup:

```
Input=[Weights=Random;
Noise=true;
Recurrency=20;
Colors={Red;Green};
Number of iterations = 10];
```

Results are presented at the Fig.5.66. Matrix of correlations between outputs.

$$\mathbf{CCor} = \begin{pmatrix} 1.0000 & 0.4653 & 0.5122 & 0.5554 \\ 0.4653 & 1.0000 & 0.3833 & 0.2354 \\ 0.5122 & 0.3833 & 1.0000 & 0.5788 \\ 0.5554 & 0.2354 & 0.5788 & 1.0000 \end{pmatrix}$$

Matrix of average correlations between outputs in case when weights of neighboring oscillators are randomly chosen.

$$\mathbf{CCor_{avg}} = \begin{pmatrix} 1.0000 & 0.5597 & 0.3447 & 0.3953 \\ 0.5597 & 1.0000 & 0.2906 & 0.3089 \\ 0.3447 & 0.2906 & 1.0000 & 0.6967 \\ 0.3953 & 0.3089 & 0.6967 & 1.0000 \end{pmatrix}$$

Experiment 10. Experiment setup:

```
Input=[Weights=Random;
Noise=true;
Recurrency=50;
Colors={Red;Green};
Number of iterations = 10];
```

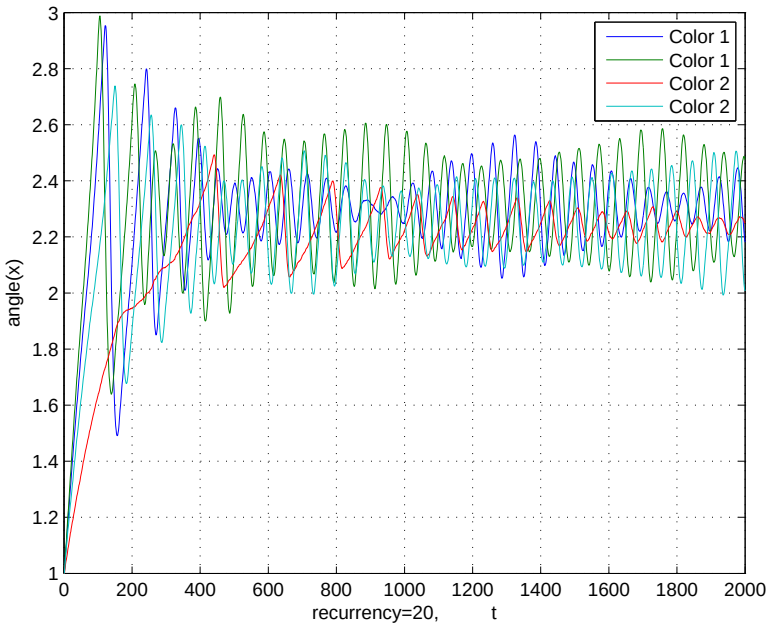

Figure 5.66: $x(t)$ for recurrency $= 20$.

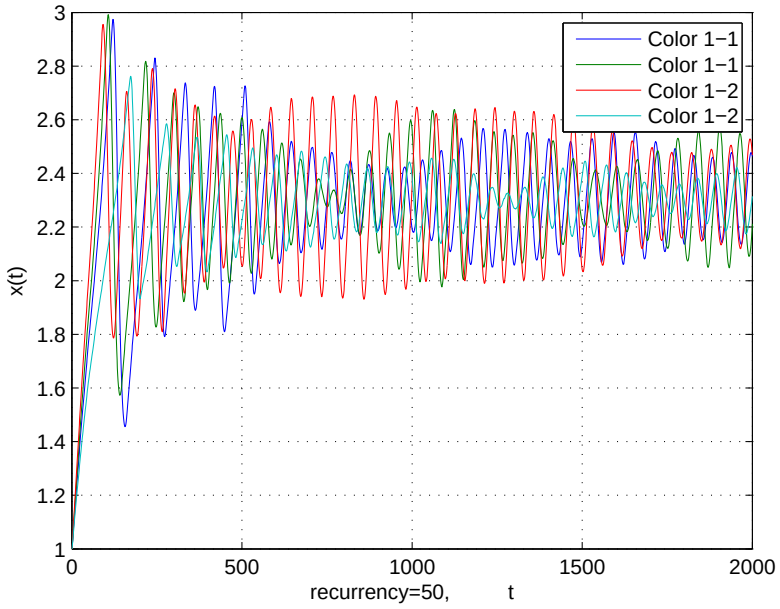

Figure 5.67: $x(t)$ for recurrency $= 50$.

Results are presented at the Fig. 5.67 Matrix of correlations between outputs.

$$\mathbf{CCor} = \begin{pmatrix} 1.0000 & 0.6637 & 0.2400 & 0.3263 \\ 0.6637 & 1.0000 & 0.1725 & 0.4310 \\ 0.2400 & 0.1725 & 1.0000 & 0.5724 \\ 0.3263 & 0.4310 & 0.5724 & 1.0000 \end{pmatrix}$$

Matrix of average correlations between outputs in case when weights of neighboring oscillators are randomly chosen.

$$\mathbf{CCor_{avg}} = \begin{pmatrix} 1.0000 & 0.3615 & 0.3850 & 0.4569 \\ 0.3615 & 1.0000 & 0.3691 & 0.4236 \\ 0.3850 & 0.3691 & 1.0000 & 0.6602 \\ 0.4569 & 0.4236 & 0.6602 & 1.0000 \end{pmatrix}$$

Results.
The results of this work are similar to the ones in the work of Maye, but for complex-valued inputs. Oscillators that represent the same segment (neighboring oscillators) produce similar outputs in terms of phase. Correlations are close to unity when the weights between neighboring oscillators are equal. If the weights are different, the outputs can differ from each other in both amplitude and phase, but in general they have the similar behavior. If the same light signal enters the system (when there are two pairs of neighboring oscillators), the outputs have high correlation at the beginning (coincide). Thus, the oscillators, which are not neighbors but have the same input signal (light or color), synchronize. The further work is to find the architecture of the neural network based on the recurrent complex-valued neural network, which will simulate the synchronization by the interaction of complex numbers instead of solving the system of differential equations as stated above.

5.9 Financial Time Series Forecasting

Prediction of the financial time series, forecasting of the leading indexes is a very important topic for the investment companies and can play an important role in the modern economics. The econophysics is a relatively new direction which is trying to build the physical models for the econometrical tasks. The main idea of econophysics is to apply the physical models to the economic data which is numerous. One of the examples of the application of the HCVNN is the prediction of the leading USA index - Dow Jones Industrial Average (DJIA). For this purpose the data in this index was taken from DJIA website [22]. This web resource allows downloading the data for free and using it for private needs. The data used for the DJIA modeling was taken for the period 01 AUG 2007 - 01 AUG 2011.

Market modeling with State Space Models

In order to predict the market one should believe that the market itself is a dynamical system which develops based on internal and external influences. The external reasons should be 'weak' in comparison to the internal ones. Thus market is a closed dynamical system which can be learned with state space models. Since the amount of the variables which drive the market is unknown one should make some assumptions or take rather big network to model the market. According to Zimmermann [73] each agent (trader) at the market plays as a neuron in the neural network which is summing the information flow with some weights and then makes biased decision in order to maximize its profit. Thus neural network is an ideal tool for the market modeling, which is a huge neural network of neurons that interact in a random way (kind of reservoir computing, which is a different topic and is enough for another thesis). State Space Models like the HCVNN can learn the market and then try to predict it for several iterations ahead, making the time teacher forcing we can correct the forecast in its angle part thus making the overall error smaller.

Results

The network was trained for 5000 epochs. The size of the HCVNN was the following: 30 states, each state has 30 hidden neurons. Activation function was selected to be *tanh*. Network had 3 outputs. All DJIA values were normalized for $[0, 1]$ interval. All results are presented for the normalized values. The figure below 5.68 shows the training set results. The figure below 5.69 shows the results

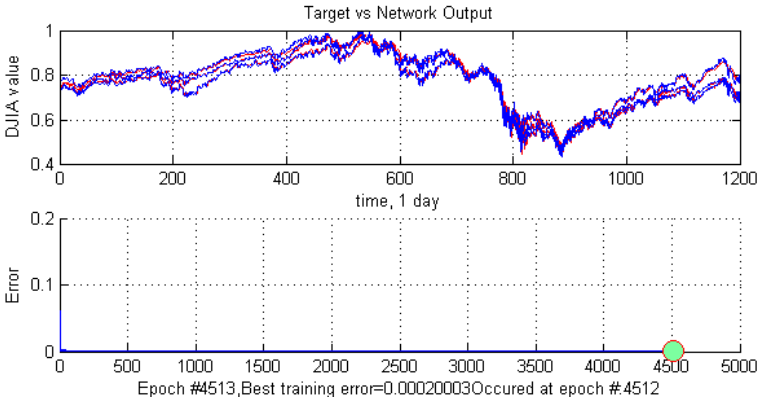

Figure 5.68: Ideal transformer (http://en.wikipedia.org/wiki/Transformer)

for the 20 steps prediction. At each prediction iteration previous predicted value was used, which means that this is pure 20 steps prediction ahead. At each iteration of prediction time teacher forcing has been applied. The Fig. 5.69

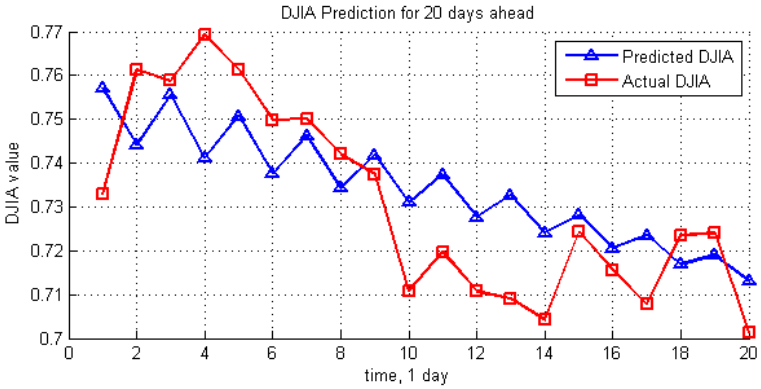

Figure 5.69: DJIA prediction for 20 steps ahead.

shows the results for the 20 steps prediction for another period of the DJIA. In

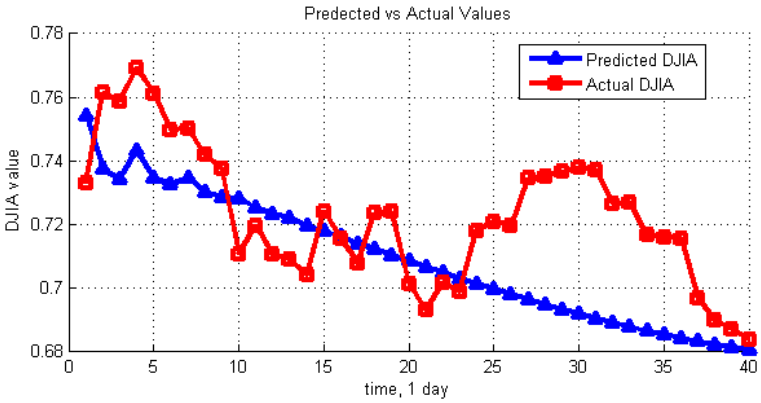

Figure 5.70: DJIA prediction for 20 steps ahead.

order to estimate the statistics the RMS and R^2 values have been used. The statistics is presented below:

- R2 is (angle)): 1
- RMS is (angle): 0
- R2 is (abs): 0.35
- RMS is (abs): 0.0002

The ideal statistics for the angle means that the teacher forcing was used, since we substitute the phase information on time with the target one. The statistics for the absolute part of the output shows that we have explained 35% of the data

behavior, which is a good result for the financial time series.

Conclusions.

The results above show that it is possible to explain 35% of the data behavior with a relatively simple and small model. One should admit that in order to predict the data better one should take into account all the variables which influence the resulting data. In our case for the prediction of the DJIA one would need the data on all its shares process, combine them in one dataset and try to predict all 30 shares prices. Such approach would simplify the prediction for the Neural Network.

Another important issue is the presence of the teacher forcing, since without the teacher forcing the models gives $R2 = 0.25$ and $RMS = 0.0004$, which means that time teacher forcing improves the prediction by 30% which is quite much, especially for the financial predictions.

6

Conclusions and Outlook

This thesis can be used as a guide to people willing to apply complex-valued neural networks. This thesis contains a strictly mathematical description of network initialization as well as a complete description of the system identification for complex-valued networks in comparison to real-valued networks. This thesis contains a solution to the nasty function properties of traditional complex functions. This thesis also contains several receipts on how to overcome the majority of problems faced by complex networks. At the end of this thesis several real world examples of the complex-valued networks are given along with the advantages of complex-valued neural networks.

6.1 Feed-Forward Architectures

As it was shown in the paper by H.-G. Zimmermann, A. Minin, V. Kusherbaeva, Comparison of the Complex-Valued and Real-Valued Neural Networks Trained with Gradient Descent and Random Search Algorithms, ESANN2011, Computational Intelligence and Machine Learning, Bruges, Belgium, 2011,pp. 213-218., complex-valued neural networks have no advantages ever in comparison to the real ones. Both produce the same statistical results and can be used depending on the input data. Both networks have very good convergence and experience similar problems such as over fitting. In this thesis these networks were compared. It was shown that the error back-propagation in such networks is much similar, despite the fact that all transpose operations must be replaced by a conjugated transpose. The activation functions remain the same. However, due to unboundness and singularities of these functions, great care must be taken with the complex representations of the functions.

6.2 Recurrent Architectures

It has been shown in H.-G. Zimmermann, A. Minin, V. Kusherbaeva, Historical Consistent Complex-Valued Recurrent Neural Network, Artificial Neural Networks and Machine Learning - ICANN 2011 - 21st International Conference on Artificial Neural Networks, Espoo, Finland, June 14-17, 2011, Proceedings, Part I. Volume 6791 of Lecture Notes in Computer Science, pages 185-192, Springer, 2011, that recurrent complex-valued architectures have clear advantages over the real-valued recurrent ones. Especially interesting is the complex-valued historical consistent neural network (HCVNN). In the paper by Minin A., Knoll A., Zimmermann H.-G.,Complex-Valued Recurrent Neural Network: From Architecture to Training, submitted to the Journal of Signal and Information Processing, it has been shown that complex-valued architectures have higher forecasting horizons and moreover the technique called time teacher forcing (further TTF), was invented in the following thesis. This TTF allows the control of the forecasting time, which allows the forecast to be corrected at each forecasting iteration and thus reduces the overall output error. One can also force the network to do the continuous time modeling. For this purpose, the TTF method should be used in a different manner. After the network has been trained with some data steps, the TTF should be used on the production set to "force" the network to produce a forecast for the needed time stamp. The network will need some time to adopt to the new time stamp, after some iterations it will tag in producing valuable results.

6.3 Brain modeling

In the paper of Minin A., Knoll A., Zimmermann H.-G., Complex-Valued Artificial Recurrent Neural Network as a Novel Approach to Model the Perceptual Binding Problem, submitted to ESANN2012 conference, it was shown that CVNN can solve the classical binding problem for oscillating units. It has also been shown that RCVNN can simulate the synchronization of weight coupling the information about the shape of the object together with its phase. Moreover, the memory, the generalization and the synchronization can be simulated by the interaction of the complex value phases via CVRNN.
In the future, more work will be done in this direction. In the future, the goal is to be able to carry out online training of the CVRNN so that information on the object can be collected by inspecting the object from several vantage points similarly to how small children learn objects. This will allow to create a big data base of images. It will then be possible to demonstrate the generalization capabilities of RCVNN for multiple object classes and to estimate the memory

capacity of the RCVNN.

6.4 Outlook

In sumary, Complex-Valued Neural Networks have a wide range of applications due to the unique properties arising from their more physical way of data processing. Time Teacher Forcing enables the modeling of continuous time systems. Recurrent architectures can model the human memory and the ability of the human brain to generalize objects. Further research must be done to establish new architectures to construct more complicated models of the human brains. Moreover, the suggested networks allow neural networks to be physically modelled. Complex Neural Networks have the potential to drastically change the way artificial intellegence is done.

References

[1] Barron A. **Approximation and Estimation Bounds for Superpositions of a Sigmoidal Function**. *IEEE transactions on Information Theory 39(3), pp.930-945*, 1994. 4, 33

[2] Fiszelew A. and Britos P. **Finding optimal neural Network Structure using genetic algorithms**. *Advances in Computer Science and Engineering, Research in Computer Science 27, pp. 15-24*, 2007. 4

[3] Maye A. **Correlated neuronal activity can represent multiple binding solutions**. *Neurocomputing*, **52-54**:73–77, 2003. 135

[4] Minin A. **Predictive Classifiers Based on Machine Learning Methods**. *Master Diploma Thesis*, page 98, 2008. 30

[5] Minin A. and Lang B. **Comparison of Neural Networks Incorporating Partial monotonicity by Structure**. *ICANN 2008, Springer, vol.2, pp. 597-607*, 2008. 1, 2

[6] Minin A. and Lang B. **Comparison of Neural Networks Incorporating Partial monotonicity by Structure**. *ICANN 2008, Lecture Notes on Computer Science*, **2**:597–606, 2008. 26

[7] Minin A. and Lang B. **Monotnic Recurrent Bounded Derivative Neural Network**. *17th European Symposium on Artificial Neural Networks, D-Side, pp.385-390*, 2008. 1, 2

[8] Minin A. and Mokhov I. **Advanced forecasting technique for rotating machinery**. *Upravlenie Bolshimi Sistemami V: VGTU*, **1**:121–128, 2008. 26

[9] Voldek A. **Electricheskie mashiny**. *Energiya*, 1978. 119, 120

[10] Marco G. Altman E. and Varetto F. **Corporate Distress Diagnosis: Comparisons Using Linear Discriminant Analysis and Neural Networks**. *Journal of Banking and Finance 18, pp. 505-529*, 1994. 72, 74

[11] B. Lang A.Minin, Y. Kuperin. **Increasing the quality of neural forecasts with the help of EMD**. *Artificial Intelligence Systems(AIS - 2007) and CAD's - 2007*, **4**:68–74, 2007. 2

[12] Lang B. **Monotonic Multi-layer Perceptron Networks as Universal Approximators. Formal Models and Their Applications**. *ICANN 2005, Springer, vol. 3697, pp. 31-37*, 2005. 2, 28

[13] A. Minin I. Mokhov Y. Kuperin B. Lang, T. Poppe and A. Mekler. **Neural Clouds for Monitoring of Complex Systems**. *Journal of Optical Memory and Neural Networks*, **17**[3]:183–192, 2008. 2, 26

[14] Benchmarks. **http://www.fizyka.umk.pl/neural/node12.html**. 63

[15] Bishop C. **Pattern Recognition and Machine Learning**. page 738, 2006. 1, 38

[16] Minin A. Zimmermann H. Knoll A. Chistyakov Y., Kholodova E. **Modeling of Electric Power Transformer Using Complex-Valued Neural Networks**. *Smart Grid Conference*, 2011. 121

REFERENCES

[17] Kuhn R. Coolen A. and Sollich P. **Theory of Neural Information Processing Systems**. page 569, 2005. 1, 38

[18] Brandwood D. **A complex gradient operator and its application in adaptive array theory**. *IEE Proceedings, F: Communications, Radar and Signal Processing*, **130**[1]:1116, 1983. 15

[19] Johnson D. **Optimization theory**. pages 1–3, 2008. 14

[20] Wolpert D. **A benchmark for how well neural net generalize**. *Biological Cybernetics, Vol 61/4, pp. 303-313*, 1989. 5, 63

[21] G.E. Hinton D.E. Rumelhart and R.J. Williams. **Learning internal representations by error propagation**. *Computational Models Of Cognition And Perception Series*, page 318362, 1986. 39

[22] DJIA. **http://finance.yahoo.com**. 147

[23] Lorenz E. **Deterministic nonperiodic flow**. *Lecture Supplement*, **20**:130141, 1963. 82

[24] Mukovskiy A. Erlhagen W. and Knoll A. **Action understanding and imitation learning in a robot-human task**. *In Proc. IEEE International Conference on Neural Networks (ICANN), pp. 261-268*, 2005. 1, 5

[25] Buzsaki G. and Draguhn A. **Neuronal Oscillations in Cortical Networks**. *Science,Vol. 304, pp. 5679*, 2004. 134

[26] Shelley G. and Wallace F. **The Relation between U.S. Money Growth and Inflation: Evidence from a Band Pass Filter**. *Economics Bulletin*, **5**[8]:1–13, 2004. 1

[27] Kalra P. Gangal A. and Chauhan D. **Performance Evaluation of Complex Valued Neural Networks Using Various Error Functions**. *World Academy of Science, Engineering and Technology*, **29**, 2007. 22, 70

[28] P. Grassberger and I. Procaccia. **Measuring the strangeness of strange attractors**. *Physica D*, **9 (1-2)**:189–208, 1983. 82

[29] Kay H. and Ungar L. **Estimating monotonic functions and their bounds**. *AIChE J.*, **46**:2426, 1997. 1

[30] Leung H. and Haykin S. **The Complex Backpropagation Algorithm**. *IEE Proceedings, Transactions on Signal Processing*, **39**[9]:2101–2104, 1991. 33

[31] A. Minin H.-G. Zimmermann and V. Kusherbaeva. **Historical consistent complex valued recurrent neural network**. *Lecture Notes in Computer Science, ICANN 2011*, **1**:185–192, 2011. 119

[32] Zimmermann H.G. and Rehkugler H. **Neuronale Netze als Entscheidungskalkl**. *Neuronale Netze in der Ekonomie*, pages 1–87, 1994. 4

[33] Zimmermann H.G. and Neuneier R. **Neural Network Architectures for the Modeling of Dynamical Systems**. *Field Guide to Dynamical Rec. Networks*, 2000. vii, 4, 38, 39, 40, 41, 46, 127

[34] Breakspear M. Honey CJ., Kter R. and Sporns O. **Network structure of cerebral cortex shapes functional connectivity on multiple time scales**. *Proc Natl. Acad Sci. USA*, **104**:10240–10245, 2007. 56

[35] Agaev I. and Kuperin Yu. **Multifractal Analysis and Local Hoelder Exponents Approach to Detecting Stock Markets**

Crashes. *http://xxx.lanl.gov/ftp/cond-mat/papers/0407/0407603.pdf*, 2004. 1

[36] Alexander I. **From Wisard to Magnus: A Family of Weightless Virtual Neural Machines, Ram based Neural Networks**. *Progress in Neural Processing 9, pp. 18-30*, 2005. 1, 5

[37] Gomez I. and Franco L. **Neural Network Architecture Selection: Can function complexity help?** *preprint*, 2009. 1, 4

[38] Anil K. Mao J. and Mohiuddin K. **Artificial Neural Networks: A Tutorial**. *Computer*, **29**[3]:31–44, 1996. 27

[39] Elman J. **Finding Structure in Time**. *Cognitive Science*, **14**[2]:179–211, 1990. vii, 38, 39

[40] Moody J. **Prediction Risk and Architecture Selection for Neural Networks**. *From statistics to Neural Networks: Theory and Pattern recognition applications, Springer*, 1994. 4, 33

[41] Sill J. **Monotonic Networks**. *Advances in Neural Information Processing Systems, vol. 10, Cambridge, MA, pp. 661-667*, 1998. 2

[42] Sill J. and Abu-Mostafa Y. **Monotonicity hints**. *Advances in Neural Information Processing Systems, vol. 9, Cambridge, MA, pp. 634-640*, 1997. 2

[43] Kreutz K. **The Complex Gradient Operator and the CR-Calculus**. *Lecture Supplement*, 2005. 14, 33

[44] Perlovsky L. **Neural Networks and Intellect, Using Model Based Concepts**. *Oxford University Press, p. 469*, 2001. 128

[45] Prechelt L. **Some notes on neural learning algorithm benchmarking**. *Neurocomputing 9, Elsevier, pp 343-347*, 1995. 1, 5, 63, 64, 73

[46] Mokhov I. Lang B. and Minin A. **Neural Clouds for Monitoring of Complex Plant Conditions**. *Neuroinformatics - 2008, Annual conference, M.: MIFI*, **1**:125–132, 2008. 26

[47] Poppe T. Lang B. and Runkler T. **Application of artificial intelligence in steel processing**. *Automatisierung in der Metallurgie, Heft 89 der Schriftenreihe der GDMB*, 2001. 1

[48] Zhaoping Li. **CA neural model of contour integration in the primary visual cortex**. *Neural Computation*, **10**:903–940, 1998. 135

[49] R.M. May. **Simple mathematical models with very complicated dynamics**. *Nature*, **261 (5560)**:459–467, 1976. 82

[50] Knoll A. Mayer H., Gomez F. and Schmidhuber J. **A system for robotic heart surgery that learns to tie knots using recurrent neural networks**. *Advanced Robotics, Special Issue: Selected Papers from IROS 2006, 21(14), pp. 1521-1537*, October 2007. 1, 5

[51] Kuperin Yu. Minin A. and Mekler A. **Classification of EEG Recordings with Neural Clouds**. *Neuroinformatics - 2008, Annual conference, M.: MIFI*, **1**:115–125, 2008. 26

[52] I. Mokhov and A. Minin. **Advanced forecasting and classification technique for condition monitoring of rotating machinery**. *Proceedings of the 8th International Conference On Intelligent Data Engineering and Automated Learning (IDEAL'07)*, **1**:37–46, December 16-19, 2007. 2

[53] Ciftcioglu O. and Turkcan E. **Neural network benchmark**. *Symposium on Nuclear reactor surveillance and Diagnostics,SMORN 7*, 1995. 5, 63

REFERENCES

[54] Dodge P. **The Oxford Dictionary of Statistical Terms**. *OUP., ISBN 0199206139*, 2003. 65

[55] Neuneier R. and Zimmermann H.G. **How to Train Neural Networks**. *Neural Networks: Tricks of the Trade*, pages 373–423, 1998. vii, 4, 27, 31, 32, 37

[56] German-Galkin S. and Kardonov G. **Electricheskie mashiny. Laboratornie raboti na PK**. 2003. 119

[57] Grasberg S. **Open Systems**. *Center of Adaptive Systems and department of Cognitive and Neural Systems*, **4**, 1997. 1, 43

[58] Haykin S. **Neural Networks: a comprehensive foundation**. *Prentice hall*, 1994. 4, 30, 33, 38

[59] Waugh S. **Generating data sets for benchmarking**. *Neural Networks 4, IEEE, pp.2145-2148*, 1995. 5, 63

[60] Zimmermann H.G. Siekmann S., Neuneier R. and Kruse R. **Neuro Fuzzy Systems for Data Analysis**. *Computing with Words in Information Intelligent Syst.*, pages 35–74, 1999. 4

[61] Johansen A. Sornette D. and Bouchaud J.-P. **Stock Market Crashes. Precursors and Replicas**. *J. Phys. I France*, **6**:167–175, 1996. 1

[62] Kondo T. and Ueno J. **Multi-layered GMDH-type NN Self-selecting optimum NN architecture and its application to 3-dimensional medical image recognition of blood vessels**. *Innovative Computing, Information and Control*, **4/1**:175–187, 2008. 4

[63] Schillen T. and Konig P. **Finding by Temporal Structure in Multiple Feature Domains of an Oscillatory Neuronal Network**. *Biological Cybernetics*, **70**:397–405, 1994. 135

[64] Grandjean B. Tarca L.A. and Larachi F. **Embedding monotonicity and concavity information in the training of multiphase flow neural network correlations by means of genetic algorithms**. *Computers and Chemical Engineering, vol. 28, issue 9, pp. 1701-1713*, 2004. 1, 2

[65] Guiver J. Turner P. and Brian L. **Introducing The State Space Bounded Derivative Network For Commercial Transition Control**. *Proceedings of the American Control Conference*, 2003. 1

[66] Kusherbaeva V. and Sushkov Yu. **Statistical investigation of Random Search**. *Stochastic optimization in information science, pp. 21-36*, 2007. 54

[67] OMP-10 web link. **http://www.orion-nt.ru/cat/transf/tr-hmel/tr-omp1.htm**. 121

[68] Wikipedia. **Complex Analysis**. 2011. 9

[69] Sushkov Yu. and Abakarov A. **The algorithm of adaptive random search for discrete-continuous optimization**. *Proc. of the 5th St. Petersburg Workshop on Simulation, pp. 11-17*, 2005. 54

[70] Erdem C. Zimmermann H.G., Mueller A. and Hoffmann R. **Prosody Generation by Causal-Retro-Causal Error Correction Neural Networks**. *Proceedings Workshop on Multi-Lingual Speech Communication, ATR*, 2000. 4

[71] Neuneier R. Zimmermann H.G. and R. Grothmann. **Multi Agent Market Modeling of FX-Rates**. *Advances in Complex Systems (ACS); Special Issue*, **4**[1], 2001. 4

[72] Neuneier R. Zimmermann H.G. and Grothmann R. **Modeling of the German Yield Curve by Error Correction Neural Networks**. *Proceedings IDEAL2000*, pages 262–267, 2000. 4

[73] Neuneier R. Zimmermann H.G. and Grothmann R. **Active Portfolio Management based on Error Correction Neural Networks**. *Proceedings of Neural Information Processing (NIPS)*, 2001. 4, 148

[74] Neuneier R. Zimmermann H.G. and Grothmann R. **Modeling of Dynamical Systems by Error Correction Neural Networks**. *Modeling and Forecasting Financial Data, Techniques of Nonlinear Dynamics*, 2002. 4

[75] Neuneier R. Zimmermann H.G. and Grothmann R. **Undershooting: Modeling Dynamical Systems by Time Grid Refinements**. *Proceedings of European Symposium on Artificial Neural Networks (ESANN) 2002*, 2002. 4

[76] R. Zimmermann H.G., Neuneier and Grothmann R. **An Approach of Multi-Agent FX-Market Modeling based on Cognitive Systems**. *Proc. of the Int. Conference on Artificial Neural Networks (ICANN)*, 2001. 4

[77] Schafer A.M. Zimmermann H.G, Grothmann R. and Tietz Ch. **Dynamical Consistent Recurrent Neural Networks**. *Proc. of the Int. Joint Conference on Neural Networks (IJCNN),pp.1537 - 1541*, 2005. viii, 47, 48, 49

[78] Tietz Ch. Zimmermann H.G. and Grothmann R. **Yield Curve Forecasting by Error Correction Neural Networks and Partial Learning**. *Proceedings of European Symposium on Artificial Neural Networks (ESANN)*, 2002. 4